

3-29-2007

Maximizing Manipulation Capabilities of Persons with Disabilities Using a Smart 9-Degree-of-Freedom Wheelchair-Mounted Robotic Arm System

Redwan M. Alqasemi
University of South Florida

Follow this and additional works at: <https://scholarcommons.usf.edu/etd>

 Part of the [American Studies Commons](#)

Scholar Commons Citation

Alqasemi, Redwan M., "Maximizing Manipulation Capabilities of Persons with Disabilities Using a Smart 9-Degree-of-Freedom Wheelchair-Mounted Robotic Arm System" (2007). *Graduate Theses and Dissertations*.
<https://scholarcommons.usf.edu/etd/599>

This Dissertation is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.

Maximizing Manipulation Capabilities of Persons with Disabilities Using a
Smart 9-Degree-of-Freedom Wheelchair-Mounted Robotic Arm System

by

Redwan M. Alqasemi

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Mechanical Engineering
College of Engineering
University of South Florida

Major Professor: Rajiv Dubey, Ph.D.
Shuh-Jing Ying, Ph.D.
Craig Lusk, Ph.D.
Wilfrido Moreno, Ph.D.
Kandethody Ramachandran, Ph.D.

Date of Approval:
March 29, 2007

Keywords: dof, adl, control, robot, rehabilitation, mobility, redundancy

© Copyright 2007, Redwan M. Alqasemi

Note to Reader

The original of this document contains color that is necessary for understanding the data. The original dissertation is on a file with the USF library in Tampa, Florida.

Dedication

To my wife, Ola, for her unconditional love and support in every possible way and for the sacrifices she made of her own needs and comfort for mine. To my children, Hiba, Lama, Rama, Ryan and Dana, who gave me that wonderful feeling that I see and feel every day when I came back from a long day in the office. Without them, I can't have any color, taste or joy in my life that charges me for success. Thank you for enduring my absence for countless days and nights.

To my loving mother and father, who gave me all the support and encouragement I needed to continue my education. I will never forget the sleepless nights you had for my comfort, and the prayers you made for my success.

To my advisor, Dr. Dubey, who was like my older brother, giving me the advice when I need it and helping me in any way he can above and beyond his duties. Your inspiration helped me achieve this work. You are truly a great professor and role model.

To my brothers and sisters who never spared any opportunity to help me when I needed their help. To my relatives and friends who gave me the comfort and confidence whenever I needed them.

Above all, to God, who showered me with his countless blessings and guided me to the right path and made me succeed throughout the way with all the obstacles I faced, thank you God.

Acknowledgments

I would like to express my gratitude to my advisor, Dr. Rajiv Dubey for giving me the precious opportunity to work with him and for sharing his knowledge and experience with me in both teaching and research. His guidance and immense patience throughout the course of my research are greatly appreciated. I would also like to thank the members of my committee Dr. Shuh-Jing Ying, Dr. Craig Lusk, Dr. Wilfrido Moreno and Dr. Kandethody Ramachandran for their valuable comments to this research.

I would like to gratefully acknowledge the important contribution and support of the Florida Department of Education, the Division of Vocational Rehabilitation, and the Center for Rehabilitation Engineering and Technology at the University of South Florida, especially Mr. Stephen Sundarrao, who provided a great help in conducting experiments and tests with people with disabilities. Many thanks go to the members of the Rehabilitation Robotics group, including Eduardo Veras, Edward McCaffrey, Kevin Edwards, Mayur Palankar, Sebastian Mahler and Steven Colbert who added some important contributions to this research.

Special thanks and appreciation go to Dr. Emanuel Donchin and Dr. Yael Arbel of the Department of Psychology at USF for lending their BCI-2000 hardware and support to integrate the BCI system into this research. I would also like to express my gratitude to Vilma Fitzhenry, Susan Britten, Shirley Tervort, Wes Frusher, Robert Smith, Thomas Gage, and James Christopher for their support in paper work and machining.

Table of Contents

List of Tables	vii
List of Figures	viii
Abstract	xvi
Chapter 1: Introduction	1
1.1. Motivation	1
1.2. Dissertation Objectives	3
1.3. Dissertation Outline	4
Chapter 2: Background	6
2.1. History of Rehabilitation Robotics	6
2.2. Rehabilitation Robotics Classification	7
2.2.1. Workstation Robotic Arms	7
2.2.2. Wheelchair-Mounted Robotic Arms	12
2.2.3. Mobile / Assistant Robots	15
2.2.4. Robots in Therapy	18
2.2.5. Smart Wheelchairs / Walkers	19
2.3. Commercial Wheelchair-Mounted Robotic Arms	20
2.3.1. The Manus	20
2.3.2. The Raptor	21
2.4. Robot Control	22
2.4.1. Redundant Robot Control	22
2.4.2. Mobile Robot Control	26
Chapter 3: Control Theory of Redundant Manipulators	41
3.1. Introduction	41
3.2. Terminology	42
3.3. Redundant Manipulators Problem Formulation	42
3.3.1. Frames of References	43
3.3.2. Denavit-Hartenberg Parameters	44
3.4. Forward Kinematics Equations	46
3.4.1. Link Transformation Matrices	46
3.4.2. Velocity Propagation and the Jacobian	48

3.5. Inverse Kinematic Equations	51
3.5.1. Closed Form Solutions	51
3.5.2. Manipulability Ellipsoid	53
3.5.3. Numerical Solutions	54
3.5.4. Redundancy Resolution	55
3.5.5. Optimization Criteria	56
3.6. Summary	57
Chapter 4: Mobility Control Theory	59
4.1. Introduction	59
4.2. Terminology	60
4.3. Mobility Problem Formulation	60
4.3.1. Frame Assignment	61
4.3.2. Wheelchair's Important Dimensions	62
4.4. Homogeneous Transformation Relations	64
4.4.1. Driving Wheels' Motion and the Turning Angle	64
4.4.2. The Radius of Curvature	66
4.4.3. Point-to-Point Transformation of the Wheelchair	69
4.4.4. Transformation to the Robotic Arm's Base	72
4.5. Wheelchair Velocities	72
4.5.1. Wheelchair Velocity Mapping to the Robotic Arm Base	73
4.5.2. Mapping the Driving Wheels' Velocities to the Wheelchair	75
4.6. Wheelchair's General Jacobian	77
4.7. Trajectory Options	78
4.8. Operator's Safety Issues	79
4.9. Summary	80
Chapter 5: Control and Optimization of the Combined Mobility and Manipulation	81
5.1. Introduction	81
5.2. Terminology	82
5.3. WMRA Assembly and Problem Definition	82
5.4. Kinematics of the Combined WMRA System	83
5.5. Jacobian Augmentation and Resolved Rate Equations Generation	84
5.6. Jacobian Changes Based on the Control Frame	88
5.6.1. Ground-Based Control	88
5.6.2. Wheelchair-Based Control	88
5.6.3. End-Effector Based Control (Piloting Option)	89
5.7. Jacobian Inversion Methods and Singularities	89
5.7.1. Inverting Using Pseudo Inverse	90
5.7.2. Inverting Using Singularity-Robust Inverse	90
5.8. Optimization Methods with the Combined Jacobian	91
5.8.1. Criteria Functions and Minimizing Euclidean Norm of Errors	92
5.8.2. Weighted Least Norm Solution	94
5.8.3. Joint Limit Avoidance	95
5.8.4. Obstacle Avoidance	99

5.8.5. Safety Conditions	99
5.8.6. Unintended Motion Effect Based on the Optimization Criteria	100
5.9. Optional Combinations for the Resolved Rate Solution	101
5.10. State Variable Options in the Control Algorithm	103
5.10.1. Seven Robotic Arm Joints, Left wheel and Right Wheel Variables	104
5.10.2. Seven Robotic Arm Joints, Forward and Rotational Motion of the Wheelchair	105
5.11. Trajectory Generation	109
5.11.1. Generator of a Linear Trajectory	109
5.11.2. Generator of a Polynomial Trajectory	113
5.11.3. Generator of a Polynomial Trajectory with Parabolic Blending Factor	115
5.12. Control Reference Frames	116
5.12.1. Ground Reference Frame	117
5.12.2. Wheelchair Reference Frame	118
5.12.3. End-Effector Reference Frame	119
5.13. Summary	120
 Chapter 6: User Interface Options	 121
6.1. Introduction	121
6.2. User Interface Clinical Testing	121
6.2.1. Representative ADL Tasks Used for the Clinical Study	122
6.2.2. The Tested User Interfaces	124
6.2.3. Population of the Chosen Users with Disabilities	125
6.2.4. Clinical Test Results on User Interfaces	126
6.3. The New WMRA User Interfaces Used	128
6.3.1. Six-Axis, Twelve-Way SpaceBall	128
6.3.2. Computer Keyboard and Mouse	129
6.3.3. Touch Screen on a Tablet PC	130
6.4. The Brain-Computer Interface (BCI) Using P300 EEG Brain Signals	131
6.4.1. The P300 EEG Signal	131
6.4.2. The Use of the BCI	132
6.4.3. The BCI-2000 Interface to the New 9-DoF WMRA System	133
6.4.4. Testing of the BCI-2000 with the WMRA Control	134
6.5. Expandability of User Interfaces	135
6.5.1. Omni Phantom Haptic Device	136
6.5.2. Sip n' Puff Device	137
6.5.3. Head and Foot Switches	138
6.6. Summary	138
 Chapter 7: Testing in Simulation	 139
7.1. Introduction	139
7.2. User Options to Control the WMRA System	139

7.3. Changing the Physical Dimensions and Constraints of the WMRA System	142
7.4. Programming Language Packages Used	142
7.4.1. Programs in C++ Programming Language	144
7.4.2. Matlab Programming Environment	144
7.4.3. Simulation with Virtual Reality Toolbox	147
7.4.4. Graphical User Interface (GUI) Program	149
7.5. Comments on Interfacing Different Programs Together	150
7.6. Summary	151
Chapter 8: Simulation Results	153
8.1. Introduction	153
8.2. Simulation Cases Tested	153
8.3. Results and Discussion of the First Five Cases	155
8.3.1. WMRA Configurations in the Final Pose of the Simulation	158
8.3.2. Displacements of the Joint Space Variables	161
8.3.3. Velocities of the Joint Space Variables	167
8.3.4. Singularities and the Manipulability Measure	169
8.4. Results and Discussion of the Second Two Cases	172
8.5. More Simulation for Optimization Methods and Criterion Function Effects	178
8.6. Simulation of the Eight Implemented Optimization Control Methods for the Case of an Unreachable Goal	184
8.7. Summary	194
Chapter 9: Experimental Testbed and Field Tests	195
9.1. Introduction	195
9.2. The New 7-DoF Robotic Arm Design and Development	195
9.2.1. Design Goals	196
9.2.1.1. Weight	196
9.2.1.2. Mount Type	196
9.2.1.3. Stiffness	197
9.2.1.4. Payload	197
9.2.1.5. Reconfigurability	198
9.2.1.6. Power Supply and Consumption	198
9.2.1.7. Cost Constraint	198
9.2.1.8. User Interface	199
9.2.1.9. Degrees of Freedom	199
9.2.1.10. Actuation and Transmission Systems	199
9.2.1.11. DC Motors as Actuators	200
9.2.2. Kinematic Arrangements and Component Selection	200
9.2.2.1. Gearhead Selection	202
9.2.2.2. Motor Selection	203
9.2.2.3. Material Selection	204
9.2.2.4. Joint Design	204

9.2.2.5. Wrist Design	204
9.2.3. Final Design Testing and Specifications	205
9.3. The New 2-Claw Ergonomic Gripper Design and Development	208
9.3.1. Paddle Ergonomic Design	209
9.3.2. Actuation Mechanism	211
9.3.3. Component Selection	212
9.3.4. Final Design and Testing	216
9.4. Modification of a Standard Power Wheelchair	219
9.5. Controller Hardware	220
9.5.1. Controller Boards	222
9.5.2. Communication and Wiring	223
9.5.3. Safety Measures	224
9.6. Experimental Testing	225
9.7. Summary	228
Chapter 10: Conclusions and Recommendations	230
10.1. Overview	230
10.2. General Discussion	231
10.3. Recommendations	234
Chapter 11: Future Work	237
11.1. Introduction	237
11.2. Quick Attach-Detach Mechanism	237
11.3. A Single Compact Controller	238
11.4. A Sensory Suite	239
11.5. Real-Time Control	239
11.6. Bluetooth Wireless Technology for Remote Wireless Teleoperation	240
11.7. Sensor Assist Functions (SAFs)	240
11.8. Pre-Set ADL Tasks	241
References	243
Appendices	249
Appendix A. Hardware Components	250
A.1. Robotic Arm Gear Motors with Encoders	250
A.2. Harmonic Drive Gearheads	252
A.3. Wheelchair Selected Encoders	264
A.4. Wheelchair Selected Friction Wheels	267
A.5. Gripper's Actuation Motor	268
A.6. Gripper's Planetary Gearhead	269
A.7. Gripper's Optical Encoder	270
A.8. Gripper's Spur Gears	272
A.9. Gripper's Slip Clutch	273
A.10. PIC Servo SC Motion Controller Board	280
A.11. SSA-485 Smart Serial Adapter	283

Appendix B. Matlab Programs and Functions	292
B.1. VRML File of the Virtual Reality Control Code	292
B.2. Matlab Functions Listed Alphabetically	297
B.3. Matlab Main Script and GUI Main File	349
Appendix C. C++ Programs and DLL Library	401
C.1. DLL Library Functions	401
C.2. DLL Library Documentation	403

About the Author

End Page

List of Tables

Table 3.1:	The D-H Parameters of the New 7-DoF Robotic Arm Built at USF.	46
Table 9.1:	HD Systems Gearhead Selections for Each Joint.	202
Table 9.2:	Arm Deflections vs. Applied Load.	206
Table 9.3:	Power Usage.	206
Table 9.4:	Summary of the Robotic Arm Specifications.	208

List of Figures

Figure 2.1:	Puma 250 Arm.	8
Figure 2.2:	Handy-1.	9
Figure 2.3:	RAID Workstation.	10
Figure 2.4:	Robot Assistive Device.	11
Figure 2.5:	ProVAR System.	11
Figure 2.6:	Weston Arm.	13
Figure 2.7:	Asimov Arm.	14
Figure 2.8:	FRIEND Robotic System.	15
Figure 2.9:	MoVAR.	16
Figure 2.10:	MoVAID.	16
Figure 2.11:	TAURO Robotic System.	17
Figure 2.12:	MIT Manus System.	18
Figure 2.13:	Mouth Opening and Closing Device.	19
Figure 2.14:	iBOT (Left) and Segway (Right) Devices.	20
Figure 2.15:	Manus Arm.	21
Figure 2.16:	Raptor Arm.	22
Figure 2.17:	Redundancy Resolution without (Left) and with (Right) Obstacle Avoidance.	24

Figure 2.18:	The Robot Visual Servoing Application Using the QP Controller.	26
Figure 2.19:	Reference Frames Used for the Manipulation LIRMM.	27
Figure 2.20:	Cooperative Control System Setup.	28
Figure 2.21:	Mobile Manipulator Model.	29
Figure 2.22:	Wheeled Mobile Manipulator with Two Arms.	30
Figure 2.23:	Nomad XR4000 with the Puma560 Mounted on Top.	32
Figure 2.24:	Mobile Manipulator.	33
Figure 2.25:	Mobile Manipulator H2BIS.	33
Figure 2.26:	LAAS Mobile Manipulator.	35
Figure 2.27:	Mobile Manipulator.	36
Figure 2.28:	Interaction Control of the Mobile Manipulator.	38
Figure 2.29:	Trajectory Tracking for a Planar 2-DoF Robot on a Differential Mobile Base.	39
Figure 2.30:	Animation of the Motion of the End-Effector and the Platform Front Point.	40
Figure 3.1:	Joint-Link Kinematic Parameters.	43
Figure 3.2:	Solid Works Model of the New 7-DoF Robotic Arm Built at USF.	45
Figure 3.3:	Frame Assignments and Dimensions of the New 7-DoF Robotic Arm.	45
Figure 3.4:	Velocity Vectors of Neighboring Links.	49
Figure 3.5:	Manipulability Ellipsoid for a 7-DoF Manipulator in a 6-DoF Euclidean Space.	54
Figure 4.1:	Wheelchair Coordinate Frames and Dimensions of Interest.	62
Figure 4.2:	Traveled Distance of a Turning Wheel.	64

Figure 4.3:	Traveled Distance with Turning Angle.	65
Figure 4.4:	Radius of Curvature in Case 1.	67
Figure 4.5:	Radius of Curvature in Case 2.	67
Figure 4.6:	Radius of Curvature in Case 3.	68
Figure 4.7:	Radius of Curvature in Case 4.	69
Figure 4.8:	Point-to-Point Transformation of Frames.	70
Figure 4.9:	The Case When “ L_3 ” is Zero.	73
Figure 4.10:	The Case When “ L_2 ” is Zero.	74
Figure 4.11:	The Case When the Left Wheel is Stationary.	76
Figure 4.12:	The Case When the Right Wheel is Stationary.	76
Figure 4.13:	The Three Sub-Motions in Motion Planning of the Wheelchair.	79
Figure 5.1:	WMRA Coordinate Frames.	83
Figure 5.2:	Four Joint Limit Boundary Conditions.	98
Figure 5.3:	Linear Trajectory Generation.	112
Figure 5.4:	Polynomial Function of 3 rd Order for Variable Ramp with Time.	114
Figure 5.5:	Polynomial Function of 3 rd Order for Blended Variable Ramp with Time.	116
Figure 5.6:	Polynomial Trajectory Generation.	117
Figure 6.1:	Four Different ADL Tasks.	123
Figure 6.2:	Four-Way Joystick for Manus.	124
Figure 6.3:	Eight-Button Keypad for Manus.	124
Figure 6.4:	Eight-Way Joystick for Raptor.	125
Figure 6.5:	Clinical Testing of the Keypad by a Power Wheelchair User.	126

Figure 6.6:	Clinical Testing of the Joystick by a Power Wheelchair User.	127
Figure 6.7:	Twelve-Way SpaceBall.	129
Figure 6.8:	A Keyboard and a Mouse.	129
Figure 6.9:	A 12-Inch Touch Screen of a Tablet PC.	130
Figure 6.10:	GUI Screen Used for the Touch Screen.	130
Figure 6.11:	Basic Design and Operation of the BCI System.	131
Figure 6.12:	The Non-Invasive BCI Device.	133
Figure 6.13:	Basic Design and Operation of the BCI System.	134
Figure 6.14:	The Phantom Omni Device from SensAble Technologies.	136
Figure 6.15:	The Sip and Puff Input Device.	137
Figure 6.16:	Head and Foot Switches.	138
Figure 7.1:	Program Flowchart.	143
Figure 7.2:	A Sample Command Prompts for Autonomous Operation Mode.	146
Figure 7.3:	Simulation Window of the WMRA System in Wire Frame.	147
Figure 7.4:	A Sample of the Virtual Reality Simulation Window.	148
Figure 7.5:	The Graphical User Interface (GUI) Screen with the Defaults.	150
Figure 8.1:	The Initial Pose of the WMRA in Simulation.	156
Figure 8.2:	Position of the WMRA During Simulation.	157
Figure 8.3:	Orientation of the WMRA During Simulation.	157
Figure 8.4:	Destination Pose for Case I, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 1]$.	159
Figure 8.5:	Destination Pose Case II, When $W = [10, 10, 10, 10, 10, 10, 10, 1, 1]$.	159
Figure 8.6:	Destination Pose Case III, When $W = [1, 1, 1, 1, 1, 1, 1, 100, 100]$.	160

Figure 8.7:	Destination Pose Case IV, When $W = [1, 1, 1, 1, 1, 1, 1, 100, 1]$.	160
Figure 8.8:	Destination Pose Case V, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 100]$.	161
Figure 8.9:	Arms' Joint Motion for Case I, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 1]$.	162
Figure 8.10:	Arms' Joint Motion for Case II, When $W = [10, 10, 10, 10, 10, 10, 10, 1, 1]$.	162
Figure 8.11:	Arms' Joint Motion for Case III, When $W = [1, 1, 1, 1, 1, 1, 1, 100, 100]$.	163
Figure 8.12:	Arms' Joint Motion for Case IV, When $W = [1, 1, 1, 1, 1, 1, 1, 100, 1]$.	163
Figure 8.13:	Arms' Joint Motion for Case V, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 100]$.	164
Figure 8.14:	Wheels' Motion for Case I, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 1]$.	165
Figure 8.15:	Wheels' Motion for Case II, When $W = [10, 10, 10, 10, 10, 10, 10, 1, 1]$.	165
Figure 8.16:	Wheels' Motion for Case III, When $W = [1, 1, 1, 1, 1, 1, 1, 100, 100]$.	166
Figure 8.17:	Wheels' Motion for Case IV, When $W = [1, 1, 1, 1, 1, 1, 1, 100, 1]$.	166
Figure 8.18:	Wheels' Motion for Case V, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 100]$.	167
Figure 8.19:	Arms' Joint Velocities for Case I, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 1]$.	168
Figure 8.20:	Wheels' Velocities for Case I, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 1]$.	168
Figure 8.21:	Manipulability Index for Case I, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 1]$.	169
Figure 8.22:	Manipulability Index for Case II, When $W = [10, 10, 10, 10, 10, 10, 10, 1, 1]$.	170
Figure 8.23:	Manipulability Index for Case III, When $W = [1, 1, 1, 1, 1, 1, 1, 100, 100]$.	170
Figure 8.24:	Manipulability Index for Case IV, When $W = [1, 1, 1, 1, 1, 1, 1, 100, 1]$.	171

Figure 8.25: Manipulability Index for Case V, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 100]$.	171
Figure 8.26: Arm Base Position When the Weights Were Equal, $W = [1, 1, 1, 1, 1, 1, 1, 1, 1]$.	173
Figure 8.27: Arm Base Orientation When the Weights Were Equal, $W = [1, 1, 1, 1, 1, 1, 1, 1, 1]$.	174
Figure 8.28: Arm Base Position for Case A, When $W = [1, 1, 1, 1, 1, 1, 1, 50, 50]$.	175
Figure 8.29: Arm Base Orientation for Case A, When $W = [1, 1, 1, 1, 1, 1, 1, 50, 50]$.	175
Figure 8.30: Arm Base Position for Case B, When $W = [1, 1, 1, 1, 1, 1, 1, 50, 1]$.	177
Figure 8.31: Arm Base Orientation for Case B, When $W = [1, 1, 1, 1, 1, 1, 1, 50, 1]$.	177
Figure 8.32: Wheels' Motion Distances for Case I.	179
Figure 8.33: Joint Angular Displacements for Case I.	179
Figure 8.34: Wheels' Motion Distances for Case II.	180
Figure 8.35: Joint Angular Displacements for Case II.	181
Figure 8.36: Wheels' Motion Distances for Case III.	181
Figure 8.37: Joint Angular Displacements for Case III.	182
Figure 8.38: Wheels' Motion Distances for Case IV.	183
Figure 8.39: Joint Angular Displacements for Case IV.	183
Figure 8.40: Manipulability Measure Case I (PI).	186
Figure 8.41: Joint Angular Displacements for Case I (PI).	186
Figure 8.42: Manipulability Measure Case II (PI-JL).	187
Figure 8.43: Joint Angular Displacements for Case II (PI-JL).	187

Figure 8.44: Manipulability Measure Case III (WPI).	188
Figure 8.45: Joint Angular Displacements for Case III (WPI).	188
Figure 8.46: Manipulability Measure Case IV (WPI-JL).	189
Figure 8.47: Joint Angular Displacements for Case IV (WPI-JL).	189
Figure 8.48: Manipulability Measure Case V (SRI).	190
Figure 8.49: Joint Angular Displacements for Case V (SRI).	190
Figure 8.50: Manipulability Measure Case VI (SRI-JL).	191
Figure 8.51: Joint Angular Displacements for Case VI (SRI-JL).	191
Figure 8.52: Manipulability Measure Case VII (WSRI).	192
Figure 8.53: Joint Angular Displacements for Case VII (WSRI).	192
Figure 8.54: Manipulability Measure Case VIII (WSRI-JL).	193
Figure 8.55: Joint Angular Displacements for Case VIII (WSRI-JL).	193
Figure 9.1: Complete SolidWorks Model of the WMRA.	201
Figure 9.2: Kinematic Diagram with Link Frame Assignments.	201
Figure 9.3: Harmonic Drive Gearhead.	203
Figure 9.4: Pittman Servo Brush Motors with Gearbox and Encoder.	203
Figure 9.5: Wrist Design: 3-Roll Wrist (Left), Orthogonal Wrist (Right).	205
Figure 9.6: WMRA SolidWorks Models and the Corresponding Positions of the Built Device.	207
Figure 9.7: The New Gripper's Ergonomic Surfaces.	209
Figure 9.8: The Gripper Design in Application Reference.	210
Figure 9.9: Extended Interior Surface Added to the Gripper.	211
Figure 9.10: The Selected Coreless Gearhead Servo Motor.	213

Figure 9.11: The Selected Slip Clutch.	214
Figure 9.12: The Assembled Actuation Mechanism.	214
Figure 9.13: The New Gripper and the Actuation Mechanism Drawing.	215
Figure 9.14: The New Gripper and the Actuation Mechanism.	215
Figure 9.15: The New Gripper When Holding a Spherical Object.	217
Figure 9.16: The New Gripper When Holding a Tapered Cup.	217
Figure 9.17: The Gripper When Opening a Door with a Lever Handle (Left) and a Knop Handle (Right).	218
Figure 9.18: The New Gripper When Handling Small Objects.	218
Figure 9.19: The New Gripper When Handling Large and Heavy Objects.	219
Figure 9.20: A Circuit Designed to Convert Digital PWM Duty-Cycle Control Signal to Analogue Signal.	220
Figure 9.21: The Designed Controller Box Installed on the Modified Wheelchair.	221
Figure 9.22: The Quick-Release Mechanism that Mounts the Robotic Arm on the Wheelchair.	222
Figure 9.23: JRKERR PIC Servo SC Controller Boards.	223
Figure 9.24: Control System Circuitry.	224
Figure 9.25: Serial Port Connection of the Joint Motors (Left) and the Gripper (Right).	225
Figure 9.26: Wheelchair Encoders and Control Communications.	226
Figure 9.27: A Person with Guillain-Barre Syndrome Driving the New WMRA System.	227
Figure 9.28: A Human Subject Testing of the BCI-2000 Interface with the WMRA System.	228

Maximizing Manipulation Capabilities of Persons with Disabilities Using a Smart 9-Degree-of-Freedom Wheelchair-Mounted Robotic Arm System

Redwan M. Alqasemi

ABSTRACT

Physical and cognitive disabilities make it difficult or impossible to perform simple personal or job-related tasks. The primary objective of this research and development effort is to assist persons with physical disabilities to perform activities of daily living (ADL) using a smart 9-degrees-of-freedom (DOF) modular wheelchair-mounted robotic arm system (WMRA).

The combination of the wheelchair's 2-DoF mobility control and the robotic arm's 7-DoF manipulation control in a single control mechanism allows people with disabilities to do many activities of daily living (ADL) tasks that are otherwise hard or impossible to accomplish. Different optimization methods for redundancy resolution are explored and modified to fit the new system with combined mobility and manipulation control and to accomplish singularity and obstacle avoidance as well as other optimization criteria to be implemented on the new system. The resulting control algorithm of the system is tested in simulation using C++ and Matlab codes to resolve any issues that might occur during the testing on the physical system. Implementation of the combined control is done on the newly designed robotic arm mounted on a modified power wheelchair and with a custom designed gripper.

The user interface is designed to be modular to accommodate any user preference, including a haptic device with force sensing capability, a spaceball, a joystick, a keypad, a touch screen, head/foot switches, sip and puff devices, and the BCI 2000 that reads the electromagnetic pulses coming out of certain areas of the brain and converting them to control signals after conditioning.

Different sensors (such as a camera, proximity sensors, a laser range finder, a force/torque sensor) can be mounted on the WMRA system for feedback and intelligent control. The user should be able to control the WMRA system autonomously or using teleoperation. Wireless Bluetooth technology is used for remote teleoperation in case the user is not on the wheelchair. Pre-set activities of daily living tasks are programmed for easy and semi-autonomous execution.

Chapter 1:

Introduction

1.1. Motivation

According to the latest data from the US Census Bureau Census Brief of 1997 [1], one of every five Americans had difficulty performing functional activities (about 53 million), half of them were considered to have severe disabilities (over 26 million). Robotic aides used in these applications vary from advanced limb orthosis to robotic arms [2]. Persons that can benefit from these devices are those with severe physical disabilities (such as cerebral palsy resulting in loss of sensation or loss of ability to control movement), acquired disabilities (such as spinal cord injury, multiple sclerosis and stroke), and mobility disabilities (such as osteoporosis and arthritis due to chronic disorders) that result in a limited or no upper extremity mobility which limit their ability to manipulate objects. These devices increase self-sufficiency, and reduce dependence on caregivers. In the case of spinal cord injury or dysfunction these aids are most appropriate for individuals with spinal deficiencies ranging from cervical spine vertebrae 3 through cervical spine vertebrae 5. Individuals with neuromuscular deficiencies, such as muscular sclerosis, or other motor dysfunctions due to accidents, disease, aging, or genetic predispositions, can benefit from these robotic devices as well.

A wheelchair mounted robotic arm (WMRA) can enhance the manipulation capabilities of individuals with disabilities that are using power wheelchairs, and reduce dependence on human aides. Individuals that require mobility assist devices such as a power wheelchair can benefit from various robotic devices because the power wheelchair provides a platform with which to mount the device as well as a power supply, using the wheelchair's batteries. There have been several attempts in the past to create commercially-viable wheelchair mounted robotic arms. Currently there are only two commercially available WMRA's available, the Manus (Exact Dynamics, Inc., Netherlands) and the Raptor (Applied Resources, Inc, NJ USA).

Unfortunately, most WMRA's have had limited commercial success due to poor usability and low payload. It is often difficult to accomplish many of the Activities of Daily Living (ADL) tasks with the WMRA's currently on the market due to its physical and control limitations and its control independence of the wheelchair's control system. This project attempts to surpass available commercial WMRA devices by offering an intelligent system that combines the mobility of the wheelchair and the manipulation of a newly designed arm in an effort to improve performance, usability, control and reduce mental load on the user while maintaining cost competitiveness.

The two commercially available WMRA's lack the integration of the robotic arm controller with the wheelchair controller, and that leads to an increased mental load on the user. Combining the control of both the power wheelchair and the robotic arm would decrease this mental burden on the user and improve the combined system usability.

It is desired to fulfill the need of such integrated systems to be used for many ADL tasks such as opening a spring-loaded door autonomously and go through it,

interactively exchange objects with a companion on the move, avoid obstacles by going around them while maneuvering objects, conveniently handle food and beverage between the fridge, Microwave oven, stove, etc. without the need to switch between the wheelchair controller and the robotic arm controller, and avoid singularities in a small working environment, such as an office, where wheelchair motion can be slightly utilized to maneuver objects while avoiding singularities (similar to a person sitting on an office chair and handling objects around him by moving his/her arm while slightly moving the chair to get closer to an object that is otherwise unreachable).

1.2. Dissertation Objectives

- 1- The main objective of this work is to develop and optimize a control system that combines the manipulation of the newly designed 7-DoF robotic arm and the mobility of a modified 2-DoF wheelchair in a smart 9-DoF control algorithm.
- 2- Redundancy resolution is to be optimally solved to avoid singularities, joint limits, obstacles and to allow larger wheelchair or manipulator motion depending on the user proximity to the goal.
- 3- This WMRA is to utilize an optimized controller that is expandable to control both WMRA and the power wheelchair.
- 4- A complex and flexible simulation program is to be developed to simulate the 9-DoF WMRA in Virtual Reality.
- 5- A 7-DoF Robotic arm is to be developed and integrated to a modified power wheelchair to include PC based control and sensory feedback.

- 6- High-level control of the 9-DoF WMRA system is to be designed to combine the WMRA's 7 DoF and the wheelchair's mobility in the new redundant 9-DoF system.
- 7- Redundancy is to be used and optimized to improve manipulation capabilities for activities of daily living (ADLs) and avoid singularities.
- 8- The new system is to be capable of executing complex pre-set tasks autonomously as well as in teleoperation mode.
- 9- The user interface in the WMRA system's teleoperation mode should be capable of using a force-reflecting haptic interface, a keypad, a Spaceball, a touch screen, a BCI 2000 brain-wave sensor device or other user interface devices that can be used as modular user interfaces with different capabilities to fit the individual user needs.
- 10- Higher level control algorithms are to be developed to interface the sensory data and the user input for an easy control of the system.
- 11- The system should be capable of future modification to use Bluetooth wireless technology for remote teleoperation so that the user can also perform some ADL tasks while not seated on the wheelchair.
- 12- A sensory suite can be in the control algorithm for feedback purposes.

1.3. Dissertation Outline

This dissertation will give a background in chapter 2 on previous work done in the field of mobile robots and redundant manipulators as well as assistive devices that can be used by individuals with disabilities. Chapter 3 will focus the redundant manipulator control theories and methods, and chapter 4 will discuss the control theory of differential drives that produce non-holonomic motion. In chapter 5, the combination of both the

redundant manipulator control and the differential drive non-holonomic mobility control theory will be discussed. Mathematical relationships and augmentation of the Jacobian to combine the mobility and manipulation will also be generated in this chapter along with the optimization methods used for this application. Chapter 6 will show the different user interfaces used for this application and some clinical studies conducted with human subjects. Chapter 7 will show the application of combined manipulation and mobility control using simulation, and the results of the simulation will be shown and discussed in chapter 8. Testbed design for experimental application of the control theory on physical WMRA system will be described in chapter 9 along with the experimental results. Chapter 10 concludes the dissertation with summary and discussion with recommendations, and chapter 11 will discuss future work that can be conducted on the WMRA system.

Chapter 2:

Background

2.1. History of Rehabilitation Robotics

The development of robots started in the 1960's with manipulators which were used for manufacturing purposes [3]. Planetary rovers and vision embedded systems took the attention of researchers in the early 1970's and were developed along side with the industrial manipulators. Starting in the past decade, researchers have focused on artificial intelligence in robotics in order to widen the use of robots and make them more intelligent in their applications. One such use is in the area of rehabilitation, where people with disabilities can take control of some of their daily needs without the need for human assistance. A key problem in robotic arms that are mounted on a mobile platform is the combination of the manipulation and mobility of these systems while they move in space, especially when redundancy is introduced. There have been various attempts over the years to create robotic assistants for individuals with various levels of disabilities. For over 30 years, research has progressed in this field with only partial commercial success.

One of the first attempts at rehabilitation robotics included the Rancho "Golden" arm [4] designed in 1969 at Rancho Los Amigos Hospital in Downey, California. The arm was an electrically-driven 6 Degree Of Freedom (DOF) robotic arm which mounted to a powered wheelchair and was controlled at the joint level by an array of tongue-operated

switches. Discussions on the topic of the controllability of the arm commented on both successes and failures of the design. The successes of the project can be attributed to the important role that proprioceptive feedback plays in the control of a persons own extremities [5]. These pioneering research projects provided a framework for future development.

2.2. Rehabilitation Robotics Classification

Assistive robotics can be grouped into one of five categories: Workstation robots which operate in stationary and well-structured environments, wheelchair-mounted robotic arms which operate on power wheelchairs to assist in activities of daily living, mobile assistive robots which travel about the room and have a manipulator arm, therapeutical robots which are used for different kinds of therapy, and smart wheelchairs and walkers.

2.2.1. Workstation Robotic Arms

The very first rehabilitation robotics applications focused on using commercially-available industrial manipulators and modifying them for rehabilitation applications. An example of these manipulators is the PUMA 250 shown in figure 2.1. A factor which limits the use of industrial robotic arms in rehabilitation is the basic difference in operational requirements. Industrial arms are designed to work at high speed in an environment where there are no humans. This reason alone would limit their use for reasons of safety of the operator. For applications in a human-intensive workspace,

assistive robotic arms need to be mechanically limited to low velocities and accelerations for the safety of the operator and the human subjects around these devices.

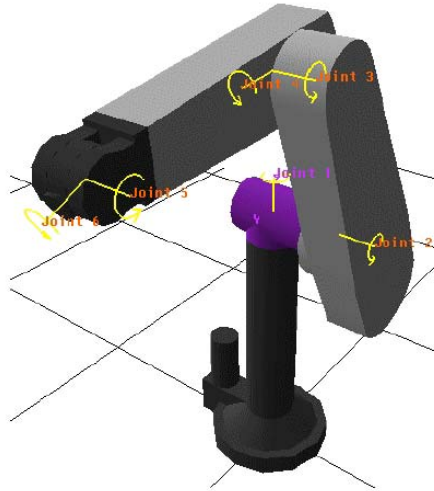


Figure 2.1: Puma 250 Arm.

The Robotic Aid Project [6] was an attempt to create a system for users with quadriplegia. The project was an integration of a PUMA 250 industrial manipulator arm, microprocessor, multi-line monochrome display and speech synthesis and recognition systems. Limitations with the speech-recognition systems and computational power of the day restricted the success of the program. The processing ability of the contemporary computers did not allow for real-time inverse kinematics of the arm. This limited the arm to merely replaying preprogrammed actions. Individual joints of the arm could be manipulated but coordinated real-time multi-joint maneuvers were impossible.

As more application-specific robotic arms and computers with increased computational power became available, arms with controllers could now be mounted onto mobile platforms. At first these systems were simply rolling bases which then increased in complexity and degrees of freedom to include powered mobile robots.

Handy-1 [7] is a robotic arm mounted to a non-powered wheeled base to assist in very specific activities of daily living (ADL). Handy-1 was developed in 1988 to provide persons with severe disabilities assistance at mealtimes. Since its initial introduction the unit has expanded capabilities and is now capable of providing assistance in a broader number of activities of daily living (ADL). Handy-1 is capable of assisting individuals with personal hygiene, eating and drinking, and the application of make-up. During user trials, women specifically asked if the unit would be capable of applying cosmetic products. Shortly after the trial, the design was upgraded with a new tray and gripper accessory. Each ADL task has a specific tray to accomplish its goal. Handy-1 is shown in figure 2.2 and is based on a 5 DOF lightly modified industrial manipulator.



Figure 2.20: Handy-1.

In the feeding mode the operator controls the robot through an interface that uses lights which move across the available food trays, and a button that selects the item desired. Once the button is pressed, the robot scoops up the selected food and brings it to a predetermined place near the operator's mouth. Once the user has consumed the food, the operator presses the button again and the robot returns to the food selection mode. This process is repeated until the operator is finished. An advancement of the technology

of Handy-1 is being explored with the Robotic Aid to Independent Living [8] (RAIL) project. RAIL improves upon the Handy-1 design by incorporating a new controller for better manipulator control, a 3D simulation tool for modeling virtual scenarios and attachment of sensors to assist set up and position error determination.

The RAID workstation [9] shown in figure 2.3 was designed to be a workstation assistive robot system. It is comprised of a 6 DOF robotic arm mounted onto a linear track in a well-controlled environment. In the figure the manipulator can be seen near the top of the shelf in the center of the cabinet.

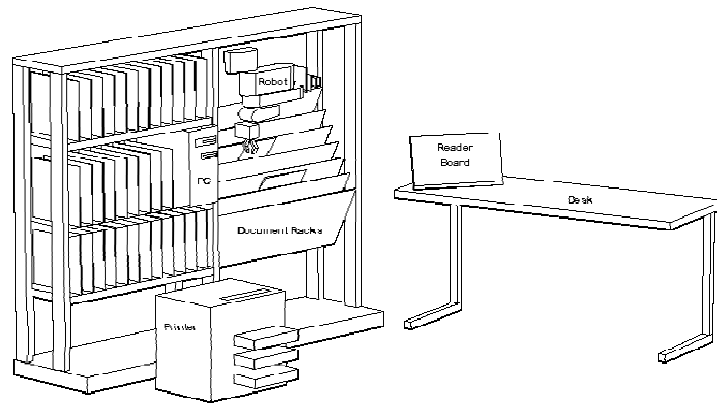


Figure 2.3: RAID Workstation.

The RAID system provides benefits that are enhanced by the formal structure provided by a workstation environment. This organization allows the manipulator arm to repeatedly move and acquire items needed by the operator using preprogrammed functions and routines. At this time the RAID system is currently under evaluation in Europe.

The Robotic Assistive Device [10], shown in figure 2.4, is a robotic arm developed by the Neil Squire Foundation in Vancouver, Canada. The RAD is a 6 DOF workspace mountable manipulator that uses a serial port computer interface. The

manipulator is controlled through a graphical user interface (GUI) utilizing icons to symbolize predefined tasks. The system consist of several modules which when combined create an arm with a cylindrical reach of approximately 55” and a height of 110”. The arm can be mounted on various surfaces and has good repeatability at 0.12” and relatively large payload capacity of 9.5 lbs. Most rehabilitation specific manipulators have maximum payloads of 5 pounds or less.



Figure 2.4: Robot Assistive Device.

The ProVAR [11] shown in figure 2.5 is a system based on a Puma 260 robotic arm designed to operate in a vocational environment.

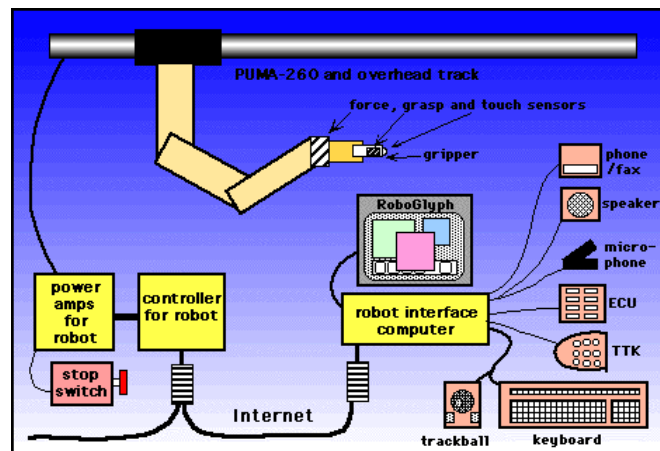


Figure 2.5: ProVAR System.

The ProVAR system uses a web-based virtual environment to model the functionality of the manipulator. In this way the operator can examine potential arm movements for a given task, and if the simulation is successful, the action can be initiated. In this way, the actions of the arm and its interactions within its workspace can be seen before any action is taken. The primary goals for ProVAR are more functionality per dollar, easier operator control, and higher system reliability compared with the previous generation of vocational assistive robots.

2.2.2. Wheelchair-Mounted Robotic Arms

Wheelchair mounted robotic arms (WMRAs) combine the idea of a workstation and a mobile robot, which mounts a manipulator arm onto a power wheelchair. In the past, industrial manipulators have been too large and heavy to be mounted onto a power wheelchair. An industrial manipulator mounted onto the wheelchair would have excessively hindered the operator's ability to maneuver the chair through doors and hallways. Several design considerations must be met before deciding on where, on a power wheelchair, to mount a robotic arm. The foremost design consideration is the safety of the operator [12]. The mount must be sturdy and rigid and not compromise the structural integrity or the functionality of the chair in any way. Next the robotic arm must be mounted in such a way that it has a minimum footprint outside the footprint of the chair itself. There are several possible mounting locations for a WMRA [13]. The mount may be in the front, side or rear of the wheelchair.

Helping Hand system [14] is a 5DOF robotic arm that can be mounted to the side of a power wheelchair. The Helping Hand operates by joint control and is manipulated by using switches to control individual joints.

The Weston robotic arm [15], shown in figure 2.6, utilizes a vertical actuator mounted to a wheelchair with the main rotary joints (shoulder, elbow, and wrist) constrained to move in the horizontal plane. This is the continuation of the trolley mounted Wessex robot arm research. A prismatic joint moves in a linear sliding motion along a track.



Figure 2.6: Weston Arm.

Another arm WMRA is the Asimov [16], which is a modular manipulator designed with the motors and controls distributed throughout the arm. A computer rendering of the Asimov is shown in figure 2.7. The modularity of the design allows for multiple mounting locations on a wheelchair or stationary application with various workspace geometries. The concept of a modular manipulator has several benefits. This provides the opportunity for one manipulator that can be used in either a mobile or

workstation environment. Different link geometries can be explored to create the optimum design for any given application. Asimov models have been shown with all three possible mounting positions: front, side and rear. Without physical models to test the efficacy of the design, it is unknown how well the design would integrate into real-world applications.



Figure 2.7: Asimov Arm.

The FRIEND [17] robotic system is a Manus arm mounted onto a wheelchair and integrated with stereo vision, dedicated computer control, and specialized software. Besides programming with a keypad or joystick, the FRIEND system, shown in figure 2.8, is capable of being programmed via a haptic interface glove. The haptic glove allows the operator / programmer to feel what the robot feels through feedback to the user. A Haptic glove is put on and the action, such as pouring a glass, is completed and stored into the computer for future use. The action can then be replayed at a later time as a pre-defined user function. The operator may also control the arm through verbal commands using an integrated voice recognition system.



Figure 2.8: FRIEND Robotic System.

2.2.3. Mobile / Assistant Robots

Mobile systems are capable of assisting individuals with disabilities. These systems include a mobile base, various sensors and a manipulator arm. An early version of one such system is the Mobile Vocational Assistant Robot (MoVAR) [18]. This system, shown in figure 2.9, utilizes an omni-directional mobile platform mounting a PUMA-250 robotic arm as well as several sensors including a remote viewing camera, force and gripper proximity sensors.

MoVAID [18] is an advanced version of the MoVAR system, designed specifically for home use. MoVAID improves upon the previous model by applying the lessons learned in laboratory testing to assist in common tasks around the home such as cleaning and food preparation. MoVAID incorporates a variety of sensing devices both mounted to the manipulator and the base. In figure 2.10, MoVAID can be seen along with the various sensors that are located on the manipulator arm. Sensors mounted to the first

link of the arm include a pair of cameras used for stereo vision and a laser localization system used in task execution.

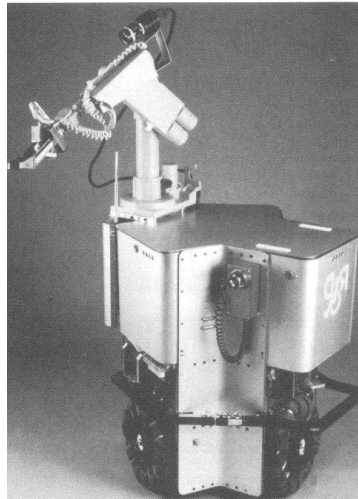


Figure 2.9: MoVAR.

The MoVAID system uses active beacons positioned within the room that provide reference data to determine its location and orientation. In addition to position detection, the unit also has ultrasonic range detectors and an active bumper that disables the device should an impact occur.



Figure 2.10: MoVAID.

The robotic arm used by MoVAID has 8 DOF and a three-fingered gripper with two degrees of freedom. The gripper was originally designed as a prosthetic device specifically to have excellent dexterity. The increased agility provided by the gripper over more traditional end-effectors allows MoVAID to be very effective in the unstructured home environment.

Another design is the TAURO [19] that is an integrated robotic system using off-the-shelf components such as a power wheelchair, Manus manipulator, ultrasonic sensors, camera and computers. TAURO is a mobile service robot being developed for inspection, stocktaking and documentation tasks in indoor environments. The TAURO system integrates the movement of the wheelchair and the operation of the manipulator. In this way if the goal is out of reach of the manipulator, the wheelchair will move on a path toward the goal until the manipulator can reach its goal. This coordinated control is a significant advance in the use of WMRAs. Although not specifically designed for rehabilitation robotics tasks, it would be readily adaptable to the task. The TAURO system can be seen in figure 2.11.



Figure 2.11: TAURO Robotic System.

2.2.4. Robots in Therapy

Assistive robotics can enhance the capability of people with disabilities by assisting them to do different activities of daily living. On the other hand, therapeutical robots can exercise the user's muscles to keep it alive or to increase its ability to function with time. One of these systems is the MIT Manus [20] shown in figure 2.12, which uses impedance control to move, guide or perturb the hand motion of the user in training. It records the position, velocity and applied forces during therapy sessions for analysis.

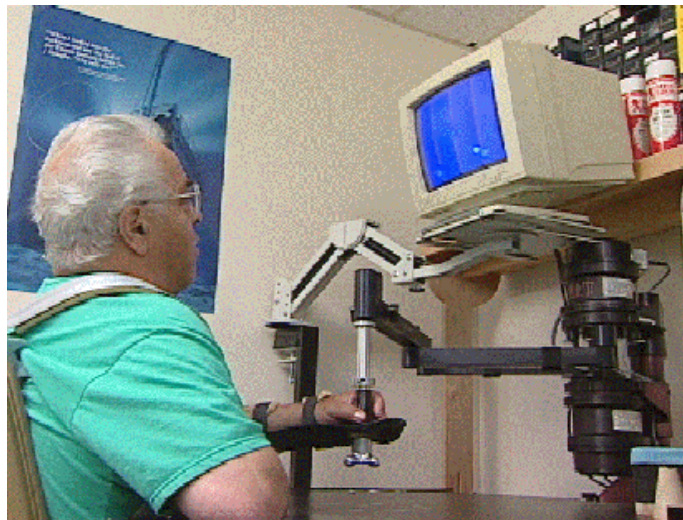


Figure 2.12: MIT Manus System.

Another therapeutical robot is the mouth-opening and closing devices conducted by Takanobu et al in 2001 [21] that is used in training as shown if figure 2.13. This robotic device uses 6 linear actuators to manipulate the U-shaped end effector. Each joint carries displacement and velocity sensors for feedback.



Figure 2.13: Mouth Opening and Closing Device.

2.2.5. Smart Wheelchairs / Walkers

The iBOT and Segway mobility systems [22] are unique gyro-balanced mobility devices that have been designed to operate on four wheels or two wheels, stabilizing the user by automatically adjusting and balancing. The iBOT offers five operating functions: a remote function that allows the user to detach the joystick, and via cable wire connection, drive the empty iBOT into the back of a vehicle for easy transporting. The stair climbing function allows the user to climb up and down stairs with or without assistance. The wheel function that allows it to climb curbs as high as 4 inches and travel over grass, gravel, sand and other forms of uneven terrain. The standard function is similar to standard power chairs. The balance function allows the user to reach high places independently in a similar manner to the Segway. Figure 2.14 shows both the iBOT and the Segway.



Figure 2.14: iBOT (Left) and Segway (Right) Devices.

2.3. Commercial Wheelchair-Mounted Robotic Arms

Currently there are two production wheelchair mounted robotic arms (WMRAs): the Manus, manufactured by Exact Dynamics, and the Raptor, manufactured by Applied Resources.

2.3.1. The Manus

The Manus manipulator [23] arm (or as recently been called ARM for Assistive Robotic Manipulator) is a fully deterministic manipulator. A fully deterministic arm can be programmed in a manner comparable to industrial robotic manipulators. The Manus has been under development since the mid 1980's and entered into production in the early 1990's. A picture of the Manus mounted onto a Permobil Max90 wheelchair is shown in figure 2.15.



Figure 2.15: Manus Arm.

This arm utilizes a front mounting location to the left of the operator's left knee, which allows for good manipulation of objects that are above the plane of the wheelchair seat, and most importantly the operator's face and lap. Manus carries 6 revolute joints with encoders, a 1-DoF gripper and a vertical lift. The Manus manipulator is controlled by a joystick and a keypad, and can perform a single-joint control or coordinated control of multiple joints.

2.3.2. The Raptor

Another production WMRA is the Raptor [24], which mounts the robotic arm to the right side of the wheelchair. This manipulator carries 4 revolute joints plus a planar gripper and can be seen mounted to a power wheelchair in figure 2.16. The arm is directly controlled by the user by either a joystick or 10-button controller. Because the Raptor does not have encoders to provide feedback to the controller, the manipulator cannot be pre-programmed in the fashion of industrial robots, and can not have Cartesian control.



Figure 2.16: Raptor Arm.

The Raptor is a side-mounted arm that is partially hidden underneath the chair. When the arm is not in use, the Raptor arm can be stowed relatively inconspicuously. Robotic arms with joint control require higher levels of concentration and eye-hand coordination from the operator than Cartesian control systems.

2.4. Robot Control

The controller design of the robotic device is as important as the design of the robotic device itself. Many researchers have explored different ways of controlling the robotic devices both in simulation and in physical systems.

2.4.1. Redundant Robot Control

When controlling a robotic device, it is essential to compare the work space capability of the robot and the task space required in operation. In general, a minimum of

6 degrees-of-freedom are required in a robot in order to accomplish a total manipulation control of objects in the workspace. In the case of planar workspace, a minimum of 3 joints are required in a robot to achieve full manipulation of that workspace. When the number of joints exceeds the number of controlled coordinates in the workspace, redundancy is introduced, and the conventional inverse kinematics for a close-form solution is no longer applicable. Redundancy resolution and optimization have been the subject of many researchers, where the use of the extra joints is employed to execute additional tasks and optimize the motion based on certain performance criteria.

Chang [25] has proposed a closed-form solution formula for inverse kinematics of redundant manipulators using Lagrange multiplier. He proposed an additional set of equations to resolve the redundancy at the inverse kinematic level in such a way that a given criteria function may be minimized or maximized. The additional equations were set in a similar way to the homogeneous solution term of the resolved rate method which uses the null space to resolve the redundancy. He used the manipulability index as the criteria function, but any criteria function can be used as long as the function can be reduced to an expression in terms of joint variables only.

Khadem et al [26] used a global optimization scheme to avoid round obstacles using the resolved rate method. Their simulation of a three-revolute-joint planar robotic arm has shown good performance in following a path while the specified robot link was avoiding a specified obstacle throughout the simulation. Figure 2.17 shows the simulation mechanism with and without the obstacle avoidance optimization criteria.

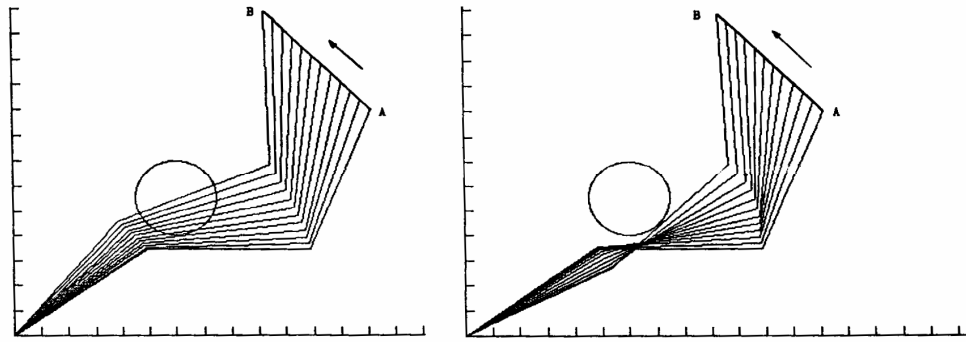


Figure 2.17: Redundancy Resolution without (Left) and with (Right) Obstacle Avoidance.

Chan et al [27] have proposed a new method to resolve the redundancy and optimize for joint limit avoidance. They used a symmetric positive definite weight matrix that carries different weights for each individual joint of the redundant robot to be included in the least-norm solution they were using to control the 7-DoF robotic arm. The weighted-least norm solution was implemented, and showed a good combination between reaching the goal with the specified trajectory accurately and avoiding the joint limits of the robotic arm. McGhee et al [28] used the weight matrix to avoid joint limits, singularities, and obstacles using the probability-based weighting of the performance criteria.

Beiner et al [29] have improved the velocity norms and the kinetic energy of their planar 3-DoF robotic crane with hydraulic actuators by using an improved Pseudo inverse solution control scheme. They used the initial manipulator configuration as an optimization parameter, and were able to reduce the actuator velocities obtained by a pseudoinverse solution and simultaneously avoid the actuators limits.

Zergeroglu et al [30] have designed a model-based nonlinear controller that was able to achieve exponential link position and subtask tracking. Their control strategy used

the pseudoinverse of the manipulator Jacobian and did not require the computation of the positional inverse kinematics. Their control strategy did not place any restriction on the self-motion of the manipulator, and hence, the extra DoF were available for their manipulability maximization, obstacle avoidance, and joint limits subtasks.

Kwon et al [31] have introduced a new method to optimize and resolve redundancy considering joint-limit constraint functions. Their Dual QCQP method used the quadratic inequality constraints to approximate linear inequality constraints which represent joint position, velocity and torque bounds. Using their method, they were able to reduce the size of the problem by reducing the number of constraints and variables. They formulated the quadratic objective function and then converted the problem into two problems by eliminating linear equality constraints and by applying the duality theory. This method was used in their simulation of a 4-joint planar robotic arm, and they were able to cut the computation time to about a tenth of that when the problem was not reduced.

Ellekilde et al [32] have introduced a new scheme for controlling robots in visual servoing applications. They employed quadratic optimization techniques to solve the inverse kinematics problem and explicitly handle both joint position, velocity and acceleration limits by incorporating these as constraints in the optimization process. Contrary to other techniques that use the redundant degrees of freedom to avoid joint limits, in their method, they incorporated the dynamic properties of the manipulator directly into the control system to use redundancy to avoid joint velocity and acceleration limits. They used the joint position limits, velocity limits and acceleration limits by converting them into the velocity domain and choosing the best case of these limits (that

satisfies all of them) at every time step to be used for the optimization function. Figure 2.18 shows the application of their method in the example of the RoboCatcher visual servoing application using the QP controller. The robot was trying to track the car which moves in a circle in the playing area. The QP control system was robust with respect to singularities which enables the robot to track the car as “good as possible” even if it is out of reach.

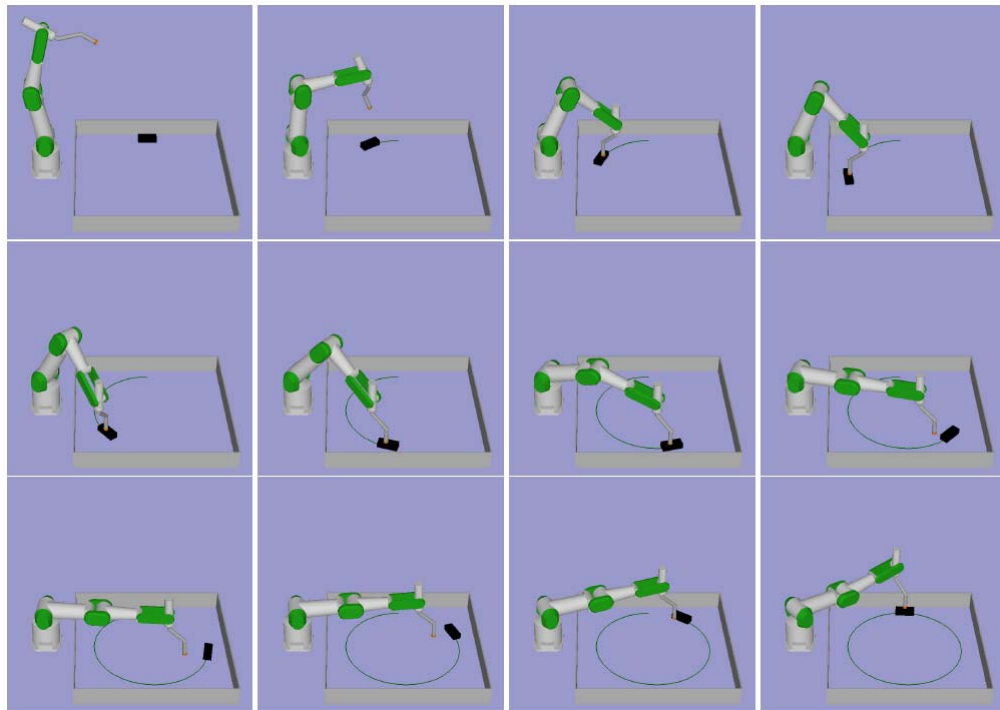


Figure 2.18: The Robot Visual Servoing Application Using the QP Controller.

2.4.2. Mobile Robot Control

In the past decade, mobile manipulators that combine the manipulation of a manipulator and the mobility of a mobile platform have been paid much attention by many researchers. Most of the works for the mobile manipulators have reported on the coordination of the mobile platform and the manipulator and the obstacle avoidance in

their environments. Recently, the importance of human-robot coexistent systems which perform cooperative works with humans and provide convenience such as house cleaning and washing has been raised.

Perrier et al [33] have proposed a method to determine a feasible path between two fixed configurations for a mobile manipulator whose vehicle is non-holonomic. For this purpose, they wrote the global displacement of the system in a symbolic way, using two representation tools: homogeneous matrices and dual quaternions. The corresponding joint parameters are computed to make the desired displacement coincide with the computed symbolic displacement. Figure 2.19 shows the frames of reference used in their robotic simulation.

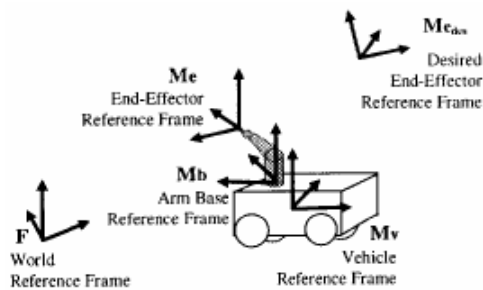


Figure 2.19: Reference Frames Used for the Manipulation LIRMM.

The simulation results about motion generation of a mobile manipulator with a non-holonomic vehicle and a six Degree-of-Freedom (DoF) arm using a global method was shown. They represented the non-holonomy of the vehicle as a constrained displacement. The method tries to make the global feasible displacement of the system correspond to the desired one. Two kinds of displacement representations were used: homogeneous matrices and dual quaternions. Trajectories obtained with the two representations were given. Both representations allowed them to compute feasible trajectories, although different.

At Chuo University in Japan [34], researchers have proposed a simple method for carrying a large object by cooperation of multiple mobile manipulators with position controllers. Manipulators on mobile platforms are used as free joint mechanisms by locking some joints and making the rest of the joints free. These free joints play the role of mechanical compliance in order to avoid excessive inner forces due to the mutual positioning errors. They found that compliance is needed for cooperation among position-controlled robots, and three feedback control laws for platforms moving on uneven ground are studied and they looked at their control performance. As shown in figure 2.20, they proposed the control laws to be used for a prototype cooperative system consisting of three moving tables driven by ball screws.

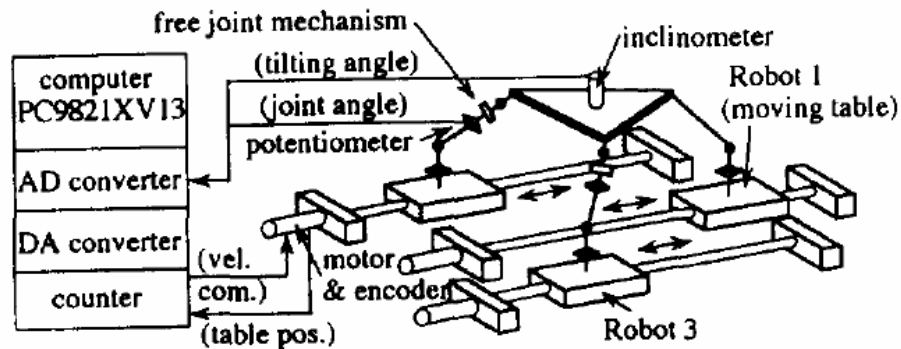


Figure 2.20: Cooperative Control System Setup.

When multiple robots hold single object at the same time, geometrical constraints by closed loop structures are imposed on each robot. Thus compliance is needed for each robot to avoid excessive inner forces caused by the mutual positioning errors.

Huang et al [35] have developed a small-sized platform as shown in figure 2.21. The problem they faced is the fact that for a small scale platforms, the mobile manipulator may fall down when moving at high speed or executing tasks in the presence

of disturbances. Therefore, it is necessary to consider both stabilization and manipulation simultaneously while coordinating vehicle motion and manipulator motion. They propose a method for coordinating vehicle motion planning considering manipulator task constraints, and manipulator motion planning considering platform stability. Specifically, first, the optimal problem of vehicle motion is formulated, considering vehicle dynamics, manipulator workspace and system stability.

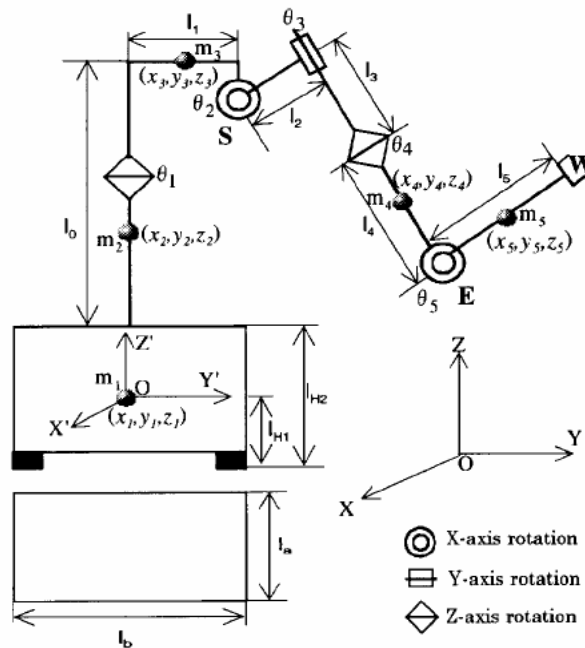


Figure 2.21: Mobile Manipulator Model.

They derived the manipulator motion considering stability compensation and manipulator configuration. Then, simulation is conducted to demonstrate effectiveness of their method. These researchers tried to derive coordinated motion so that the mobile manipulator can move stably and follow a given desired end-point trajectory (path, velocity) at an optimal configuration. They derived the redundant manipulator motion, considering stability compensation and manipulator configuration, and provided simulation results. When considering the compatibility of stabilization and manipulation,

it is first necessary to maintain system stability. Then, based on the assurance of system stability, the mobile manipulator should execute tasks with an optimal configuration.

The researchers in Noval Postgraduate School, and Tokai University [36] have presented a unified approach to the task space analysis of a wheeled mobile manipulator interacting with the environment as shown in figure 2.22. The system considered is a double- articulated manipulators atop a wheeled mobile platform handling a common object.

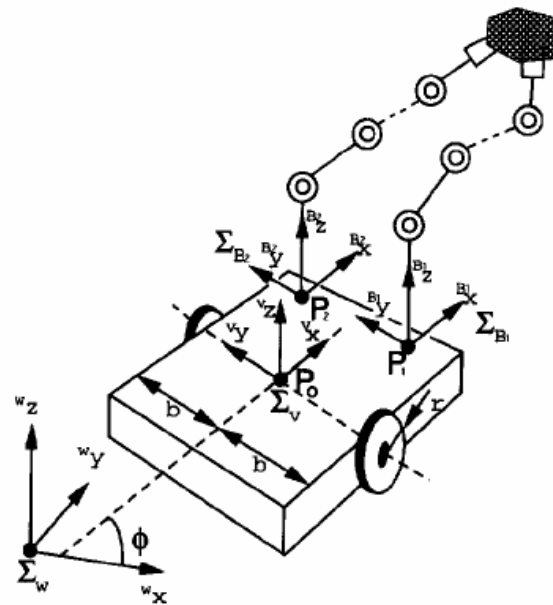


Figure 2.22: Wheeled Mobile Manipulator with Two Arms.

They derived the task space ellipsoid, both kinematic and dynamic, for a wheeled mobile manipulator which takes into account manipulation and locomotion. The ellipsoid is able to visualize how the manipulator and the platform can contribute to a task execution by integrating the mobility of the platform with the manipulability of the arms as one unified measure. This measure can be useful not only for the task space analysis of

a single mobile manipulator, but also for the coordination of multiple-arms mobile robots or mobile manipulators.

Separating manipulation from mobility makes control and planning problems easier, but it will be much more effective and efficient if the manipulator can execute a given task while the platform is moving. These researchers have proposed a coordination algorithm of mobile manipulators which utilizes manipulability measure of the arm. They treated both locomotion and manipulation in the same framework from the view-point of task space. Contribution of the manipulator and platform is represented by the task space ellipsoid at the end effector point or at the center of the object to be handled. The ellipsoid not only displays how a given task is contributed by the arm and the platform, but also the shape of the ellipsoid, kinematic or dynamic, naturally reflects constraint equations to which the platform is subjected. They also derived the motion equations of the two-arm mobile manipulator and the object separately. Motion equations of the mobile manipulator which itself consists of multiple subsystems are obtained by adding dynamic interaction terms to pre-existing motion equations to get the global equations of motion for the whole system.

Royal Institute of Technology researchers [37] have proposed a platform-independent control approach for mobile manipulation and coordinated trajectory following. Given a path for the gripper to follow, another path is planned for the base in such a way that it is feasible with respect to manipulability. The base and the end-effector then follow their respective reference trajectories according to error-feedback control algorithms, while the base is placed in such a way that the end-effector trajectory always is within reach for the manipulator. The experimental platform that they have used for

this is a Nomadic XR4000 base platform together with a Puma560 manipulator arm, as shown in figure 2.23.

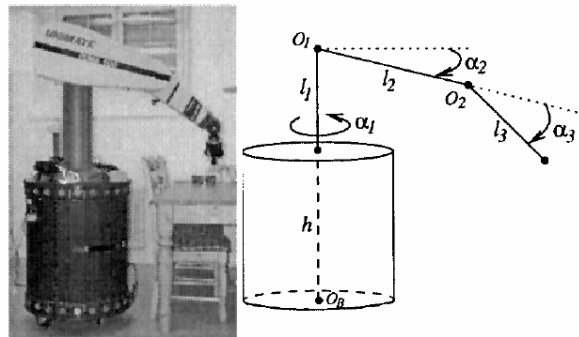


Figure 2.23: Nomad XR4000 with the Puma560 Mounted on Top.

Given a path for the gripper to follow, the idea is to plan another path for the base, online, in such a way that the end-effector trajectory always lies in the dextrous workspace. These two paths are then tracked using a virtual vehicle approach, where the motions of the reference points on the desired base and gripper paths are governed by their own dynamics, containing both position error feedback as well as coordination terms.

Researchers in Kyushu University [38] have studied the planning method of a trajectory of a mobile manipulator such as shown in figure 2.24. They derived the dynamics of the mobile manipulator considering it as the combined system of the manipulator and the mobile platform. The planning problem is formulated as an optimal control problem. To solve the problem, they used the concept of the order of priority. A gradient-based iterative algorithm which synthesizes the gradient function in a hierarchical manner based on the order of priority is used. The simulation results of the 2-link planar non-holonomic mobile manipulator are given to show the effectiveness of their algorithm. A mobile manipulator composed of a manipulator and a mobile platform

has a much larger workspace than a fixed-base manipulator due to the mobility provided by the platform. The trajectory planning problem of the non-holonomic mobile manipulator dynamics has been taken into consideration.

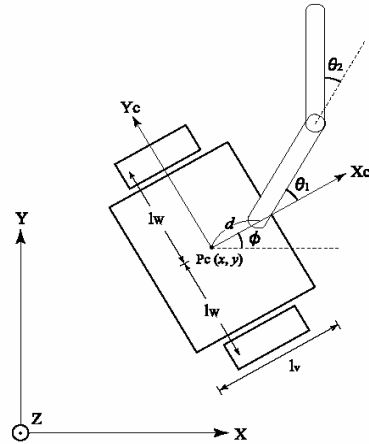


Figure 2.24: Mobile Manipulator.

Researchers in LAAS-CNRS in France [39] have extended the standard definition of manipulability to the case of a nonholonomic mobile manipulator built from an n joint robotic arm and a nonholonomic mobile platform as shown in figure 2.25. The effects of mounting the arm on a nonholonomic platform are shown through the analysis of the manipulability.

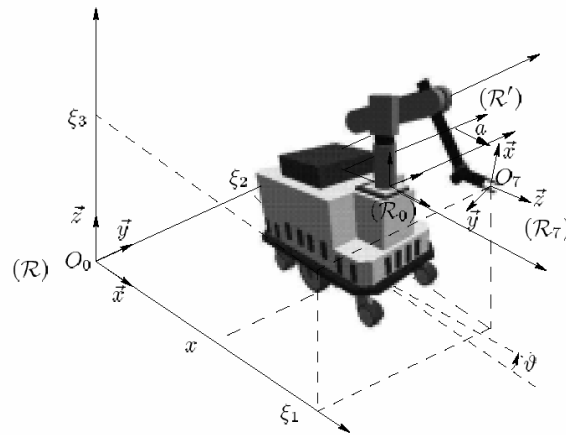


Figure 2.25: Mobile Manipulator H2BIS.

Applications of criteria inherited from manipulability considerations are given to justify design and to generate the controls of their system. Their study was motivated by the generation of the mobile manipulator velocities to execute a given operational path. The inversion of the direct instantaneous kinematic model of the mobile manipulator allowed them to solve the problem and to take into account some additional criteria. As a usual criterion, they considered the manipulability measure, which is very useful to characterize the instantaneous kinematics of a given system. These researchers showed how the notion of manipulability can be extended in that case to represent the possible operational motions in a given configuration of the system. Some simulations gave an idea of the effects of the platform on the shape of manipulability ellipsoids, with an obvious dependence on nonholonomy.

Those same researchers [40] have also proposed a generic scheme to solve the kinematic control problem of wheeled mobile manipulators when the operational motion is imposed. They generalized the Additional Task Method to solve the control problem of these redundant nonholonomic systems. They integrated any number of additional user-defined constraints to the operational task and proposed a generic approach to control a large class of mobile manipulators as well as other methods to express the additional tasks corresponding to real experimental constraints. Thus the control designer can use purposely redundancy, particularly to avoid obstacles. They illustrated the Additional Task Method by a collision free simulation. This simulation is done in a 3D environment and uses an efficient collision detector. Figure 2.26 represents the mobile manipulator of LAAS.

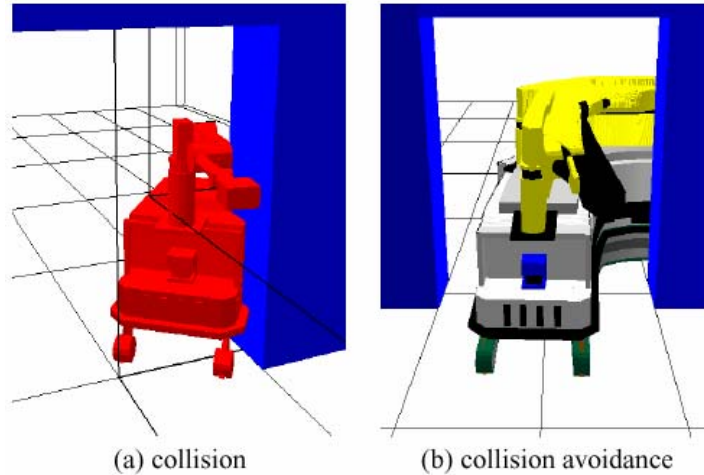


Figure 2.26: LAAS Mobile Manipulator.

Figure 2.26 (a) represents the collision in motion of the mobile manipulator without taking into account the additional collision avoidance constraint. Figure 2.26 (b) shows the achievement of the motion with the use of the proposed method.

Researchers at Okayama University [41] have conducted research to realize the motion planning for an intelligent mobile manipulator shown in figure 2.27. To plan a mobile manipulator's motion, it is popular that the base robot motion is regarded as manipulator's extra joints, and the whole system is considered as a redundant manipulator. In this case, the locomotion controller is a part of the manipulator controller. However it is difficult to implement both controllers as one controller in the implementation because of the difference of actuators character. In this research they have focused on a path planning algorithm for a mobile base with keeping manipulability at the tip of the mounted manipulator. In this case, the locomotion controller is independent from the manipulator controller and a cooperative motion is realized by the communication between both controllers.

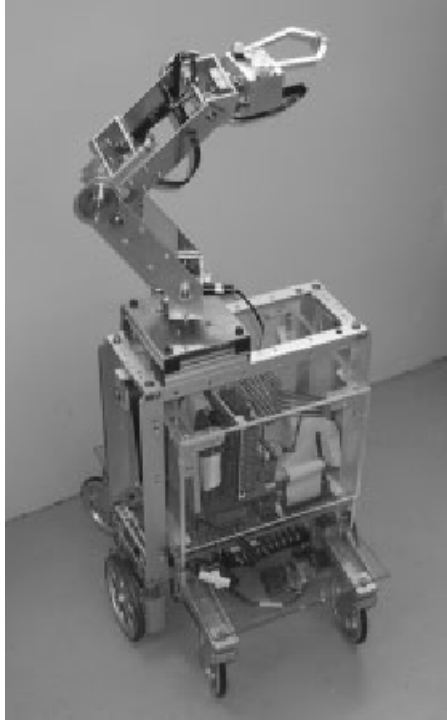


Figure 2.27: Mobile Manipulator.

One of their general approaches is to consider the locomotion as extra joints of the manipulator. Manipulability was defined as a valuation of difficulty of manipulator's operation. The goal of these researchers is to draw large objects on a wall by a mobile manipulator based on the above approach. Figure 2.27 shows an overview of the mobile manipulator considered for their research task. To realize the above task, one of the biggest problems is that an accumulated error of estimated base robot's position affects position accuracy at the end effector. Therefore, the manipulator should have the capability to adjust its pose when the base robot detects positioning errors. The motion planning approach is reasonable enough to cope with above conditions because manipulability is considered. Each pose of the manipulator is calculated by inverse kinematics at each layer. To verify the result, they have executed the program in the

motion simulator. The end effector traces the desired slope segment while the mobile base moves.

Seraji [42] has simulated the motion control of a mobile 2-link planar manipulator mounted on a non-holonomic mobile platform. He combined the Jacobian of the mobile manipulator with that of the robotic arm. To resolve the redundancy, he added two additional variables to the task space, the platform angle, and the elbow angle between the forearm and the wrist. Accordingly, he augmented the Jacobian to include these two variables' Jacobian in his control equations. This kind of redundancy resolution increases the need for trajectory planning for these extra task variables and makes it computationally expensive.

Chung et al [43] have approached the control problem of non-holonomic mobile manipulators in special workspace by decomposing the mobile manipulator into two subsystems, the mobile platform and the manipulator. According to their redundancy resolution scheme, the manipulator was commanded to follow the desired trajectory given in task space and the platform was responsible for positioning the manipulator at a specified point in the workspace to avoid singular configurations of the manipulator. To coordinate the two separate motions together, they developed an interaction control algorithm, as shown in figure 2.28, in which two nonlinear controllers were designed for the subsystems, based on the redundancy resolution scheme. The interaction controller consisted of a robust adaptive controller for the manipulator and an input-output linearizing controller for the mobile platform. Their simulation results demonstrated a good performance of the interaction controller based on their trajectory-following task.

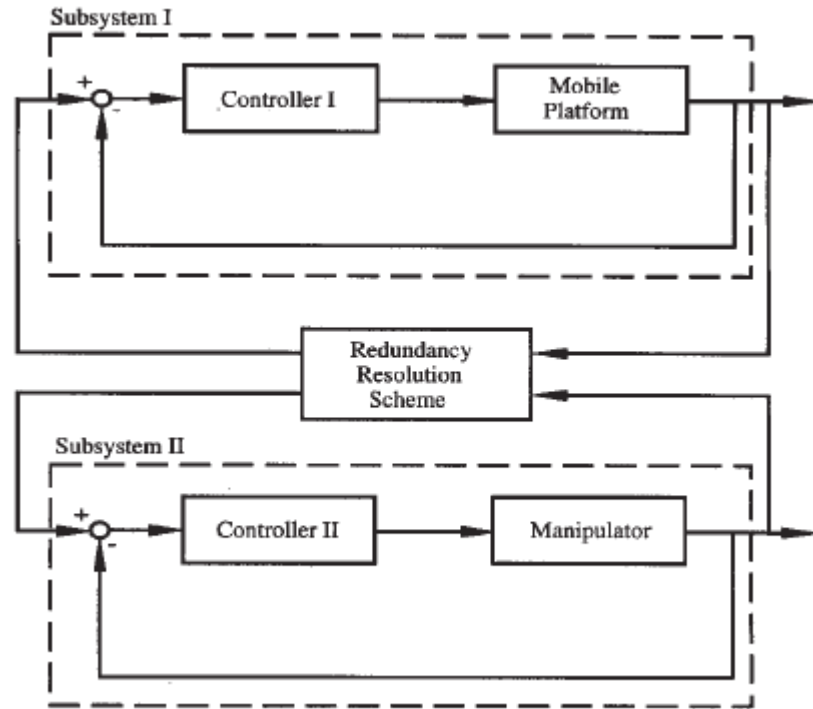


Figure 2.28: Interaction Control of the Mobile Manipulator.

Gardner et al [44] and Bayle et al [45] presented a systematic unified kinematic analysis for manipulator arms mounted on mobile platforms. They extended the definition of manipulability by scaling joint velocities by their maximum values. To find the best possible mounting location of the mobile platform, they simulated a mobile manipulator with differential drive that carries a variable placement of the arm base on the mobile platform. They were able to illustrate the manipulability measure based on the manipulator mounting position on the overall mobility of the system.

Xu et al [46] have proposed a new strategy to deal with the mobility and manipulation combination problem in a mobile manipulator that has redundant DoF. In their control, they decomposed the position and orientation of the end-effector into two parts. The position and orientation of the sub-vectors of the manipulator projected on the global z-axis was defined, then the mobile base and the manipulator were moved along

the main direction of the desired path and the sub-vectors on axes x-axis and y-axis in the world frame. The simulation results showed that the small working ranges of the joints of the manipulator have seriously limited the application.

Luca et al [47] have tested the extension of conventional redundancy resolution methods to include non-holonomic mobile platforms at the base of the redundant arm using an augmented Jacobian. They have used the singularity analysis and redundancy resolution methods available for standard manipulators to compare the reduced gradient method and the projected gradient method. Their simulation has shown the superior optimization performance of the reduced gradient over the projected gradient method. The desired tasks for the robotic system were executed by the combined motion of all the configuration variables. Figure 2.29 shows the simulation of their implementation using the reduced gradient method on a 4-DoF planar system following a circular path while keeping the end-effector pointing to certain direction.

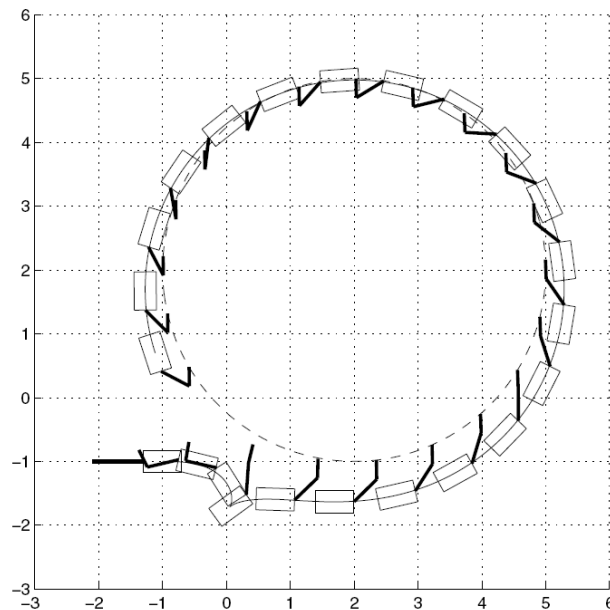


Figure 2.29: Trajectory Tracking for a Planar 2-DoF Robot on a Differential Mobile Base.

Papadopoulos et al [48] have tested their control algorithm on mobile manipulators mounted on differential-drive platforms as well as car-like platform. Both system platforms were equipped with a two link manipulator. The differential kinematics for the two mobile systems were written so as to map platform and end-effector velocities to the driven wheel velocities, without violation of the non-holonomic constraints. This allowed specification of trajectories for both the platform and the end-effector and computation of actuator commands. Orthogonal complements and the Lagrangian methodology were used to obtain the reduced equations of motion for the differentially-driven system. Figure 2.30 shows the desired path of the end effector and that of the front point on the platform along with the actual trajectory-following simulation.

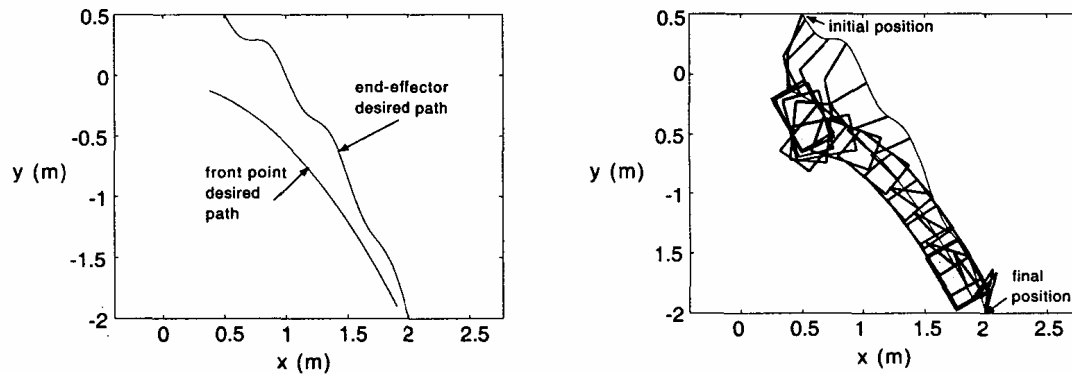


Figure 2.30: Animation of the Motion of the End-Effector and the Platform Front Point.

Based on these equations, a model-based controller was designed to eliminate tracking errors. The controller was applied successfully to a simple crack-sealing example, and showed accurate simulation.

Chapter 3:

Control Theory of Redundant Manipulators

3.1. Introduction

Control issues in simple robotic systems can be resolved easily when we try to control Cartesian coordinates using a robotic system that has the same number of joints as these coordinates to be controlled. For instance, a robotic arm with two revolute joints can be used to control the x-axis and y-axis coordinates of the end-effector in a planar workspace using simple mathematical relations that relate the joint motion to the end-effectors' motion. Trying to control a third variable, such as the angle of approach in a planar workspace will be difficult and some times impossible if we use the same 2-DoF. This is because the manipulator carries less DoF than the workspace. On the contrary, when we try to control the two variables mentioned above using a manipulator with three or more joints, then we will face control problems since the solution to the equations of motion carries an infinite number of solutions. This is because the manipulator carries more DoF than the workspace.

In this chapter, we will look at different ways to control a robotic arm that carries more DoF than the workspace, and we will provide different solution choices that can be chosen from these infinitely many solutions.

3.2. Terminology

When talking about the degrees of freedom (DoF), Craig [49] defines it as the number of independent position variables which would have to be specified in order to locate all parts of the mechanism. For example, a 4-bar mechanism has a single DoF even though there are three moving members of the mechanism. In a typical manipulator, the number of DoF is the same as the number of joints since it is an open kinematic chain. Degrees of redundancy, on the other hand, are referred to when the number of joints is greater than the dimension of the manipulation variable [50].

The end-effector, or sometimes referred to as the “gripper” defined as the free end of the chain of links that make the manipulator [49]. A work space is referred to as the space of which the manipulator’s end-effector can reach, or as Craig [49] defines it, it is the existence or non-existence of a kinematic solution of a given manipulator.

3.3. Redundant Manipulators Problem Formulation

Redundant manipulators can be of any size and shape that use revolute or prismatic joints among others, but in this chapter, we will limit our research on a seven DoF redundant robotic arm that has a full six DoF Cartesian workspace. By definition, this robotic arm has one degree of redundancy. The six controlled Cartesian variables in this case are the three positions in the x, y and z coordinates, and the three orientations or angles about the x, y and z axes. To be compatible with the test bed and the simulation that we will discuss in details in the coming chapters, a complete description of the manipulator’s physical characteristics will be discussed.

3.3.1. Frames of References

The first step in studying the kinematics of any robotic manipulator is to assign coordinate frames of references according to one of the known conventions and generate the kinematic parameters based on the selected convention. The most widely used convention is the standard convention used by Paul [51], but in our case, we will use the modified convention by Craig [49]. The results will still be the same, but it is only a matter of preference. Referring to figure 3.1, frames of each link should be attached in the following manner:

- 1- Assign the Z_i axis pointing along the i^{th} joint axis. If that link has no joints, such as the ground or the end-effector, any direction is permitted.
- 2- Assign the X_i axis pointing along the Z_i - Z_{i+1} common perpendicular. If the axes intersect, then X_i can be normal to the plane containing Z_i - Z_{i+1} axes.
- 3- Assign the Y_i axis according to the right-hand coordinate system.

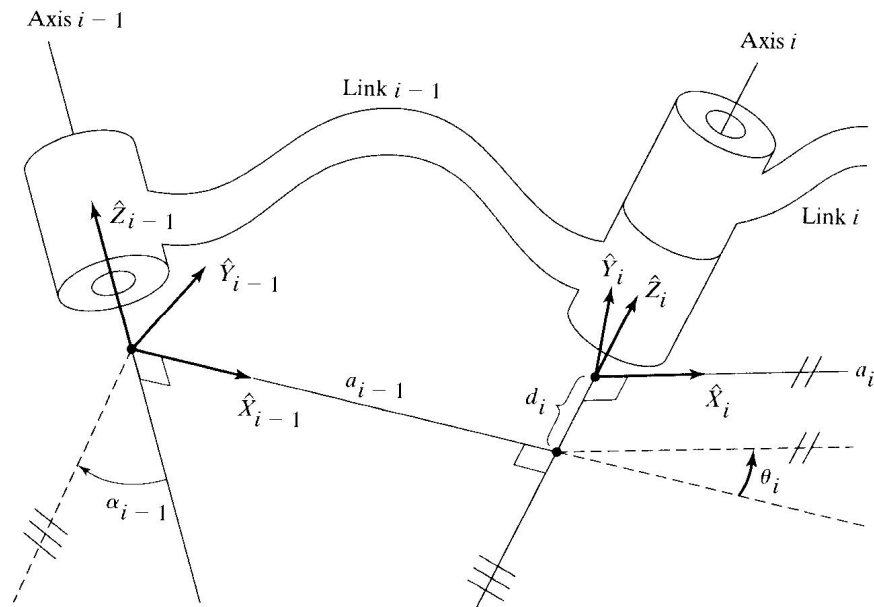


Figure 3.1: Joint-Link Kinematic Parameters.

3.3.2. Denavit-Hartenberg Parameters

There are four parameters that fully describe the kinematic relations between every neighboring joints and links in a manipulator, as shown in figure 3.1. These four parameters are:

- 1- Two neighboring joint relations:
 - a. The link length (parameter a).
 - b. The link twist angle (parameter α).
- 2- Two neighboring link relations:
 - a. The link offset (parameter d).
 - b. The joint angle (parameter θ).

These kinematic parameters are called the Denavit-Hartenberg parameters, or for short, D-H parameters. Gathering these parameters for all coordinate frames in a table allows a better view of the kinematic characteristics of the robotic arm. Assigning the frames as shown in the above steps will allow us to define the neighboring joint-link parameters as follows:

- 1- The value of " a_i " is defined as the distance from Z_i to Z_{i+1} measured along X_i .
- 2- The value of " α_i " is defined as the angle from Z_i to Z_{i+1} measured about X_i .
- 3- The value of " d_i " is defined as the distance from X_{i-1} to X_i measured along Z_i .
- 4- The value of " θ_i " is defined as the angle from X_{i-1} to X_i measured about Z_i .

The robotic arm at hand consists of seven revolute joints of which the rotation axes of every two immediate joints intersect. Figure 3.2 shows a Solid Works drawing of the new robotic manipulator that was designed and built at the University of South

Florida from the ground up [52]. For that manipulator, frame assignment for each link is shown in figure 3.3, and the D-H parameters are pointed out.

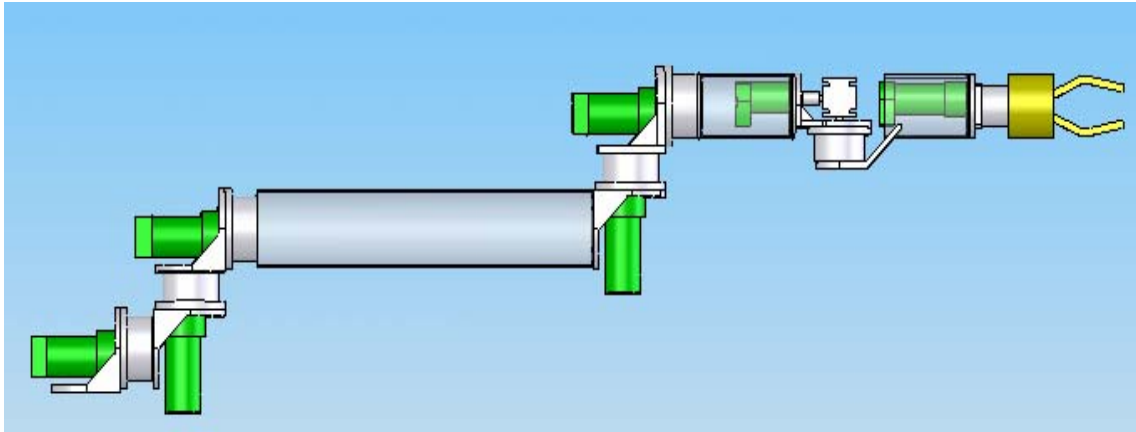


Figure 3.2: Solid Works Model of the New 7-DoF Robotic Arm Built at USF.

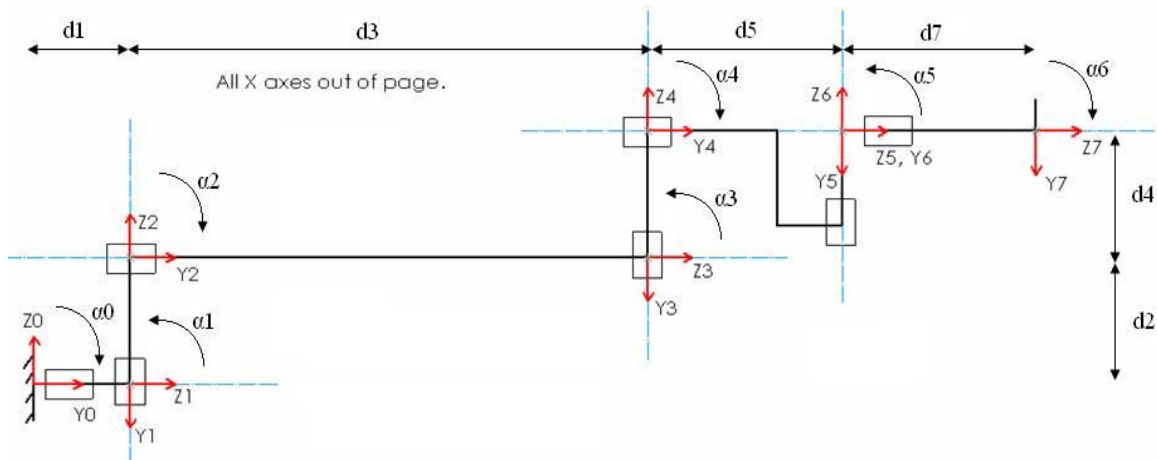


Figure 3.3: Frame Assignments and Dimensions of the New 7-DoF Robotic Arm.

Note that the rotational axes of the last three joints intersect at one point, this setup gives a mechanical advantage to the wrist both in calculations and in manipulation. The D-H parameters of the above manipulator are shown in table 3.1. Note that the value assigned to d_6 is zero since X_5 and X_6 are at the same axis line and the distance between them is zero. More details on this particular joint design will be discussed in a later chapter.

Table 3.1: The D-H Parameters of the New 7-DoF Robotic Arm Built at USF.

i	α_{i-1} (degrees)	a_{i-1} (mm)	d_i (mm)	θ_i (degrees)
1	-90	0	110	θ_1
2	90	0	146	θ_2
3	-90	0	549	θ_3
4	90	0	130	θ_4
5	-90	0	241	θ_5
6	90	0	0	θ_6
7	-90	0	179	θ_7

3.4. Forward Kinematics Equations

The aim of the forward kinematics is to solve the transformation equations for the end-effectors' Cartesian position and orientation or velocities when the joint angles and velocities are given. Even though this kind of control is not practical if used for task execution, but it is a step that has to be done before thinking of doing the inverse kinematic control.

3.4.1. Link Transformation Matrices

Homogeneous transformation matrices that transform the motion from one coordinate frame reference to the other can be easily obtained from the D-H parameters using the conventional equations [49] that relate every two consecutive frames to each other as follows:

$${}^{i-1}T_i = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i \cdot c\alpha_{i-1} & c\theta_i \cdot c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} \cdot d_i \\ s\theta_i \cdot s\alpha_{i-1} & c\theta_i \cdot s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} \cdot d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.1)$$

Where “s” is sine, and “c” is cosine of the angle. Applying the above formula to all seven reference coordinate frames gives the following homogeneous transformations:

$$\begin{aligned} {}^0_1T &= \begin{bmatrix} c\theta_1 & -s\theta_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ -s\theta_1 & -c\theta_1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, & {}^1_2T &= \begin{bmatrix} c\theta_2 & -s\theta_2 & 0 & 0 \\ 0 & 0 & -1 & -d_2 \\ s\theta_2 & c\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \\ {}^2_3T &= \begin{bmatrix} c\theta_3 & -s\theta_3 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ -s\theta_3 & -c\theta_3 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, & {}^3_4T &= \begin{bmatrix} c\theta_4 & -s\theta_4 & 0 & 0 \\ 0 & 0 & -1 & -d_4 \\ s\theta_4 & c\theta_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \\ {}^4_5T &= \begin{bmatrix} c\theta_5 & -s\theta_5 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ -s\theta_5 & -c\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, & {}^5_6T &= \begin{bmatrix} c\theta_6 & -s\theta_6 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s\theta_6 & c\theta_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{ and} \\ {}^6_7T &= \begin{bmatrix} c\theta_7 & -s\theta_7 & 0 & 0 \\ 0 & 0 & 1 & d_7 \\ -s\theta_7 & -c\theta_7 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \quad (3.2)$$

These homogeneous transformations describe the kinematic behavior of the robotic system at any instance of time. For instance, to find where frame 4 lies based on frame 3 when joint 4 is at certain angle, substituting that angle in the specified transformation matrix gives the position and orientation of frame 4 based on frame 3. The first 3x3 rows and columns of the homogeneous transform describes frame 4's unit

vectors projection on frame 3, and the first three rows of the last column of the homogeneous transform describes the position of frame 4's center based on frame 3. Propagating these matrices from one frame to the other gives us the forward kinematics of the robotic arm that describes the end-effectors' frame based on the base frame as follows:

$${}^0T_7 = {}^0T_1 \cdot {}^1T_2 \cdot {}^2T_3 \cdot {}^3T_4 \cdot {}^4T_5 \cdot {}^5T_6 \cdot {}^6T_7 \quad (3.3)$$

From this point on, we will use these transformation matrices as noted above. The rotation matrices and the frame's center coordinates extracted from these homogeneous transformation matrices will also be used as follows:

$${}^i_{i+1}T = \begin{bmatrix} {}^i_{i+1}R & {}^i_{i+1}P \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.4)$$

Where "R" is the 3x3 rotation matrix representation of the transform, and "P" is the vector containing the X, Y and Z coordinates of the origin of the frame.

3.4.2. Velocity Propagation and the Jacobian

Forces and velocities acting on the joints are crucial for the control of robotic manipulators. When a manipulator is controlled by sending a torque value to its joint motors, precise knowledge of the acting torques and forces on each joint is needed. The same is true when velocities are used to control the manipulators, each joints' velocity need to be determined so that the task can be executed as desired by the operator. Figure 3.4 shows the linear (v) and angular (ω) velocity vectors acting on neighboring links. These velocities are related together by the physical dimensions of the link that holds these two neighboring joints, and these dimensions are the same ones we obtained in the

previous sub-heading in the form of homogeneous transformation. The angular velocity of link “i+1” with respect to frame “i+1” can be defined as:

$${}^{i+1}\omega_{i+1} = {}^{i+1}R_i \cdot {}^i\omega_i + \dot{\theta}_{i+1} \cdot {}^{i+1}Z_{i+1} \quad (3.5)$$

Where “ $\dot{\theta}$ ” is the joint angular velocity, and “Z” is the projection of the Z-axis on its own frame of reference. This Z is usually $[0, 0, 1]^T$. Similarly, the linear velocity of the origin of frame “i+1” with respect to frame “i+1” can be defined as:

$${}^{i+1}v_{i+1} = {}^{i+1}R_i \cdot ({}^i v_i + {}^i\omega_i \times {}^i P_{i+1}) \quad (3.6)$$

Propagating these velocities throughout the joints will result in a full description of all velocities acting on every joint at any moment of time when the joint angles are provided.

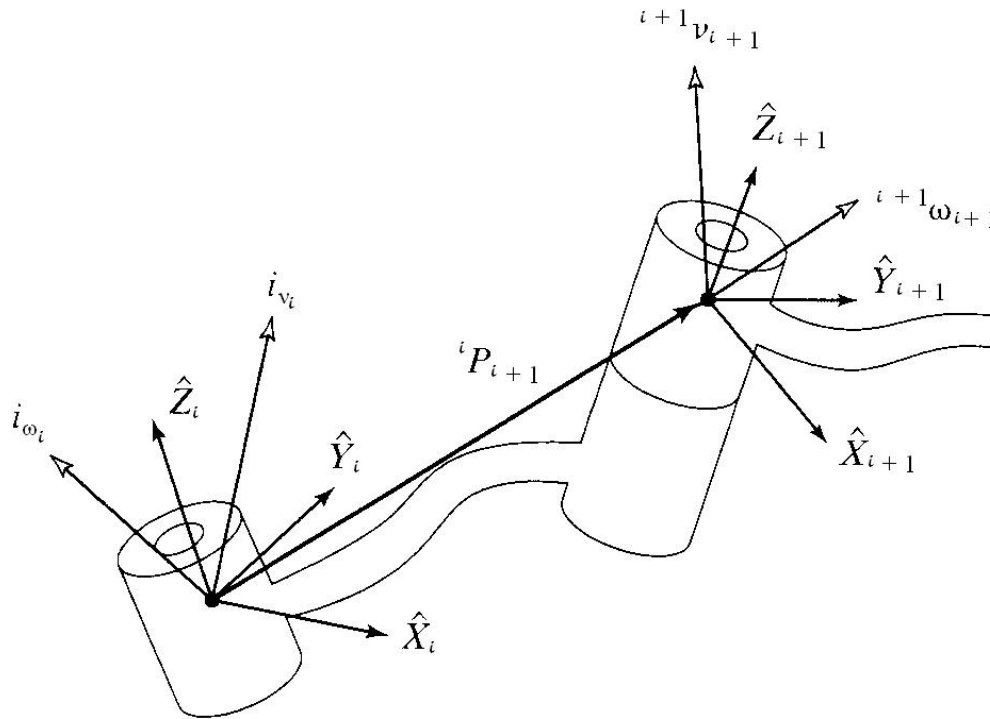


Figure 3.4: Velocity Vectors of Neighboring Links.

The same way we found the velocities at each joint, we can derive these relations in general using the Jacobian. The end-effectors' Cartesian coordinates are direct functions of the joint angles along the manipulator as follows:

$$\underline{X} = F(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7) \quad (3.7)$$

Where X is the 6x1 vector contains the 3 position and 3 orientation dimensions of the end-effector with respect to the base frame. To convert the dimensions into velocities, we can partial differentiate each of the Cartesian variables with respect to each of the joint angles. That gives:

$$\begin{aligned} \delta x &= \frac{\partial f_1}{\partial \theta_1} \delta \theta_1 + \frac{\partial f_1}{\partial \theta_2} \delta \theta_2 + \dots + \frac{\partial f_1}{\partial \theta_7} \delta \theta_7 \quad , \\ \delta y &= \frac{\partial f_2}{\partial \theta_1} \delta \theta_1 + \frac{\partial f_2}{\partial \theta_2} \delta \theta_2 + \dots + \frac{\partial f_2}{\partial \theta_7} \delta \theta_7 \quad , \\ \delta z &= \frac{\partial f_3}{\partial \theta_1} \delta \theta_1 + \frac{\partial f_3}{\partial \theta_2} \delta \theta_2 + \dots + \frac{\partial f_3}{\partial \theta_7} \delta \theta_7 \quad , \\ \delta \alpha &= \frac{\partial f_4}{\partial \theta_1} \delta \theta_1 + \frac{\partial f_4}{\partial \theta_2} \delta \theta_2 + \dots + \frac{\partial f_4}{\partial \theta_7} \delta \theta_7 \quad , \\ \delta \beta &= \frac{\partial f_5}{\partial \theta_1} \delta \theta_1 + \frac{\partial f_5}{\partial \theta_2} \delta \theta_2 + \dots + \frac{\partial f_5}{\partial \theta_7} \delta \theta_7 \quad , \\ \delta \gamma &= \frac{\partial f_6}{\partial \theta_1} \delta \theta_1 + \frac{\partial f_6}{\partial \theta_2} \delta \theta_2 + \dots + \frac{\partial f_6}{\partial \theta_7} \delta \theta_7 \end{aligned} \quad (3.8)$$

Or these can be re-written as:

$$\delta X = \frac{\partial F}{\partial \theta} \delta \theta \quad (3.9)$$

Note that we only have six equations, which are the Cartesian positions and orientations, and seven unknowns, which are the seven joint angles. This gives us an over-defined system, and we will talk about this in more details later in the chapter. Dividing both sides of (3.9) by the time increment (δt) gives the velocities as:

$$\frac{\delta X}{\delta t} = \frac{\partial F}{\partial \theta} \cdot \frac{\delta \theta}{\delta t} \quad \equiv \quad \dot{X} = \frac{\partial F}{\partial \theta} \cdot \dot{\theta} \quad \equiv \quad V = J(\theta)\dot{\theta} \quad (3.10)$$

Where “J(θ)” is the Jacobian matrix that relates the joint angular velocities to the end-effectors’ Cartesian velocities based on the base frame. At any time step, knowing the joint velocities and joint angles allows us to translate directly to the end-effector’s Cartesian velocities using the Jacobian. This Jacobian can also be used to relate the acting forces and moments at each joint to the end-effectors’ acting forces and moments.

3.5. Inverse Kinematic Equations

The aim of the inverse kinematics is to solve the transformation equations for the joint angles or velocities when the end-effectors’ Cartesian position and orientation or velocities are given. Most industrial robotic manipulators implement this kind of control for its practical use. In order to read the joint angles while the robot is running and supply it to the controller, encoders are necessary to be mounted at each joint for joint feedback. The solution of such equations can vary from a close-form solution to different numerical solutions depending on the size of the joint domain as compared to the size of the task domain.

3.5.1. Closed Form Solutions

A close form solution is an exact solution to the set of equations that relates the joint rates to the Cartesian velocities. This is possible when the number of joints in the manipulator is equal to the number of Cartesian variables to be controlled in the task space. In that case, the transformation matrix in the left-hand side of (3.3) is given, and we need to find the joint angles included in the right-hand side of that equation. This can

be done easily for simple robotic systems. For instance, it can be easy to find the joint angles of a two-link planar robotic manipulator when the X and Y coordinates of the tip of the arm are given by solving the two equations for the two unknowns. Doing so for more complex manipulator can result in very lengthy equations that will require extensive calculations and can be time consuming, which might be costly in terms of real-time control.

Another way of finding a close-form solution is by using the Jacobian given in equation (3.10). Inverting the Jacobian when it exists can directly give the joint rates if the Cartesian velocities and the current joint angles are given as follows:

$$\dot{\theta} = J^{-1}(\theta)V \quad (3.11)$$

A solution does not exist when the Jacobian is not at full rank, or when redundancy is introduced since the Jacobian in that case will not be a square matrix. When the number of joints exceeds the number of controlled coordinates in the workspace, the conventional inverse kinematics for a close-form solution is no longer applicable. Redundancy resolution and optimization schemes have been the subject of many researchers, where the use of the extra joints is employed to serve additional task executions and optimize the motion based on certain criterion.

Some researchers have altered these equations by adding more constraints based on certain criterion so that the number of equations matches the number of joint variables as Chang [25] did. He proposed a closed-form solution formula for inverse kinematics of redundant manipulators using Lagrange multiplier by proposing an additional set of equations to resolve the redundancy at the inverse kinematic level in such a way that a given criterion function may be minimized or maximized. The additional equations were

set in a similar way to the homogeneous solution term of the resolved rate method which uses the null space to resolve the redundancy. Although he used the manipulability index [53] as the criterion function, but any criterion function can be used as long as the function can be reduced to an expression in terms of joint variables only.

3.5.2. Manipulability Ellipsoid

For a Cartesian coordinate solution of the end-effector to exist, it is important to stay away from singular configurations of the robotic arm. A good way to ensure that a singular configuration is not reached is to find the determinant of the Jacobian matrix and make sure it is as far away from zero as possible. The closer the determinant to zero, the higher the joint velocities required to produce the desired Cartesian motion.

In the case where the Jacobian is not at full rank due to the fact that the matrix is not square, a different measure is required to ensure a smooth motion of the arm with no singularities along the way. Yoshikawa [53] have proposed a method that measures the manipulability measure for any manipulator with any size Jacobian. Consider the set of end-effectors' velocities \dot{X} that are accomplishable by the set of joint velocities such that the Euclidean norm satisfies:

$$\|\dot{\theta}\| = \sqrt{\dot{q}_1^2 + \dot{q}_2^2 + \dot{q}_3^2 + \dot{q}_4^2 + \dot{q}_5^2 + \dot{q}_6^2 + \dot{q}_7^2} \leq 1 \quad (3.12)$$

This set is an ellipsoid in the 6-dimensional Euclidean space shown in figure 3.5. The end-effector can move at high speed along the direction of the major axis of the ellipsoid, and only at low speed along the minor axis of the ellipsoid. Also, the larger the ellipsoid is, the faster the end-effector can move. A representative measure of how the

manipulator is able to move at a certain configuration is the volume of the ellipsoid at that particular configuration, and that can be represented by a scalar value as follows:

$$w = \sqrt{\det(J(\theta) \cdot J(\theta)^T)} \quad (3.13)$$

Where “w” is the manipulability measure at the configuration specified by the set of joint angles “ θ ”. This measure represents how far the manipulator is from singularities, the larger this measure is, the farther away from singularity the manipulator is. When this measure reaches zero, a manipulator is said to be at a singular configuration.

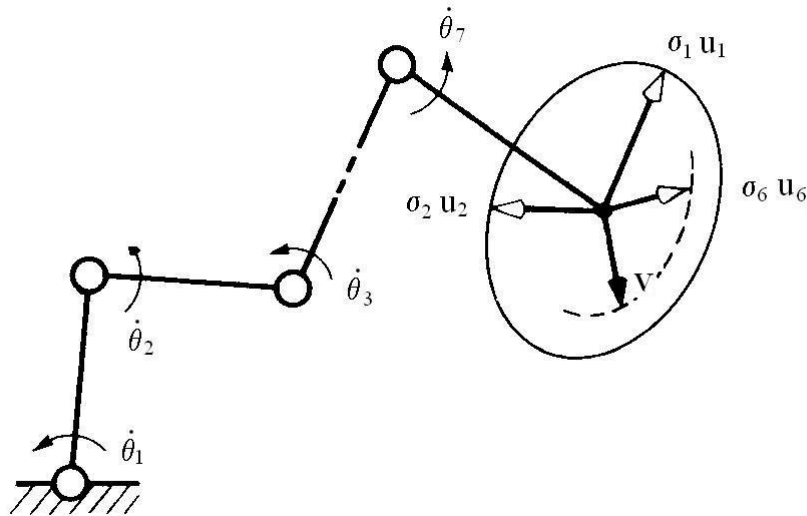


Figure 3.5: Manipulability Ellipsoid for a 7-DoF Manipulator in a 6-DoF Euclidean Space.

3.5.3. Numerical Solutions

Numerical solutions, such as Gauss’ elimination method, are less computationally expensive than the inverse solutions of the Jacobian [54]. These solutions can be implemented using the Jacobian to follow the user’s directional motion commands or to follow the desired trajectory. In the case of redundancy, the Jacobian is not a square

matrix any more, and that makes it un-invertible and not at full rank. Since the Jacobian is the key relation that relates the Cartesian space and the joint space variables, it is important to use different methods to invert this kind of non-square matrix. One of the most used methods in redundant manipulator controls is the Pseudo inverse, which can be used for numerical solutions as follows:

$$J^* = J^T \cdot (J \cdot J^T)^{-1} \quad (3.14)$$

To use the above equation, it is required that the rank of the jacobian matrix “J” is equal to the number of rows of that matrix. Redundancy can then be resolved using Pseudo inverse of the Jacobian to obtain a numerical solution of the joint angle rates using the following equation:

$$\dot{\theta} = J^*(\theta) \cdot V \quad (3.15)$$

This is one of the numerical solutions adopted in the earlier versions of the control system of the arm, which minimizes the Euclidean norm of errors as the optimization criterion. Other numerical solutions and optimization methods will be discussed later in this chapter and in chapter 5.

3.5.4. Redundancy Resolution

One of the first problems that needs to be taken care of in robotic manipulators is the singularity problem, which is the case when a solution does not exist at certain parts of the trajectory due to an odd configuration of the arm. Even with redundant manipulators, singular configurations may be reached along the process of following certain pre-specified trajectories. As mentioned earlier, manipulability measure is used as a factor to measure how far the current configuration is from singularity. In the case when

we have a redundant manipulator, achieving the same point in the work space can be done in an infinitely many configurations of the arm. Choosing one of these configurations can be done using different optimization criteria to be satisfied in case multiple solutions exist. When the trajectory is satisfied, and there exists a null space that can be used to select one of the infinite solutions that satisfy the trajectory, and at the same time satisfy a chosen criterion function, a valid configuration is selected that is solved based on both original requirement and the criteria constraints. One of these criterion functions or restrictions can be the maximization of the manipulability measure since it is crucial to the task execution.

Optimizing the solution can be achieved by adding an additional term to equation (3.15) that carries a sub-task to be considered in case more than one solution to the basic task exists. This secondary task term is added as follows:

$$\dot{\theta} = J^*(\theta) \cdot V + (I_7 - J^*(\theta) \cdot J(\theta)) \cdot f \quad (3.16)$$

Where “f” is a 7x1 vector representing the secondary task, “J^{*}” is a 7x6 inversion of the Jacobian matrix, and I₇ is the identity matrix of size 7. The choice of the criterion function can range from a scalar quantity, such as the manipulability measure, or can be a set of functions, such as joint limit avoidance conditions.

3.5.5. Optimization Criteria

As mentioned earlier, optimization can take the form of a scalar or a set of equations to be considered in the optimization process. When an exact solution does not exist, equation (3.16) covers all the least square solutions that minimize the Euclidean norm of errors while maintaining minimum joint velocity norms as follows:

$$\text{Fulfil } (\min \|V - J(\theta) \cdot \dot{\theta}\|) \text{ while maintaining } (\min \|\dot{\theta}\|) \quad (3.17)$$

At many instances, the main objective function to follow a certain trajectory does not require the use of all available joints of the manipulator, For instance, rotating the wrist around the end-effector's rotation axis requires only the last joint (joint 7) to move. This leaves all 6 other joints available for optimization, and at that moment, the arm would have six degrees of redundancy. The null space in this case containing the other six joints can be used to optimize for more than one criterion. Some of these criteria are:

- 1- Maximizing the manipulability measure.
- 2- Minimizing the joint velocities.
- 3- Minimizing the energy.
- 4- Avoiding obstacles.
- 5- Avoiding joint limits.
- 6- Pointing at certain point while moving along a required trajectory.

Some of the above criteria can be used together in a priority-based level that will realize the higher priority criterion while the main task is being executed, and then go on to the lower priority tasks if null space still exists until all joints are being used or all criteria have been realized.

3.6. Summary

In this chapter, a mathematical model of a 7-DoF redundant robot is described. The arm consists of seven revolute joints with intersecting axes of rotation between every two neighboring joints. The problem was formulated by assigning coordinate frames to each one of the links according the modified convention of frame assignments. The D-H

parameters of the system were generated to calculate the relations between every two consecutive joints and links. The forward kinematic equations were generated, and the total homogeneous transformation matrix of the robotic arm was created. The velocity relations between the links were propagated to find the end-effectors' Cartesian velocity relations to the joint velocities, and these relations lead to the Jacobian matrix that relates the work space and the joint space together. Inverse kinematic equations were generated to find the joint positions or velocities in case the required Cartesian coordinates are given. Different methods of doing the inverse kinematics were discussed, and the optimized redundancy resolution scheme used in the control was shown.

Chapter 4:

Mobility Control Theory

4.1. Introduction

“Mobile Manipulator” is a widespread term that refers to robots that combine capabilities of locomotion and manipulation. When these systems are devoted to indoor tasks, they are often equipped with powered wheels. The arrangement of the wheels and their actuation device determine the holonomic or non-holonomic nature of this locomotion system. Some wheeled mobile manipulators built from an omni-directional platform are holonomic, and many of them are non-holonomic. The tasks assigned to these systems are often translated in terms of end-effectors’ motion. Although this concept is well known for robotic arms, it is quite different in the case of non-holonomic systems [50, 54]. A distinction between the two types of motion is that the holonomic motion can sufficiently move in any direction of its workspace, whereas the non-holonomic motion can not move in arbitrary directions of its workspace.

In this chapter, we will deal with the non-holonomic motion, which is the opposite problem of redundancy discussed in chapter three. Different control methods will be discussed, and the chosen motion planning strategy will be derived so that the lost degree-of-freedom can be compensated by using trajectory planning.

4.2. Terminology

Holonomic motion refers to the relationship between the controllable and total number of degrees of freedom of a given platform. If the controllable DoF is greater than or equal to the total DoF in the workspace, then the platform is said to be holonomic [55]. If the controllable DoF is less than the total DoF in the workspace, then the platform is said to be non-holonomic [56]. Examples of non-holonomic platforms are cars, power wheelchairs and other mobile platforms that can, at any given moment, move in two dimensions out of the three planar dimensions (i.e. motion along the X-axis, Y-axis and rotation about the Z axis). In contrast, holonomic platforms are platforms that can move at any moment of time in all three planar dimensions, such as platforms that are equipped with three power omni-directional wheels.

Three different variables are available for the planar motion of the mobile platform moving on the ground, two positional variables along the X-axis and Y-axis, and one rotational variable about the Z-axis. The power wheelchair used in this work is a non-holonomic wheelchair that can move along the X-axis and rotate about the Z-axis.

4.3. Mobility Problem Formulation

The wheelchair used in this work is an “Action Ranger X Storm Series” power wheelchair. This wheelchair accomplishes its non-holonomic motion using a differential drive that carries two independently-driven wheels in the back of the power wheelchair. The front of the wheelchair has two passive castors that are placed to support the wheelchair’s motion. This makes the wheelchair a 2-DoF system that moves in plane

[48]. A full description of the wheelchair and its important dimensions will be discussed in this section.

4.3.1. Frame Assignment

Three important points of interest were assigned on and around this wheelchair, and coordinate frames were assigned on these three points. These three frames are the wheelchair's coordinate frame assigned at the center of the driving wheels' axle, the ground frame assigned at an arbitrary location on the ground floor, and another frame called frame "A" assigned at the point where the 7-DoF robotic arm will be mounted. Figure 4.1 shows two-dimensional top and side views of the Solid Works™ model of the wheelchair with the key dimensions and the frame assignments. Note that these three frames are independent and the frame assignment rules discussed in chapter 3 do not apply. For simplicity, the orientation of the wheelchair's frame and the "A" frame were assigned such that the rotation matrix between the two is identity.

Throughout the development of the equations in this section and in the subsequent sections, these assigned frames will be used to define the relationships between the ground, the wheelchair, the arm base and the end-effector (gripper). The assignment of the ground frame is arbitrary because it doesn't change any of the kinematics of the WMRA system. The wheels' axle frame is assigned because of its importance in the generation of equations between the ground and the arm base. The arm base coordinate frame is assigned to link the end of the wheelchair kinematics to the robotic arm kinematics as it is mounted on the wheelchair. The end-effector's frame is the frame that will carry on the assigned tasks in the Cartesian space.

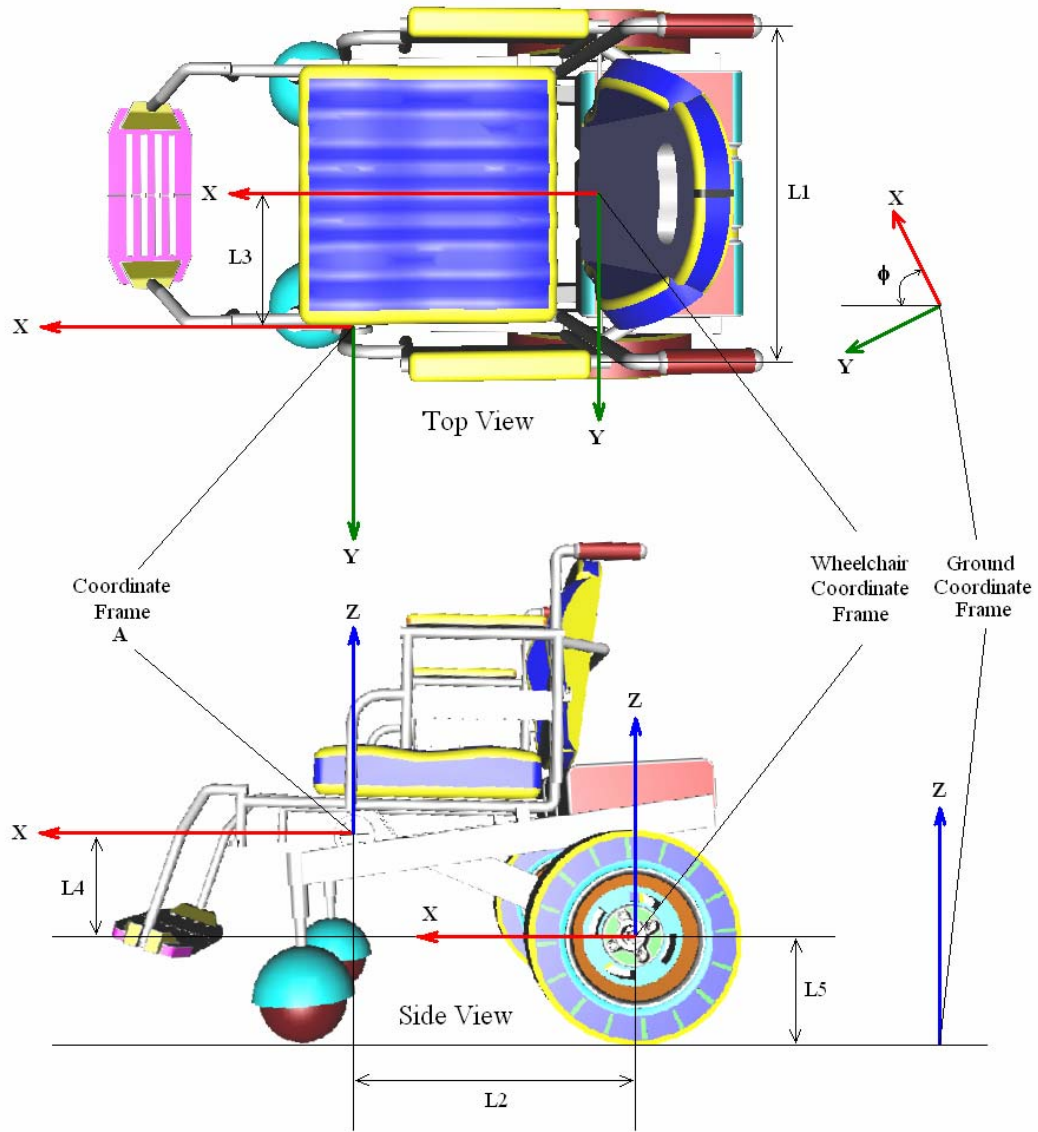


Figure 4.1: Wheelchair Coordinate Frames and Dimensions of Interest.

4.3.2. Wheelchair's Important Dimensions

There are five important dimensions that will be used in the derivation of coordinate relations in the next section. These dimensions are shown in figure 4.1, and they represent the physical dimensions of the wheelchair as well as the coordinate frame distances. These dimensions can be described as follows:

- 1- The distance between the centers of the two driving wheels along the differential drive axle. This distance is noted as “L1” and its value is 560 mm.
- 2- The offset distance from the center of the differential drive to the center of frame “A” along the wheelchair’s X-axis. This distance is noted as “L2” and its value is 440 mm.
- 3- The offset distance from the center of the differential drive to the center of frame “A” along the wheelchair’s Y-axis. This distance is noted as “L3” and its value is 230 mm.
- 4- The offset distance from the center of the differential drive to the center of frame “A” along the wheelchair’s Z-axis. This distance is noted as “L4” and its value is 182 mm.
- 5- The offset distance from the center of the differential drive to the center of the ground frame along the wheelchair’s Z-axis, which is the same as the wheelchair’s driving wheels’ radius. This distance is noted as “L5” and its value is 168 mm.

An important note to mention at this point is that the transformation between the wheelchair’s coordinate frame and the “A” coordinate frame is constant since both frames were attached to the power wheelchair independently from its wheels’ motion. On the other hand, the transformation between the ground frame and the wheelchair’s frame depends on three variables, the distance along the X-axis and Y-axis and the orientation of the wheelchair about the Z-axis, denoted by “ Φ ”. These variables represent the mobility of the wheelchair on the planar ground surface.

4.4. Homogeneous Transformation Relations

The homogeneous transformations between the coordinate frames assigned to the wheelchair, the ground and the robotic arm base depend on the motion of the two differentially-driven wheels. The details of generating these relations will be discussed thoroughly in this section.

4.4.1. Driving Wheels' Motion and the Turning Angle

In the application of mobile robots, wheel slippage can be considered when the wheels' characteristics are constant, but in our application, we will assume that slip is compensated by the user. The classical way of obtaining the distance travelled from an initial position to the final position of a wheel that is turning with angle “ θ ” as shown in figure 4.2 can be written as:

$$d = L_5 \cdot \theta \quad (4.1)$$

Where “ d ” is the travelled distance, and “ L_5 ” is the wheel's radius. This is the case when the wheel moves in a straight motion with no turning.

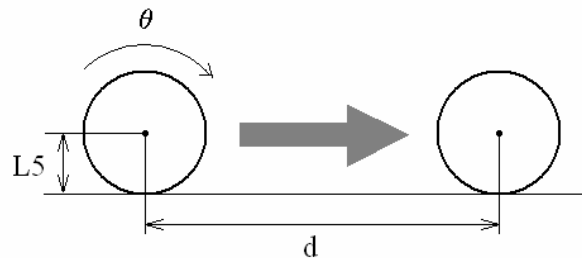


Figure 4.2: Traveled Distance of a Turning Wheel.

In our case, the wheelchair is equipped with two wheels, and the above motion is a special case that commands the wheelchair to move in a straight forward fashion. In general, each independent wheel moves independently at its own velocity, and a turning

angle is introduced. To approach the formulation of the turning angle, let's assume that the left wheel is stationary and the right wheel is turning as shown in figure 4.3.

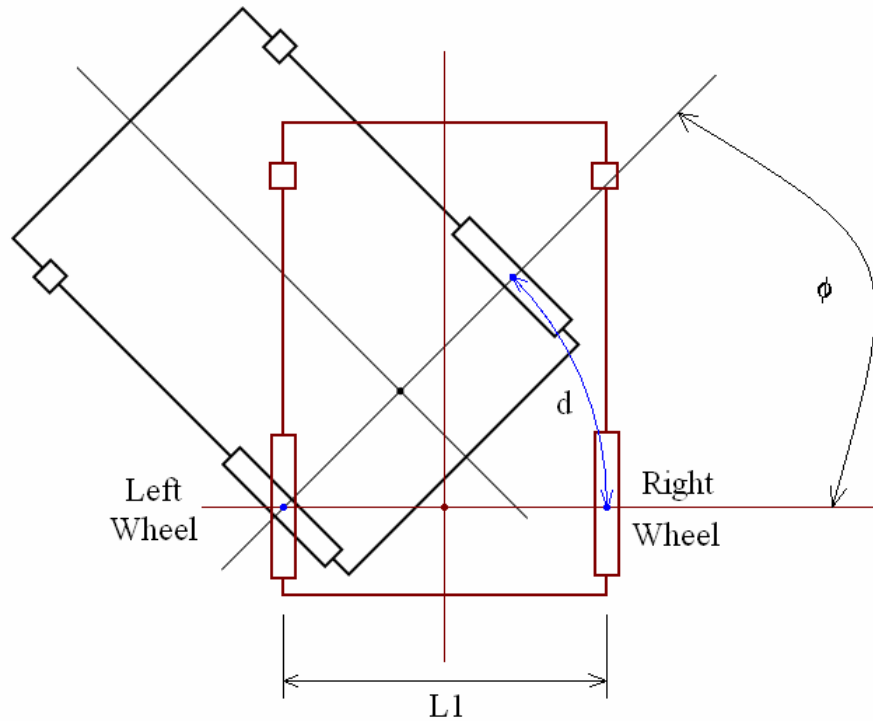


Figure 4.3: Traveled Distance with Turning Angle.

The curved blue line in the figure is the actual traveled distance of the right wheel when the left wheel is stationary. The angle of rotation in this case is:

$$\phi = \frac{d}{L_1} = \frac{L_s}{L_1} \cdot \theta_r \quad (4.2)$$

Where " L_1 " is the wheelbase width and " θ_r " is the right wheel turning angle. In the case when the left wheel is moving while the right wheel is stationary, the turning angle would be the same as in (4.2) with a negative sign. When both wheels are turning at different amplitude, the turning angle would be directly related to the difference between the two angles as follows:

$$\phi = \frac{L_5}{L_1} \cdot (\theta_r - \theta_l) \quad (4.3)$$

Where “ θ_l ” is the rotation angle of the left wheel. The above relation is not enough to describe consecutive motion steps that will be implemented in the Virtual Reality simulation and in the implementation of the physical system. We need to relate the previous step to the current step at any moment of time to relate all steps together. If we have the turning angle from the previous step, we can assume that that angle was the starting angle, and that the coming step will carry the translation through next turning angle increment. This can be realized as follows:

$$\phi = \phi_0 + \frac{L_5}{L_1} \cdot (\theta_r - \theta_l) \quad (4.4)$$

Where “ ϕ_0 ” is the resultant turning angle from all previous steps added together. This gives us a continuous angle tracking throughout the motion of the power wheelchair even when the turning angle was fluctuating.

4.4.2. The Radius of Curvature

The turning angle is not the only factor needed in this non-holonomic motion of the platform. The radius of curvature “ r ” is also needed [61], and it is critical to the calculations of the transformation matrices. Four cases are to be considered for the radius of curvature. These cases are as follows:

1- When $r \geq 0$:

As shown in figure 4.4, this case happens when the left wheel is turning less than the right wheel. In this case, the radius of curvature is defined as:

$$r = L_5 \cdot \frac{\theta_l}{\phi} \quad (4.5)$$

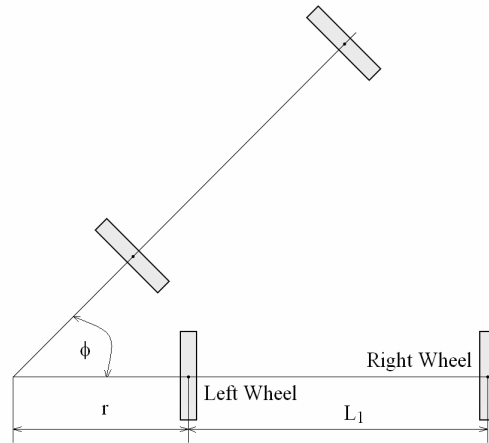


Figure 4.4: Radius of Curvature in Case 1.

2- When $-L_1 \leq r \leq 0$:

As shown in figure 4.5, this case happens when the left wheel is turning in the opposite direction of the right wheel. In this case, the radius of curvature is negative since the left wheel is moving in the negative direction. The radius of curvature in this case is defined as:

$$r = L_5 \cdot \frac{\theta_l}{\phi} \quad (4.6)$$

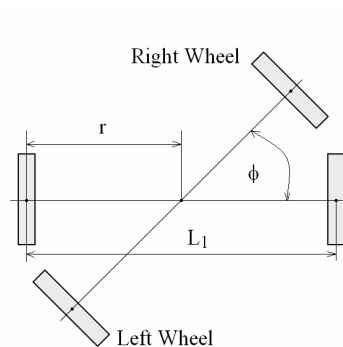


Figure 4.5: Radius of Curvature in Case 2.

3- When $r \leq -L_1$:

As shown in figure 4.6, this case happens when the right wheel is turning less than the left wheel. In this case, the radius of curvature will be negative since the turning angle of the wheelchair is in the negative direction. The radius of curvature in this case is defined as:

$$r = L_5 \cdot \frac{\theta_r}{\phi} - L_1 \quad (4.7)$$

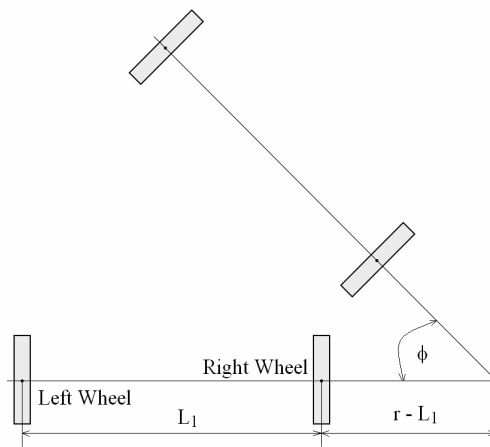


Figure 4.6: Radius of Curvature in Case 3.

4- When $r \cong \pm\infty$

As shown in figure 4.7, this case happens when the left wheel is turning in the same direction and amplitude as that of the right wheel. In this case, the radius of curvature will be infinity since there is no turning angle involved in this kind of motion.

The above four cases cover all motion possibilities the wheelchair is capable of. At this point, we have all the necessary information to generate the transformation matrices along all the points of interest that were pointed out earlier in the chapter.

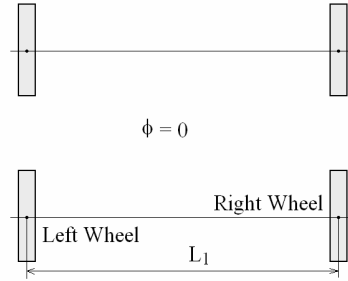


Figure 4.7: Radius of Curvature in Case 4.

4.4.3. Point-to-Point Transformation of the Wheelchair

To transform the wheelchair's coordinate frame during motion, we assume that the initial position and orientation of the frame is known, and we need to find the new position and orientation for the next time step. Let the initial coordinate frame of the wheelchair be “W₀” and the next coordinate frame after moving one step is “W₁” as shown in figure 4.8. Lets also define point “O” as the point where the extension of the two Y-axes intersect on plane XY.

Knowing the transformation between “W₀” and “W₁” gives us the perspective that we were looking for when the wheelchair is in motion. To accomplish this transformation, three sub-transformations are to be performed:

- 1- Transformation along “Y₀” by the amount of $r + \frac{L_1}{2}$ to reach point “O”.
- 2- Rotation about “Z₀” by the amount of “ ϕ_1 ” to reach the orientation of “W₁”.
- 3- Transformation along “Y₁” by the amount of $-r - \frac{L_1}{2}$ to reach point “W₁”.

These three transformations define coordinate frame “W₁” based on coordinate frame “W₀”, which is:

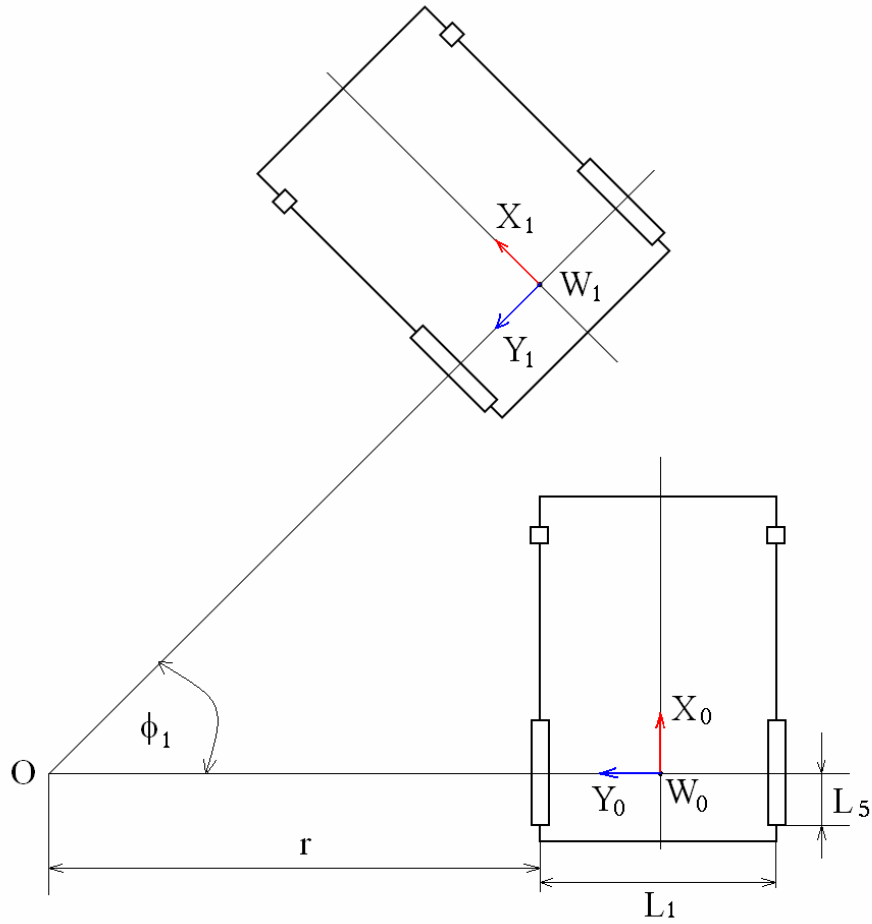


Figure 4.8: Point-to-Point Transformation of Frames.

$${}_{w_1}^{w_0}T = D_y\left(r + \frac{L_1}{2}\right) \cdot R_z(\phi_1) \cdot D_y\left(-r - \frac{L_1}{2}\right), \text{ or}$$

$${}_{w_1}^{w_0}T = \begin{bmatrix} C\phi_1 & -S\phi_1 & 0 & S\phi_1 \cdot \left(r + \frac{L_1}{2}\right) \\ S\phi_1 & C\phi_1 & 0 & (1 - C\phi_1) \cdot \left(r + \frac{L_1}{2}\right) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.8)$$

If we assume that the initial coordinate frame of the wheelchair “W₀” was a result of previous transformation from the ground origin “G” as illustrated in figure 4.1, the

resulting homogeneous transformation from the ground frame “G” to the wheelchair’s initial frame “W₀” can be expressed as:

$${}^G T_{W_0} = \begin{bmatrix} C\phi_0 & -S\phi_0 & 0 & P_{0x} \\ S\phi_0 & C\phi_0 & 0 & P_{0y} \\ 0 & 0 & 1 & L_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.9)$$

Multiplying (4.8) and (4.9) together results in the relation between the ground coordinate frame “G” and the final coordinate frame of the wheelchair “W₁” as follows:

$${}^G T_{W_1} = {}^G T_{W_0} \cdot {}^{W_0} T_{W_1}, \text{ or}$$

$${}^G T_{W_1} = \begin{bmatrix} C(\phi_0 + \phi_1) & -S(\phi_0 + \phi_1) & 0 & S\left(\frac{\pi}{2} + \phi_0 + \frac{\phi_1}{2}\right) \cdot \left(r + \frac{L_1}{2}\right) \cdot \frac{S\phi_1}{C(\phi_1/2)} + P_{0x} \\ S(\phi_0 + \phi_1) & C(\phi_0 + \phi_1) & 0 & -C\left(\frac{\pi}{2} + \phi_0 + \frac{\phi_1}{2}\right) \cdot \left(r + \frac{L_1}{2}\right) \cdot \frac{S\phi_1}{C(\phi_1/2)} + P_{0y} \\ 0 & 0 & 1 & L_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.10)$$

In the case when the wheelchair is moving in a straight line as described in the previous sub-heading in “case 4”, the relation in (4.8) can be simplified to a pure translation as follows:

$${}^{W_0} T_{W_1} = \begin{bmatrix} 1 & 0 & 0 & L_5 \cdot \theta_r \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.11)$$

Following the same procedure above, a simpler solution can be reached for this special case that relates the ground coordinate frame to the wheelchair’s coordinate frame.

4.4.4. Transformation to the Robotic Arm's Base

So far, we have found the homogeneous transformation matrix that relates the ground coordinate frame to the wheelchair's coordinate frame. For the purpose of the robotic arm to be mounted on the wheelchair, one more transformation is required between the wheelchair's coordinate frame and the robotic arm base coordinate frame where it attaches to the wheelchair. This transformation will be constant since the arm base and the wheelchair are both attached together in a rigid mounting bracket at point "A" as shown in figure 4.1. This constant transformation is basically a translation as follows:

$${}^{w_1}T_A = \begin{bmatrix} 1 & 0 & 0 & L_2 \\ 0 & 1 & 0 & L_3 \\ 0 & 0 & 1 & L_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.12)$$

Post multiplying (4.10) by (4.12) gives the transformation needed to describe the robotic arm base at any moment based on the ground frame. This resultant matrix is required to be evaluated at every time step to know the current position and orientation of the wheelchair and arm base referenced to the world coordinate frame.

4.5. Wheelchair Velocities

In wheelchair motion, velocities are mapped from the wheels' motion to the robotic arm base motion so that the results can be ready when the robotic arm is put together with the wheelchair. In this subsection, the velocity relations will be generated and the Jacobian matrix will be derived. Depending on the location of the robotic arm base on the wheelchair, three cases will be studied to make the general form that we were

looking for in the Jacobian. When the wheelchair is moving in a straight line, the velocity component at any point on the wheelchair is the same. But in the case of rotation, velocity components throughout the different points on the wheelchair will be different [44]. We will take the case of pure rotation to derive the velocity relations, and then we will add the component that is coming from the straight line motion to the velocities gathered from the rotation.

4.5.1. Wheelchair Velocity Mapping to the Robotic Arm Base

The relation between the wheelchair's coordinate frame and the robotic arm base frame can be found in three cases as follows:

1- Case I: When the Robotic Arm's Offset is in the X-Direction:

In this case, the length " L_3 " is set to zero as shown in figure 4.9, and the robotic arm is installed directly in front of the wheelchair along the line that divides the wheelchair into two symmetrical halves. Wheelchair motion in this context produces three relative motions at point "A", two linear motions along X-axis and Y-axis, as well as a rotational motion about Z-axis. These three components are:

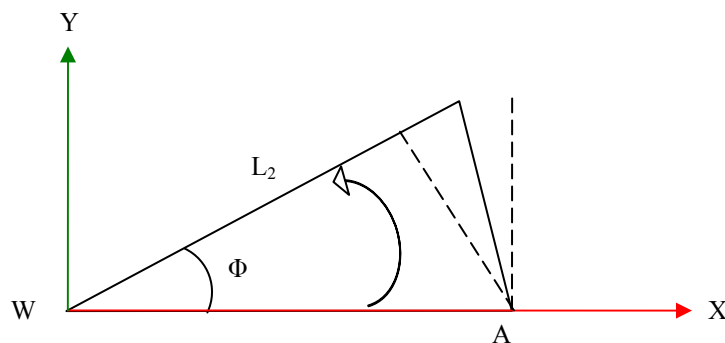


Figure 4.9: The Case When " L_3 " is Zero.

$$\begin{aligned}
\dot{X}_A &= -L_2 \cdot \dot{\phi}_W \cdot S\phi_W \\
\dot{Y}_A &= L_2 \cdot \dot{\phi}_W \cdot C\phi_W \\
\dot{\phi}_A &= \dot{\phi}_W
\end{aligned}
\tag{4.13}$$

2- Case II: When the Robotic Arm's Offset is in the Y-Direction:

In this case, the length “ L_2 ” is set to zero as shown in figure 4.10, and the robotic arm is installed directly in on the wheelchair's axle along the line of rotation of the two driving wheels. Wheelchair motion in this context also produces three relative motions at point “A”, two linear motions along X-axis and Y-axis, as well as a rotational motion about Z-axis. These three components are:

$$\begin{aligned}
\dot{X}_A &= -L_3 \cdot \dot{\phi}_W \cdot C\phi_W \\
\dot{Y}_A &= -L_3 \cdot \dot{\phi}_W \cdot S\phi_W \\
\dot{\phi}_A &= \dot{\phi}_W
\end{aligned}
\tag{4.14}$$

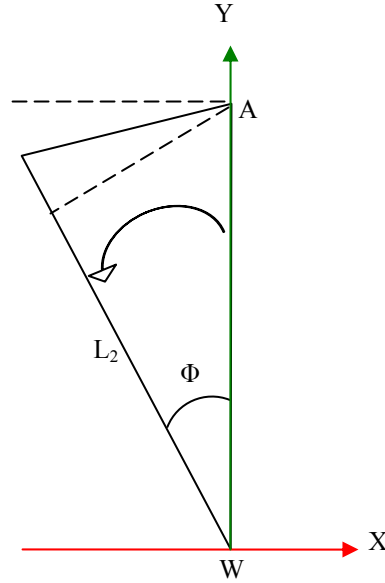


Figure 4.10: The Case When “ L_2 ” is Zero.

3- Case III: When the Robotic Arm's Offset is in Both Directions:

This case combines the above two cases with a general understanding of the used arm location on the wheelchair. In this case, both offsets are accounted for and algebraically added together with their signs. Wheelchair motion in this context also produces three relative motions at point “A”, two linear motions along X-axis and Y-axis, as well as a rotational motion about Z-axis. Adding the component of linear motion of the wheelchair produces the general motion formula that relates the wheelchair frame velocities to the arm bas frame velocities as follows:

$$\begin{aligned}\dot{X}_A &= \dot{X}_W - L_2 \cdot S\phi_W \cdot \dot{\phi}_W - L_3 \cdot C\phi_W \cdot \dot{\phi}_W \\ \dot{Y}_A &= \dot{Y}_W + L_2 \cdot C\phi_W \cdot \dot{\phi}_W - L_3 \cdot S\phi_W \cdot \dot{\phi}_W \\ \dot{\phi}_A &= \dot{\phi}_W\end{aligned}\quad (4.15)$$

4.5.2. Mapping the Driving Wheels' Velocities to the Wheelchair

The relation between the wheelchair's coordinate frame and the two driving wheels can be found by studying two cases, one case is when the left wheel is stationary while the right wheel is rotating as shown in figure 4.11, and the other case is when the right wheel is stationary and the left wheel is moving as shown in figure 4.12.

Algebraically adding the two terms together gives the general relationship between the wheels' motion and the wheelchair frame's motion as follows:

$$\begin{aligned}\dot{X}_W &= \frac{L_5}{2} \cdot C\phi_W \cdot \dot{\theta}_l + \frac{L_5}{2} \cdot C\phi_W \cdot \dot{\theta}_r \\ \dot{Y}_W &= \frac{L_5}{2} \cdot S\phi_W \cdot \dot{\theta}_l + \frac{L_5}{2} \cdot S\phi_W \cdot \dot{\theta}_r \\ \dot{\phi}_W &= \frac{-L_5}{L_1} \cdot \dot{\theta}_l + \frac{L_5}{L_1} \cdot \dot{\theta}_r\end{aligned}\quad (4.16)$$

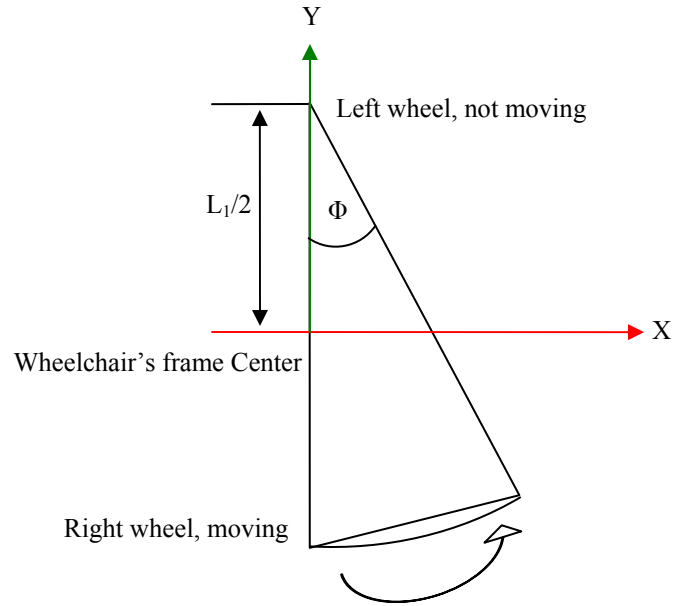


Figure 4.11: The Case When the Left Wheel is Stationary.

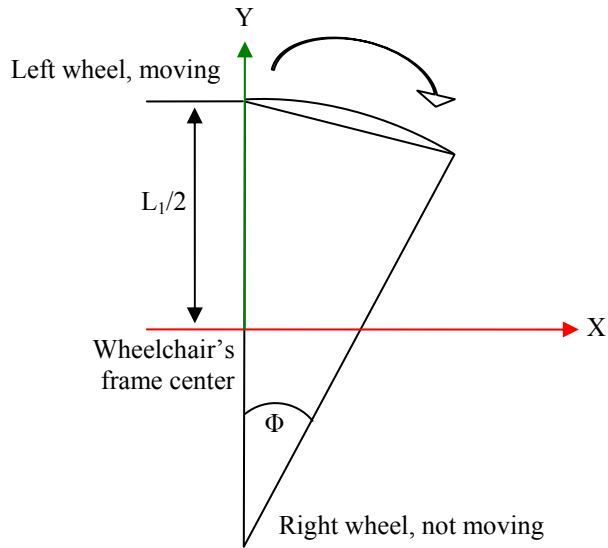


Figure 4.12: The Case When the Right Wheel is Stationary.

Equation (4.16) carries a full mapping between the velocities of the wheels and the wheelchair velocities, while equation (4.15) carries a full mapping between the velocities of the arm base and the wheelchair velocities.

4.6. Wheelchair's General Jacobian

To simplify the velocity relations and find the Jacobian, let us rewrite equation (4.15) in the form of a matrix as follows:

$$\begin{bmatrix} \dot{X}_A \\ \dot{Y}_A \\ \dot{\phi}_A \end{bmatrix} = \begin{bmatrix} 1 & 0 & -(L_2 \cdot S\phi + L_3 \cdot C\phi) \\ 0 & 1 & L_2 \cdot C\phi - L_3 \cdot S\phi \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \dot{X}_W \\ \dot{Y}_W \\ \dot{\phi}_W \end{bmatrix} \Rightarrow \dot{V}_A = J_{WA} \cdot \dot{V}_W \quad (4.17)$$

Where “ J_{WA} ” is the Jacobian that relates the wheelchair's Cartesian velocities to the arm base Cartesian velocities. Also, equation (4.16), can be rewritten in a matrix form as follows:

$$\begin{bmatrix} \dot{X}_W \\ \dot{Y}_W \\ \dot{\phi}_W \end{bmatrix} = \frac{L_5}{2} \cdot \begin{bmatrix} C\phi & C\phi \\ S\phi & S\phi \\ -\frac{2}{L_1} & \frac{2}{L_1} \end{bmatrix} \cdot \begin{bmatrix} \dot{\theta}_l \\ \dot{\theta}_r \end{bmatrix} \Rightarrow \dot{V}_W = J_{whW} \cdot \dot{\theta}_{wh} \quad (4.18)$$

Where “ J_{whW} ” is the Jacobian that relates the wheelchair's motion and the driving wheels' motion. To obtain the general Jacobian that relates the wheels' velocities to the arm base frame velocities, a dot product of the two Jacobians can be performed as follows:

$$\dot{V}_A = J_{WA} \cdot J_{whW} \cdot \dot{\theta}_{wh} \Rightarrow \dot{V}_A = J_{whA} \cdot \dot{\theta}_{wh}, \text{ or} \quad (4.19)$$

$$\begin{bmatrix} \dot{X}_A \\ \dot{Y}_A \\ \dot{\phi}_A \end{bmatrix} = \frac{L_5}{2} \cdot \begin{bmatrix} C\phi + \frac{2}{L_1} \cdot (L_2 \cdot S\phi + L_3 \cdot C\phi) & C\phi - \frac{2}{L_1} \cdot (L_2 \cdot S\phi + L_3 \cdot C\phi) \\ S\phi - \frac{2}{L_1} \cdot (L_2 \cdot C\phi - L_3 \cdot S\phi) & S\phi + \frac{2}{L_1} \cdot (L_2 \cdot C\phi - L_3 \cdot S\phi) \\ -\frac{2}{L_1} & \frac{2}{L_1} \end{bmatrix} \cdot \begin{bmatrix} \dot{\theta}_l \\ \dot{\theta}_r \end{bmatrix} \quad (4.20)$$

The above equation will be used with the numerical methods to produce the motion commanded by the user in Cartesian coordinates after calculating the wheels' velocities required to realize the commanded motion.

4.7. Trajectory Options

As discussed earlier in this chapter, non-holonomic constraints on mobile platforms restrict the system's ability to control all DoF in the workspace. The above equations give the user the choice to control two out of three variables in the planar Cartesian coordinates. To make sense of the two chosen variables, it is more important to select the two position variables of the wheelchair rather than the orientation and one of the two positions. This way, the wheelchair is free to move in the plane in both direction, but without proper orientation.

Adding trajectory planning to the motion can compensate for the lack of control variables by dividing the motion into sub-motions to realize two of the three commanded variables at every sub-motion. Suppose that the wheelchair is commanded to move the arm's base reference frame from " T_0 " position to " T_1 " position, where " T " is the homogeneous transformation matrix of that position, the motion can be planned in three steps to realize the X-direction motion, Y-direction motion and the Z-direction orientation. The following steps can be programmed to execute these three motions:

- 1- From the initial point of the arm base at " T_0 ", find the corresponding wheelchair's frame transformation matrix at that pose using equation 4.12.
- 2- From the destination point of the arm base at " T_1 ", find the corresponding wheelchair's frame transformation matrix at that pose using equation 4.12.

- 3- Draw a line between the two new frame transformations of the wheelchair's frame, and find the angle of that line using the transformation resultant between the two.
- 4- Command the wheelchair to rotate to the angle of the new line with no translation.
- 5- Command the wheelchair to move in a straight line from the initial position to the final position of the wheelchair, ignoring the orientation.
- 6- Command the wheelchair to rotate from the angle of the new line to the angle of the final position.

The above steps with the three sub-motions produced as shown in figure 4.13 will make the user capable of controlling all three DoF in the workspace of the wheelchair's planar motion. Simulation testing of this method has been done and was successful to move to any position and orientation on the ground plane.

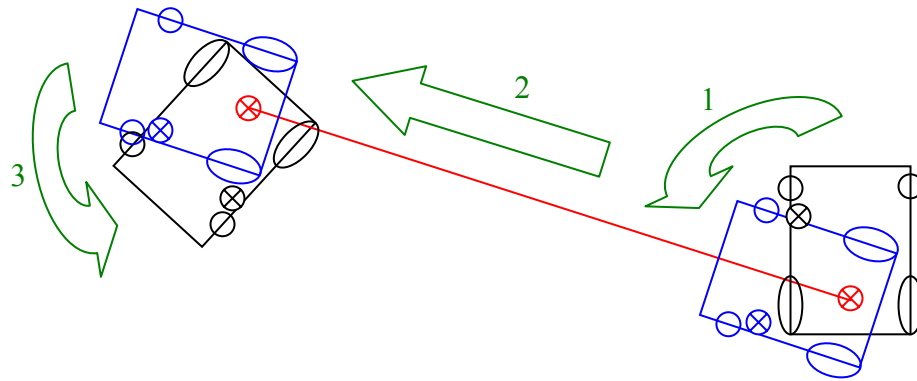


Figure 4.13: The Three Sub-Motions in Motion Planning of the Wheelchair.

4.8. Operator's Safety Issues

In the simulation process of the wheelchair motion, two separate concerns were noticed and can be summarized as follows:

- 1- In the case of directional motion with no rotation, the wheelchair might end up in an odd angle that might be useless to the user if he/she is using this kind of control in the autonomous motion of the actual wheelchair.
- 2- In the case when a trajectory is planned and a motion is divided into three sub-motions, it was noticed that when the wheelchair is in the first or second rotational motion, the front of the wheelchair may run into close-by objects such as a wall or a human in the close proximity of the wheelchair. It would also be dangerous if a drop in the ground elevation is nearby, such as a stair step down. Since this process is done autonomously, sensory information is important to be added for the safety of the operator.

4.9. Summary

In this chapter, the wheelchair's motion was analyzed and the points of interest in the wheelchair relations were pointed out. Coordinate frames were assigned to these points of interest and transformations between the assigned frames were generated. Velocity propagation from one coordinate frame to the other were conducted to find the corresponding points of interests' velocities based on the differentially driven wheel's velocities according to the non-holonomic motion rules. The system's Jacobian was generated to relate the robotic arm base frame Cartesian velocities to the wheels' velocities. After that, trajectory planning was shown to compensate for lack of full coordinate control in the workspace of the wheelchair. Safety issues were addressed in the wheelchair control methods and suggestions were given to address them.

Chapter 5:

Control and Optimization of the Combined Mobility and Manipulation

5.1. Introduction

In the previous two chapters, mathematical models of the 7-DoF robotic arm and the 2-DoF power wheelchair have been derived. The homogeneous transformation relationships between different key points on both of them were developed, and the relative velocities were calculated. The Jacobians of both systems were developed and numerical solution schemes to calculate the inverse kinematics of each of them were derived. It is often helpful to use robotic devices to help people with disabilities to do their activities of daily living without the need for an assistant. The idea of mobile robotic manipulators is something that can help the user, especially if the user is confined to a wheelchair. Having two independent controls for the wheelchair and the arm significantly limits the use of the arm in terms of controllability and the executable tasks.

In this chapter, we will combine the two systems together in an effort to produce a 3-degree of redundancy, 9-DoF system with a single control structure that can be used to control the combined wheelchair-mounted robotic arm (WMRA) system. This gives the WMRA system much more flexibility and it can be controlled autonomously or using teleoperation with the two sub-systems cooperating together in the control scheme.

5.2. Terminology

In the combination of mobility and manipulation, several interpretations can be done to accomplish cooperated motion in the Cartesian space. In this work, the term “combined mobility and manipulation” is used to indicate that the combination is done from the lowest control level and mathematical models up to the advanced control algorithms to integrate both units into a single system. In this context, a single Jacobian of both sub-systems that combines the characteristics of mobility and manipulation will be derived.

5.3. WMRA Assembly and Problem Definition

In a previous work, mounting locations of the robotic arm on a power wheelchair were studied to determine the effect of the mounting location on the manipulability measure [57]. In this work, the selected mounting location provided the best location to execute tasks related to activities of daily living (ADL). Figure 5.1 shows the WMRA system with the frame assignments at each one of the points of interest. The reference frame of the ground was assumed to be stationary, and all other frames of the WMRA system are related to it.

The aim of this chapter is to combine the seven joints of the robotic arm and the two joints (wheels) of the wheelchair into a single vector, and generate the corresponding individual transformations to obtain the general homogeneous transformation that relates the ground frame to the end-effector’s frame. Velocities will also be propagated through both sub-systems to find the total Jacobian that represents the velocity mapping from the nine joint variables to the six Cartesian space variables.

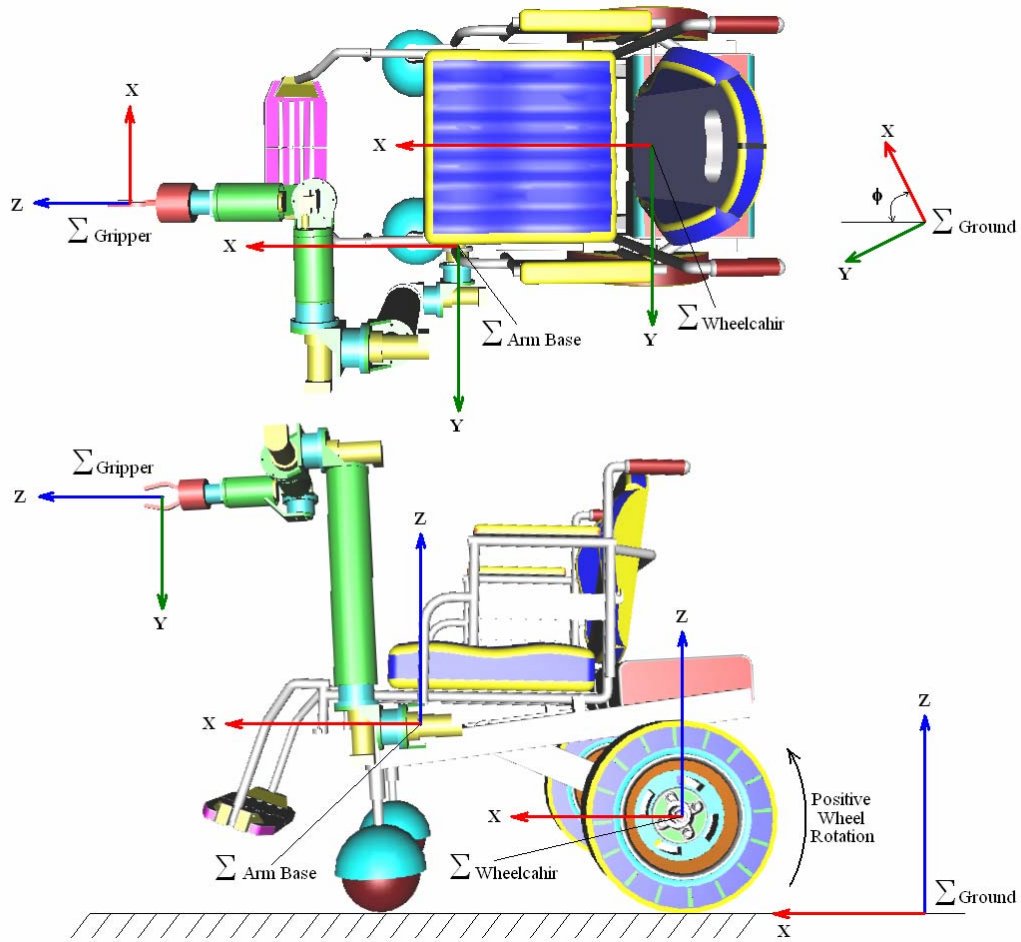


Figure 5.1: WMRA Coordinate Frames.

5.4. Kinematics of the Combined WMRA System

The total homogeneous transformation matrix of the WMRA system will be used later in the implementation of this theory, and defining it appropriately insures accuracy of the results. In our application, we will define it one way now, and then we will redefine it later in this chapter when we have other options and choices of variables to control. For the time being, let's define the joint space of the robotic arm as:

$$q_A = [\theta_1 \ \theta_2 \ \theta_3 \ \theta_4 \ \theta_5 \ \theta_6 \ \theta_7]^T \quad (5.1)$$

Let's also define the joint space of the wheelchair (wheels' rotation angles) as:

$$q_c = [\theta_l \quad \theta_r]^T \quad (5.2)$$

The combination of the two joint spaces can also be defined as:

$$q = \begin{bmatrix} q_A \\ q_c \end{bmatrix} = [\theta_1 \quad \theta_2 \quad \theta_3 \quad \theta_4 \quad \theta_5 \quad \theta_6 \quad \theta_7 \quad \theta_l \quad \theta_r]^T \quad (5.3)$$

Where θ_1 through θ_7 are the robotic joint angles from the arm base to the wrist respectively, θ_l and θ_r are the rotation angles of the left and right wheels respectively. Since we defined the transformation between the robotic arm base frame and the end-effector's frame in equation (3.3), and the transformation between the ground frame and the wheelchair's frame in equation (4.10), as well as the constant transformation between the wheelchair's frame and the arm base frame in (4.12), we can use these equations to find the total transformation matrix as follows:

$${}^G_7T = {}^G_wT \cdot {}^w_0T \cdot {}^0_1T \cdot {}^1_2T \cdot {}^2_3T \cdot {}^3_4T \cdot {}^4_5T \cdot {}^5_6T \cdot {}^6_7T \quad (5.4)$$

Matrix G_7T represents the 4X4 homogeneous transformation between the ground and the end-effector's frame in terms of the WMRA joint space. At any moment of time, for a given the joint space vector in (5.3), substituting joint values in (5.4) gives a clear description of the position and orientation of the end-effector. This calculation is important when we compare the target position in the workspace to the current location of the end-effector.

5.5. Jacobian Augmentation and Resolved Rate Equations Generation

The velocity relation developed in chapter 4 leads to the Jacobian of the non-holonomic wheelchair that relates the wheels' velocities to the three Cartesian velocities

in the planar motion of the wheelchair as described in equation (4.20). It is important to modify this equation to include all six Cartesian velocities in space so that we can combine the motion of the arm with the motion of the wheelchair. To find the new Jacobian of the wheelchair, let's define the task space vector of the wheelchair as:

$$r_C^A = f(q_C) = [X_C \ Y_C \ Z_C \ \alpha_C \ \beta_C \ \gamma_C]^T \quad (5.5)$$

This vector represents the task space vector at the robotic arm base frame (A), and can be found by modifying (4.20) to include all Cartesian velocities as follows:

$$\dot{r}_C^A = J_C \cdot J_W \cdot \dot{q}_C \quad (5.6)$$

Where “ J_W ” is the wheelchair's planar Jacobian:

$$J_W = \frac{L_5}{2} \cdot \begin{bmatrix} C\phi + \frac{2}{L_1} \cdot (L_2 \cdot S\phi + L_3 \cdot C\phi) & C\phi - \frac{2}{L_1} \cdot (L_2 \cdot S\phi + L_3 \cdot C\phi) \\ S\phi - \frac{2}{L_1} \cdot (L_2 \cdot C\phi - L_3 \cdot S\phi) & S\phi + \frac{2}{L_1} \cdot (L_2 \cdot C\phi - L_3 \cdot S\phi) \\ -\frac{2}{L_1} & \frac{2}{L_1} \end{bmatrix} \quad (5.7)$$

and “ J_C ” is defined to map the Jacobian from the three Cartesian coordinates to six Cartesian coordinates as follows:

$$J_C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T \quad (5.8)$$

Equation (5.6) relates the wheels' velocity vector to the Cartesian task space at the robotic arm base. Since the task will not be performed by the arm base, we need to map the motion to the end-effector's frame. Let's define the task space vector at the end-effector's frame (E) as:

$$r = f(q_A, q_C) = [X \ Y \ Z \ \alpha \ \beta \ \gamma]^T \quad (5.9)$$

Differentiating (5.9) with respect to time gives:

$$\dot{r} = \frac{\partial f}{\partial q_A} \cdot \dot{q}_A + \frac{\partial f}{\partial q_C} \cdot \dot{q}_C \quad (5.10)$$

Note that the Jacobian in equation (5.6) relates the wheels' velocity vector to the Cartesian task space at the robotic arm base frame, and what we need is the equivalent relationship defined at the end-effector's frame. This will be done to the wheelchair's motion by introducing a new Jacobian that relates the wheelchair's motion at the arm base frame to the end-effector's frame motion. This Jacobian will depend only on the first two and last Cartesian coordinates of the end-effector based on the arm base frame as follows:

$$\dot{r}_C^E = \begin{bmatrix} I_2 & [0] & -(P_{xg} \cdot S\phi + P_{yg} \cdot C\phi) \\ [0] & & P_{xg} \cdot C\phi - P_{yg} \cdot S\phi \\ & & I_4 \end{bmatrix} \cdot \dot{r}_C^A = J_G \cdot \dot{r}_C^A \quad (5.11)$$

Where P_{xg} and P_{yg} are the x-y coordinates of the end-effector based on the arm base frame, and Φ is the angle of the arm base frame, which is the same as the angle of the wheelchair based on the ground frame. Substituting (5.6) into (5.11) gives:

$$\dot{r}_C^E = J_G \cdot J_C \cdot J_W \cdot \dot{q}_C \quad (5.12)$$

Now the Jacobians are augmented separately and ready for combination. Substituting the rates of change by the Jacobians found earlier for the arm and the Jacobians for the wheelchair into equation (5.10) gives:

$$\dot{r} = J_A \cdot \dot{q}_A + J_G \cdot J_C \cdot J_W \cdot \dot{q}_C \quad (5.13)$$

Putting (5.13) in a matrix form results in the following:

$$\dot{r} = [J_A \quad J_G \cdot J_C \cdot J_W] \begin{bmatrix} \dot{q}_A \\ \dot{q}_C \end{bmatrix}, \text{ or } \dot{r} = J \cdot \dot{q} \quad (5.14)$$

Equation (5.14) combines the two Jacobians together to combine the mobility of the wheelchair and the manipulation of the arm. It is important to mention here that care must be taken in implementing this method in simulation or in the actual WMRA system. The combined Jacobian is related to joint angle rates, while the wheelchair is effectively performing linear motion that results from the two driving wheels. Combination of the Jacobian in this context requires three processes to be done before and after evaluating equation (5.14), these processes must be followed, or the solution will not converge, and the system will be out of control. The three processes should be done in the following sequence:

- 1- Before using the Jacobian, convert the linear velocities at the left and right wheels into angular velocities using the equations:

$$\dot{\theta}_l = \frac{\dot{D}_l}{L_5}, \text{ and } \dot{\theta}_r = \frac{\dot{D}_r}{L_5}. \quad (5.15)$$

- 2- Use the Jacobian with the new angles of the left and right wheels.
- 3- After using the Jacobian, convert the angular velocities of the left and right wheels into linear velocities using the equations:

$$\dot{D}_l = \dot{\theta}_l \cdot L_5, \text{ and } \dot{D}_r = \dot{\theta}_r \cdot L_5. \quad (5.16)$$

The reason we are converting the angular into linear velocities is to avoid oscillation of the system when evaluating the sines and cosines of the wheels' angles when they complete a full rotation. Having the angular velocity values converted into linear velocities eliminates the oscillation in the system when the mobility and manipulation are combined, and produce smooth and compatible motion with the task

trajectory. Running these equations in simulation showed satisfactory results, this will be discussed later.

5.6. Jacobian Changes Based on the Control Frame

It is always noticed in mobile robotic applications that the user may need to control the end-effector based on its own frame rather than the ground frame or the wheelchair frame. In this application, we considered all three possibilities to be included for user convenience. These possibilities require a slight modification to the Jacobian of both the wheelchair and the robotic arm. Note that so far, we defined the robotic arm's Jacobian based on its base frame, and the wheelchair's Jacobian based on the ground frame. These definitions will be changed based on the controlled frame.

5.6.1. Ground-Based Control

This is best used in autonomous control mode, where the trajectory to the target is always defined based on the ground frame. The Jacobian of the wheelchair in this case stays the same, and the Jacobian of the robotic arm becomes:

$$[J_A]_{new} = \begin{bmatrix} {}^G_0R & 0 \\ 0 & {}^G_0R \end{bmatrix} \cdot [J_A]_{original} \quad (5.17)$$

5.6.2. Wheelchair-Based Control

This is best used in teleoperation control mode, when the user is controlling the wheelchair most of the time. The Jacobian of the robotic arm in this case stays the same, and the Jacobian of the wheelchair becomes:

$$[J_G \cdot J_C \cdot J_W]_{new} = \begin{bmatrix} {}^G R^T & 0 \\ 0 & {}^G R^T \end{bmatrix} \cdot [J_G \cdot J_C \cdot J_W]_{original} \quad (5.18)$$

5.6.3. End-Effector Based Control (Piloting Option)

This is best used in teleoperation control mode, when the user is controlling the end-effector most of the time to perform ADL tasks. This mode is called the pilot mode since the end-effector is compared to a flying object with its own frame-based control.

The Jacobian of the robotic arm in this case will be changed as follows:

$$[J_A]_{new} = \begin{bmatrix} {}^0 R^T & 0 \\ 0 & {}^0 R^T \end{bmatrix} \cdot [J_A]_{original} \quad (5.19)$$

And the Jacobian of the wheelchair will be changed as follows:

$$[J_G \cdot J_C \cdot J_W]_{new} = \begin{bmatrix} {}^G R^T & 0 \\ 0 & {}^G R^T \end{bmatrix} \cdot [J_G \cdot J_C \cdot J_W]_{original} \quad (5.20)$$

5.7. Jacobian Inversion Methods and Singularities

At this point, we produced a Jacobian that combines both the mobility of the power wheelchair and the manipulation of the 7-DoF arm. To make effective use of the arm and execute Cartesian space tasks, an inversion of the Jacobian is necessary. Often times singular configurations that produce high joint rates and lead to instability occur while trying to execute a given task. Inverting the Jacobian while trying to avoid singularities would give reasonably effective results. Two methods of Jacobian inversions were done for the combined system, one of them uses Pseudo inverse, and the other uses Singularity-Robust inverse.

5.7.1. Inverting Using Pseudo Inverse

Numerical solutions were implemented using the Jacobian to follow the user's directional motion commands or to follow the desired trajectory. Redundancy can be resolved using Pseudo inverse of the Jacobian [54], and singularity is avoided by maximizing the manipulability measure [53] discussed earlier in chapter 3. When the Jacobian matrix is of rank 6, which is the number of rows in the 6X9 combined Jacobian matrix that we have, Pseudo inverse can be written as:

$$J^* = J^T \cdot (J \cdot J^T)^{-1} \quad (5.21)$$

When this inverse was implemented in simulation, it showed good results when the manipulability measure was far from zero. Since this method carries a guaranteed valid solution only at a singular configuration and not around it, the results can carry high joint velocities when singularity is approached.

5.7.2. Inverting Using Singularity-Robust Inverse

Another inversion method was tried with the new combined WMRA system since singularity needs to be addressed. A method that starts to change the arm configuration as it approaches singularities was tested with the new system, that method is called the Singularity-Robust (S-R) inverse of the Jacobian [50]. Using this method allowed the use of redundancy resolution schemes for different subtasks, while singularities are taken care of at the Jacobian inversion level. The S-R inverse of the Jacobian used to carry out the inverse kinematics can be written as:

$$J^* = J^T \cdot (J \cdot J^T + k \cdot I_6)^{-1} \quad (5.22)$$

where I_6 is a 6x6 identity matrix, and k is a scale factor. It has been known that this method reduces the joint velocities near singularities, but compromises the accuracy of the solution by increasing the joint velocities error. Choosing the scale factor k is critical, if it is too high, the error will be too high and the system might destabilize, and if it is too small, the joint rates will go too high, and the system might destabilize. Since the point in using this factor is to give approximate solution near and at singularities, an adaptive scale factor is updated at every time step to put the proper factor as needed. This factor can be defined as:

$$k = \begin{cases} k_0 * (1 - \frac{w}{w_0})^2 & \text{for } w < w_0 \\ 0 & \text{for } w \geq w_0 \end{cases} \quad (5.23)$$

Where w_0 is the manipulability measure at the start of the boundary chosen when singularity is approached, and k_0 is the scale factor at singularity. The optimum values of w_0 and k_0 for our system were found by simulation to be 0.034 and 13×10^{-9} respectively.

5.8. Optimization Methods with the Combined Jacobian

One of the most beneficial advantages of redundancy in robotic manipulators is the fact that its motion can be optimized in many different ways. A human arm, for instance, is a redundant system because it has 7 degrees of freedom (3 in the shoulder, 1 in the elbow and 3 in the wrist) and there are only 6 physical degrees of freedom in the task of placing the hand in any position and orientation in space (x, y, z, roll, pitch and yaw). If a human arm had only 6 joints, the arm will be stationary if the wrist is fixed, but since the human arm carries 7th joint, we were able to still move our arm while fixing the wrist at a fixed point.

Now that the singularity is taken care of using the S-R Inverse of the Jacobian, we can use the joint redundancy to optimize for a secondary task or to set motion preference weights on the joint domain while following the Cartesian trajectory. Three methods of optimization will be discussed and tried for this system.

5.8.1. Criteria Functions and Minimizing Euclidean Norm of Errors

Redundancy can be resolved using any of the two inverses discussed above to obtain a numerical solution of the joint angle rates. If we have the desired task space variables, and we need to obtain the desired joint space variables, we can use the simplest form of the resolved rate methods using the following equation:

$$\begin{bmatrix} \dot{q}_A \\ \dot{q}_C \end{bmatrix}_d = J^* \cdot \dot{r}_d \quad (5.24)$$

This solution minimizes the norm of the joint velocities and the Euclidean norm of end-effector velocity errors, which is the difference between the commanded Cartesian space variables and the actual Cartesian space variables achieved. Adding more tasks to the optimization process can be done by adding additional terms to include another optimization function as follows:

$$\begin{bmatrix} \dot{q}_A \\ \dot{q}_C \end{bmatrix}_d = J^* \cdot \dot{r}_d + (I_9 - J^* \cdot J) \cdot \dot{q}_0 \quad (5.25)$$

Where “ \dot{q}_0 ” is a 9x1 vector representing the secondary task, “ J^* ” is the modified 6x9 Jacobian matrix, and I_9 is the identity matrix of size 9. The choice of the criterion function can range from a scalar quantity, such as the manipulability measure “w”, or can be a set of functions, such as joint limit avoidance conditions. The secondary task can

either be the desired trajectory in the case of pre-set task execution, or it can be a criterion function such that:

$$\dot{q}_0 = a \nabla_q H(q) \quad (5.26)$$

Where $H(q)$ is the optimization criterion $y = H(q)$. The existence of the mobile platform means that “ \dot{q}_0 ” may not exist for non holonomic constraint such as that of the wheelchair. To go around this limitation [47] proposed the following: Differentiate the optimization criterion function “H” with respect to time as follows:

$$\dot{y} = \dot{H}(q) = \frac{\partial H}{\partial q_A} \cdot \dot{q}_A + \frac{\partial H}{\partial q_C} \cdot \dot{q}_C, \text{ or,} \quad (5.27)$$

$$\dot{y} = \nabla_q^T H \begin{bmatrix} I & 0 \\ 0 & J_W \end{bmatrix} \begin{bmatrix} \dot{q}_A \\ \dot{q}_C \end{bmatrix} \quad (5.28)$$

In this case, the value of “ \dot{q}_H ” that improves the objective function can be defined as:

$$\dot{q}_H = \pm a \begin{bmatrix} I & 0 \\ 0 & J_W^T \end{bmatrix} \nabla_q H(q) \equiv \dot{q}_0 \quad (5.29)$$

And that velocity vector can be used for optimization. This gives a good representation of the arm joints’ velocities and the wheels’ velocities of the wheelchair. The null space may contain more variables than what is required for the secondary task. In this case, another secondary task can be used to optimize for more than one criterion. The sign in (5.29) was taken as positive in case the optimization function needs to be maximized, and negative in case it needs to be minimized depending on the function and its requirement in the control algorithm.

5.8.2. Weighted Least Norm Solution

Weighted Least Norm solution can also be used as proposed by [27] to resolve the system redundancy. In order to put a motion preference of one joint rather than the other (such as the wheelchair wheels and the arm joints), a weighted norm of the joint velocity vector can be defined as:

$$|\dot{q}|_W = \sqrt{\dot{q}^T \cdot W \cdot \dot{q}} \quad (5.30)$$

Where “W” is a 9X9 symmetric and positive definite weighting matrix. For simplicity, the weight matrix can be a diagonal matrix that represents the motion preference of each joint of the system. For the purpose of analysis, the following transformations are introduced:

$$J_W = J \cdot W^{-1/2} \quad \text{and} \quad \dot{q}_W = W^{-1/2} \cdot \dot{q} \quad (5.31)$$

Using (5.31), we can rewrite (5.14) and (5.30) respectively as:

$$\dot{r} = J_W \cdot \dot{q}_W, \quad \text{and} \quad (5.32)$$

$$|\dot{q}|_W = \sqrt{\dot{q}_W^T \cdot \dot{q}_W} \quad (5.33)$$

In this case, the Least Norm solution of (5.32) is:

$$\dot{q}_W^* = J_W^* \cdot \dot{r} \quad (5.34)$$

Using the second part of (5.31), joint velocities can be redefined as:

$$\dot{q}_W = W^{-1/2} \cdot \dot{q}_W^* \quad (5.35)$$

Using Pseudo inverse, it can be shown that the Weighted Least Norm solution is calculated as follows:

$$\dot{q}_W = W^{-1} \cdot J^T \cdot (J \cdot W^{-1} \cdot J^T)^{-1} \cdot \dot{r} \quad (5.36)$$

The above method has been used in simulation of the 9-DoF WMRA system with the nine state variables “ \dot{q}_d ” that represent the seven joint velocities of the arm and the two wheels’ velocities of the power wheelchair. It was found that latter two state variables are of limited use since they tend to unnecessarily rotate the wheelchair back and forth during a long forward motion due to their equal weights. Changing the weights of these two variables will only result in a preference of one’s motion over the other. More on this will be discussed later in the chapter.

5.8.3. Joint Limit Avoidance

The criterion function used for optimization can be defined based on the physical joint limits of the WMRA system, and minimizing that function results in limiting the joint motion to its limit. A mathematical representation of joint limits in robotic manipulators has been a topic of many researchers. One of these representations were proposed [27] as follows:

$$H(q) = \sum_{i=1}^7 \frac{1}{4} \cdot \frac{(q_{i,\max} - q_{i,\min})^2}{(q_{i,\max} - q_{i,\text{current}}) \cdot (q_{i,\text{current}} - q_{i,\min})} \quad (5.37)$$

This criterion function becomes “1” when the current joint angle is in the middle of its range, and it becomes “infinity” when the joint reaches either of its limits. When used in (5.29) the function can be minimized by choosing the negative sign for it so that the null space is used to choose the minimum value of the function that satisfies the main objective function.

Using this optimization function in (5.36) can be accomplished through the weight matrix used for optimization. Rather than choosing arbitrary weight values for

each individual joint based on the user preference, an additional value can be added to represent the optimization criterion function as follows:

$$W = \begin{bmatrix} w_{1,u} + \left| \frac{\partial H(q)}{\partial q_1} \right| & 0 & \dots & 0 \\ 0 & w_{2,u} + \left| \frac{\partial H(q)}{\partial q_2} \right| & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & w_{9,u} + \left| \frac{\partial H(q)}{\partial q_9} \right| \end{bmatrix} \quad (5.38)$$

Where “ $w_{i,u}$ “ is the user-defined weight preference to joint “ i ”, and the second term in each element is the gradient projection of the criterion function defined as:

$$\frac{\partial H(q)}{\partial q_i} = \frac{(q_{i,\max} - q_{i,\min})^2 \cdot (2 \cdot q_{i,\text{current}} - q_{i,\max} - q_{i,\min})}{4 \cdot (q_{i,\max} - q_{i,\text{current}})^2 \cdot (q_{i,\text{current}} - q_{i,\min})^2} \quad (5.39)$$

When any particular joint is in the middle of the joint range, (5.39) becomes zero, for that joint, and the only weight left is the user defined weight. On the other hand, when any particular joint is at its limit, (5.39) becomes “infinity”, which means that the joint will carry an infinite weight that makes it impossible to move any further. When the user prefers to move robotic arm with minimal wheelchair motion, heavy weight can be assigned by the user to the two wheelchair state variables. When any of the robotic arm joints gets close to its limit and its weight approaches infinity, the wheelchair’s weight will be much less than that of the joint, and hence it will be more free to move than the joint that is close to its limit.

It is important to note two different deficiencies that may lead to unintended operation or joint lock when using this method. The first deficiency is that the joint is penalized with higher weight whether it is approaching its limit or getting away from it.

This may cause the robotic arm to use the null space inefficiently by preferring to move a joint with heavy weight going towards its limit rather than moving a joint with heavier weight that is moving away from its limit. This problem can be eliminated if two conditions were imposed on the criterion function as shown in figure 5.2. These conditions are as follows:

- 1- When the joint limit is being approached from outside the limit and moving towards the limit (i.e. the weight difference between two consecutive steps is positive and the current joint limit is not exceeded) then give the weight as calculated by (5.38).
- 2- When the joint limit is being approached from outside the limit and moving away from the limit (i.e. the weight difference between two consecutive steps is negative and the current joint limit is not exceeded) then give the weight as “ $w_{i,u}$ ”.

The second deficiency is that the precise joint limit that takes the weight to infinity may never be reached, instead, the numerical solution with its relatively coarse step size may jump from a joint value close to the joint limit before it is reached to a joint value close to the joint limit after it is reached. This will result in a heavy weight that will slowly get lower as the joint gets away from the set limit towards its actual limit. If the previous two conditions were applied alone, a dangerous motion can happen by giving the weight as the user chosen weight only since the joint is getting away from its limit from inside that limit. This can either break the joint or lock it when it reaches its actual physical limit with the hard stop. To overcome this deficiency, two more conditions need to be imposed on the system:

- 3- When the joint limit is being approached from inside the limit and moving away from the limit (i.e. the weight difference between two consecutive steps is negative and the current joint limit is exceeded) then give the weight as infinity since no further motion inside should be allowed.
- 4- When the joint limit is being approached from inside the limit and moving towards the limit (i.e. the weight difference between two consecutive steps is positive and the current joint limit is exceeded) then give the weight as " $w_{i,u}$ " since the joint is actually getting away from its limit.

Imposing the above four conditions on the weight matrix to perform on the optimization criterion gave the control mechanism much better results in terms of joint limit avoidance and user-preferred motion of each individual variable in the joint space.

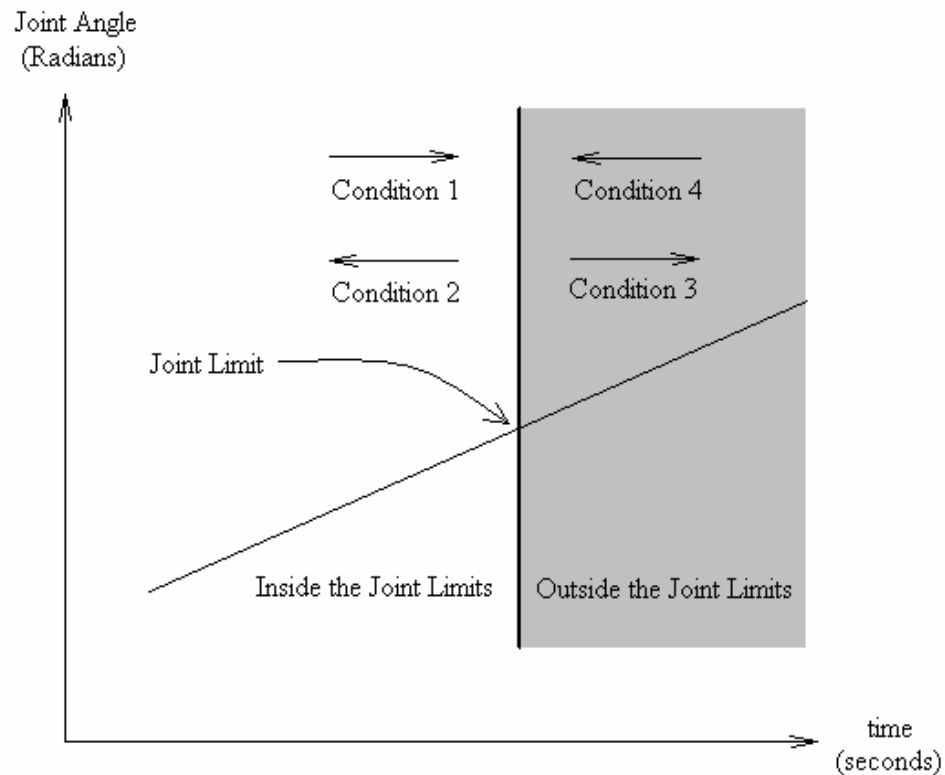


Figure 5.2: Four Joint Limit Boundary Conditions.

5.8.4. Obstacle Avoidance

The same criterion function can also be used to optimize the control algorithm based on obstacle avoidance. An important mathematical representation of the obstacles around the WMRA system is necessary to formulate these criteria functions. For the immediate objects around the WMRA system, there are three complex shapes that need to be avoided. These shapes are the wheelchair, the human user and the robotic arm itself. Modelling these shapes mathematically can be challenging since they are not fixed shapes. In this case, sensory suite can be used to recognize the obstacle and avoid it as the WMRA moves in its workspace.

5.8.5. Safety Conditions

In order to create a comprehensive representation of the physical environment within the WMRA and in its immediate surroundings, several safety conditions should be imposed to avoid joint limits both in position and velocities, and to avoid the arm from hitting the human user or the wheelchair or itself. These conditions were put in place as follows:

- 1- Stop the joint if it reaches its maximum or minimum limit:

$$\text{if } q_i \geq q_{i,\max} \quad \text{OR} \quad q_i \leq q_{i,\min} \quad \text{then } \dot{q}_{i,\text{commanded}} = 0 \quad (5.40)$$

- 2- Slow down the joint if it reaches its velocity limit (which is also useful in case the WMRA reached singularity and went out of control):

$$\text{if } |\dot{q}_i| \geq |\dot{q}_{i,\max}| \quad \text{then } \dot{q}_{i,\text{commanded}} = \text{sign}(\dot{q}_{i,\text{commanded}}) \cdot \dot{q}_{i,\max} \quad (5.41)$$

- 3- Slowly reverse all joint velocities in case any robotic arm joint frame approaches collision with the ground, the wheelchair's side, the wheelchair's wheels, the wheelchair's human driver's shoulder, his/her lap, his/her legs, and the wheelchair's battery pack.
- 4- Slowly reverse all joint velocities in case the robotic arm's upper arm approaches collision with its forearm.

Considering the above four conditions in the control algorithm ensures the safety of the human operator as well as the WMRA system from physical damage. Condition number "3" above has been expanded into "16" sub-conditions that address the physical relations between the reachable space of each joint frame and the physical presence of obstacles in that particular reachable space.

5.8.6. Unintended Motion Effect Based on the Optimization Criteria

It is important to the user when operating the WMRA system in teleoperation mode to have total control with predictable motion. When equation (5.25) is used with " $q_0=0$ ", the optimization will be based on the minimization of the Euclidean norm of errors. Observing that equation shows that the commanded joint velocities will be "zeros" if the user does not command the Cartesian variables of the end-effector to move. Depending on the chosen criterion function of " q_0 " and its dependency on different variables in both the Cartesian space and the joint space, it is possible for the WMRA to move even when the user does not command it to move.

In the case of joint limit avoidance, the criterion function in (5.37) depends only on the joint variables. That makes the second term in (5.25) non-zero even when the user

commands the WMRA system not to move by setting the first term to “zero” (or by not touching the controller interface to send $\dot{r}=0$). The logical interpretation of that equation means that the user will see the WMRA in motion as soon as the system is powered up if any of the joints is not in the middle of its operational range. This motion will drive the arm to its joint mid-range angles and then stop. The reaction caused by such optimization can be dangerous on the user and the surrounding subjects if it is used the way it is.

On the other hand, looking at equation (5.36) when using the Weighted Least Norm solution, when the commanded Cartesian velocity vector $\dot{r}=0$, the commanded joint velocity vector will be “zero” no matter what the weight function is. This ensures that the optimization will not take effect unless the user started to command the arm to move in its workspace. Even though both solutions were integrated in the control algorithm of the WMRA, care must be taken when using the first solution since it can result in unpredictable motion.

5.9. Optional Combinations for the Resolved Rate Solution

Each of the optimization schemes mentioned earlier can be used for different purposes, and other schemes might be added to resolve redundancy. To make the control algorithm flexible in terms of optimization, all methods were implemented in the high-level control so that the user can choose his/her preference. Two things are needed for redundancy resolution, the first is finding the inverse of the Jacobian, and the second is implementing the optimization function to find the joint rates. The following eight combinations were programmed for redundancy resolution options:

- 1- Pseudo inverse solution (PI): This selection evaluates the inverse of the Jacobian using Pseudo inverse to find joint rates for the next time step when given the current Cartesian coordinates of the end-effector.
- 2- S-R inverse solution (SRI): This selection evaluates the inverse of the Jacobian using the Singularity-Robust inverse to find joint rates for the next time step when given the current Cartesian coordinates of the end-effector.
- 3- Weighted Pseudo inverse solution (WPI): This selection evaluates the inverse of the Jacobian using Pseudo inverse, and then applies the weighted least norm solution to find joint rates for the next time step when given the current Cartesian coordinates of the end-effector.
- 4- Weighted S-R inverse solution (WSRI): This selection evaluates the inverse of the Jacobian using the Singularity-Robust inverse, and then applies the weighted least norm solution to find joint rates for the next time step when given the current Cartesian coordinates of the end-effector.
- 5- Pseudo inverse with gradient projection solution (PI-GP): This selection evaluates the inverse of the Jacobian using Pseudo inverse and adds the projection of the null space with the optimization criterion function to find joint rates for the next time step when given the current Cartesian coordinates of the end-effector.
- 6- S-R inverse with gradient projection solution (SRI-GP): This selection evaluates the inverse of the Jacobian using the Singularity-Robust inverse and adds the projection of the null space with the optimization criterion function to

find joint rates for the next time step when given the current Cartesian coordinates of the end-effector.

7- Weighted Pseudo inverse with criterion function optimization solution (WPI-CF): This selection evaluates the inverse of the Jacobian using Pseudo inverse, and then applies the weighted least norm solution with the criterion optimization function included in the weight matrix to find joint rates for the next time step when given the current Cartesian coordinates of the end-effector.

8- Weighted S-R inverse with criterion function optimization solution (WSR-CF): This selection evaluates the inverse of the Jacobian using the Singularity-Robust inverse, and then applies the weighted least norm solution with the criterion optimization function included in the weight matrix to find joint rates for the next time step when given the current Cartesian coordinates of the end-effector.

In the event that other optimization criteria or Jacobian inversion methods were added to the control algorithm, more choices can be added to the program, and the user will be given the prompt to choose which optimization selection to select.

5.10. State Variable Options in the Control Algorithm

In any control problem, state variables are chosen such that changing any of them during the control process gives a result that makes sense for the general objective of the control algorithm. In our application, the state variables selected are the seven joint limits of the robotic arm and the two wheelchair wheels' angles. This is not necessarily the best

way to control the WMRA system. Other state variables may make more sense in the overall control objectives, and may be used in a better way in terms of optimization.

5.10.1. Seven Robotic Arm Joints, Left wheel and Right Wheel Variables

This is the default state variables selection that applies the resolved rate scheme to find “ \dot{q} ” that contains the nine joint variables selected. These joint variables are the seven joints of the robotic arm, the left driving wheel angle, and the right driving wheel angle of the wheelchair. As far as the robotic arm is concerned, controlling its seven joints is the only available selection, but when it comes to the wheelchair, there is another way to control its wheels’ angles. In this sub-section, the selected variables for the wheelchair motion gave undesired motion that was not necessary to execute the trajectory-following command.

When used with the weighted least norm solution optimization scheme, a weight matrix that contains preference weights to each of the nine state variables was created such that the wheelchair wheels’ motion carry higher weights than the robotic arm’s joints. This is done so that if the task is within the workspace of the robotic arm and can be executed without the need to move the wheelchair, the control process would result in minimal wheelchair motion and maximum arm motion for task execution. In the event that the user is working in an office environment, he/she will not be inconvenienced by constantly moving his wheelchair unnecessarily.

The problem found with the selection of these state variables is that when motion of the wheelchair is necessary for task execution, and both wheels are given equal weights, the wheelchair turns unnecessarily. This happens because the robotic arm is

mounted on the side of the wheelchair, and when it follows a long trajectory and the arm reaches its workspace boundaries, the left wheel starts to move since it is the closest one to the arm base that will cause the arm to continue following the trajectory. This results in a wheelchair rotation since the right wheel is not necessary to move at that point. When moving the left wheel is no longer sufficient to give the robotic arm the necessary motion to follow the trajectory, the right wheel starts moving and causes the wheelchair to turn again unnecessarily. Even though the end-effector precisely followed its trajectory and the wheelchair moved as it was necessary, its motion was unpleasant.

In a test of moving the end-effector in a straight line trajectory extending 10 meters in the forward direction of the wheelchair, the motion that makes sense is that the arm should extend forward, and the two wheels of the wheelchair move at the same speed as necessary, but because of the wheel proximities to the robotic arm base, they were giving different speeds to the wheels over the simulation period of time until the destination point was reached. Giving the two wheels different weights based on their proximity of the arm base doesn't solve the problem since the trajectory can be different from the straight line, and in the case of left or right hand motion of the end-effector, the wheelchair will not behave in the best way possible.

5.10.2. Seven Robotic Arm Joints, Forward and Rotational Motion of the Wheelchair

In avoiding this kind of behavior in the wheelchair motion response, the two wheelchair's state variables should be changed. Since the user does not care about the wheels motion or which wheel moves faster than the other, two different state variables must be chosen, and then related to the wheels' motion. The choice that makes sense in

this context is to choose polar coordinate control of the wheelchair, and that can be done by controlling the linear distance and the angle of the wheelchair. If having the linear motion and the rotational motion of the wheelchair as the two state variables rather than the two wheels' motions, it can be used with the weights to give motion that is more convenient to the user. A linear motion of the wheelchair corresponds to equal velocities of both wheels in the same direction, while a rotational motion corresponds to equal velocities of both wheels in opposite directions. A combination of linear and rotational velocities corresponds to algebraically adding the resultant motion of both in each of the wheels. To make this option available, let's re-define equation (5.2) by the two new state variables as follows:

$$q_c = \begin{bmatrix} \text{Translation}(X) \\ \text{Rotation}(\phi) \end{bmatrix} \quad (5.42)$$

For pure rotation of the wheelchair, both wheels will be running to the same angle with opposite directions, the resulting wheelchair's inclination angle is:

$$\phi = \frac{L_s}{L_1} \cdot (\theta_r - \theta_l) \quad (5.43)$$

Since “ θ_l ” and “ θ_r ” are equal in value, but opposite in direction, let's take the right wheel angle as the positive angle, that is:

$$\theta_r = -\theta_l = \theta \quad (5.44)$$

This simplifies (5.43) to:

$$\phi = \frac{L_s}{L_1} \cdot (2 \cdot \theta) = \frac{2 \cdot L_s \cdot \theta}{L_1} \quad (5.45)$$

Or,

$$\theta = \frac{L_1 \cdot \phi}{2 \cdot L_5} \quad (5.46)$$

Using (5.44) and (5.46) gives the relations:

$$\theta_l = \frac{-L_1 \cdot \phi}{2 \cdot L_5}, \text{ and } \theta_r = \frac{L_1 \cdot \phi}{2 \cdot L_5} \quad (5.47)$$

For pure translation of the wheelchair, both wheels will be running to the same angle “ θ ” with the same direction, the resulting wheelchair’s linear distance is:

$$X = L_5 \cdot \theta, \text{ or } \theta_l = \theta_r = \frac{X}{L_5} \quad (5.48)$$

For the general case when we have both rotation and translation, we can add the wheels’ angles in (5.47) and (5.48) together. Now we need to relate the old state variable to the new state variables in the wheelchair as follows: let the new wheelchair’s state

variables be defined as $q_C = \begin{bmatrix} X \\ \phi \end{bmatrix}$, then the two wheels’ rotational angles will be:

$$\theta_l = \frac{-L_1}{2 \cdot L_5} \cdot \phi + \frac{X}{L_5}, \text{ and } \theta_r = \frac{L_1}{2 \cdot L_5} \cdot \phi + \frac{X}{L_5} \quad (5.49)$$

Differentiating (5.49) with respect to time gives the rates of the angular motion of the wheels as:

$$\dot{\theta}_l = \frac{-L_1}{2 \cdot L_5} \cdot \dot{\phi} + \frac{\dot{X}}{L_5}, \text{ and } \dot{\theta}_r = \frac{L_1}{2 \cdot L_5} \cdot \dot{\phi} + \frac{\dot{X}}{L_5} \quad (5.50)$$

Remembering that $\frac{\dot{X}}{L_5} = \dot{\theta}$ for linear motion, using the angle instead of the

distance makes the Jacobian compatible in units. Putting (5.50) in a matrix form gives a better prospective as follows:

$$\begin{bmatrix} \dot{\theta}_l \\ \dot{\theta}_r \end{bmatrix} = \begin{bmatrix} \frac{1}{L_5} & \frac{-L_1}{2 \cdot L_5} \\ \frac{1}{L_5} & \frac{L_1}{2 \cdot L_5} \end{bmatrix} \cdot \begin{bmatrix} \dot{X} \\ \dot{\phi} \end{bmatrix} \quad (5.51)$$

This gives the Jacobian that relates the two wheels to the angle of inclination and the travelled linear distance of the wheelchair. The Jacobian used in (5.7) then can be modified as follows:

$$[J_W]_{New} = [J_W]_{Old} \cdot \begin{bmatrix} \frac{1}{L_5} & \frac{-L_1}{2 \cdot L_5} \\ \frac{1}{L_5} & \frac{L_1}{2 \cdot L_5} \end{bmatrix} \quad (5.52)$$

The rest of the derivation can be carried on in the same way as done to combine the manipulability of the wheelchair and the manipulation of the arm done earlier this chapter. The new state variables will become:

$$q = \begin{bmatrix} q_A \\ q_C \end{bmatrix} = [\theta_1 \quad \theta_2 \quad \theta_3 \quad \theta_4 \quad \theta_5 \quad \theta_6 \quad \theta_7 \quad X \quad \phi]^T \quad (5.53)$$

Care must be taken again here when using the angle “ θ ” in the Jacobian instead of the linear distance “ X ”, a conversion from distance to angle must be done before processing the Jacobian, and then another conversion back to distance after processing the Jacobian.

The combination of the above two variables would be sufficient to describe any forward and rotational motion of the wheelchair. Having these two state variables in vector “ q ” instead of the wheels’ velocities gives a greater advantage in controlling the preferred rotation or translation of the wheelchair. The wheelchair’s Jacobian in (5.6) is changed for the new state variables before augmenting it to the arm’s Jacobian, and the

results are much better in terms of desired control. When these new state variables were used in simulation with the weight matrix, we were able to avoid user inconvenience problem that happened in the previous method by assigning heavier weight to the rotation of the wheelchair than the weight of the translation so that when a trajectory is followed, rotation only occurs as necessary. We will show examples of that in simulation results chapter.

5.11. Trajectory Generation

Trajectory generation is an important step in performing autonomous tasks using any robotic device. Since this 9-DoF robotic system can be used in both autonomous motion and teleoperation motion, it was essential to explore on different trajectory generation schemes. Four different trajectory generators were developed for this WMRA system, and the user were given the choice to choose one of them for any particular task.

5.11.1. Generator of a Linear Trajectory

It is important to mention here that the transformation from one point to the other in space is a non-linear process if it involves rotation. For this reason, the trajectory generator must take that into consideration when dealing with rotations. A typical homogeneous transformation matrix consists of three rotational vectors and one translational vector as follows:

$$T = \begin{bmatrix} n_x & o_x & a_x & P_x \\ n_y & o_y & a_y & P_y \\ n_z & o_z & a_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.54)$$

Where “n” is the projection of the unit “X” axis on the reference frame, “o” is the projection of the unit “Y” axis on the reference frame, “a” is the projection of the unit “Z” axis on the reference frame, and “P” is the coordinates of the frame’s origin on the reference frame. Since vector “P” is a linear vector that involves only distances, it needs no modification. The modification is needed for the unit axes projections on the reference frame since they include non-linear sine and cosine functions of the three angles of rotation. The three Euler angles of rotations in the homogeneous transform can be represented by a single rotation about a new single axis in space [51]. Finding that axis of rotation and the new single rotational angle makes it easier to divide that single angle into angle steps along the trajectory. The single angle of rotation can be found from the homogeneous transform as [51]:

$$\theta = \tan^{-1} \left(\frac{\sqrt{(o_z - a_y)^2 + (a_x - n_z)^2 + (n_y - o_x)^2}}{(n_x + o_y + a_z - 1)} \right) \quad (5.55)$$

Using the function (Atan2) instead of (Atan) would give a single value of the angle based on its position in the quadrant of rotation. Once this angle is found, we can now find the new axis of rotation by defining its unit vector projection on the reference frame. This can be done through three different conditions:

- 1- When the rotation angle is zero or very small, in this case, there is no rotation, and the axis of rotation can be arbitrary. For simplicity, we can define it as:

$$k = [1 \quad 0 \quad 0]^T \quad (5.56)$$

- 2- When the rotation angle is less than 90° , in this case, the axis of rotation can be defined as:

$$k = \left[\begin{array}{ccc} \frac{o_z - a_y}{2 \cdot \sin \theta} & \frac{a_x - n_z}{2 \cdot \sin \theta} & \frac{n_y - o_x}{2 \cdot \sin \theta} \end{array} \right]^T \quad (5.57)$$

3- When the rotation angle is more than 90°, in this case, the axis of rotation can be defined as:

$$k = \left[\begin{array}{c} \text{sign}(o_z - a_y) \cdot \sqrt{\frac{n_x - \cos \theta}{1 - \cos \theta}} \\ \text{sign}(a_x - n_z) \cdot \sqrt{\frac{o_y - \cos \theta}{1 - \cos \theta}} \\ \text{sign}(n_y - o_x) \cdot \sqrt{\frac{a_z - \cos \theta}{1 - \cos \theta}} \end{array} \right] \quad (5.58)$$

Where “sign” indicates the sign of the difference of what is inside the bracket. These values are not always true [51], and adjustments must be made based on which one of the projection components is the largest value as follows:

a- If “k_x” is the largest, then the other two are:

$$k_y = \frac{n_y + o_x}{2 \cdot k_x \cdot (1 - \cos \theta)} \quad \text{and} \quad k_z = \frac{a_x + n_z}{2 \cdot k_x \cdot (1 - \cos \theta)} \quad (5.59)$$

b- If “k_y” is the largest, then the other two are:

$$k_x = \frac{n_y + o_x}{2 \cdot k_y \cdot (1 - \cos \theta)} \quad \text{and} \quad k_z = \frac{o_z + a_y}{2 \cdot k_y \cdot (1 - \cos \theta)} \quad (5.60)$$

c- If “k_z” is the largest, then the other two are:

$$k_x = \frac{a_x + n_z}{2 \cdot k_z \cdot (1 - \cos \theta)} \quad \text{and} \quad k_y = \frac{o_z + a_y}{2 \cdot k_z \cdot (1 - \cos \theta)} \quad (5.61)$$

It is important to remember that the transformation matrix of the task is usually defined based on the global coordinate frame, and that the current end-effector’s frame is defined based also on the global coordinate frame. The rotation from the end-effector’s

current position “ R_i ” to the desired task position “ R_d ” should be found, and that rotation is the one that should be processed for trajectory generation of the angle between the two, and that can be found as:

$$R = R_i^T \cdot R_d \quad (5.62)$$

Once we have the single angle of rotation and the axis of rotation, we can generate the trajectory in a linear line. The approach used to generate the trajectory utilizes a constant transformation change along the trajectory, which means that the trajectory will be divided into “ n ” transformation matrices, with “ δT ” transformations between every two consecutive points in the trajectory as shown in figure 5.3.

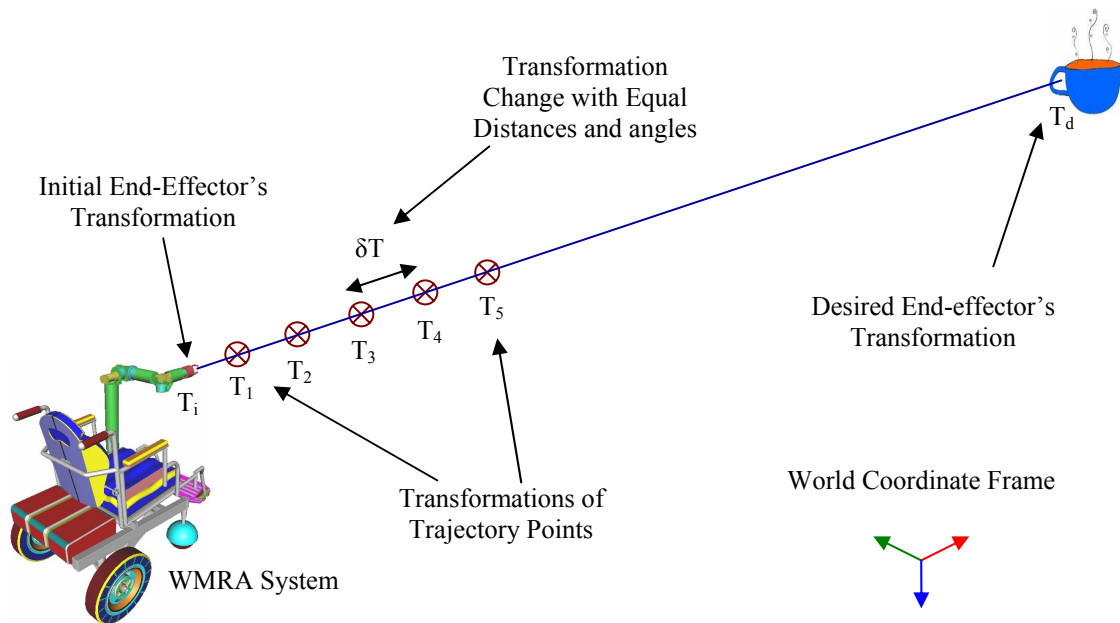


Figure 5.3: Linear Trajectory Generation.

To find the constant transformation change along the trajectory, the four variables (d_x, d_y, d_z, d_θ) that represent the constant distance change and the constant angle change must be defined as follows:

$$\begin{bmatrix} dx \\ dy \\ dz \\ d\theta \end{bmatrix} = \frac{1}{n} \cdot \begin{bmatrix} Px_d - Px_i \\ Py_d - Py_i \\ Pz_d - Pz_i \\ \theta \end{bmatrix} \quad (5.63)$$

From these values, we can now construct the rotation change along the trajectory using the following equation:

$$dR = \begin{bmatrix} k_x^2(1-\cos d\theta) + \cos d\theta & k_x k_y(1-\cos d\theta) - k_z \sin d\theta & k_x k_z(1-\cos d\theta) + k_y \sin d\theta \\ k_x k_y(1-\cos d\theta) + k_z \sin d\theta & k_y^2(1-\cos d\theta) + \cos d\theta & k_y k_z(1-\cos d\theta) - k_x \sin d\theta \\ k_x k_z(1-\cos d\theta) - k_y \sin d\theta & k_y k_z(1-\cos d\theta) + k_x \sin d\theta & k_z^2(1-\cos d\theta) + \cos d\theta \end{bmatrix} \quad (5.64)$$

Using (5.63) and (5.64), we can now find the transformation change between any two consecutive points along the trajectory as follows:

$$\delta T = \begin{bmatrix} & dx \\ dR & dy \\ & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.65)$$

To find the transformation matrix of any point along the trajectory line, the following equation will be used:

$$T_t = T_i \cdot \delta T^t, \quad 1 \leq t \leq n \quad (5.66)$$

Where “ T_t ” is the trajectory point transformation and “ T_i ” is the initial transformation matrix of the end-effector. The above scheme was coded into a program in the form of a function for trajectory generation and used for WMRA autonomous control.

5.11.2. Generator of a Polynomial Trajectory

When the robotic arm starts moving from rest, it is impossible to move it from zero to full speed in virtually no time. When the linear trajectory was generated,

simulation revealed that the joint space variables are commanded to move at infinite accelerations at the beginning of simulation to reach the desired velocity in no time. To take care of this issue, a polynomial trajectory is introduced so that when the arm starts from rest, the trajectory points are very close to each other, and then the arm will reach a maximum speed and ramp back down to zero velocity when it reaches the destination as shown in figure 5.4.

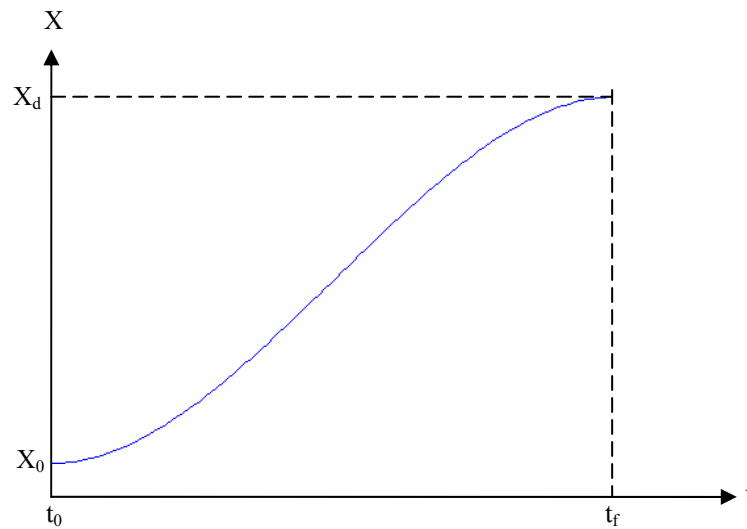


Figure 5.4: Polynomial Function of 3rd Order for Variable Ramp with Time.

The governing equation for such a polynomial [49] can be written for any variable that needs to be ramped as follows:

$$X(t) = X_0 + \frac{3}{t_f^2} \cdot (X_f - X_0) \cdot t^2 + \frac{2}{t_f^3} \cdot (X_f - X_0) \cdot t^3 \quad (5.67)$$

The above equation was implemented in the trajectory generator, and all four variables from (5.63) were divided to non-linear segments, and the results were satisfactory since the transformation change is no longer constant, but rather variable with time. This was added to the program as a second choice, and the previous linear option was also kept as the first choice.

5.11.3. Generator of a Polynomial Trajectory with Parabolic Blending Factor

When using the polynomial function to generate a non-linear trajectory, efficiency of the trajectory-following task was not acceptable. The reason is that the desired velocity is reached at the mid-point of the trajectory, and ramping that velocity up and down takes a very long time. To overcome this problem, a polynomial blending procedure was adopted [49]. The blending factor accelerates the velocities at the beginning of the trajectory, and then set the acceleration to zero throughout the major part of the trajectory following procedure when the desired velocity is reached, and then decelerate the velocity back down to zero at the end of the trajectory-following task as shown in figure 5.4 with a blending factor of “5”. Since the middle segment is linear, we use the linear function definition to define that segment. The beginning and ending segments of the trajectory are assumed to have the same duration and at the same constant acceleration with opposite signs. We first begin by defining the blending factor “b”, in our case, we chose 5. Then we define the acceleration during blending as:

$$\ddot{X}_b = 4 \cdot b \cdot \frac{X_f - X_i}{t_f^2} \quad (5.68)$$

Where “ t_f ” is the time at which the trajectory-following task is completed. Then we define the time when blending ends as:

$$t_b = \frac{t_f}{2} - \frac{\sqrt{\ddot{X}^2 \cdot t^2 - 4 \cdot \ddot{X} \cdot (X_f - X_i)}}{2 \cdot \ddot{X}} \quad (5.69)$$

$$\text{The velocity at that point would be: } \dot{X}_b = \ddot{X}_b \cdot t_b \quad (5.70)$$

$$\text{And the variable's value at blending would be: } X_b = X_i + 0.5 \cdot \ddot{X}_b \cdot t_b^2 \quad (5.71)$$

Substituting these values in to the 3rd degree polynomial gives a smooth trajectory as shown in figure 5.5. When the blending factor is “1”, the constant velocity region becomes a point, and that returns us back to the polynomial with no blending as described in the previous subsection.

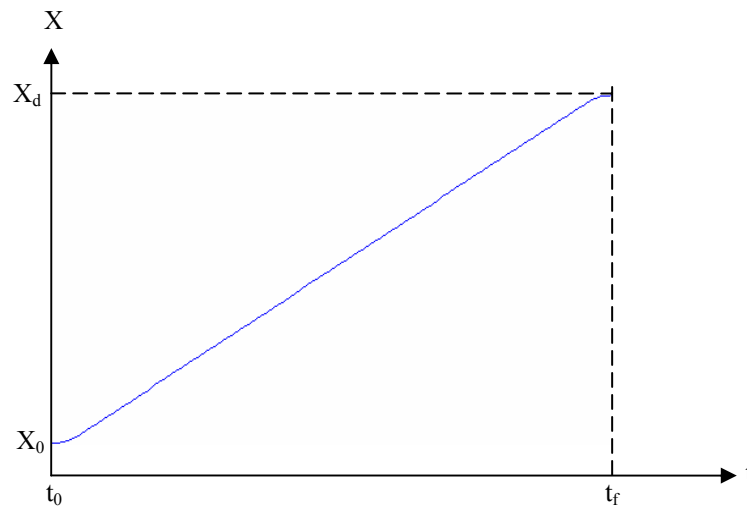


Figure 5.5: Polynomial Function of 3rd Order for Blended Variable Ramp with Time.

A slight modification was made to make the blending factor vary based on the length of the trajectory and the commanded velocity so that the curve is not too steep and at the same time it doesn't, the acceleration is reasonable during the blending regions. The second blending region is done the opposite way of the first one. Figure 5.6 shows the trajectory generation with polynomial function.

5.12. Control Reference Frames

At the beginning of solving the manipulation and mobility combination problem, the commanded motion was referenced to the ground frame. In reality, this doesn't always help the user in the most effective way. For instance, when the robotic arm is used in autonomous mode, it is better to reference the motion to the ground frame, but when

the user is controlling the WMRA system using teleoperation, he/she will be confused about which direction represents the forward or right turn and so forth. For that reason, three different control reference frames were programmed so that the user can choose the most convenient one based on the task at hand. The ground frame is most suited for autonomous operation with pre-set tasks, the wheelchair's frame is most suited for wheelchair motion in the most part, and the end-effector's frame is most suited for teleoperation using the end-effector. Refer to figure 5.1 for the Cartesian coordinate frame of references that we will be working on to transform the Cartesian task space and the Jacobian from one frame to the other.

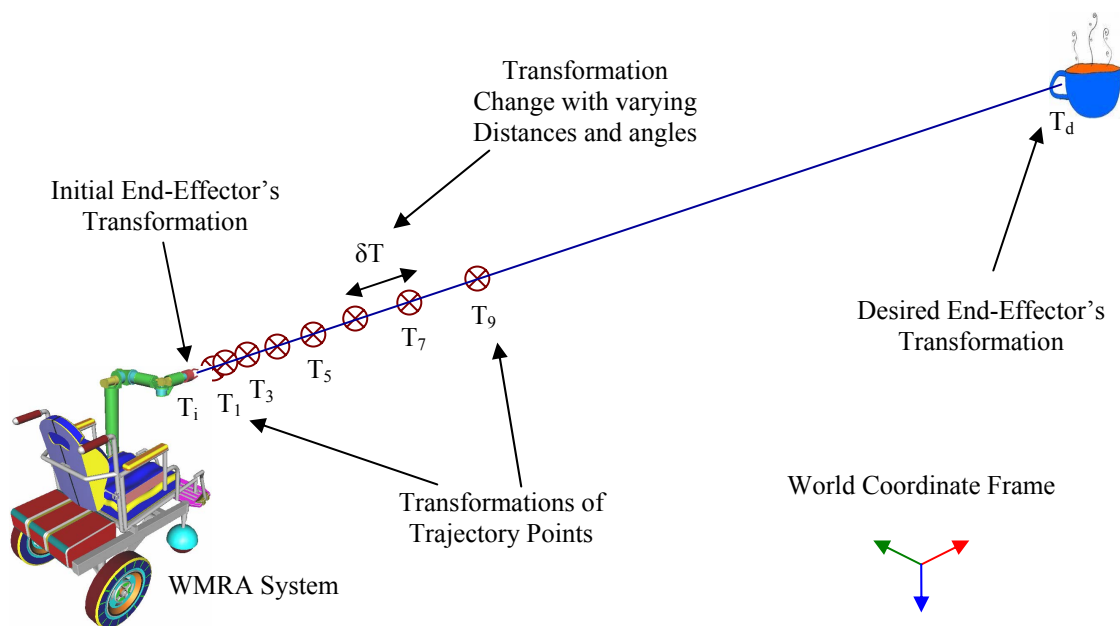


Figure 5.6: Polynomial Trajectory Generation.

5.12.1. Ground Reference Frame

This is the default option that was used for reference in all the previously generated equations. The commanded Cartesian positions expressed in a homogeneous

transformation matrix were based on the ground coordinate frame, and the Jacobian was also based on the ground frame. If the user is operating in autonomous mode, the trajectory will also be generated based on the ground frame. If the user uses this option, it may cause some inconvenience in the sense that if the wheelchair rotates 180° and the user tries to move forward in his/her mind, the actual motion would be backwards since the positive X-axis of the ground reference frame is stationary and will be pointing backwards from the wheelchair's prospective.

5.12.2. Wheelchair Reference Frame

In this option, the user would like to use the wheelchair's frame as the reference frame for the motion of the WMRA system. Two transformations have to be made, one that transforms the commanded Cartesian positions from the wheelchair's reference frame to the ground's reference frame, and that would be modified as:

$${}^G T = {}^G T \cdot {}^W T \quad (5.72)$$

And the other transformation is to transform the Jacobian from the ground frame to the wheelchair's frame as follows:

$${}^A J_W = \begin{bmatrix} {}^G R^T & [0] \\ [0] & {}^G R^T \end{bmatrix} \cdot {}^G J_W, \text{ and } {}^A J_A = \begin{bmatrix} {}^G R^T & [0] \\ [0] & {}^G R^T \end{bmatrix} \cdot {}^G J_A \quad (5.73)$$

When the user is using the WMRA system to perform pre-set tasks of activities of daily living in autonomous mode, a third transformation is necessary to transform the trajectory generation from the wheelchair's reference frame to the ground's reference frame as follows:

$$T_{t,new} = {}^G T_{current}^{-1} \cdot {}^G T_{initial} \cdot T_{t,old} \quad (5.74)$$

This gives the user the feeling of controlling the WMRA system based on where the wheelchair moves. A very good choice to for this option would be when the user is controlling the wheelchair alone without autonomous motion and without using the robotic arm.

5.12.3. End-Effector Reference Frame

In this option, the user uses the end-effector's frame as the reference frame for the motion of the WMRA system. Two transformations have to be made here as well, one that transforms the commanded Cartesian positions from the end-effector's reference frame to the ground's reference frame, and that would be modified as:

$${}^G T = {}^G T \cdot {}^7 T \quad (5.75)$$

And the other transformation is to transform the Jacobian from the ground frame to the end-effector's frame as follows:

$${}^7 J_W = \begin{bmatrix} {}^G R^T & [0] \\ [0] & {}^G R^T \end{bmatrix} \cdot {}^G J_W, \text{ and } {}^7 J_A = \begin{bmatrix} {}^G R^T & [0] \\ [0] & {}^G R^T \end{bmatrix} \cdot {}^G J_A \quad (5.76)$$

When the user is using the WMRA system to perform pre-set tasks of activities of daily living in autonomous mode, a third transformation is necessary to transform the trajectory generation from the end-effector's reference frame to the ground's reference frame as follows:

$$T_{t,new} = {}^G T_{current}^{-1} \cdot {}^G T_{initial} \cdot T_{t,old} \quad (5.77)$$

This gives the user the feeling of controlling the WMRA system based on where the end-effector is pointing. This option is a very good choice when the user controls the WMRA to do activities of daily living using the end-effector.

5.13. Summary

The main aim of this chapter is to show the theory of combining the mobility of 2-DoF the wheelchair and the manipulation of the 7-DoF robotic arm to form a single control structure of the 9-DoF wheelchair-mounted robotic arm. We derived the combined forward kinematics of the system, and showed the relations between the two subsystems with each other and with the ground. The new WMRA system with 3 degrees of redundancy was controlled using the resolved rate after augmenting the Jacobian to include both subsystems. Several methods of inversion were implemented to find the inverse of the jacobian to solve the inverse kinematic problem. Optimization was then done using several techniques and the control algorithm was designed to use any optimization method or criterion function.

Two different state variables were implemented in the WMRA system to reduce the unintended motion of the wheelchair while executing a task of a long trajectory. It was found that using the polar-coordinate type variables in the mobility side of the problem gave more efficient results when used with the robotic arm. Trajectory generation was done using different linear and polynomial functions with or without parabolic blending. The control reference frame was also shown in three different bases of reference, and each one of them was useful in different kinds of setups.

Chapter 6:

User Interface Options

6.1. Introduction

One of the important aspects of controlling robotic devices is the user interface, especially if the user is a person with disabilities. If the design of the robot is good, the control algorithm is very sophisticated, but the user interface is insufficient, the user might not utilize the robot up to its capabilities due to lack of control. In this chapter, we will talk about the clinical tests done on two WMRA user interfaces that are used in commercial WMRA systems. Then we will talk about the user interface devices used for this new WMRA that was included in the program software, as well as some other possibilities that can be added later as the user prefers.

6.2. User Interface Clinical Testing

Two different user interfaces were tested in two different commercial wheelchair-mounted robotic arms, Manus and Raptor. These two user interfaces are the double-axis joystick as well as a keypad. A test procedure was put in place to do the testing, and human subjects with disabilities were tested to perform these tasks using the chosen user interfaces. Cognitive load was addressed based on the different user interfaces for different users and the type of control they are using (Cartesian or joint control). The

effectiveness of the two commercial WMRA based on these aspects and the execution times for different ADL tasks were addressed.

6.2.1. Representative ADL Tasks Used for the Clinical Study

In WMRA applications to perform tasks of daily living, the user-specific needs should be taken into consideration to make a proper selection of a WMRA [57]. This criterion is based on user's age, size, weight, disability and prognosis. Other characteristics should also be addressed such as the cosmetics features, cost and payload capacity needed. In this clinical study, a representative population of individuals with different functional limitations who use a personal aid to do their ADL tasks were observed and interviewed while performing reach and manipulation tasks associated with their ADL using Manus and Raptor in separate occasions. Their opinions on the ease of use and comfort-related aspects were obtained.

Based on the uses of WMRA devices for ADL tasks, a test bed was designed to evaluate the user interface and control of each WMRA. In order to assess the functional use, different ADL related tasks, as shown in figure 6.1, were designed as follows:

- 1- Relocating an object on a level plane: This task consisted of moving the WMRA from the home position (H), picking up an object from quadrant-1 and positioning it in quadrant-2. Each quadrant was 8" deep and 11.5" wide. The quadrants were on a table surface 30" above the floor surface.
- 2- A pronation / supination function to simulate a pouring function: This task consisted of moving the WMRA from the home position (H), picking up a

water bottle from quadrant-1 and pouring the water into a cup in quadrant-2. Each quadrant was 11.5” deep and 8” wide.

- 3- Accessing a higher level cabinet: This task consisted of moving the WMRA from the home position (H), picking up the object from the surface of a shelf (24”) above the surface of the table and placing it on the table surface.
- 4- Picking up an object from the floor: This task consisted of moving the WMRA from the home position (H) to the floor (1), picking up the object and placing it on the table surface (2).

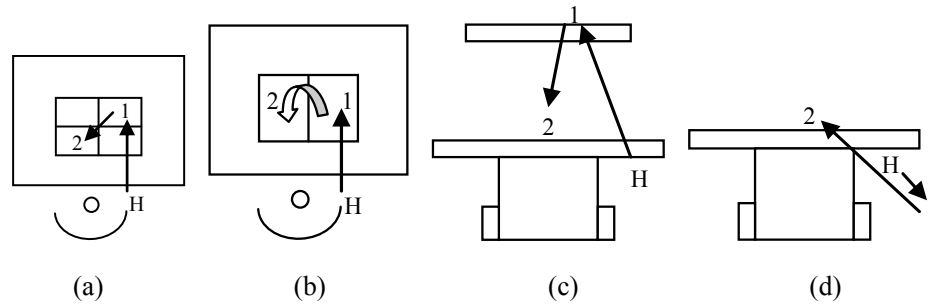


Figure 6.1: Four Different ADL Tasks.

Each task was performed three times and the time was recorded. The Raptor was tested with the joystick interface and the Manus with the joystick and the keypad interfaces. The input device for Raptor consists of an eight-way joystick. An additional set of switches controlled the opening and closing of the gripper. The four-way joystick of Manus controls the arm and the gripper using four different menus in the Cartesian control mode. The user accesses the menus by a quick tap of the joystick in either direction. The keypad interface consists of 16 buttons that the user can activate for control of the WMRA. Both the keypad and joystick systems offered a visual display to the user indicating the menu and function.

6.2.2. The Tested User Interfaces

Three user interface devices were tested in this clinical study, two different joysticks and one keypad. These devices were supplied by the manufacturer of these robotic arms, and they had no flexible robotic system to take third-party user interfaces. Manus uses a two dimensional joystick, four-way joystick as shown in figure 6.2, that is easy to move in all directions in its circular motion.



Figure 6.2: Four-Way Joystick for Manus.

Manus was also equipped with another user interface that uses a 4X4 matrix of buttons on a keypad as shown in figure 6.3. This device works individually from the joystick, and it has a clear display of button functions.



Figure 6.3: Eight-Button Keypad for Manus.

The third user interface came with Raptor, which is a two-dimensional, eight-way joystick as shown in figure 6.4. That joystick can travel only along the eight directions to control the arm, and if the direction does not match one of these eight ways, the joystick doesn't move.



Figure 6.4: Eight-Way Joystick for Raptor.

6.2.3. Population of the Chosen Users with Disabilities

In this clinical study, two individuals with disabilities (C5-6 quadriplegia) of similar size and weight, who have been power wheelchair users for 25 years, were selected to test the evaluation test bed. The home position for each WMRA was chosen as the stored position in which the user would normally place the arm when travelling. After each user was trained on the use of the WMRA device, they were timed on the performance of each task. Errors were noted if the object was dropped, placed 6" beyond the destination or the task was incomplete. In the second phase, cognitive load was added to determine the effect on the time. This was done by asking the subjects a series of questions using a telephone headset as they performed each task.

6.2.4. Clinical Test Results on User Interfaces

In the tests conducted by this study, the users were active power wheelchair users, and they favoured a joystick interface, but did not like using a two dimensional control for three dimensional output. A space-ball or glove with voice recognition and macro controls would be far more efficient. The tests showed that the Raptor's 2D, 8-way joystick control interface was the easiest to understand and learn. However the users found it difficult to activate the secondary switch for opening and closing the gripper. The Manus with joystick was the most difficult to learn and errors were caused by the user when using the joystick to access the menu structure. The keypad offered direct control and was most efficient. However the users had difficulty with the size of the buttons and shape on the keypad as shown in figure 6.5.



Figure 6.5: Clinical Testing of the Keypad by a Power Wheelchair User.

In moving the object in the same plane, the users had difficulty picking up objects from quadrant 1 and took an average of 180 seconds to pick up the object using Raptor as

shown in figure 6.6. Once the object was picked up, positioning it in quadrant two was done in 15-30 seconds and the return to home occurred in 20 seconds or less. The same task took about half the time when Manus was used with the keypad. Cognitive loading interestingly did not affect the initial phase of the task (H-1), but significantly increased the time trebled for the remainder of the task. The users had difficulty with diagonal movement of the arms.



Figure 6.6: Clinical Testing of the Joystick by a Power Wheelchair User.

The pouring task was the most difficult and the operation of tilting the bottle often caused the water to spill outside the receiving cup when Raptor was used. The model presented was a useful test to evaluate WMRA. Both the degrees of freedom and the control interface are critical for an efficient WMRA use. Performance was best in the case of the Manus with the keypad, where sufficient degrees of freedom existed with the least complicated user interface control. However, the performance can be greatly

enhanced by a more intuitive control with less cognitive load. Other user interfaces may put these two arms in a better usability when used by people with disabilities.

6.3. The New WMRA User Interfaces Used

One of the most difficult situations that can affect the outcome of the use of WMRA systems is the existence of two individual user interfaces to individually control the power wheelchair and the robotic arm. In the new WMRA system, since both the wheelchair and the arm are being controlled in the same control algorithm, a single user interface can be used to perform ADL tasks without the need of user-cooperated motion from two different interfaces. In the program structure, flexibility was one of the objectives in the design of user interfaces so that a wider range of these interfaces can be used based on the user's abilities and preference.

6.3.1. Six-Axis, Twelve-Way SpaceBall

This user interface makes a three dimensional motion that corresponds to the six Cartesian space variables used in the WMRA. Three translational directions in their positive and negative values, and three rotational directions in their positive and negative values. Figure 6.7 shows the SpaceBall device that is programmed and used in the control software implementing the control algorithm discussed in this work. In addition to the SpaceBall's main functionality, twelve fully programmable buttons were added for any sub-commands that might be used in the control of the WMRA system. The problem in this device is that it is stiff and might be hard to move by people with disabilities.



Figure 6.7: Twelve-Way SpaceBall.

6.3.2. Computer Keyboard and Mouse

Another user interface is the computer keyboard and mouse, where the user might prefer the use of these devices as shown in figure 6.8. This option was programmed in the control algorithm in case a computer is equipped with special software, such as speech recognition software, that the user might be using for other functions.



Figure 6.8: A Keyboard and a Mouse.

6.3.3. Touch Screen on a Tablet PC

This is one of the main user interfaces that can be used in this application since a tablet PC is already installed as part of the control hardware. The implementation of the control structure included this option in both simulation and actual WMRA motion. The touch screen used is of a Fujitsu Lifebook tablet PC equipped with a 12-inch active digitizer as shown in figure 6.9.



Figure 6.9: A 12-Inch Touch Screen of a Tablet PC.

This user interfaces can be used with the newly developed graphical user interface (GUI) program shown in figure 6.10, where the user can touch any of the directional buttons to activate the proper Cartesian directional motion of the WMRA system.

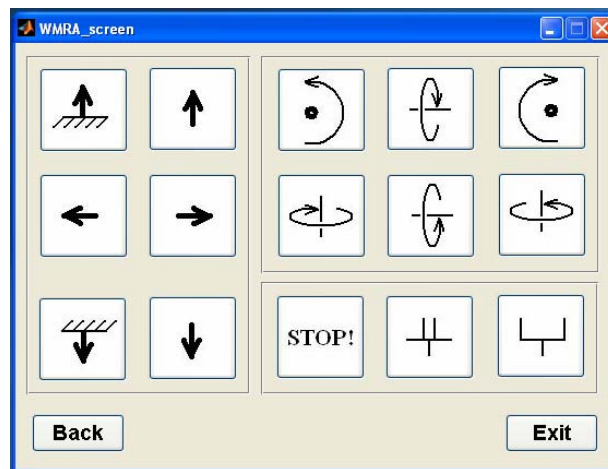


Figure 6.10: GUI Screen Used for the Touch Screen.

6.4. The Brain-Computer Interface (BCI) Using P300 EEG Brain Signals

Many people with severe motor disabilities need augmentative communication technology to enable them to control different devices independently. Those who are totally paralyzed, or “locked-in,” cannot use conventional augmentative technologies, all of which require some measure of muscle control. Over the past two decades, a variety of studies has evaluated the possibility that brain signals recorded from the scalp or from within the brain could provide new augmentative technology that does not require muscle control [58] for a comprehensive review. These BCI systems measure specific features of brain activity and translate them into device control signals as shown in figure 6.11.

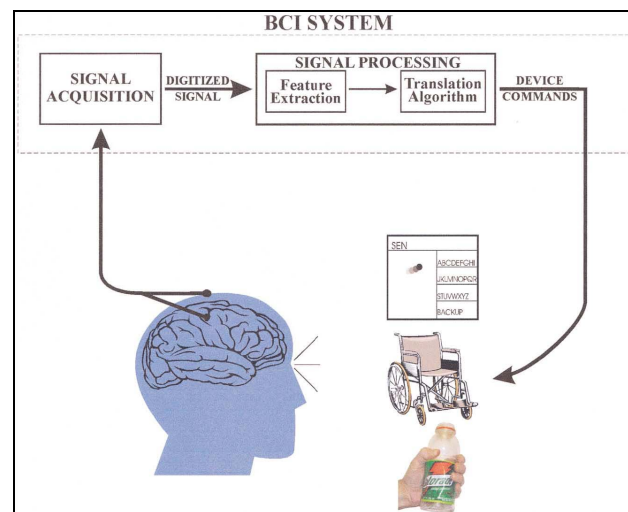


Figure 6.11: Basic Design and Operation of the BCI System.

6.4.1. The P300 EEG Signal

The P300 is a neural evoked potential component of the electroencephalogram, or EEG [59]. This event-related potential (ERP) appears as a positive deflection of the EEG voltage at approximately 300 ms. It dominates at parietal electrode sites. The P300 is

supposed to follow unexpected sensory stimuli or stimuli that provide useful information to the subjects according to his/her task.

The P300 only peaks in the vicinity of 300 millisecond for very simple decisions [59]. More generally, its latency appears to reflect the amount of time necessary to come to a decision about the stimulus. The P300 also has useful properties of being larger to rare stimuli, especially if they are targets. The amplitude of the P300 therefore gives information about how the person is categorizing the stimuli and how rare they are considered to be subjectively. The P300 is only seen when the person is actively keeping track of the stimulus so it also gives information about what they are paying attention to, which makes it useful for BCI applications. A further parameter is the method of feedback used and this is shown in studies of P300 signals. Patterns of P300 waves are generated involuntarily (stimulus-feedback) when people see something they recognize and may allow BCIs to decode categories of thoughts without training patients first.

6.4.2. The Use of the BCI

The features used in studies to date include slow cortical potentials, P300 evoked potentials, sensory motor rhythms recorded from the scalp, event-related potentials recorded on the cortex, and neuronal action potentials recorded within the cortex. These studies show that non-muscular communication and control is possible and might serve useful purposes for those who cannot use conventional technologies. To people who are locked-in (e.g., by end-stage amyotrophic lateral sclerosis, brainstem stroke, or severe polyneuropathy) or lack any useful muscle control (e.g., due to severe cerebral palsy), a BCI system, as shown in figure 6.12, could give the ability to answer simple questions

quickly, control the environment, perform slow word processing, or even operate a neuron-prosthesis or orthosis [58]. For easier, non-invasive use of this neuro-imaging technology, the user wears a head mask fitted with several electrodes to measure the P300 EEG signals from the activities of the brain as shown in figure 6.12.



Figure 6.12: The Non-Invasive BCI Device.

6.4.3. The BCI-2000 Interface to the New 9-DoF WMRA System

In collaboration with the Department of Psychology at the University of South Florida, we were able to develop a new user interface that uses the portable BCI-2000 device to control the new WMRA system even for people who are paralyzed from the neck down. The screen shown in figure 6.13 was developed to give the user the proper perspective of what to control, and at the same time to serve as the user feedback for the selected image. The BCI-2000 scans the rows and columns of the screen choices shown in figure 6.13 at high frequency so that one row or one column is shown at a time. The user is asked to look at the symbol that he/she would like to use to control the WMRA

system and count the number of times the/she saw that symbol. Every time the user counts one more view of the symbol, the P300 EEG signal is recorded, and the corresponding row or column that was shown at that moment was also recorded. In about 15 seconds, the BCI-2000 gives the selected row and column of the shown matrix on the screen. Once this value is received by the WMRA control program, it translates it into a Cartesian velocity in the proper direction and executes the algorithm to move the arm.

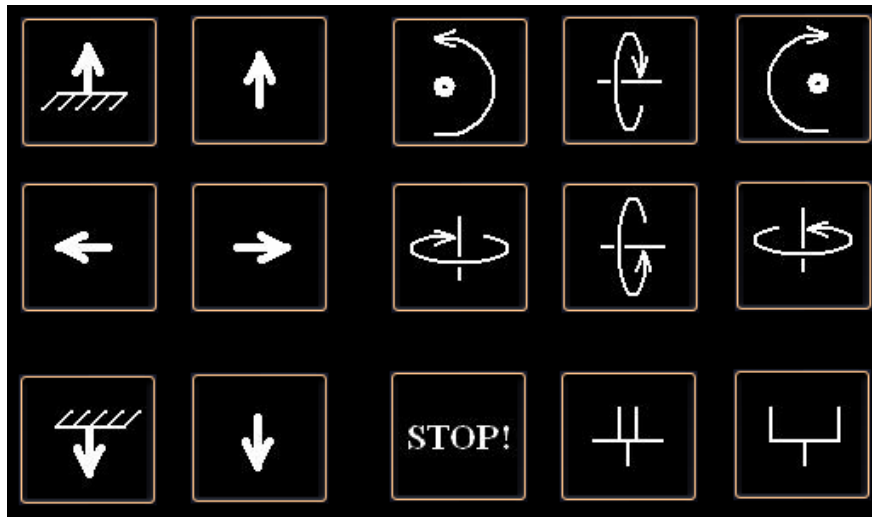


Figure 6.13: Basic Design and Operation of the BCI System.

6.4.4. Testing of the BCI-2000 with the WMRA Control

Before the BCI-2000 was tested to control the robotic arm, a volunteer human subject was trained properly to use the device. The BCI-2000 was also trained to be optimized for that particular human subject, and it showed high accuracy of the selected choice (ranging from 92% to 100 %). These gains were recorded to be used for the actual test. During the testing phase, a successful control with high accuracy of the motion response was apparent. Few potential problems were noticed as follows:

- 1- Every full scan of a single user input takes about 15 second, and that might cause a delay in the response of the WMRA system to change direction on time as the human user wishes. This 15 second delay may cause problems in case the operator needs to stop the WMRA system for a dangerous situation such as approaching stairs.
- 2- After an extended period of time in using the BCI-2000 system, fatigue starts to appear on the user due to his concentration on the screen when counting the appearances of his chosen symbol. This tiredness on the user's side might be a potential problem.
- 3- In case the wrong selection was made by the BCI-2000, the user will be frustrated to return back to his/her original choice.
- 4- When the user is constantly looking at the screen and concentrating on the chosen symbol, he/she will not be looking at where the WMRA is going, and that poses some danger on the user.

Despite the above noted problems, a successful interface with a good potential for a novel application was developed.

6.5. Expandability of User Interfaces

During the programming and implementation of the designed control algorithm, modularity and flexibility of the WMRA system was taken into account. For this purpose, all mentioned user interfaces are converted into a corresponding vector that is interfaced to the main program through a single function. Changing the user interface, or adding other interfaces is very easy in this context since the output of any new user interface can

be reformatted to the proper vector format in a new function that will directly interface to the main program and be used as a new interface selection. Other possible interface selections can be added, including the following devices.

6.5.1. Omni Phantom Haptic Device

The force-feedback enabled Phantom device from SensAble Technologies shown in figure 6.14 can be used as one of the user interface devices. It carries a stylus mounted on a six-joint mechanism with encoders and force transducers. The Cartesian coordinate velocities of the tip of the stylus can be mapped into an input to the commanded Cartesian velocities of the WMRA system.



Figure 6.14: The Phantom Omni Device from SensAble Technologies.

The Phantom allows users to actually feel virtual objects if integrated with a sensory suite. The Phantom contains 3 motors, which control the x, y, and z forces exerted on the user's fingertip. Mounted on each motor is an optical encoder to determine the x, y, and z position of the user's fingertip. The torque from the motors is transmitted

through a proprietary transmission cable to a stiff, lightweight linkage. Incorporated at the end of this linkage is a passive 3 degree-of-freedom set of gimbals attached to a thimble [60]. The passive gimbals allow the thimble to rotate so that a user's fingertip may assume any comfortable orientation.

6.5.2. Sip n' Puff Device

The Sip n' Puff is a term used to describe a dual-switch system which utilizes pneumatic switches. A single piece of tubing, accessible to the user, controls both switches as shown in figure 6.15. A slight pressure (puff) operates one switch, while a slight vacuum (sip) operates the other, and the proper signal to the controlled device is sent through an RS232 serial port. This device is widely used in assistive technology applications for control. A disadvantage of this device is the fact that it acts as an on/off switch, which means that its use will be very complicated for the user to control functions that need many input choices.



Figure 6.15: The Sip and Puff Input Device.

6.5.3. Head and Foot Switches

Head and foot switches such as the ones shown in figure 6.16 can also be used for a user interface to the WMRA system in case the user's foot or head muscles are the strongest controllable muscles in his/her body. Some of the foot switches allow the user to rest the activating body part on top of the switch between activations. The head switch can be activated with a light pressure exerted by the user's head,



Figure 6.16: Head and Foot Switches.

6.6. Summary

In this chapter, several user interface options were presented. A clinical study of the user interface devices used by the commercially available WMRA was presented, and a test procedure was described. The high-level control algorithm of the WMRA can be interfaced with many user interfaces, but the ones tested were the SpaceBall, the Keyboard and mouse, the touch screen, and the Brain-Computer interface (BCI) that reads the P300 EEG signal from the brain to control the WMRA just by paying attention to a visual display. Other devices that can easily be adapted to the WMRA control include the Phantom Omni haptic devices, the Sip n' Puff devices, and the head and foot switches among others.

Chapter 7:

Testing in Simulation

7.1. Introduction

When new concepts in control are developed, it is important to validate them by means of simulation. In our case, the control methods that combined the manipulation and mobility of the newly developed WMRA were tested in simulation before applying them to the actual WMRA system. This step is very important for debugging and inspecting the methods before applying them into the actual arm so that no harm to the physical system is done in case of unexpected errors. In this chapter, we will show the different ways this theory was implemented, and the different programming packages used for this purpose. Figure 7.1 shows a flowchart of the program procedure.

7.2. User Options to Control the WMRA System

In the control software, several options were made available to include the modularity, re-configurability and flexibility requirements for this WMRA system. These options were programmed to work in combination with any possibilities that make sense of the control as follows:

- 1- What to control: This option gives the user the option to control both position and orientation of the end effector with its six Cartesian variables, or control

the position only and ignore the orientation to bring the Cartesian space variables down to three variables and use the resultant six-degree of redundancy system for different sub-task optimization.

- 2- What to run: In this option, the user is given three choices. These choices are to run the robotic arm only while freezing the wheelchair, to run the wheelchair only while freezing the arm in certain configuration specified by the user, or to run the combined WMRA system that uses both the robotic arm and the power wheelchair.
- 3- What is the control coordinate reference frame: This option gives the user the choice of controlling the end effector in reference to the ground coordinate frame, the wheelchair coordinate frame, or the gripper's coordinate frame.
- 4- What kind of simulation to run: This option gives the user to run Virtual Reality simulation, Matlab wire frame simulation, both simulations together, or no simulation at all.
- 5- Run the actual WMRA: This option gives the user the option to run the WMRA system or not when running the control software.
- 6- Print diagnostic plots: This option allows the user to print out the various states of the system variables in terms of position, velocity and acceleration of the points of interest in the WMRA system, as well as the manipulability measure of both the arm and the WMRA system.
- 7- Optimization Method: This option gives the user the option to use the eight optimization combination methods discussed in chapter five. It also allows the

user to select the joint limit avoidance and/or the joint limit and obstacle safety stop options on the control system.

- 8- Close all when completed: This option gives the user the option to close or keep open the simulation windows, the diagnostic plots, and the WMRA control DLL library that connect to the actual WMRA.
- 9- User interface options: This option allows the user to choose autonomous operation using position control, velocity control of the end effector. It also allows the user to choose teleoperation control using the SpaceBall, the BCI-2000 system, or the touch screen.
- 10- Trajectory generator: This option allows the user to choose the trajectory to be linear, polynomial, or polynomial with parabolic blending.
- 11- Where to start: This option gives the user the option to start the WMRA system at the ready position, the current position or a user-specified position.
- 12- Include pre-set task motion: This option gives the user the option to initialize the system from its parking position to its ready position, and when the user is finished using the WMRA system, it gives the option to go back to the ready position and the park position, respectively.

The above user choices were adequate to allow the user to choose the most comfortable options based on his/her preference so that the WMRA could be used efficiently with as many user-specific needs as possible. When the user chooses to control the wheelchair only, the wheelchair motion is slow relative to the normal wheelchair velocities. If the user needs a normal operation of the wheelchair, the control system can shut down, and the control switch can be switched to the standard wheelchair controller.

7.3. Changing the Physical Dimensions and Constraints of the WMRA System

It was noted in previous programming experience that a program can be extremely difficult to modify by other than the developer if any physical changes or modifications to the system occur. Since this is a project that could involve several changes, it is important to store few files that describe the physical characteristics of the system, and have each function or script programmed to read from these files so that any physical changes to the system can be easily accommodated in the control software. Three files were dedicated for this purpose as follows:

- 1- Wheelchair dimensions: A file named “WMRA_WCD.m” was designed to carry the physical dimensions of the wheelchair that are used in the program, as well as the mounting location of the robotic arm on the wheelchair.
- 2- Robotic arm parameters: A file named “WMRA_DH.m” was designed to carry the D-H parameter table of the robotic arm.
- 3- Robotic arm joint limits: A file named “WMRA_Jlimit.m” was designed to carry the maximum and minimum joint limits of the robotic arm.

7.4. Programming Language Packages Used

In order to fulfill the need of implementing the program in simulation and in the physical arm, it was important to choose compatible programs whenever possible. For the physical arm, the communication protocols and functions that send the commands and receive the sensory information from the controller boards use C++ with certain DLL library functions. On the other hand, simulation is best done using Matlab 2006b from MathWorks because of its powerful toolboxes that include good packages for simulation.

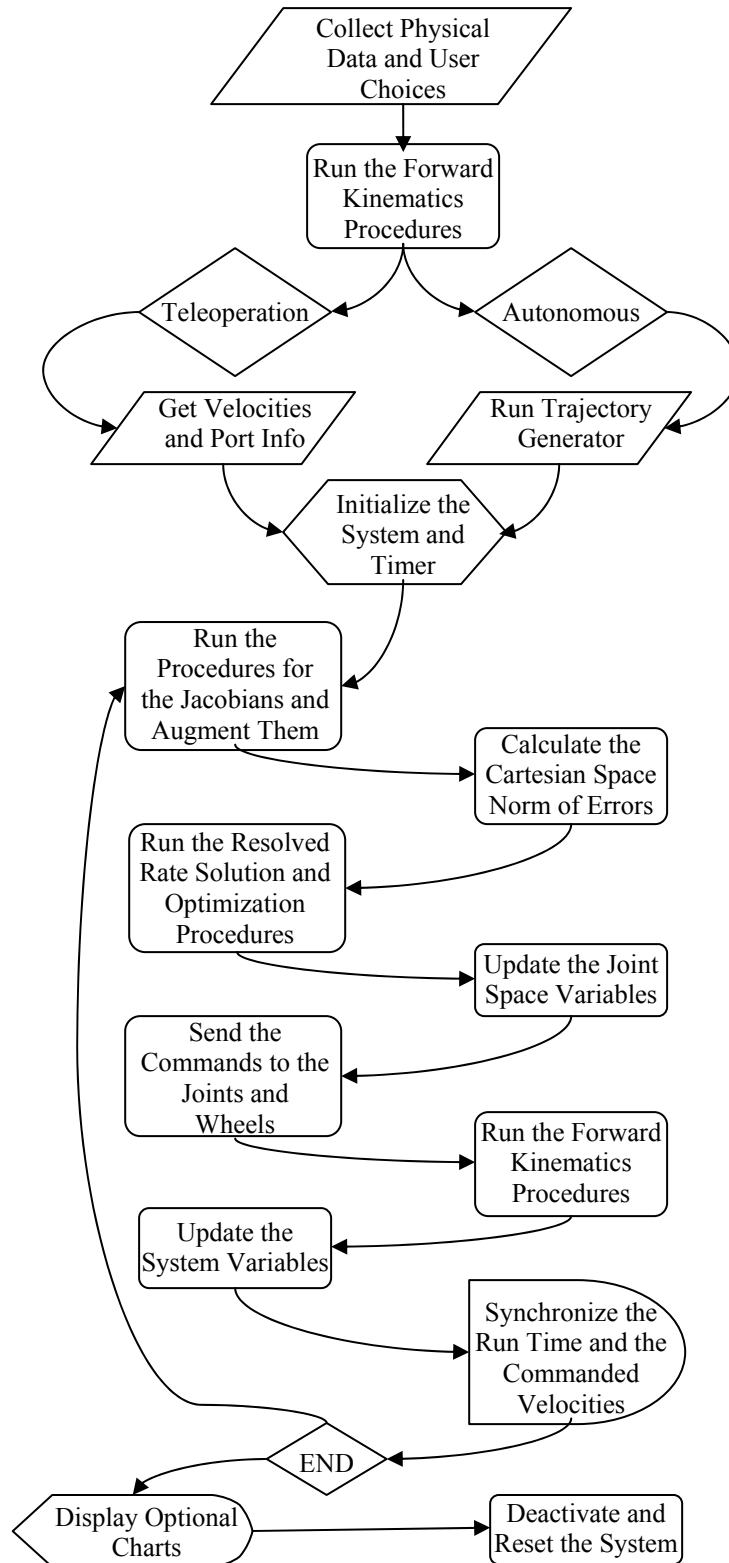


Figure 7.1: Program Flowchart.

To be compatible with the goal of modularity of the physical system, a separate file that includes the physical characteristics of the robotic arm and the wheelchair was created, and every thing in the program refers to that file to read the piece of information needed for its calculations. In this sense, the program can be easily modified for any other WMRA system with different physical characteristics by editing that file and changing the information from the current system to the new system.

7.4.1. Programs in C++ Programming Language

Since the PIC Servo SC controller boards were interfaced to the PC using DLL libraries that are programmed in C++ as functions, it was clear that this programming language had to be used for communication with the PC. In terms of simulation, these are not needed since we did not need to communicate with the actual WMRA to perform the simulation. However, for the Space Ball to be implemented and integrated with the system, C++ programming was required as its drivers were compatible with a C++ library. The program is designed to run the driver of the SpaceBall and collect the user inputs from the device and send it to a Matlab environment as a vector variable that is changed constantly as the user moves the SpaceBall.

7.4.2. Matlab Programming Environment

The main programming language used to implement the control system is Matlab since it includes a lot of libraries and simulation capabilities. The program was coded into Matlab code that includes the main script as well as several functions created for certain purposes. Each one of the function was created in such a way that it was simple to

understand, and at the same time easy to be modified for future changes. The main script runs in Matlab command prompt and starts by asking the user questions and collecting the answers so that it runs according to the user preferences. When the program runs in simulation mode, the user can chose between simulation in wire frame graphics or in Virtual reality graphics. In the wire frame graphics, precise Cartesian coordinate lines were drawn and simulated through time for the ground, wheelchair, arm base and end-effector coordinate frames. Another coordinate frame was added in case autonomous operation was required and the desired destination coordinate frame can be shown. In this case, at the end of simulation, the end-effector's coordinate frame coincides with the destination coordinate frame. This gives a precision representation of the simulation accuracy.

At the end of simulation, optional plots can be displayed, including the positions, velocities and accelerations of the joint space variables, the Cartesian coordinates of both the end-effector and the wheelchair, and the manipulability measure throughout the simulation period. These plots are very useful to understand the behavior of the system and the response throughout the simulation and when certain characteristics or methods are chosen over others. They also help in diagnosing any problems or potential problems that may appear when the control is implemented. Figure 7.2 shows a sample of the command prompts when the user uses the program from the common command line of Matlab to use the simulation in autonomous mode. Figure 7.3 shows the simulation window of the WMRA wire frame with the Cartesian coordinate frames attached and color-coded. The results of the simulation will be shown and discussed in the next chapter.

```

>> WMRA_Main
Choose what to control:
For combined Wheelchair and Arm control, press "1", For Arm only control, press "2", For Wheelchair
only control, press "3". 1

Choose whose frame to base the control on:
For Ground Frame, press "1", For Wheelchair Frame, press "2", For Gripper Frame, press "3". 1

Choose the cartesian coordinates to be controlled:
For Position and Orientation, press "1", For Position only, press "2". 1

Please enter the desired optimization method:
(1= SR-I & WLN, 2= P-I & WLN, 3= SR-I & ENE, 4= P-I & ENE) 1

Do you want to include Joint Limit Avoidance? (1= Yes, 2= No) 1

Do you want to include Joint Limit/Velocity and Obstacle Safety Stop? (1= Yes, 2= No) 1

Choose the control mode:
For position control, press "1", For velocity control, press "2", For SpaceBall control, press "3",
For Psychology Mask control, press "4", For Touch Screen control, press "5". 1

Please enter the transformation matrix of the desired position and orientation from the control-based frame
(e.g. [0 0 1 1455;-1 0 0 369;0 -1 0 999; 0 0 0 1]) [ 1 0 0 800 ; 0 1 0 -500 ; 0 0 1 350 ; 0 0 0 1 ]

Please enter the desired linear velocity of the gripper in mm/s (e.g.50) 50

Chose the Trajectory generation function:
Press "1" for a Polynomial function with Blending, or
press "2" for a Polynomial function without Blending, or press "3" for a Linear function. 1

Choose animation type or no animation:
For Virtual Reality Animation, press "1", For Matlab Graphics Animation, press "2",
For BOTH Animations, press "3", For NO Animation, press "4". 2

Would you like to run the actual WMRA? For yes, press "1", For no, press "2". 2

Press "1" if you want to start at the "ready" position,
or press "2" if you want to enter the initial joint angles. 1

Press "1" if you want to include "park" to "ready" motion, or press "2" if not. 1

Press "1" if you do NOT want to plot the simulation results, or press "2" if do. 1

Simula. time is 7.460476 seconds. Elapsed time is 7.513704 seconds.

Do you want to go back to the "ready" position? Press "1" for Yes, or press "2" for No. 1

Do you want to go back to the "parking" position? Press "1" for Yes, or press "2" for No. 1

Do you want to close all simulation windows and arm controls? Press "1" for Yes, or press "2" for No. 1

>>
>>

```

Figure 7.2: A Sample Command Prompts for Autonomous Operation Mode.

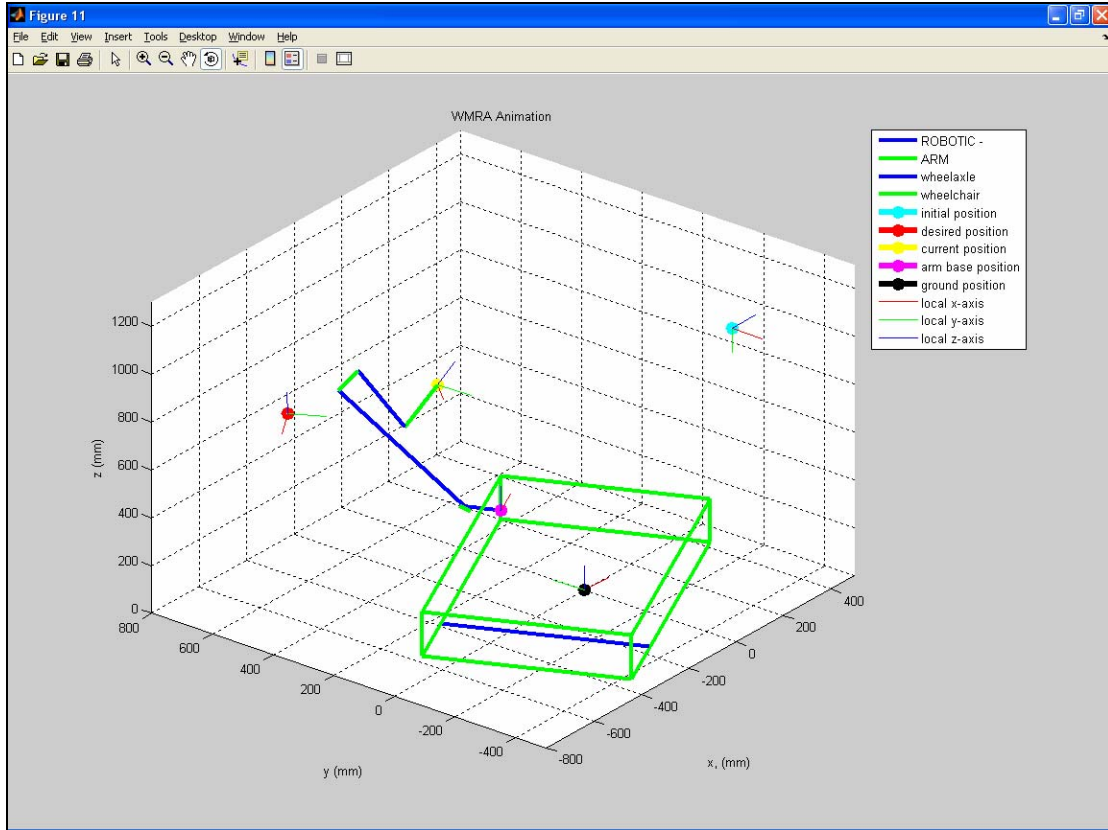


Figure 7.3: Simulation Window of the WMRA System in Wire Frame.

7.4.3. Simulation with Virtual Reality Toolbox

The same simulation discussed in the previous sub-section was also programmed and simulated using Virtual Reality simulation. SolidWorks models of each one of the link segments of the robotic arm were drawn, as well as the wheelchair model and the two driving wheels separately. All drawn models are then converted into WMRL files that use WMRL language. A new VRML program was created to call each individual segment of the WMRA system in a hierarchy, and relate them together using variable positions of the joint space variables. In that environment, enhancements were made to make the background and the floor look realistic in simulation. The new VRML file

created was then called during Matlab simulation and updated with the new joint space variables so that the view of the WMRA change as the simulation progresses.

Different view points were created to view the system in Virtual Reality. Unlike workstation robots, this WMRA is not stationary, and it eventually gets out of the simulation window if the wheelchair is driven too far. For this reason, several dynamic views are also developed to follow the wheelchair as it moves so that it stays within the viewing area of the window. These views can be changed during the simulation, and snapshots or videos can be recorded. Figure 7.4 shows a static view of the Virtual Reality program window that shows the WMRA in the ready position.



Figure 7.4: A Sample of the Virtual Reality Simulation Window.

The Virtual Reality model used for simulation was tested using several user interfaces, including the SpaceBall, the keyboard and mouse, the brain-computer interface (BCI2000) and the touch screen interface. The program performed in a satisfactory way with precise and fast simulations with no noticeable delays.

7.4.4. Graphical User Interface (GUI) Program

Using the main program in Matlab to control the robotic device was hard for a user with disabilities to accomplish because of the initial questions asked by the program to bring the control up to the user preference. A new GUI program was created to ease this process and make it practical and user friendly for persons with disabilities. The main program was integrated with a GUI with default values so that the user can store the default values in the main program and use it directly as the software opens. This feature dramatically reduces the burden on the user to fill out the initial options every time he/she wants to use the WMRA system. Figure 7.5 shows the graphical user interface with its default options. To make it even easier and less confusing to the user, different windows or buttons will disappear if they don't apply to the user's selected option or when next options do not apply to the currently chosen mode. Since the tablet PC is equipped with a touch screen, the user can easily tap the selections. When a touch-screen user interface control is selected, another screen appears with the functions and directions that the user can choose appear as shown in figure 6.8. This screen accepts commands by touching the intended button, or by pressing the button by mouse or the equipped touch pad.

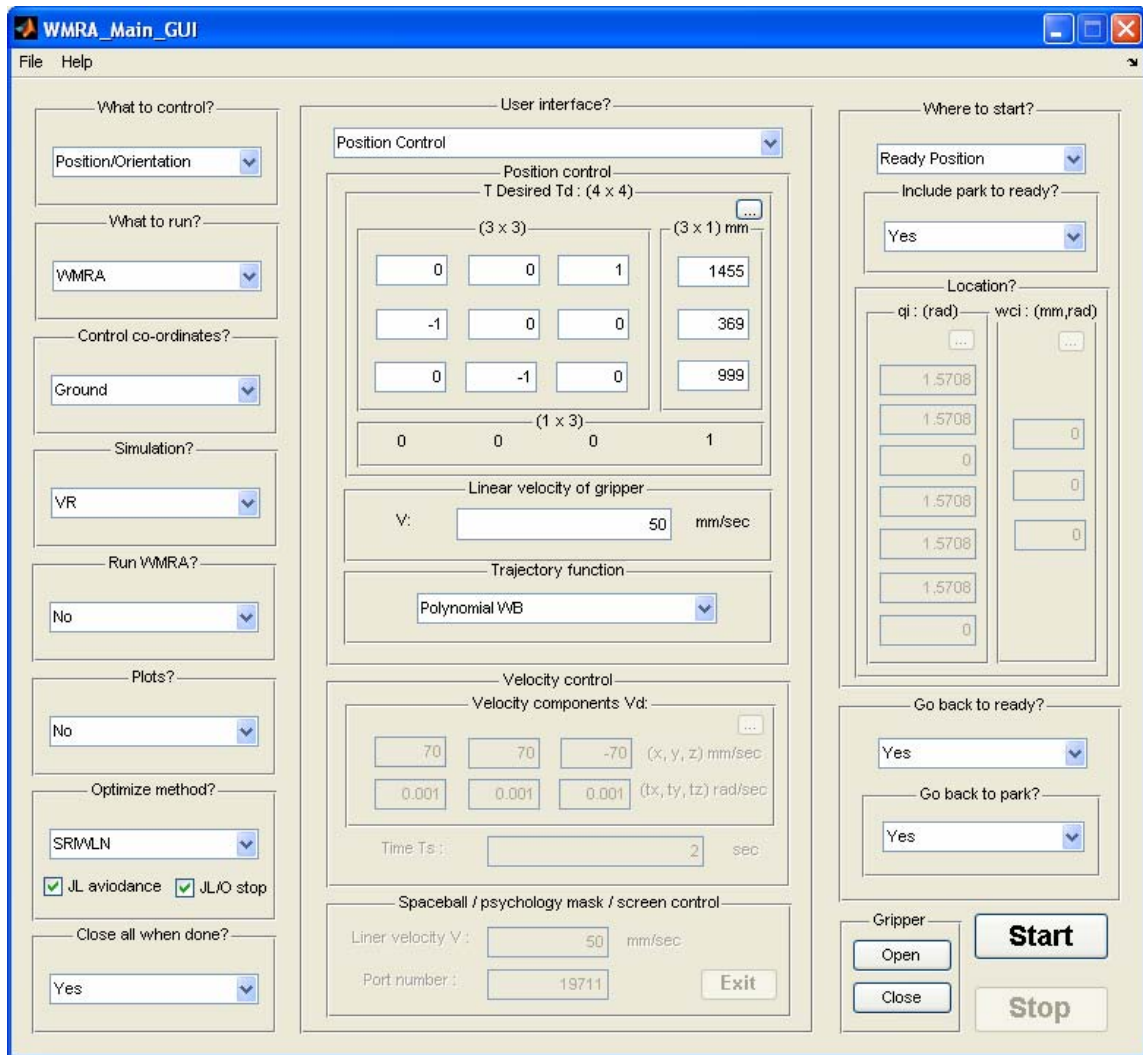


Figure 7.5: The Graphical User Interface (GUI) Screen with the Defaults.

7.5. Comments on Interfacing Different Programs Together

When this program was created, we knew that communication problems would occur between software and hardware or software and software. The first problem was integrating the SpaceBall and interfacing it with Matlab. DLL libraries that are written in C++ are possible to read and use the functions they contain, but the problem comes when these functions use different data structure than Matlab while compiling. This means that

the functions are either unusable or very hard to use. In the case of SpaceBall, a new C++ program was created to send the data to a Matlab environment and make it ready for use.

Another problem came when we were going to use the program to operate the actual WMRA system. Since it uses functions from complex DLL libraries, we had to re-create functions in C++ and compile them into DLL files in a data structure that is compatible with C++, and then use them in Matlab and call these functions to communicate with the PIC Servo SC controller boards used in controlling the WMRA. This works out well, except that some times the virtual link between Matlab and the DLL library fails, and that results in unresponsive WMRA when commanded to do a task. This problem can be taken care of if the program controlling the arm is separated from the program that simulates the arm. This way, the program that controls the arm can be rewritten in C++ so that less interfacing problems will appear.

The BCI 2000 user interface also uses a C++ program for processing and sending the data out. In this case, a networked TCP/IP port was dedicated to communicate between the BCI2000 and the computer that is running the control algorithm, and Matlab was interfacing with the TCP/IP port to get the data and use it in the control software.

7.6. Summary

In this chapter, a description of the simulation software was presented and discussed. Different programming languages and packages were used to create different applications and interface them together. The main program was written in Matlab, and two different graphic simulation were used. Wire frame graphical simulation of WMRA was created for precise inspection of the simulation and its results, and Virtual reality

simulation was created for its realistic look and appearance. Several plots can be shown to describe the system behavior during the simulation period.

The main program can be run in two different ways, one is through the common command line of Matlab, and the other through a graphical user interface (GUI). The GUI was more user-friendly and easier for use by people with disabilities. Several communication and interfacing problems were faced during programming different parts of the WMRA system together with the control software. The solutions to these problems were presented.

Chapter 8:

Simulation Results

8.1. Introduction

Simulation of many different cases to test the theory developed in chapters 3, 4, and 5 is important to validate the control algorithm and the methods used for control, especially if these algorithms are going to be used to control the actual WMRA system built at USF. In this chapter, simulation of these cases will be shown, and the effects of different control schemes and values will be discussed. Many plots of Cartesian space variables and joint space variables will be shown in positions, velocities and accelerations of these variables throughout the simulation period. The effectiveness of the singularity avoidance schemes will be shown by plotting the manipulability measure of the robotic arm and the combined WMRA system. The control system of the 9-DoF WMRA system is implemented in simulation using Matlab 7.0.4 with Virtual Reality toolbox installed on a PC running Windows XP.

8.2. Simulation Cases Tested

Several cases were tested in this simulation using the Weighted Least Norm solution control with Singularity-Robust inverse of the Jacobian since this was the most effective way of controlling the WMRA system. Five different values were tried for the

diagonal elements of the weight matrix (W) to implement the control system and to verify its effectiveness. These values were expressed in the following five cases:

- 1- Case I: The weight matrix of the first case carried in its diagonal elements the same value “1” for all 9 variables. That means that all seven joints of the arm and the two wheelchair position and orientation variables will have equal potential of motion.
- 2- Case II: In the second case, “ W ” carried “10” for each of the arm’s seven joints, and “1” for wheelchair’s position and orientation variables, which means that the wheelchair’s two variables are 10 times more likely to move than the arm’s joints.
- 3- Case III: The third case carried weights of “1” for the arm’s joints, and “100” for the wheelchair’s two variables in “ W ”, which means that the arm is 100 times more likely to move than the wheelchair.
- 4- Case IV: In the fourth case, “ W ” carried weights of “1” for the arm’s seven joints and the wheelchair’s orientation variable, and a weight of “100” for the wheelchair’s position. This means that the forward or backward motion of the wheelchair is 100 times less likely than the motion of the rest of the system
- 5- Case V: The last case was the opposite of the fourth case, where the orientation of the wheelchair took a weight of “100”, and the other eight variables took a weight of “1”. This means that the wheelchair’s rotational motion is 100 times less likely to occur than the motion in the arm’s joints and the wheelchair’s translational motion.

To show the effect of choosing the state variables of the wheelchair's non-holonomic motion in the planar Cartesian coordinates as the linear position and angular orientation rather than the two wheelchair wheel angles, two other cases were added for comparison of the WMRA system's behaviour when either method was used as follows:

- 1- Case A: When the state variables representing the wheelchair's motion were selected as the two angles of the wheelchair's driving wheels.
- 2- Case B: When the state variables representing the wheelchair's motion were selected as the linear forward motion and the angular motion of the wheelchair in the planar Cartesian space.

Each one of these individual cases will be discussed, and the results will be shown to express the difference between these cases and the effectiveness of the methods and variables chosen.

8.3. Results and Discussion of the First Five Cases

The first five cases dealing with different weight values in the weight matrix "W" will be discussed in this section. The simulation was tested by commanding the WMRA system to move the gripper's frame from its ready position defined by the following homogeneous transformation matrix based on the ground frame:

$$T_r = \begin{bmatrix} 0 & 0 & 1 & 455 \\ -1 & 0 & 0 & -131 \\ 0 & -1 & 0 & 899 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (8.1)$$

Moving the arm from its ready position defined above to the desired position defined by the following homogeneous transformation matrix based on the ground frame:

$$T_d = \begin{bmatrix} 1 & 0 & 0 & 455 \\ 0 & 0 & 1 & 970 \\ 0 & -1 & 0 & 550 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (8.2)$$

Figure 8.1 shows the initial pose of the WMRA system at the beginning of the simulation when it was at the ready position. The end-effector's position and orientation on the Cartesian space were the same in all trials since the trajectory was the same for all five cases tested. Figure 8.2 shows the end-effector's position and figure 8.2 shows the end-effector's orientation during simulation as it moves from the initial pose to the commanded point in the workspace. The motions of the individual variables in the joint space were completely different for each one of the cases depending on the selected weight for each variable so that we can get the desired behaviour of the WMRA system.



Figure 8.1: The Initial Pose of the WMRA in Simulation.

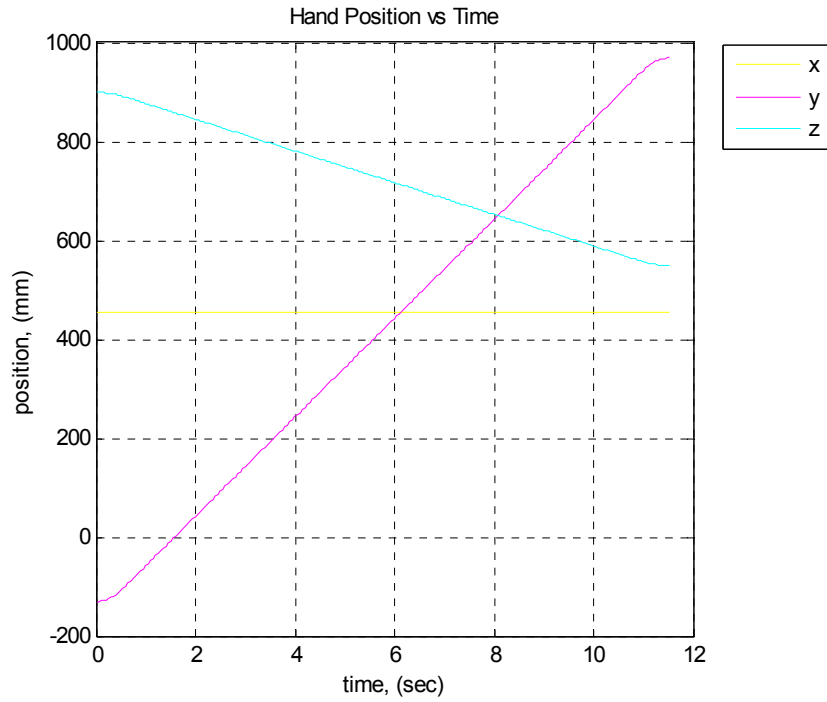


Figure 8.2: Position of the WMRA During Simulation.

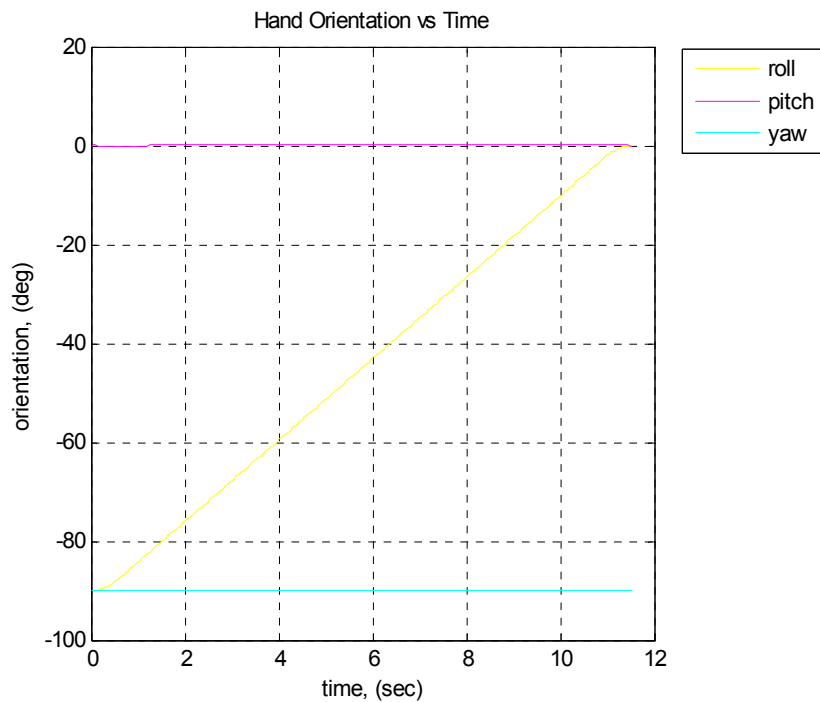


Figure 8.3: Orientation of the WMRA During Simulation.

8.3.1. WMRA Configurations in the Final Pose of the Simulation

During simulation, each case behaved differently in terms of solved values of the joint space variables. Figures 8.4, 8.5, 8.6, 8.7 and 8.8 show the final poses of the WMRA system after the end-effector reached the desired destination for the five cases studied. Observing the figures, it was apparent from the first case compared to the others that all seven joints of the arm and the two wheelchair's position and orientation variables had equal potential of motion as shown in figure 8.4. In the second case, the wheelchair's two variables were 10 times more likely to move than the arm's joints, and that is apparent in the results shown in figure 8.5. In the third case, the arm was 100 times more likely to move than the wheelchair, and that can be clearly seen in figure 8.6, where the wheelchair had a minimal motion and the arm did most of the motion.

The beauty of this simulation comes apparent in the last two cases, where in the fourth case, the forward or backward motion of the wheelchair was 100 times less likely than the motion of the rest of the system, and figure 8.7 shows how the wheelchair's forward motion was minimal. Figure 8.8 shows the last case, which is the opposite of the fourth case, where the wheelchair's rotational motion was 100 times less likely to occur than the motion in the arm's joints and the wheelchair's translational motion.

These poses clearly show the property of combining the wheelchair's motion and the robotic arm's motion under the optimization and redundancy resolution schemes discussed in earlier chapters. It was also observed from running other tasks that took the WMRA system out of its reach in the vertical direction that this method was stabilized by ignoring some of the trajectory's orientation or position errors as needed so that the system doesn't go out of control by producing high velocities in the joint domain.



Figure 8.4: Destination Pose for Case I, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 1]$.



Figure 8.5: Destination Pose Case II, When $W = [10, 10, 10, 10, 10, 10, 10, 1, 1]$.

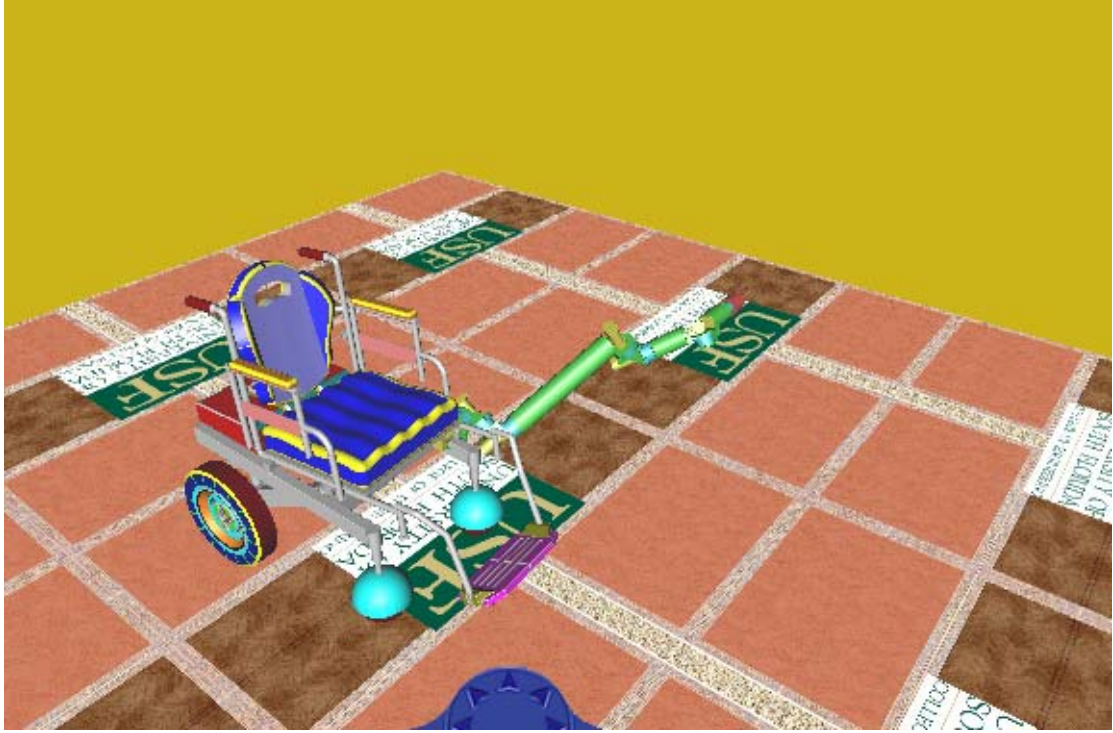


Figure 8.6: Destination Pose Case III, When $W = [1, 1, 1, 1, 1, 1, 1, 100, 100]$.



Figure 8.7: Destination Pose Case IV, When $W = [1, 1, 1, 1, 1, 1, 1, 100, 1]$.



Figure 8.8: Destination Pose Case V, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 100]$.

8.3.2. Displacements of the Joint Space Variables

The simulation program was designed to give different useful values and plots throughout the simulation process for observation and diagnosis of any potential problems that might occur during the task execution whether the physical arm is running or if it is just the simulation. Among these plots are the joints' angular displacements and velocities. Figures 8.9 through 8.13 show the angular displacement versus time for the arm's seven joints throughout the simulation period for all five cases. The first case in figure 8.9 shows the normal weights with no preference to any of the nine variables. In the second case shown in figure 8.10, when the arm was assigned large weight in the weight matrix, it was clear that the seven arm joints had minimal motion that was necessary for the destination to be reached. That end-effector destination was impossible

to reach by using the two wheelchair variables only. The last three cases shown in figures 8.11, 8.12 and 8.13 show an easy arm motion as compared to that of the wheelchair.

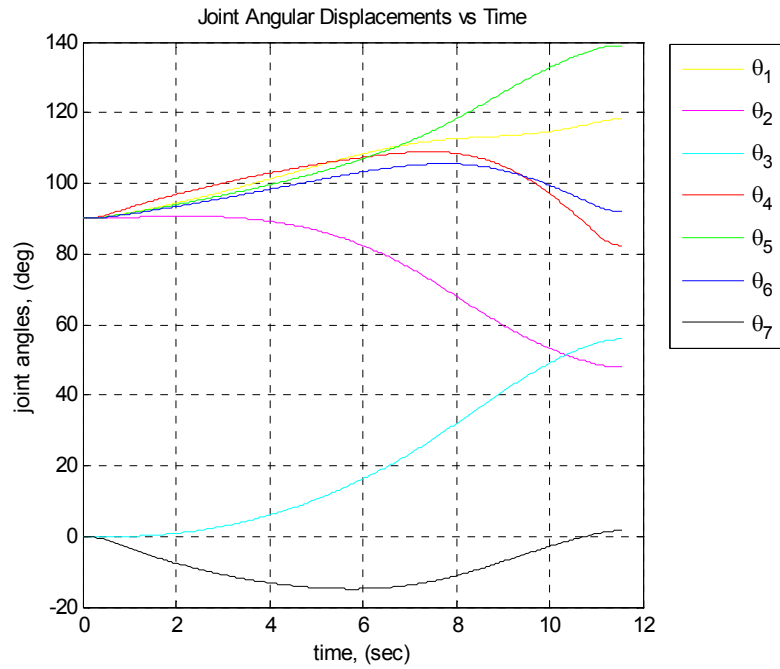


Figure 8.9: Arms' Joint Motion for Case I, When $W = [1, 1, 1, 1, 1, 1, 1, 1]$.

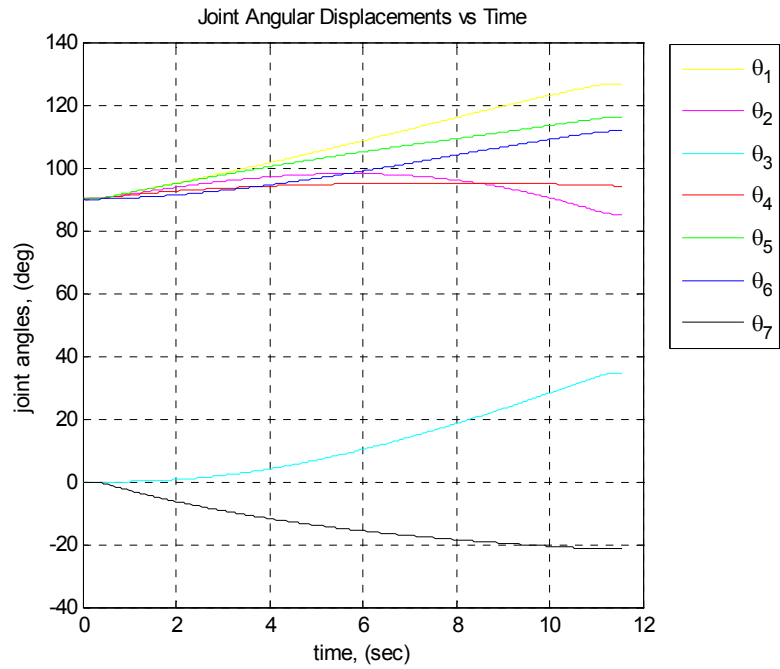


Figure 8.10: Arms' Joint Motion for Case II, When $W = [10, 10, 10, 10, 10, 10, 1, 1]$.

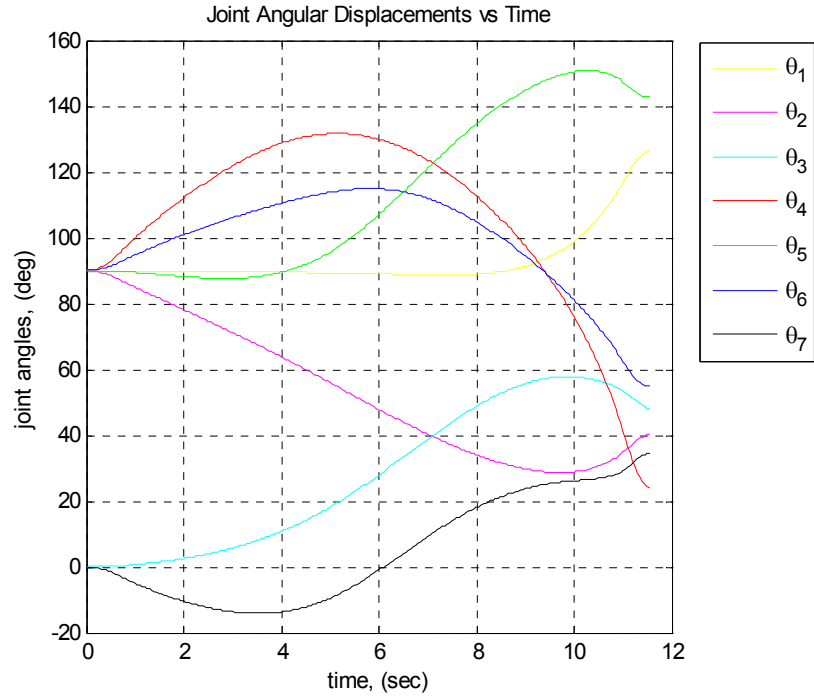


Figure 8.11: Arms' Joint Motion for Case III, When $W = [1, 1, 1, 1, 1, 1, 1, 100, 100]$.

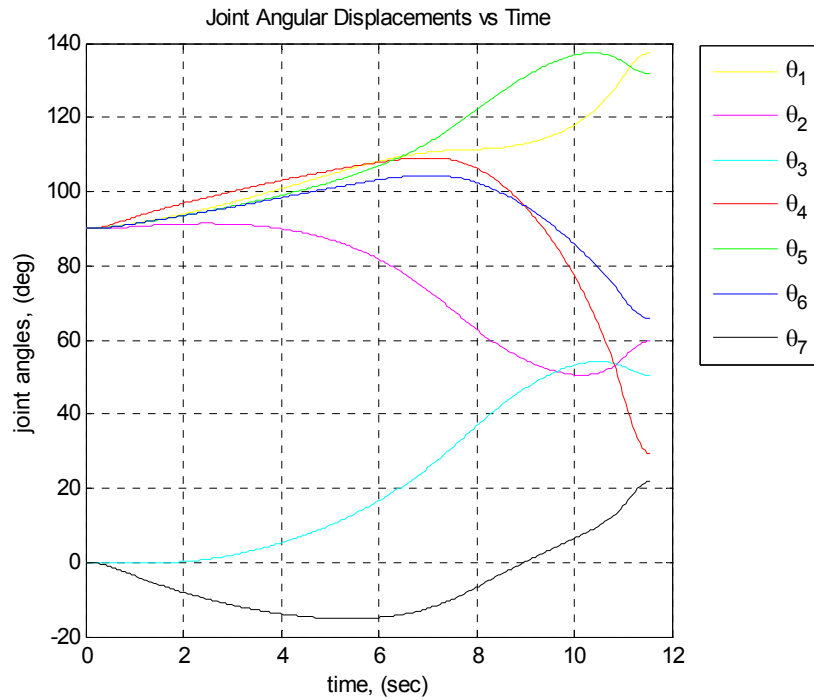


Figure 8.12: Arms' Joint Motion for Case IV, When $W = [1, 1, 1, 1, 1, 1, 1, 100, 1]$.

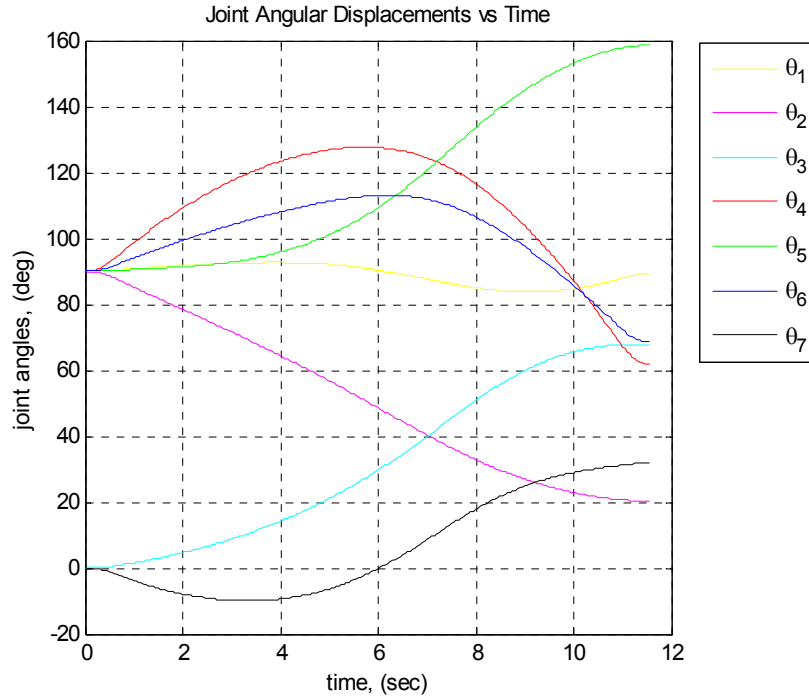


Figure 8.13: Arms' Joint Motion for Case V, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 100]$.

Another plot that was given in the simulation program was the track distances drawn by each of the two wheels of the wheelchair. These plots were useful in particular to observe the wheelchair's motion. Figures 8.14 through 8.18 show these distances driven through the simulation for all five cases. An important property of this optimization method was apparent during simulation, and can be seen in figure 8.14, which was minimization of singularity. As the arm was moving to the destination and the left wheel was moving backwards, it reversed its motion in the middle of the simulation period when the arm approached singularity as seen in figure 8.21. The maximum wheelchair motion occurred in the second case as shown in figure 8.15, where the higher weight was assigned to the arm, and the wheelchair was free to move. Figure 8.16 shows the opposite, where the wheelchair moved the least among all cases since the weight was assigned to the wheelchair's motion and the arm did most of the motion.

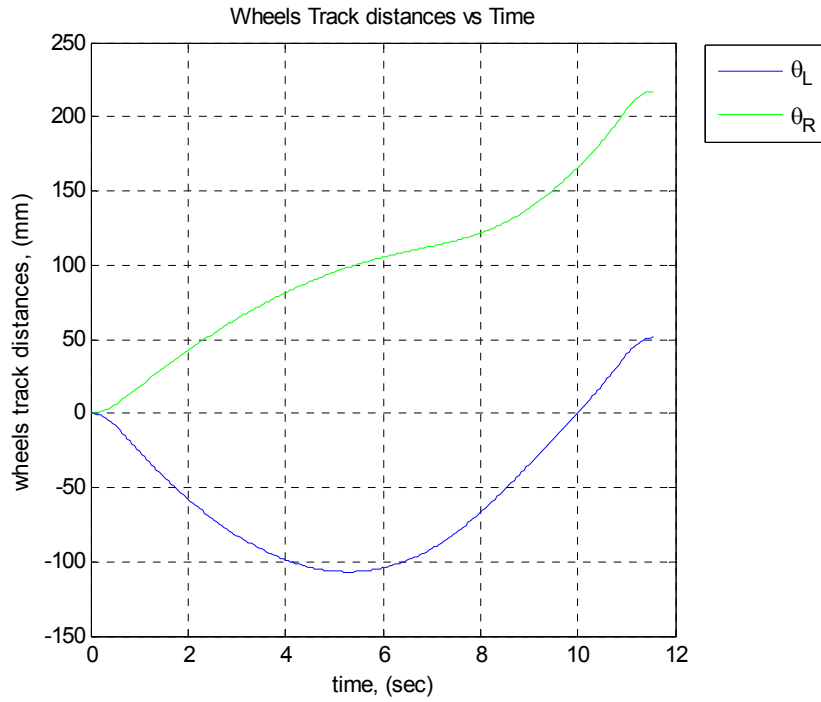


Figure 8.14: Wheels' Motion for Case I, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$.

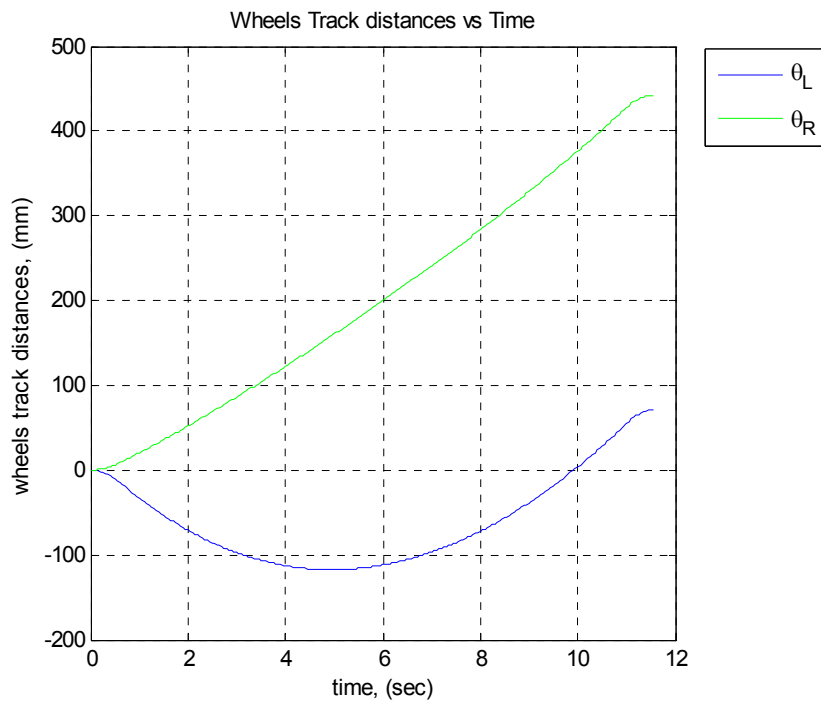


Figure 8.15: Wheels' Motion for Case II, When $W = [10, 10, 10, 10, 10, 10, 10, 10, 1, 1]$.

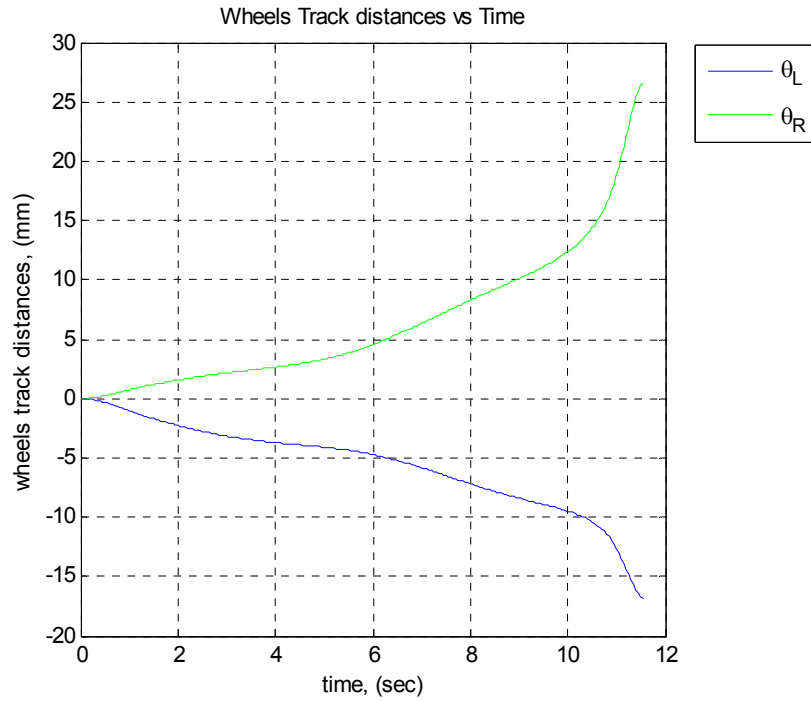


Figure 8.16: Wheels' Motion for Case III, When $W = [1, 1, 1, 1, 1, 1, 1, 100, 100]$.

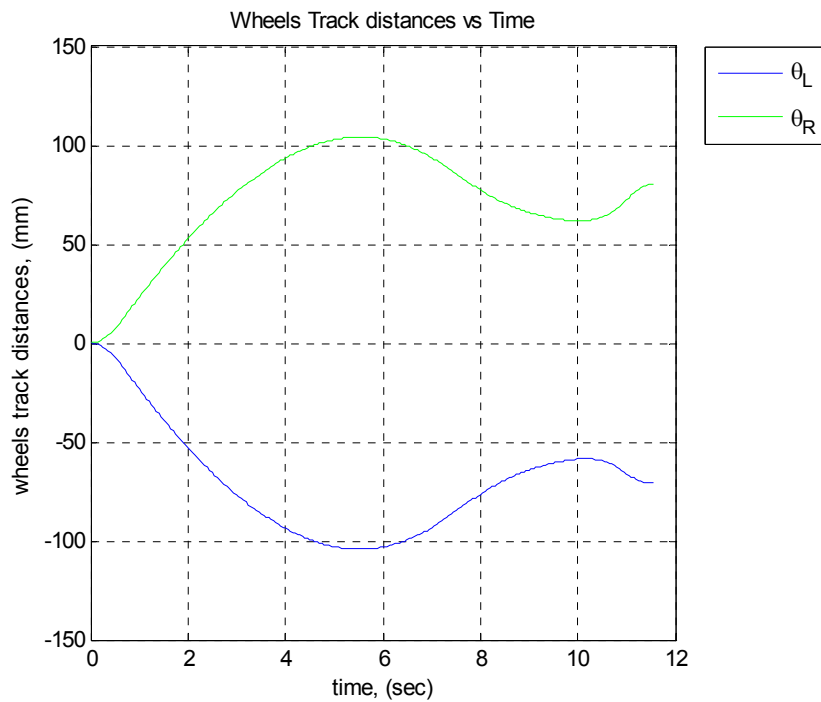


Figure 8.17: Wheels' Motion for Case IV, When $W = [1, 1, 1, 1, 1, 1, 1, 100, 1]$.

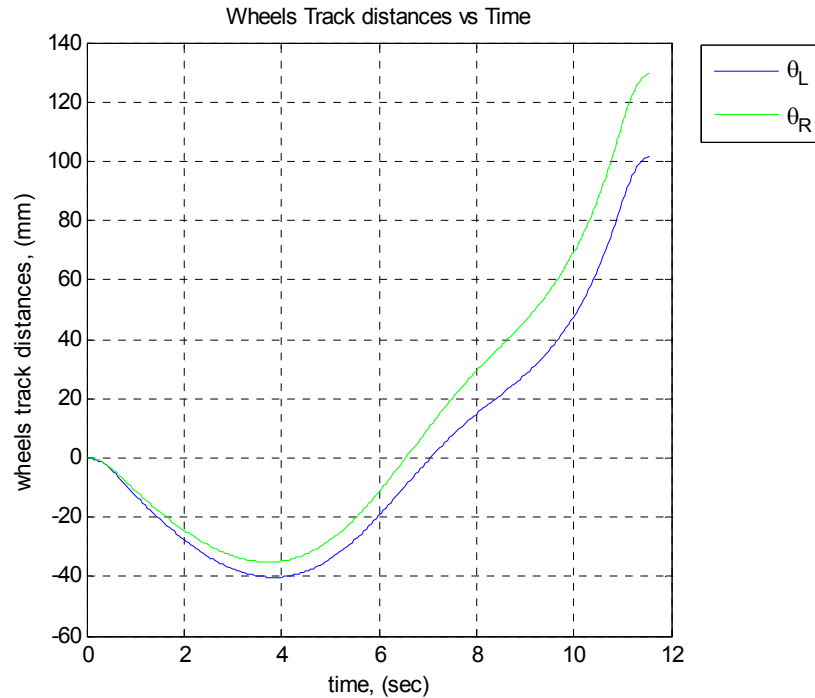


Figure 8.18: Wheels' Motion for Case V, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 100]$.

Observing figures 8.17 and 8.18 shows how the opposite weights carried by the position and orientation variables of the wheelchair in these two cases led to a rotation as observed in figure 8.17, where both wheels carried the same but opposite motion, and a translation as observed in figure 8.18, where both wheels carried the same motion.

8.3.3. Velocities of the Joint Space Variables

The velocity profiles of the five cases were observed, but the beauty of the trajectory generator was apparent. Figures 8.19 and 8.20 show the velocity profiles of the seven arm joints and the two wheelchair wheels respectively for case I. When using a 3rd order polynomial with parabolic blending, velocities ramped up or down at a constant acceleration rather than going from zero to the desired joint velocities in no time. This option was used in all simulation cases.

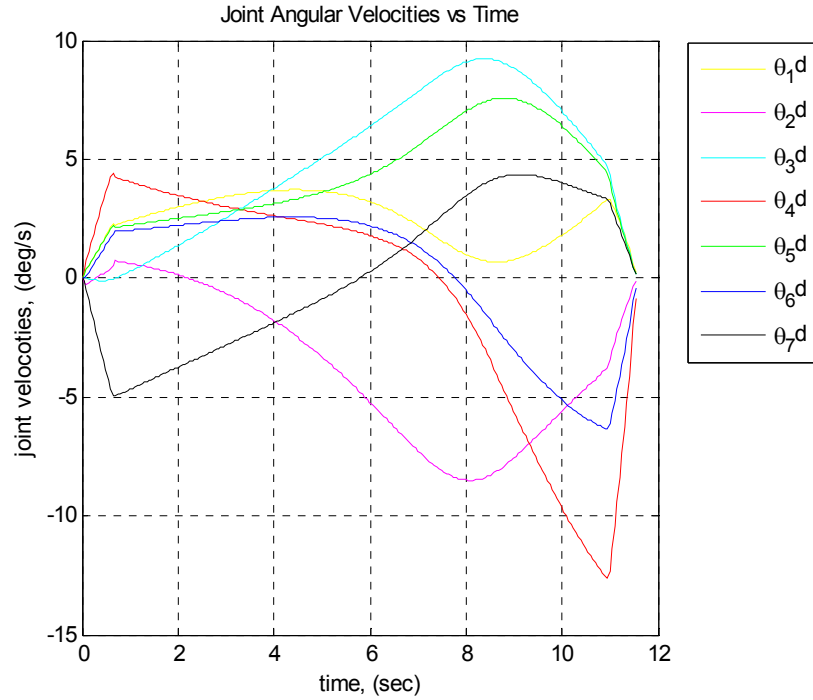


Figure 8.19: Arms' Joint Velocities for Case I, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 1]$.

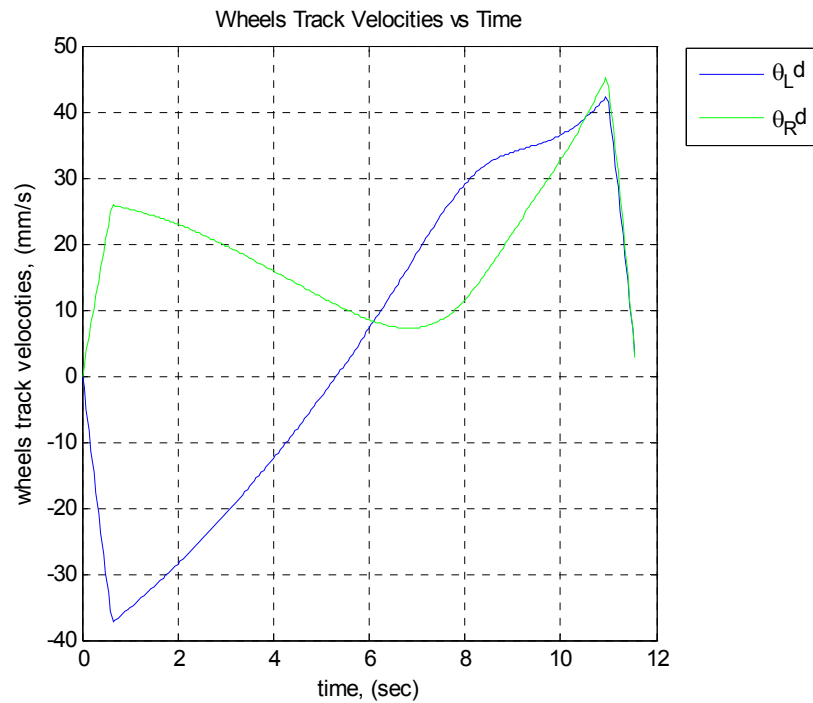


Figure 8.20: Wheels' Velocities for Case I, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 1]$.

8.3.4. Singularities and the Manipulability Measure

Figures 8.21 through 8.25 show the manipulability index of both arm only and the combined WMRA system. It is important to note here that these values were multiplied by (10^{-9}) to get the normalized manipulability measure. It is clear that the manipulability is much higher for the WMRA system than that of the arm only due to the fact that the WMRA system carries two more degrees of freedom. In all five cases, the manipulability measure was maximized based on the weight matrix. Figure 8.22 shows the manipulability of the arm as nearly constant because of the minimal motion of the arm. Figure 8.23 shows how the wheelchair started moving rapidly later in the simulation (see figure 8.16) as the arm approached singularity, even though the weight of the wheelchair motion was heavy. This helped in improving the WMRA system's manipulability.

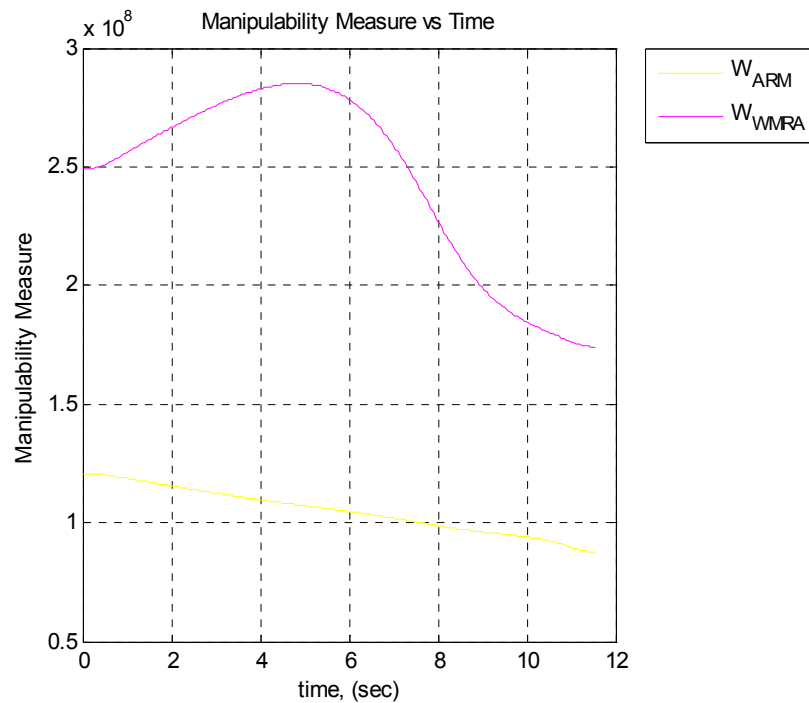


Figure 8.21: Manipulability Index for Case I, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$.

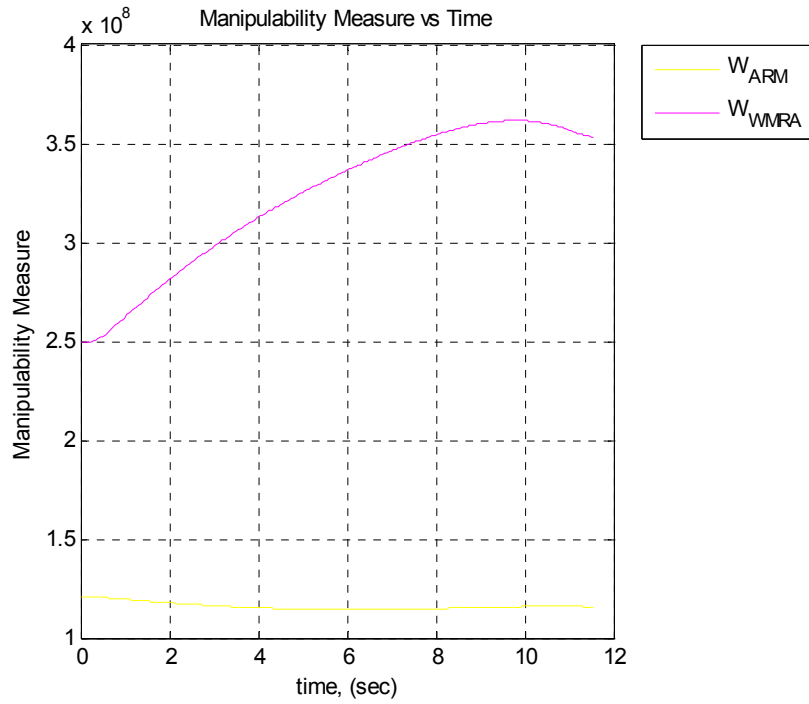


Figure 8.22: Manipulability Index for Case II, When $W = [10, 10, 10, 10, 10, 10, 10, 1, 1]$.

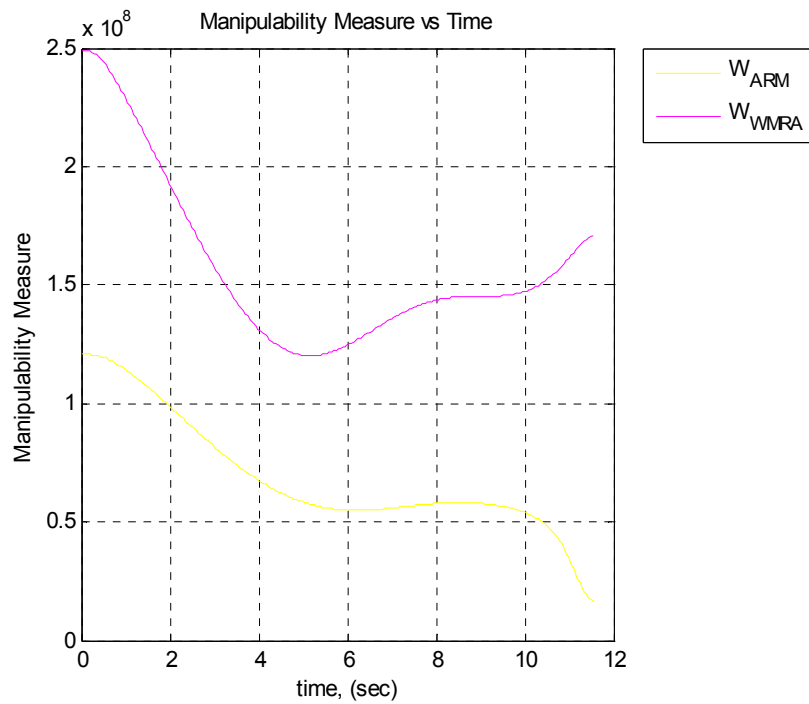


Figure 8.23: Manipulability Index for Case III, When $W = [1, 1, 1, 1, 1, 1, 1, 100, 100]$.

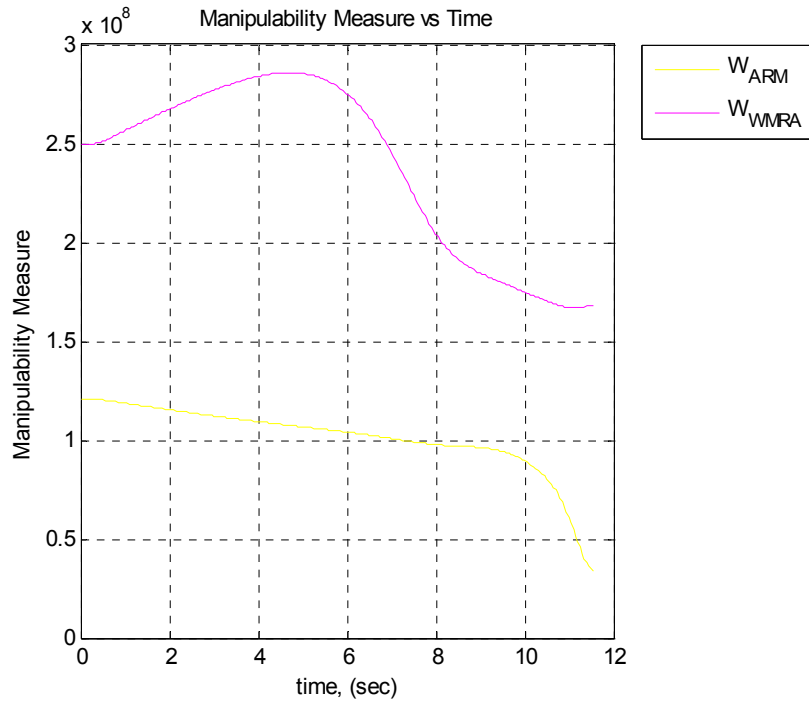


Figure 8.24: Manipulability Index for Case IV, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 100, 1]$.

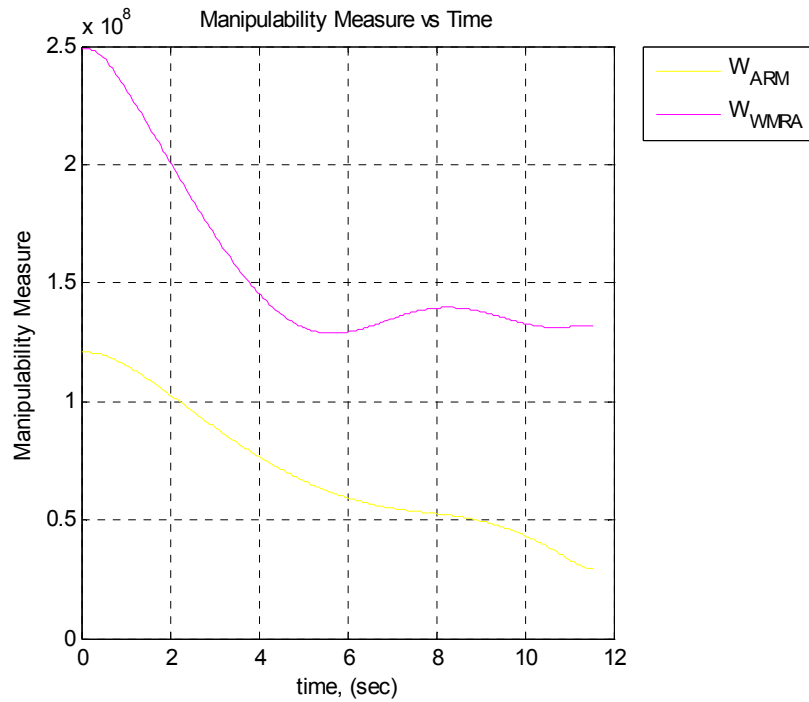


Figure 8.25: Manipulability Index for Case V, When $W = [1, 1, 1, 1, 1, 1, 1, 1, 1, 100]$.

It is important to mention that changing the weights of each of the state variables gives motion priority to these variables, but may lead to singularity if heavy weights are given to certain variables when they are necessary for particular motions. For example, when all the seven joints of the arm were given a weight of “1000” and the task required rapid motion of the arm, singularity occurred since the joints were nearly stationary. Changing these weights dynamically in the control loop depending on the task in hand leads to a better performance.

8.4. Results and Discussion of the Second Two Cases

The two other cases tested in simulation were done to show the effect of choosing the state variables of the wheelchair’s non-holonomic motion in the planar Cartesian coordinates as the linear position and angular orientation rather than the two wheelchairs wheel angles. In the first case (A), the state variables representing the wheelchair’s motion were selected as the two angles of the wheelchair’s driving wheels. In the second case (B), the state variables representing the wheelchair’s motion were selected as the linear forward motion and the angular motion of the wheelchair in the planar Cartesian space. In this simulation test, the WMRA system was commanded to move the gripper forward on a straight line along the global “X” direction for one meter (1000 mm), i.e., it was moved from the ready position shown in equation 8.1 to the following desired position:

$$T_d = \begin{bmatrix} 0 & 0 & 1 & 1455 \\ -1 & 0 & 0 & -131 \\ 0 & -1 & 0 & 899 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (8.3)$$

The natural response that the operator would expect is to move the wheelchair forward without turning since the trajectory is in a straight line in front of the wheelchair. What actually happened in the first case (A) was different. First, when the weights of all joint variables were the same, the response was the same in both cases since it didn't make a difference what the state variables were if you assigned the same weights to all variables. Figure 8.26 shows the position of the robotic arm's base that is mounted on the power wheelchair. Observe that the arm had to move about 650 mm forward and 400 mm to the side of the wheelchair. Also, figure 8.27 shows the orientation of the robotic arm's base that is mounted on the power wheelchair. Observe that the arm had to turn unnecessarily about 28 degrees clockwise, and then turn again about 9 degrees counter clockwise. This unnecessary motion can be avoided using the weight matrix only if the state variables are selected in the proper way to be controlled.

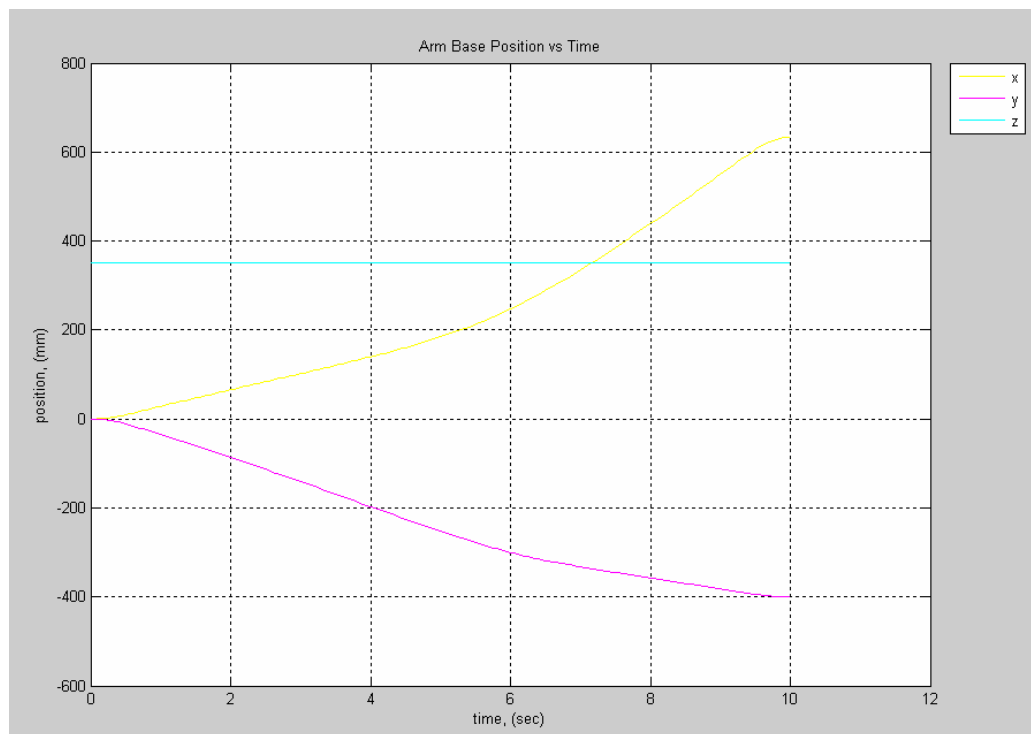


Figure 8.26: Arm Base Position When the Weights Were Equal, $W = [1, 1, 1, 1, 1, 1, 1, 1, 1]$.

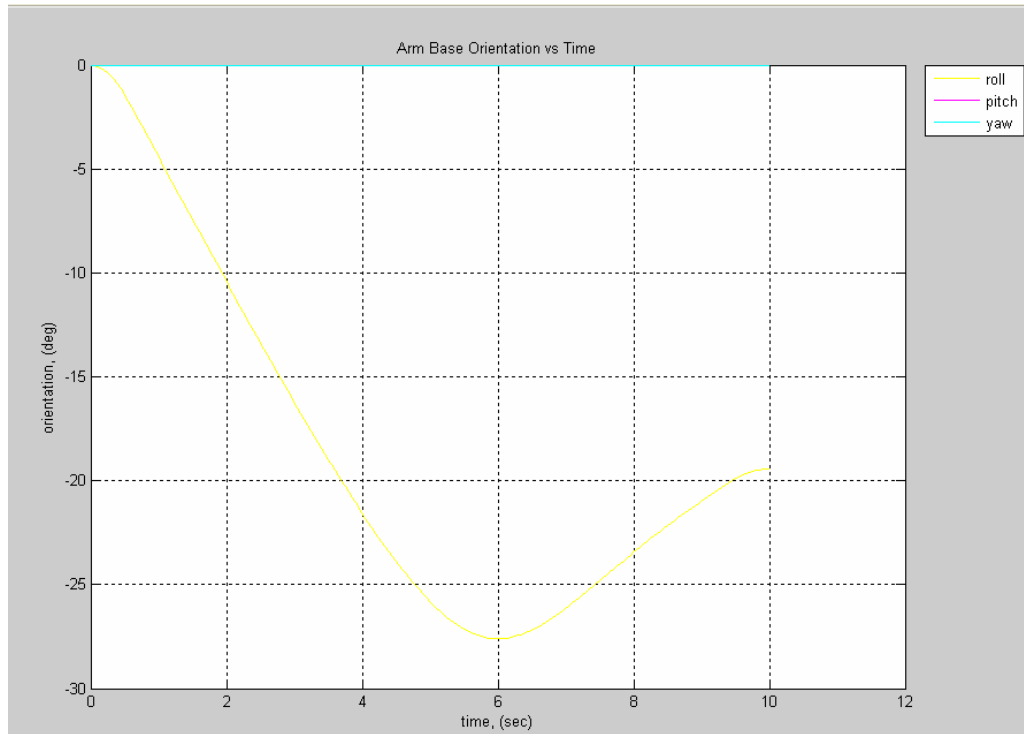


Figure 8.27: Arm Base Orientation When the Weights Were Equal, $W = [1, 1, 1, 1, 1, 1, 1, 1, 1]$.

In case (A), the state variables were the two wheels of the wheelchair, and the only way to control these two variables were by assigning heavy weights on both of them so that the wheelchair doesn't move unnecessarily. The weights assigned were "50" to both wheels, and "1" to the seven robotic arm joints. Figure 8.28 shows the position of the robotic arm's base that is mounted on the power wheelchair. In this case, the arm had to move about 625 mm forward and 250 mm to the side of the wheelchair. Even though the side motion was not necessary, but it was less than the motion when the weights were equal. Also, figure 8.29 shows the orientation of the robotic arm's base. Observe that the arm had to turn unnecessarily about 16 degrees clockwise, and then turn again about 2 degrees counter clockwise. Even though this unnecessary rotation happened, it was still less than that motion when the weights were equal.

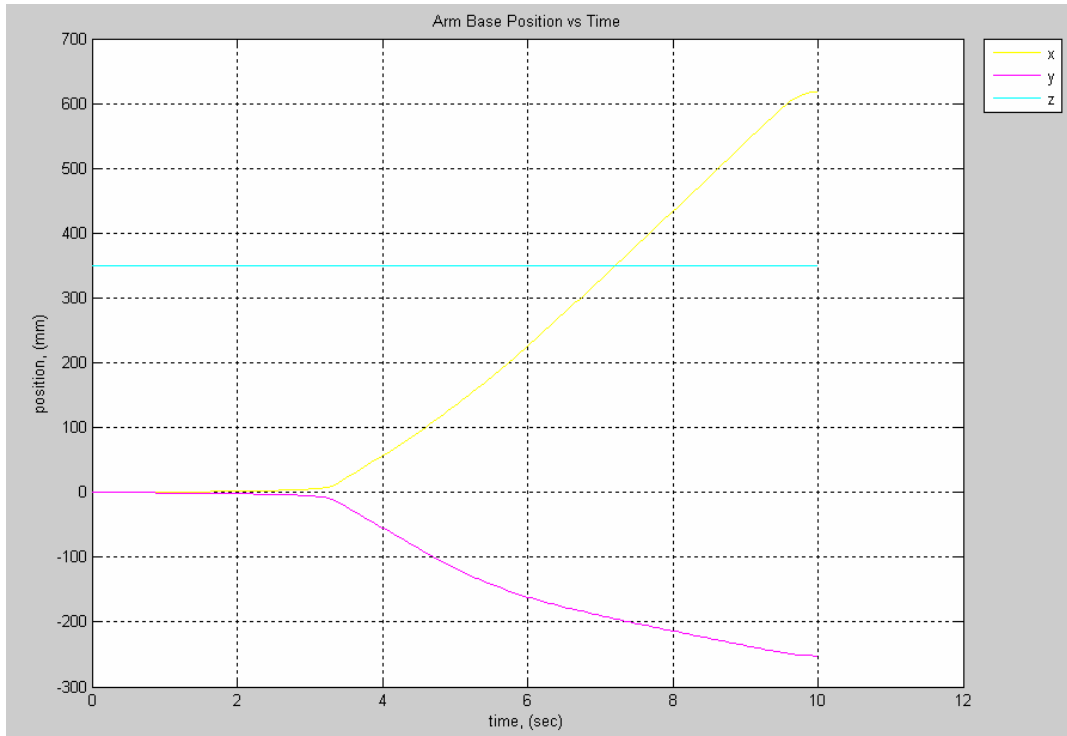


Figure 8.28: Arm Base Position for Case A, When $W = [1, 1, 1, 1, 1, 1, 1, 50, 50]$.

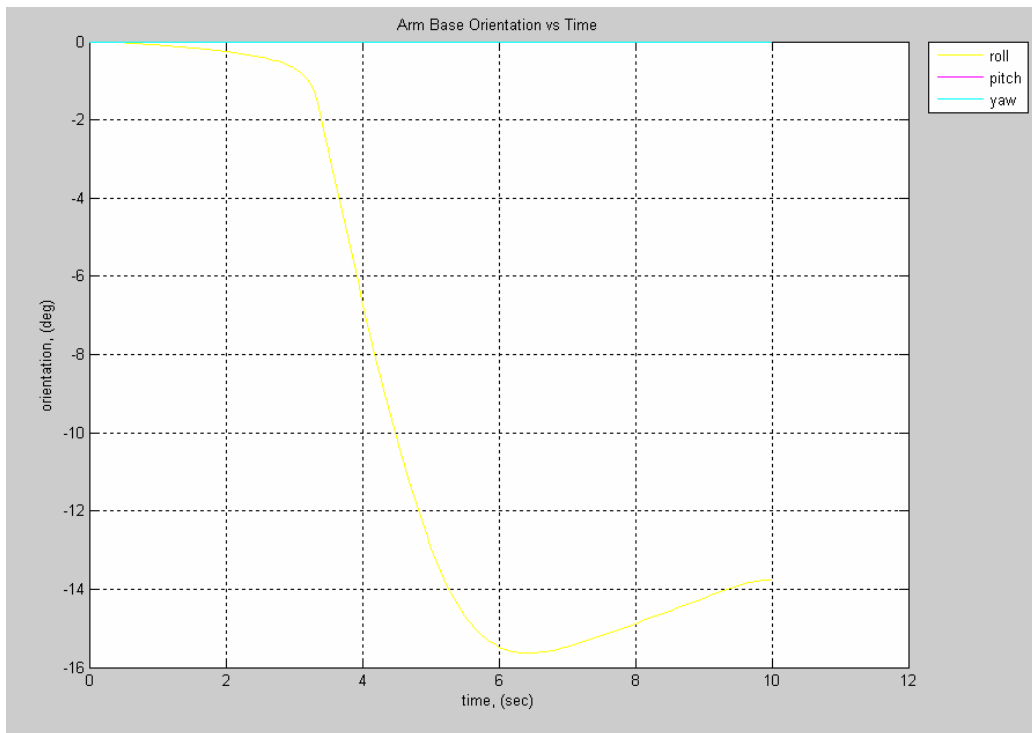


Figure 8.29: Arm Base Orientation for Case A, When $W = [1, 1, 1, 1, 1, 1, 1, 50, 50]$.

Having control over the forward motion and the orientation of the wheelchair separately allows greater and more meaningful behavior of the system response. In case (B), the state variables were the wheelchair's linear motion and its rotational orientation, and these two variables can be controlled separately to give preference to the rotation or the forward motion separately by assigning heavier weights on the variable that should not change unnecessarily. The weights assigned in this case were "50" to the wheelchair's rotational motion, and "1" to the wheelchair's forward motion and the seven robotic arm joints. Figure 8.30 shows the position of the robotic arm's base that is mounted on the power wheelchair. In this case, the arm had to move about 700 mm forward and 100 mm to the side of the wheelchair. Even though the side motion was not necessary, it was significantly less than that for case (A). The wheelchair moved more forward to compensate for the unwanted side motion. Also, figure 8.31 shows the orientation of the robotic arm's base. The orientation change was minimal, and it was less than 8 degrees clockwise. This turn was less than half of that in case (A) since heavier load was given to the orientation rather than all wheelchair motion. Notice that the orientation change not only was minimal, but it didn't change direction to go counter clockwise as what happened in case (A). This shows that the apparently unnecessary rotation that happened was in fact necessary to follow the trajectory without getting close to singular configurations.

The observations of these cases emphasize the importance of choosing the variables based on the convenience of the user. The adapted variables for WMRA control in the actual arm were the forward position and the rotational orientation of the WMRA.

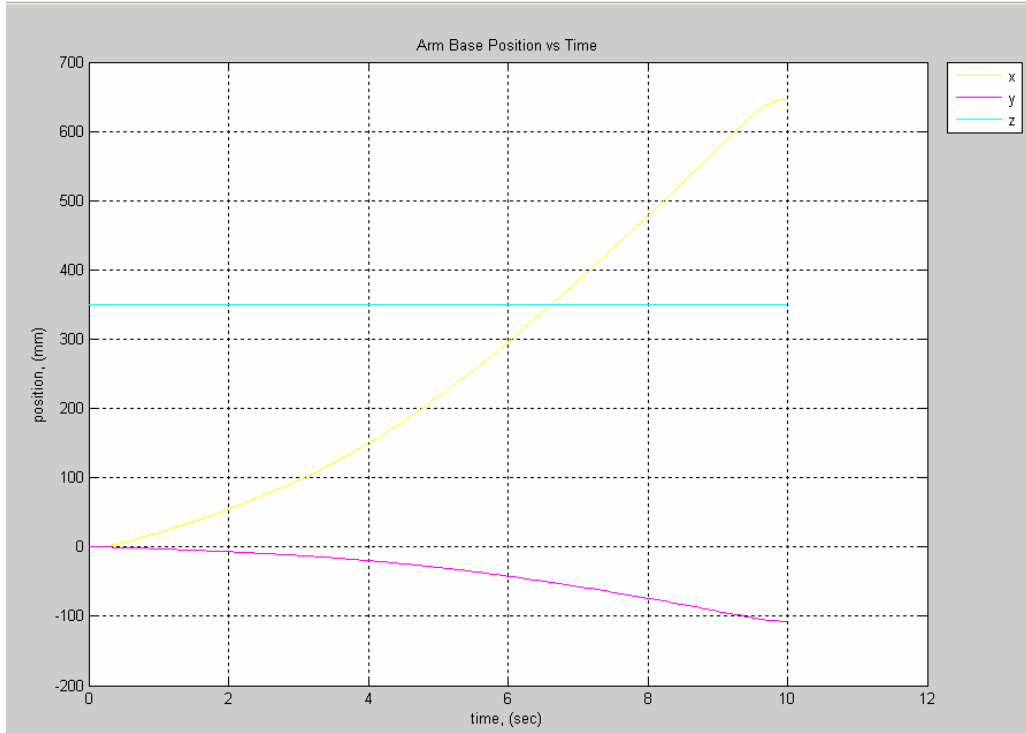


Figure 8.30: Arm Base Position for Case B, When $W = [1, 1, 1, 1, 1, 1, 1, 50, 1]$.

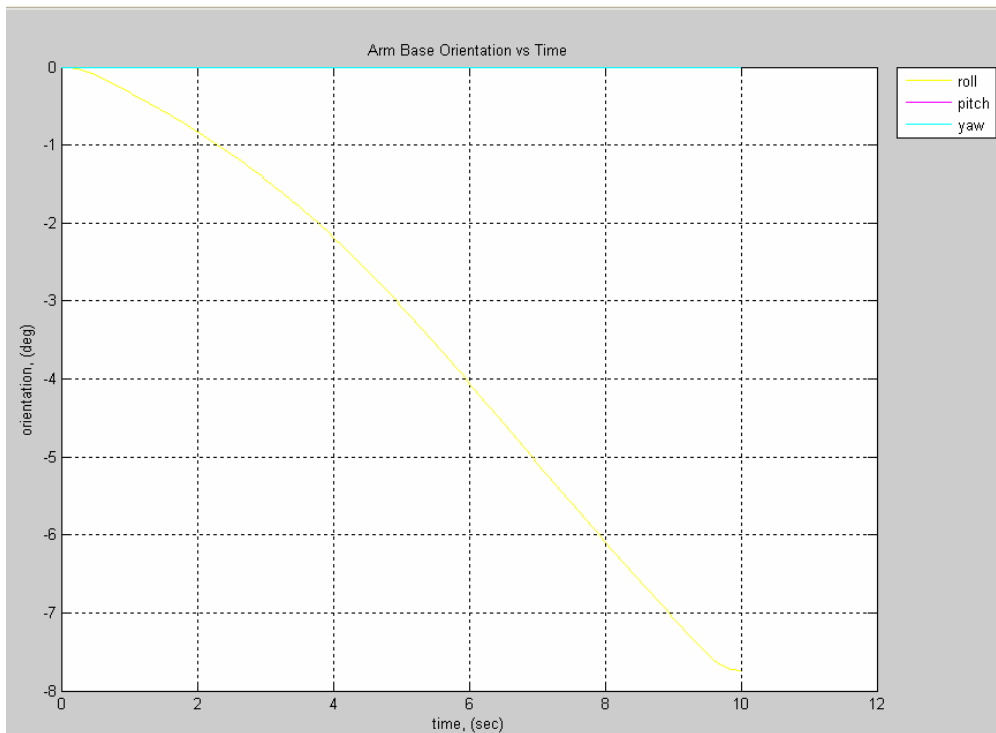


Figure 8.31: Arm Base Orientation for Case B, When $W = [1, 1, 1, 1, 1, 1, 1, 50, 1]$.

8.5 More Simulation for Optimization Methods and Criterion Function Effects

To show the effects of optimization method and the criterion function on the simulation results, four different cases of optimization methods with different criterion function were tested in simulation. These cases are:

- 1- Case I: The minimization of Euclidean norm of errors using Pseudo inverse of the Jacobian.
- 2- Case II: Joint limit avoidance based optimization with Pseudo inverse of the Jacobian and the gradient projection term.
- 3- Case III: The Weighted Least Norm optimization solution with S-R inverse of the Jacobian.
- 4- Case IV: The Weighted Least Norm optimization solution with S-R inverse and joint limit avoidance.

The four tested cases showed different joint reaction in the WMRA system based on the method used and the optimization criteria selected. The WMRA system was commanded to move in autonomous mode from its initial position before simulation to a point that is (-1500, -400, 100) mm away from the ground's frame with the same orientation as the ground frame's orientation. Figures 8.32 and 8.33 show the results of the first case, where the wheels and the joints of the WMRA system moved minimally to achieve the destination point. Note that joint six can not move more than 100° from the center of the joint range, and not including the joint limit avoidance made it cross that limit, while the rest of the joints had plenty of room to move to achieve the destination and did not move.

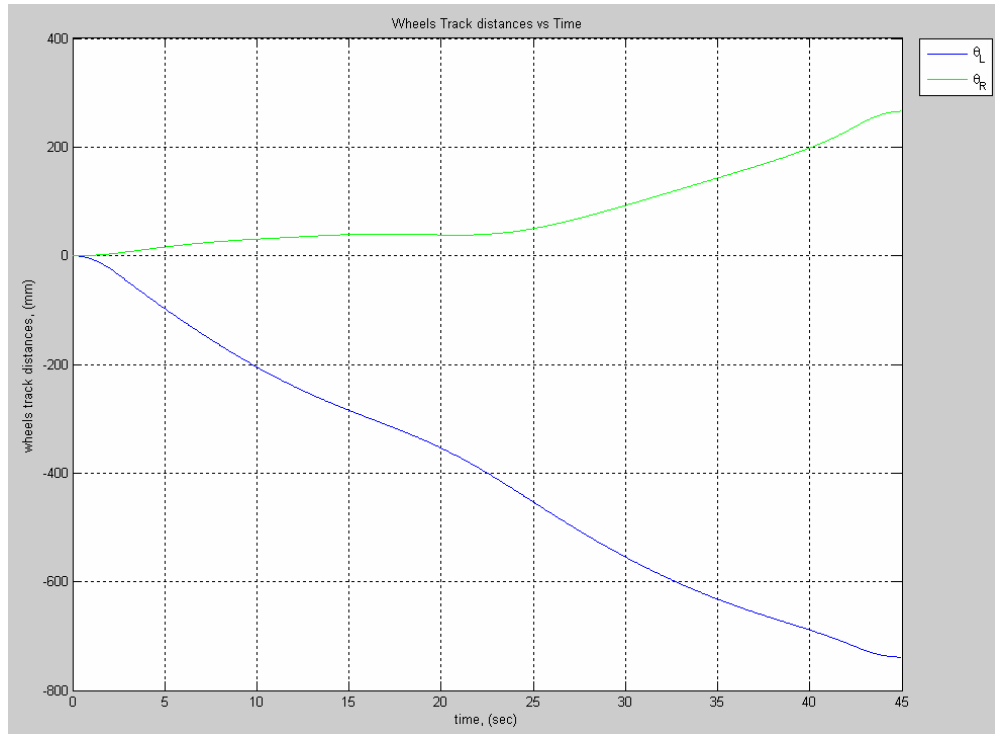


Figure 8.32: Wheels' Motion Distances for Case I.

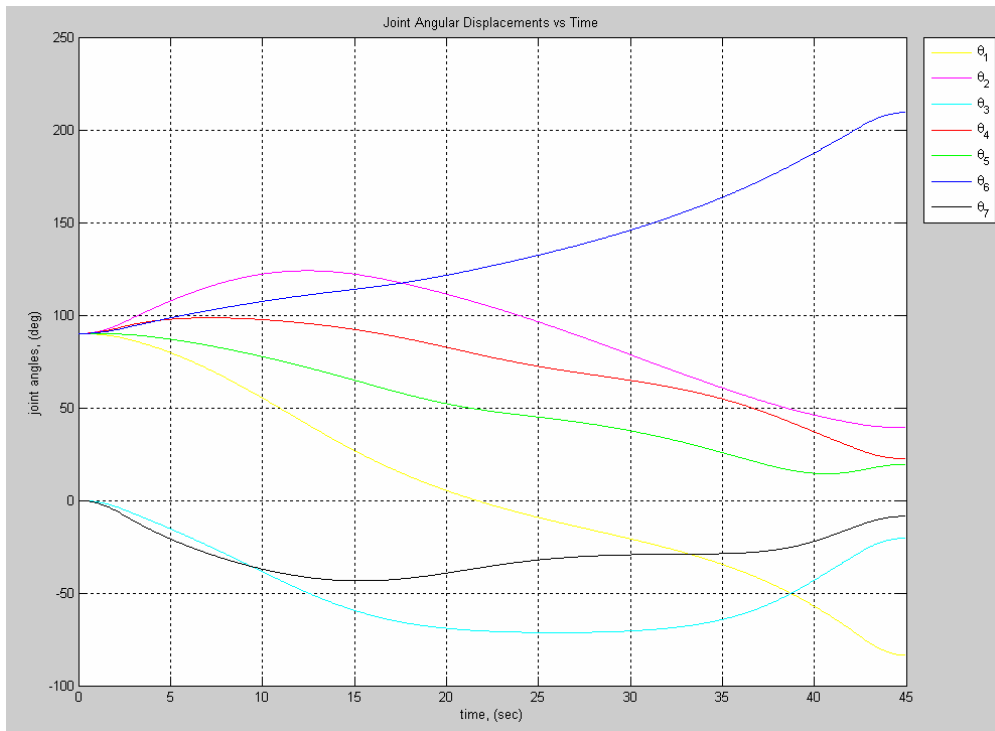


Figure 8.33: Joint Angular Displacements for Case I.

When this same test was conducted with the joint limit avoidance as the criterion function as discussed in case II, joint limits were successfully avoided as shown in figures 8.34 and 8.35. The wheelchair moved more than in the first case, and joint one came close to its limit of 170° to compensate for the other joints for the limited motion in joint six.

In case III, a different optimization method was used without the joint limit avoidance criterion function imbedded in the weight matrix. The weight matrix considered of the user-defined weights of $W = \text{diagonal} [1, 1, 1, 1, 1, 1, 1, 20, 20]$. Figures 8.36 and 8.37 show the motion of the wheels as they occurred later on during the simulation, and the joint angles travelled to reach the destination at the end effector. It can be seen that joint six went over its limits of 100° since the weight matrix does not reflect the joint limit avoidance as the optimization criterion function.

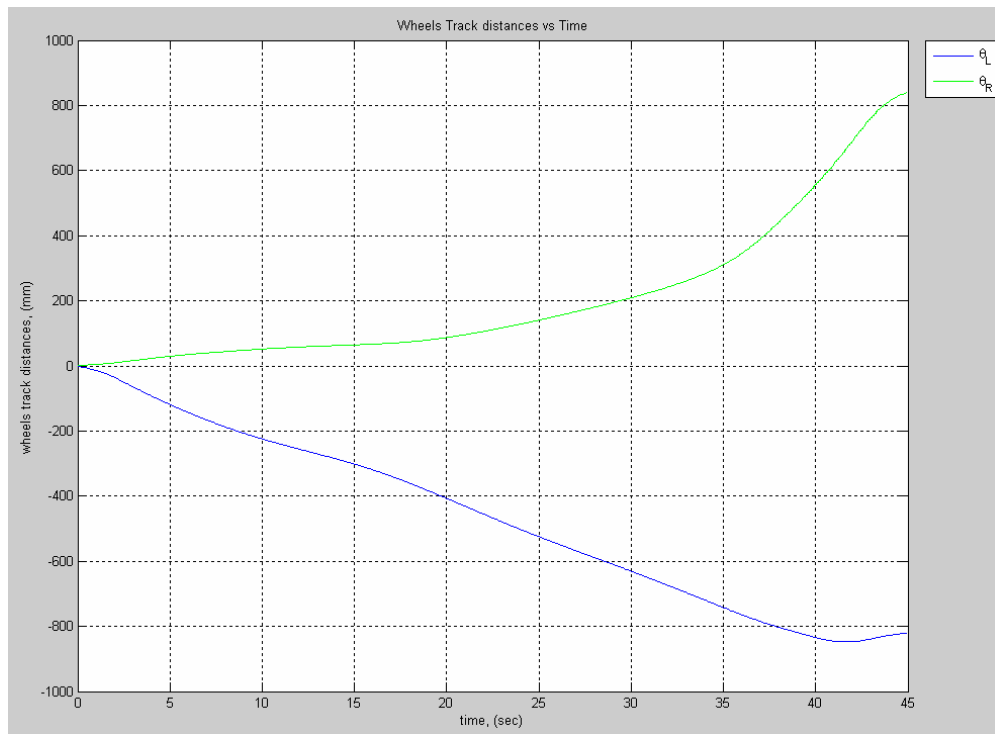


Figure 8.34: Wheels' Motion Distances for Case II.

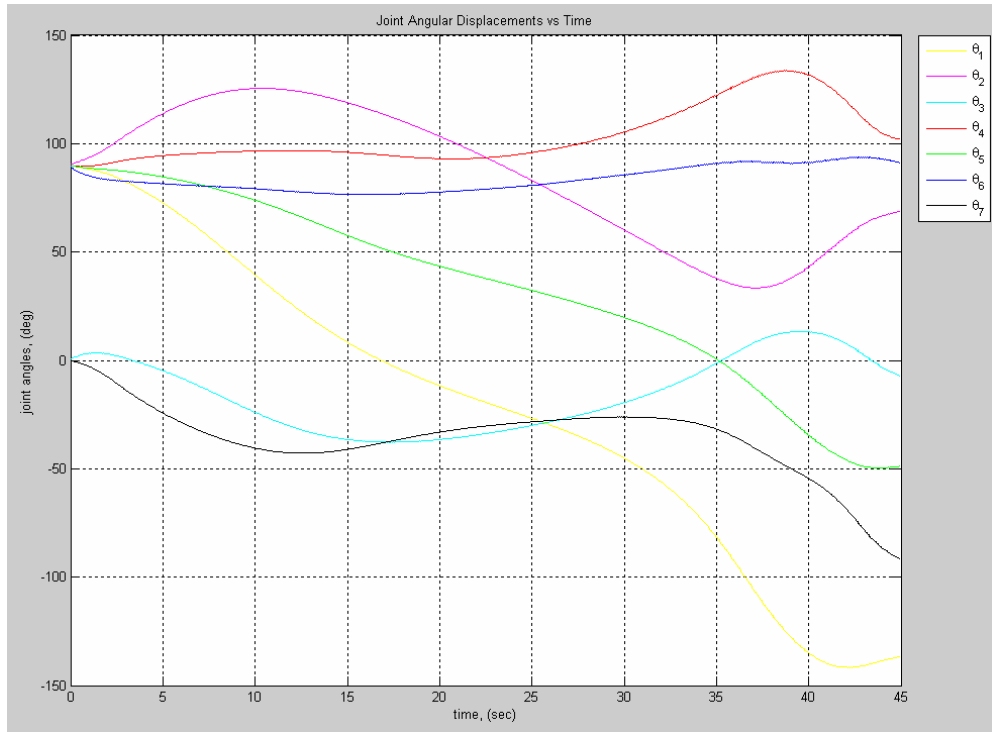


Figure 8.35: Joint Angular Displacements for Case II.

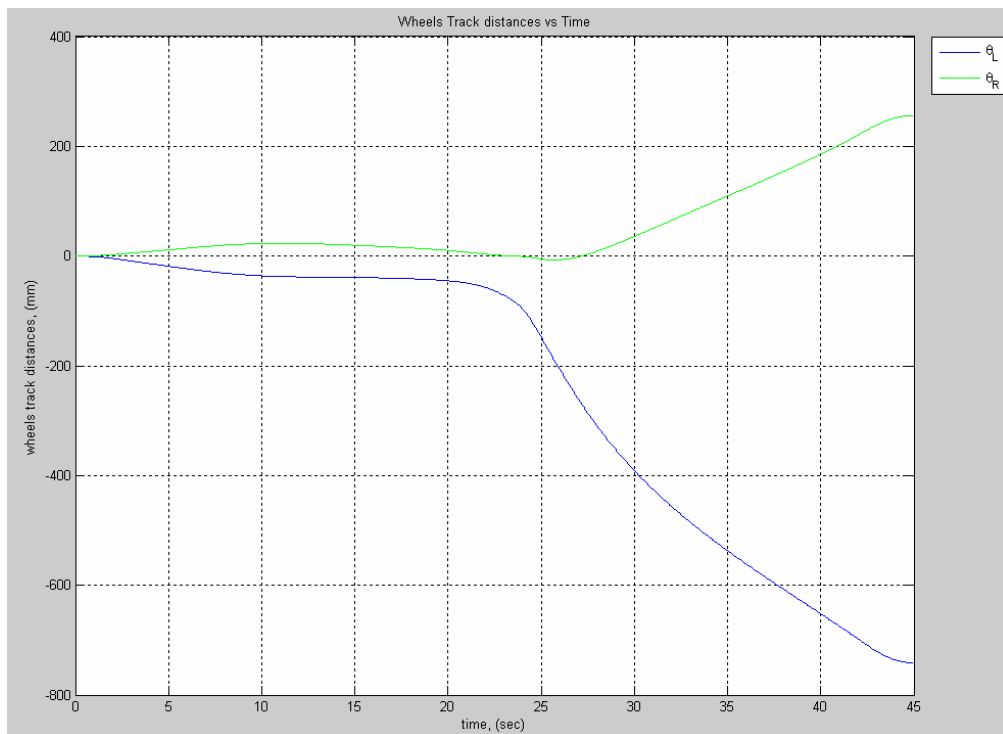


Figure 8.36: Wheels' Motion Distances for Case III.

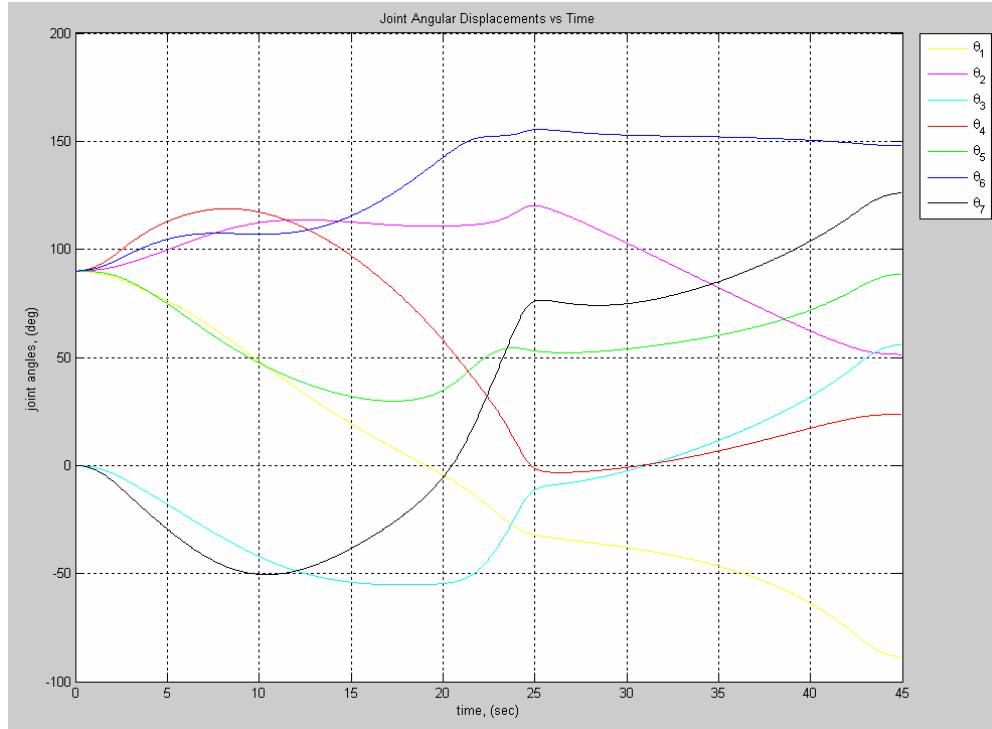


Figure 8.37: Joint Angular Displacements for Case III.

In case IV, the optimization criterion function was included in the weight matrix to avoid joint limits with all the four conditions discussed in chapter 5. Figures 5.38 and 5.39 show how the wheelchair moved significantly more to compensate for the joints that reached their limits and their weights went to infinity. Joint three reached its limit of 170° and joint six reached its limit of 100° . This resulted in a smoother simulation with joint limit implementation while keeping the user's preference of minimal wheelchair motion as expressed in terms of the user-defined portion of the weight matrix discussed in chapter five. These test cases reflect the usability of the system and its reaction to different control algorithms as it is used based on the user's preference. It is noted here that when the user chose case II with teleoperation mode, the system started moving before the user touched the controls since the system was still optimized to keep the joints close to the middle of their range of motion.

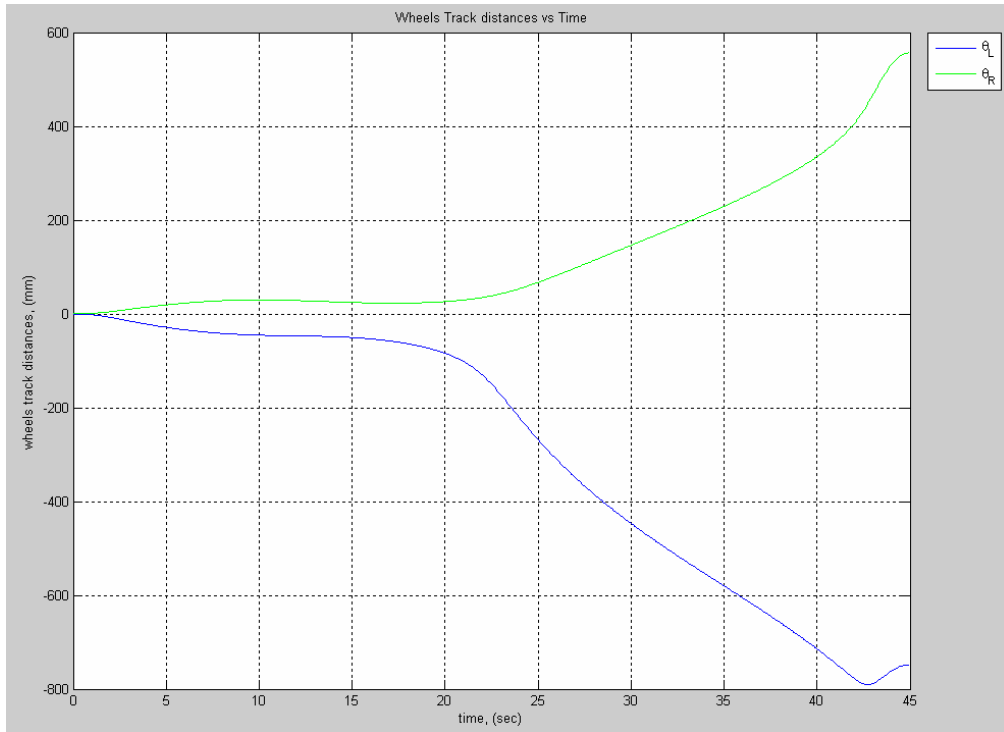


Figure 8.38: Wheels' Motion Distances for Case IV.

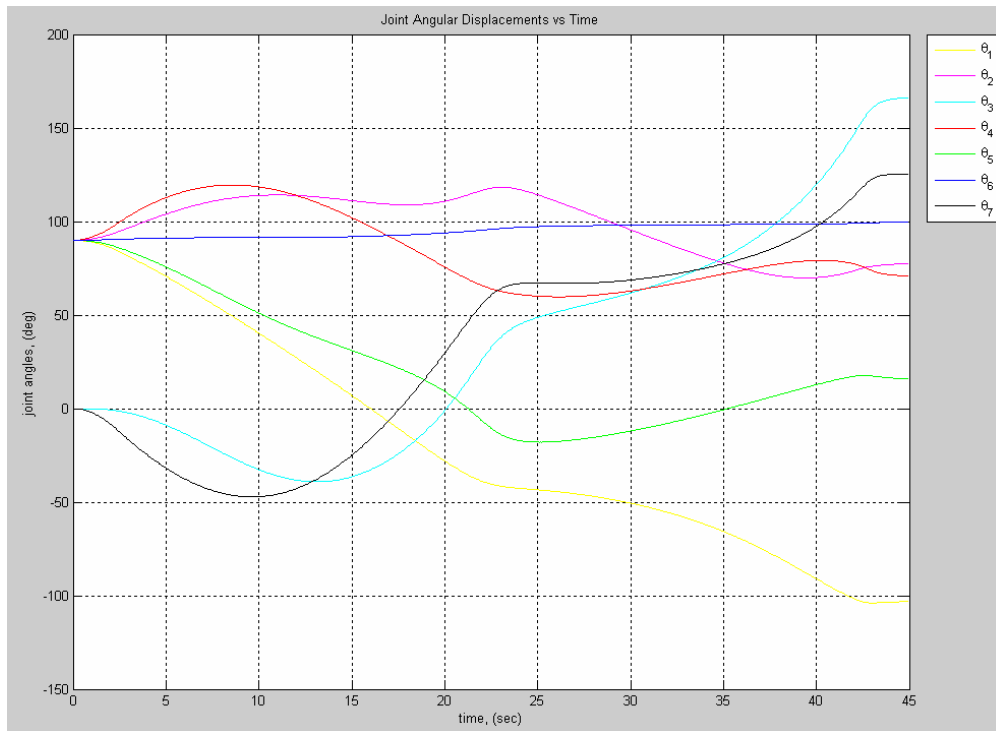


Figure 8.39: Joint Angular Displacements for Case IV.

8.6. Simulation of the Eight Implemented Optimization Control Methods for the Case of an Unreachable Goal

To test the difference in the system response in case the WMRA system is commanded to reach a point that is physically unreachable, eight different cases were simulated, each uses a different control method. The end-effector was commanded to move horizontally and vertically upwards to a height of 1.3 meters from the ground. This height is physically unreachable, and the WMRA system will reach singularity. The response of the system can avoid that singularity depending on the method used. Singularity, joint limits and preferred joint-space weights were the three factors we focused on in this simulation. The eight control cases simulated were as follows:

- 1- Case I: Pseudo inverse solution (PI).
- 2- Case II: Pseudo inverse solution with the gradient projection term for joint limit avoidance (PI-JL).
- 3- Case III: Weighted Pseudo inverse solution (WPI).
- 4- Case IV: Weighted Pseudo inverse solution with joint limit avoidance (WPI-JL).
- 5- Case V: S-R inverse solution (SRI).
- 6- Case VI: S-R inverse solution with the gradient projection term for joint limit avoidance (SRI-JL).
- 7- Case VII: Weighted S-R inverse solution (WSRI).
- 8- Case VIII: Weighted S-R inverse solution with joint limit avoidance (WSRI-JL).

From these cases, we observe the following results in terms of singularity expressed by the manipulability measure, joint limit avoidance (joint 6 should not exceed

+/- 100°), and the user option of preferred weights of motion (1 is used for the arm and 10 for the wheelchair):

- 1- Case I: (PI) In this case, the system was unstable, the joints went out of bounds, and the user had no weight assignment choice (see figures 8.40 and 8.41).
- 2- Case II: (PI-JL) In this case, the system was unstable, the joints stayed in bounds, and the user had no weight assignment choice (see figures 8.42 and 8.43).
- 3- Case III: (WPI) In this case, the system was unstable, the joints went out of bounds, and the user had weight assignment choices (see figures 8.44 and 8.45).
- 4- Case IV: (WPI-JL) In this case, the system was unstable, the joints stayed in bounds, and the user had weight assignment choices (see figures 8.46 and 8.47).
- 5- Case V: (SRI) In this case, the system was stable, the joints went out of bounds, and the user had no weight assignment choice (see figures 8.48 and 8.49).
- 6- Case VI: (SRI-JL) In this case, the system was unstable, the joints stayed in bounds, and the user had no weight assignment choice (see figures 8.50 and 8.51).
- 7- Case VII: (WSRI) In this case, the system was stable, the joints went out of bounds, and the user had weight assignment choices (see figures 8.52 and 8.53).
- 8- Case VIII: (WSRI-JL) In this case, the system was stable, the joints stayed in bounds, and the user had weight assignment choices (see figures 8.54 and 8.55).

It is clear that case number 8 showed the best performance since it fulfilled all the important control requirements. This method avoided singularities while keeping the joint limits within bounds and satisfying the user-specified weights as much as possible.

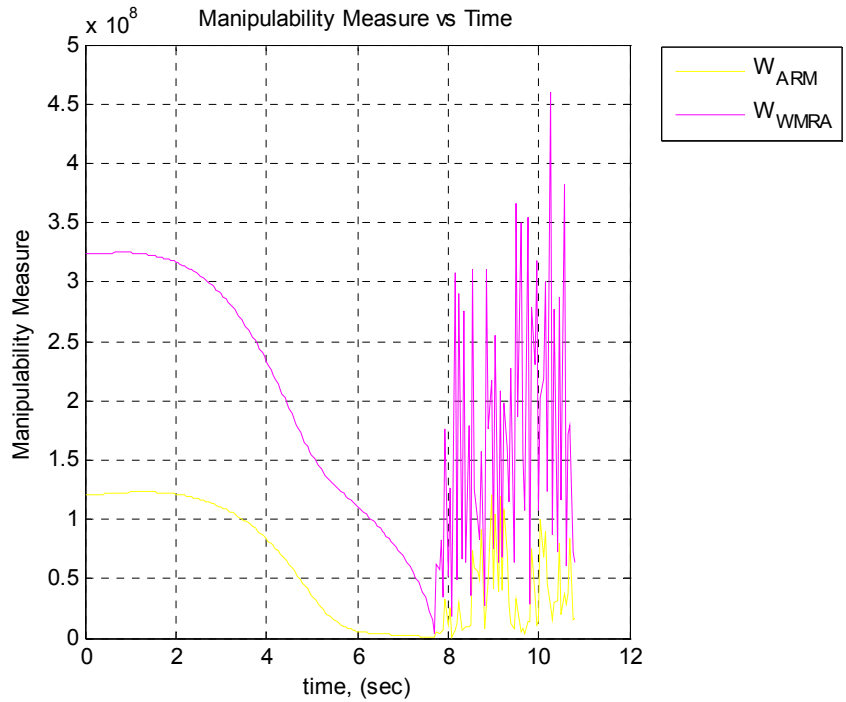


Figure 8.40: Manipulability Measure Case I (PI).

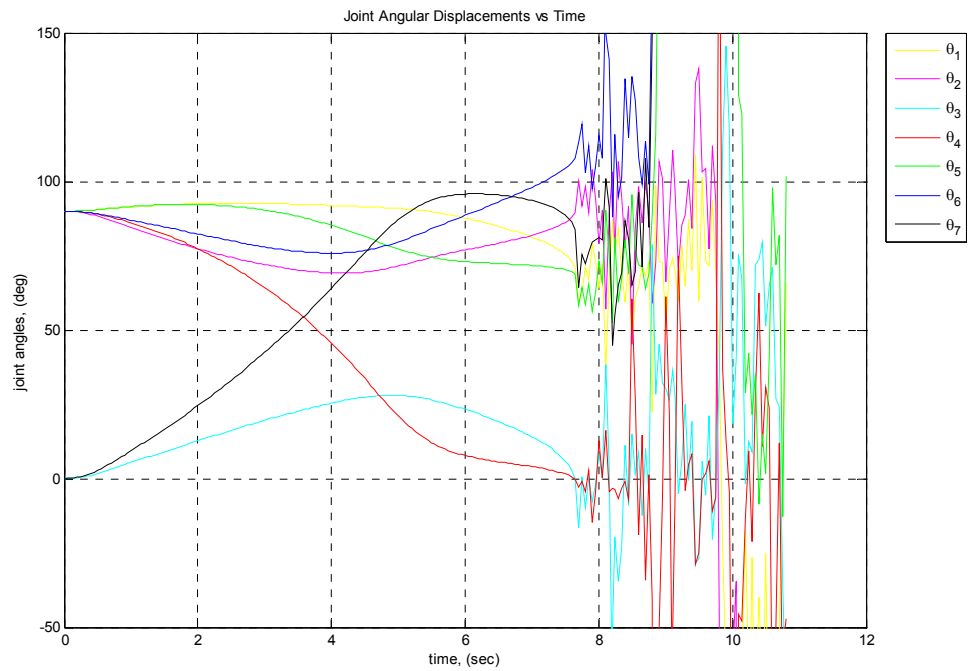


Figure 8.41: Joint Angular Displacements for Case I (PI).

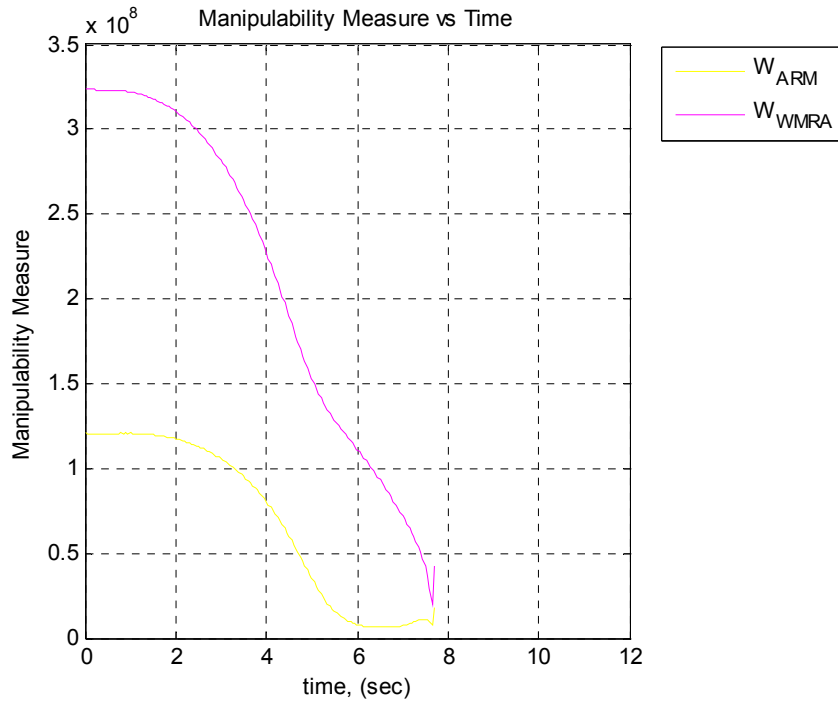


Figure 8.42: Manipulability Measure Case II (PI-JL).

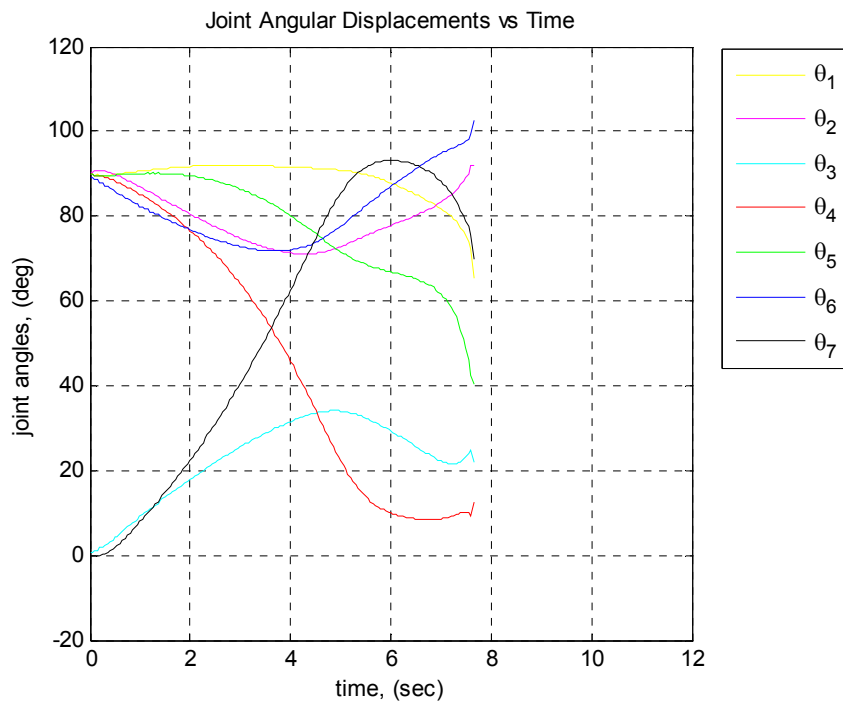


Figure 8.43: Joint Angular Displacements for Case II (PI-JL).

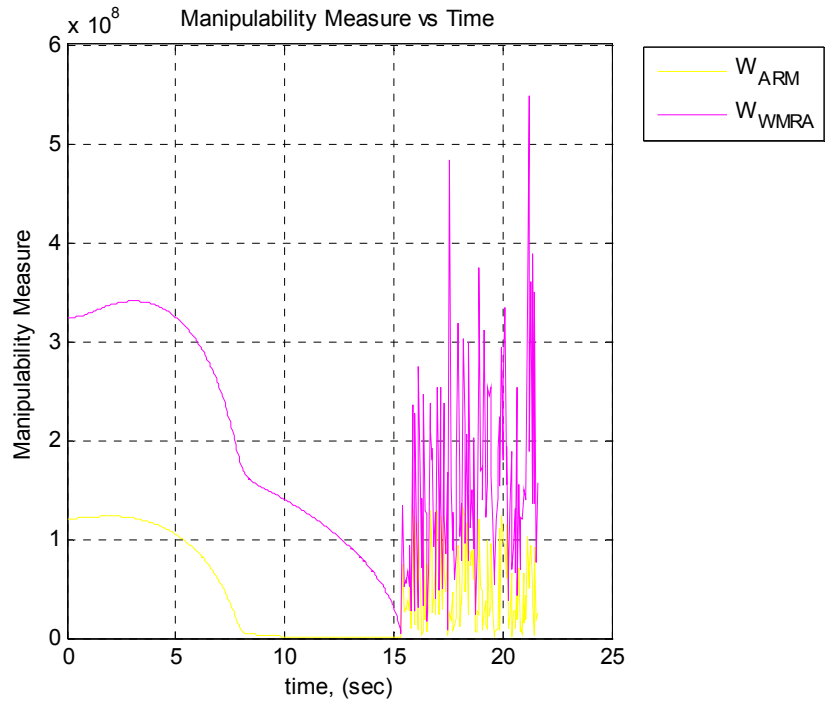


Figure 8.44: Manipulability Measure Case III (WPI).

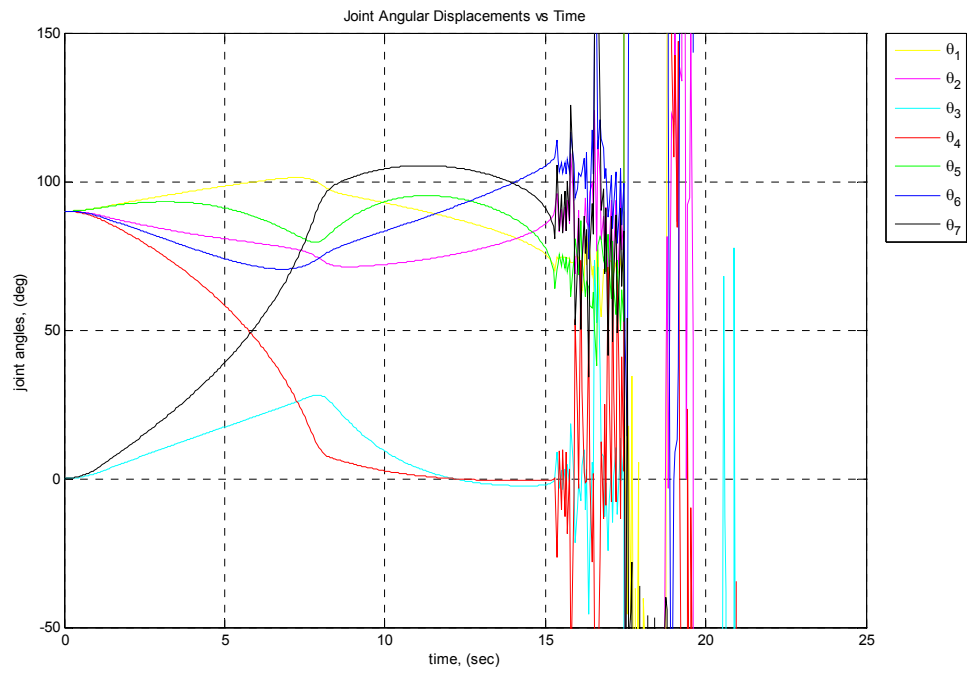


Figure 8.45: Joint Angular Displacements for Case III (WPI).

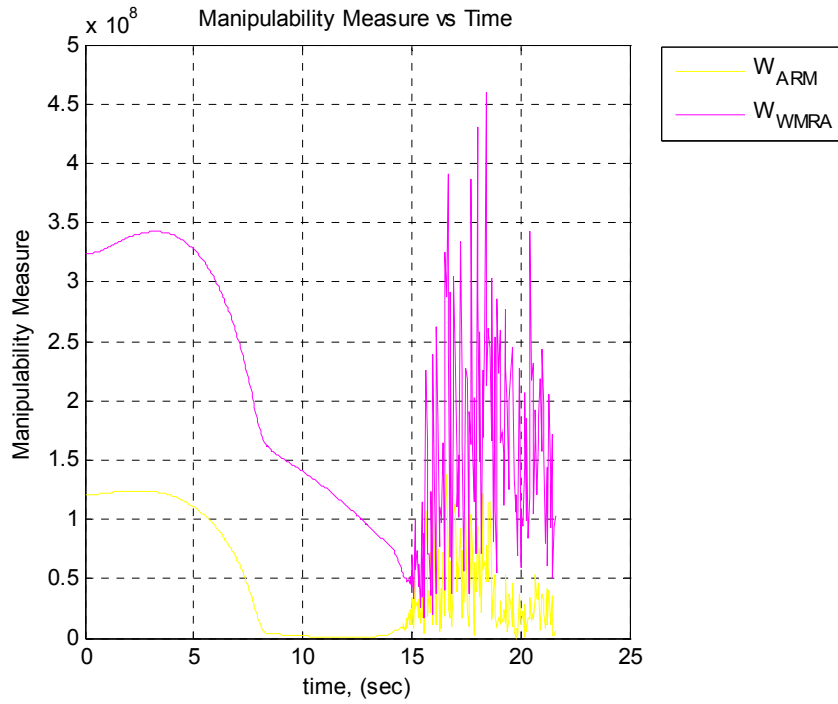


Figure 8.46: Manipulability Measure Case IV (WPI-JL).

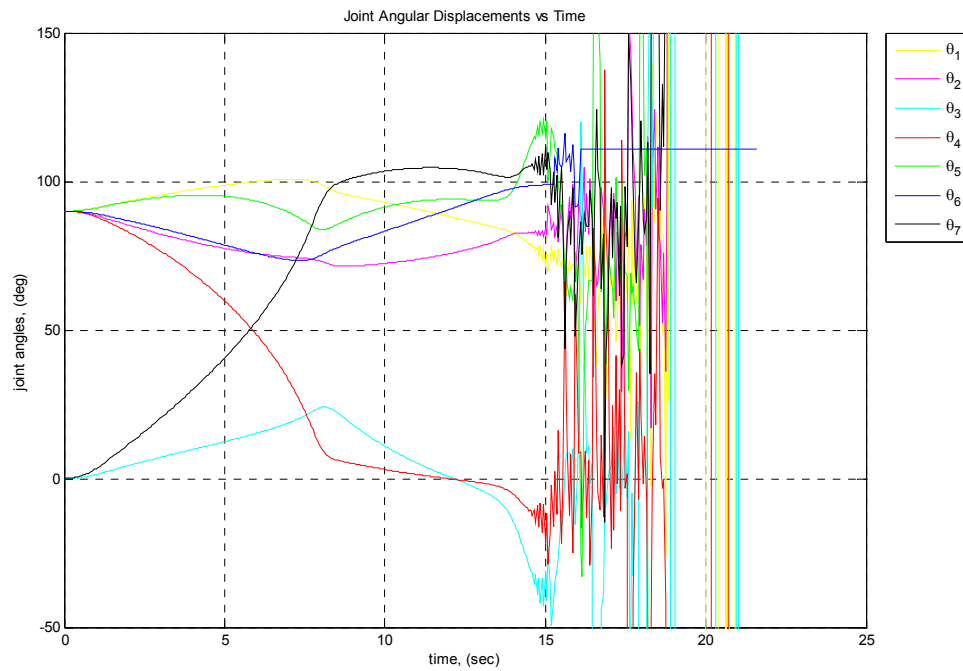


Figure 8.47: Joint Angular Displacements for Case IV (WPI-JL).

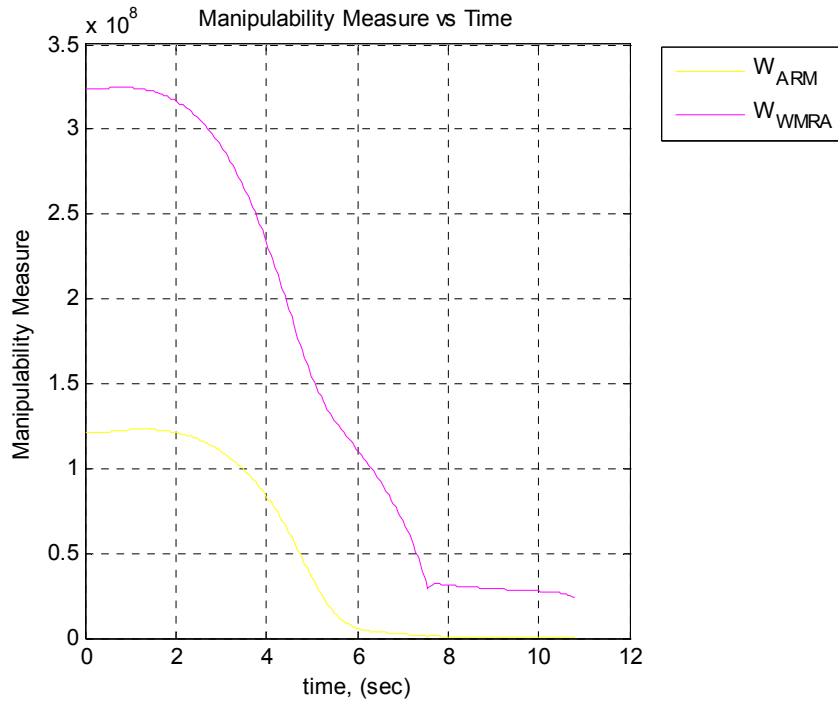


Figure 8.48: Manipulability Measure Case V (SRI).

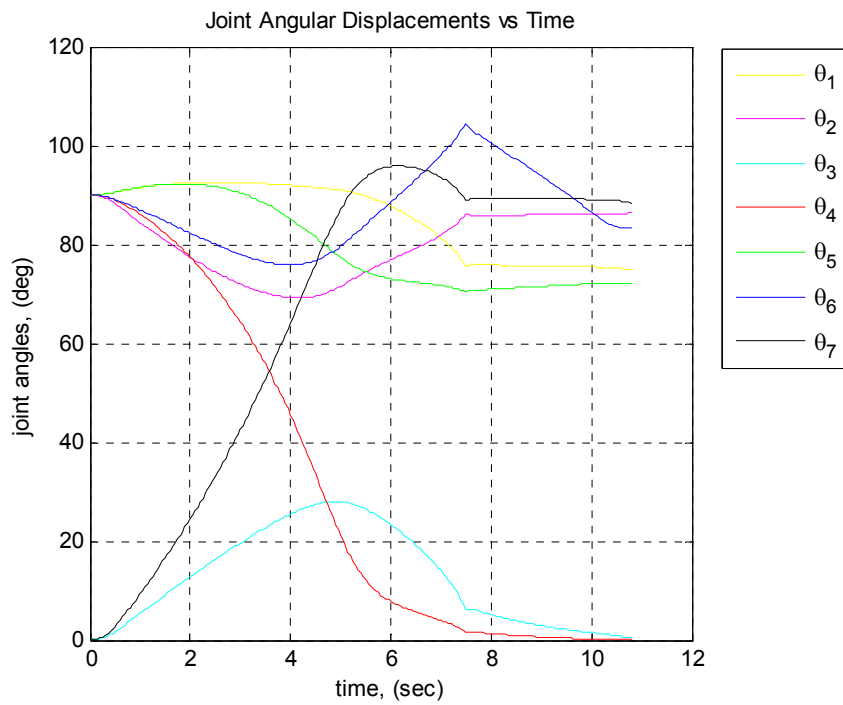


Figure 8.49: Joint Angular Displacements for Case V (SRI).

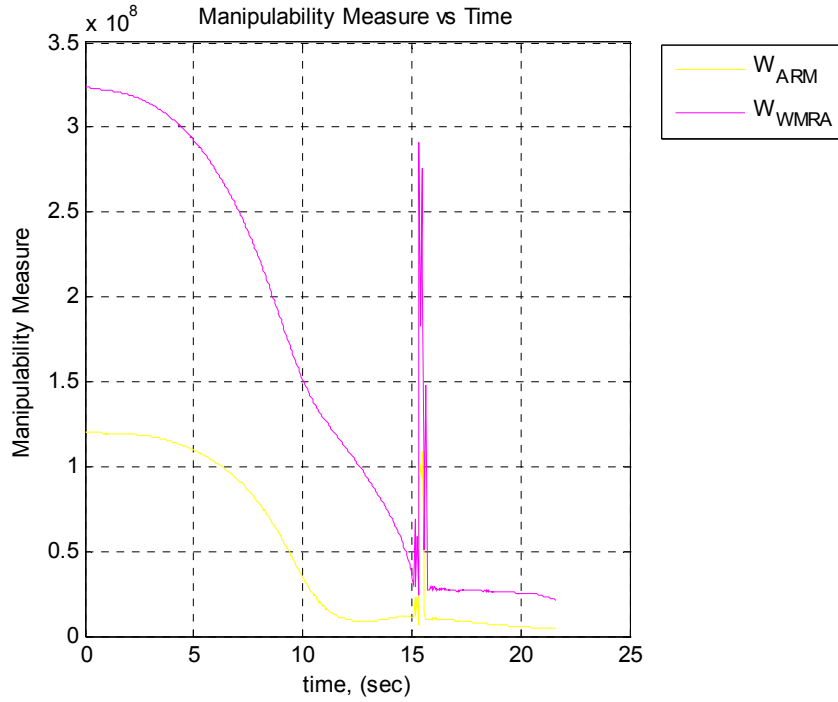


Figure 8.50: Manipulability Measure Case VI (SRI-JL).

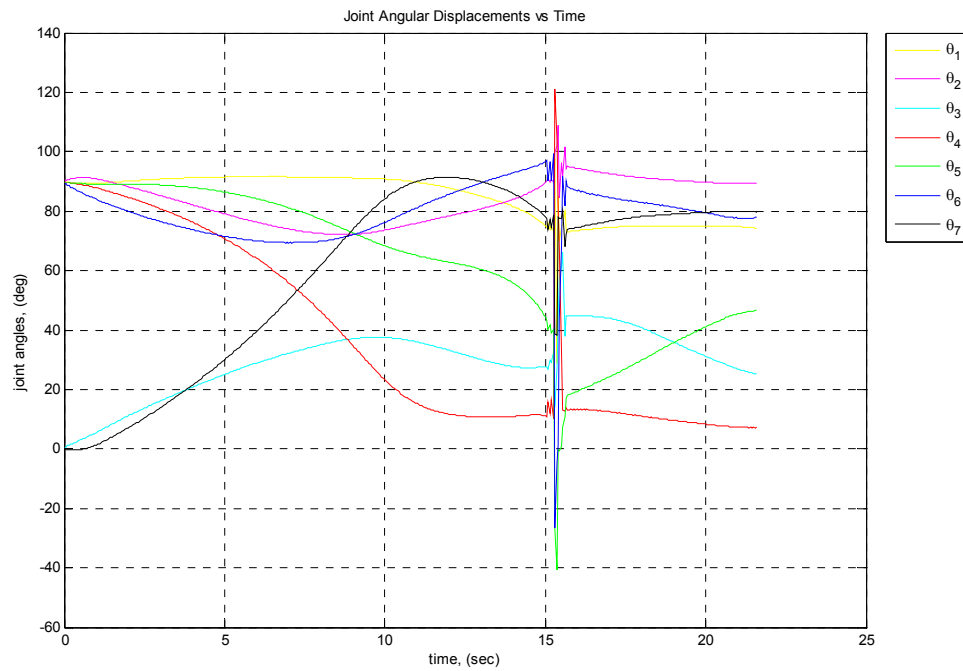


Figure 8.51: Joint Angular Displacements for Case VI (SRI-JL).

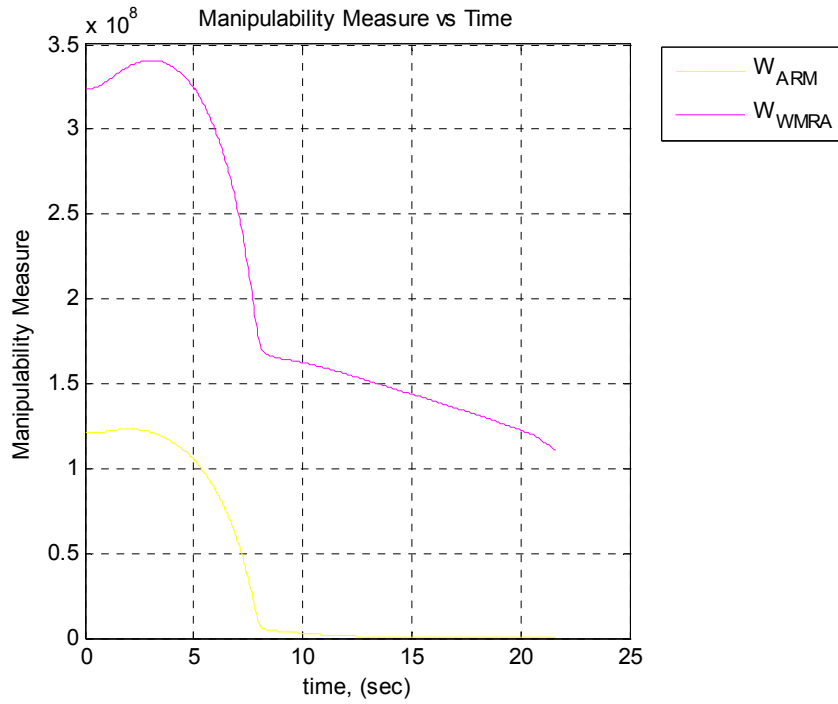


Figure 8.52: Manipulability Measure Case VII (WSRI).

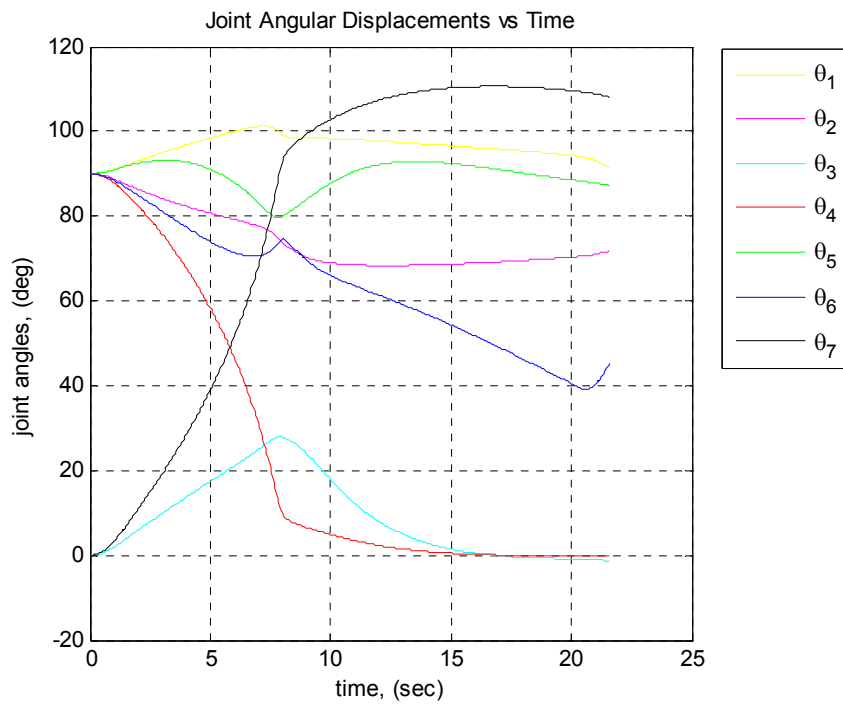


Figure 8.53: Joint Angular Displacements for Case VII (WSRI).

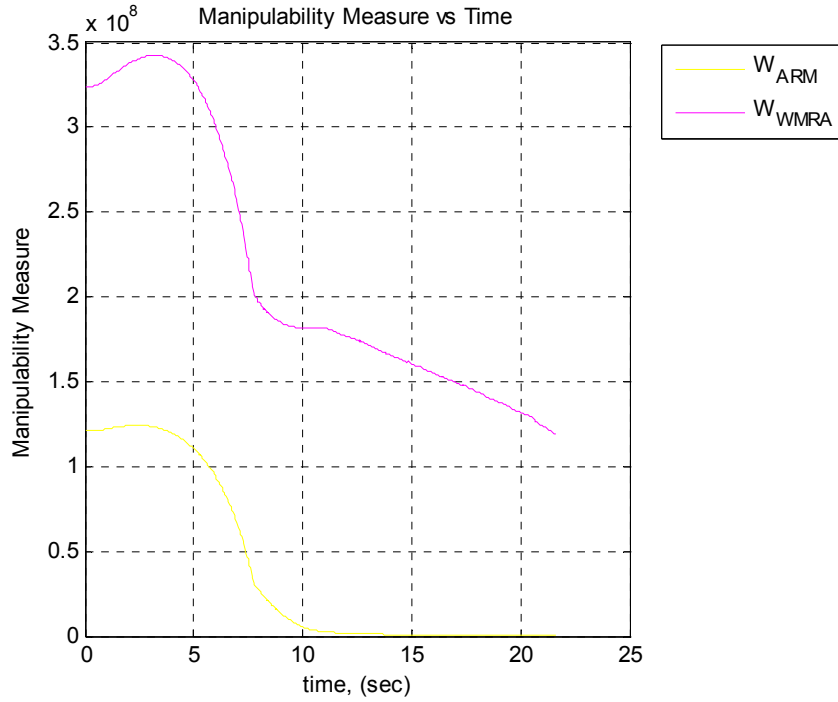


Figure 8.54: Manipulability Measure Case VIII (WSRI-JL).

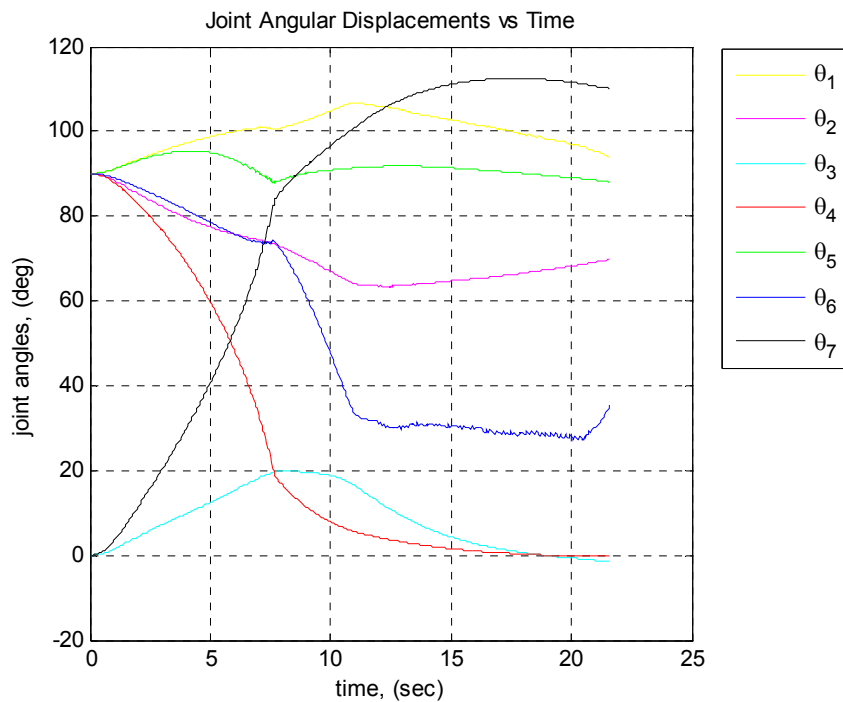


Figure 8.55: Joint Angular Displacements for Case VIII (WSRI-JL).

8.7. Summary

Simulation results of the implementation of the methods of combining mobility and manipulation and redundancy resolution were shown in this chapter. Several cases were defined for simulation, and observation of the simulation results were shown and discussed for the effectiveness of the solutions. In all cases, the trajectory was generated to move the end-effector from the initial to the final position following the specified optimization choices. Final configurations of the WMRA system were shown for all cases, and the joint space variables were studied. The effect of the 3rd degree polynomial with parabolic blending in generating the trajectory points were shown in the velocities of the joint space variables. This led to constant accelerations or decelerations of the variables so that smooth motion occurred. A couple of other simulations shown in this chapter verified the idea behind the proper choice of the state variables so that the control of these variables makes more sense than any arbitrary choice of variables that may produce undesirable system behavior.

Four comparison cases were presented to compare four different control optimization methods when used within the workspace. Another eight cases were presented to show the different behaviors of the system response in case the WMRA system was commanded to go to a point that was physically out of its reach. These twelve cases clearly identified the advantage of using the WSRI method with joint limit avoidance over all other optimization methods.

Chapter 9:

Experimental Testbed and Field Tests

9.1. Introduction

The combination of mobility and manipulation in robotics as assistive devices would be better used in actual products if testing on physical systems was done after theories and simulation results were developed. In this chapter, we will discuss the testbed comprise of a new 7-DoF robotic arm design, a modified wheelchair, a new gripper designed specifically for activities of daily living (ADL), and a control hardware that controls all these equipment using a tablet PC running Windows XP. Design aspects and components will be shown, and communication and wiring the system together will be discussed.

9.2. The New 7-DoF Robotic Arm Design and Development

A 7-DoF wheelchair-mounted robotic arm (WMRA) was designed and built to be integrated with a power wheelchair to help people with disabilities to do their activities of daily living independently or with minimal help. The mechanical design incorporates DC servo drive motors with actuator hardware at each individual joint, allowing reconfigurable link lengths [52]. It has seven degrees of freedom and uses a side mount on a power wheelchair. The control system allows coordinated Cartesian control, and

offers expandability for future research, such as coordinated motion with the wheelchair itself.

9.2.1. Design Goals

A new WMRA was developed, designed and built. The goal was to produce an arm that has better manipulability, greater payload, and easier control than current designs. The arm is also reconfigurable, which increases the number of applications for our design. The following design goals were set for the hardware:

9.2.1.1. Weight

In a mobile application, minimal weight is of primary importance. Power wheelchairs have a rated payload, and a heavy arm reduces the payload available for the user. Based on this criterion, our goal was to have a total system mass under 14 kg, including the arm, controller, and all peripherals.

9.2.1.2. Mount Type

As found in previous research [57], side mounting is preferable overall because it provides the best balance between manipulability and unobtrusiveness. However, care must be taken to prevent widening of the power chair. The new arm is mounted as far forward and upward as possible while still in a side mount configuration, and only increases chair width by 7.5cm. This mounting location allows the arm to be stowed by folding it back and then wrapping the forearm behind the seat. It virtually disappears

when not in use, especially when the arm is painted to match the chair. This helps avoid the stigma that these devices can bring.

9.2.1.3. Stiffness

It is one of the major differences between this WMRA and a typical industrial manipulator. As anticipated, teleoperation will be the most common control mode for the robot, and therefore great precision is not required. With a human participating at all times, inaccuracy due to a compliant structure is easily and transparently corrected. Recognizing this allowed the structure to be made much lighter than an industrial manipulator with the same payload. However, the low stiffness and large backlash of other WMRAs is an impediment to accurate coordinated control. With this design, we attempted to find an optimal balance, stiffer than other WMRAs, but less stiff than an industrial manipulator.

9.2.1.4. Payload

This manipulator is intended for use in Activities of Daily Living (ADL), and for job tasks of a typical office environment. As such, it is important that the arm be strong enough to move objects that are common in these environments. A gallon (4 Liters) of milk is a good upper limit for a typical around-the-house object that must be manipulated. As this is an approximately 4 kg mass, this was set as the baseline payload for the arm at full horizontal reach at rest. Then, an extra margin of 2 kg was added to allow for a choice of end-effector capable of this load. The 4 kg useful payload is much larger than the 1 kg payload of the Raptor.

9.2.1.5. Reconfigurability

Even though a side mount was chosen for this prototype, it is important to note that the basic design can be adapted to a front or rear wheelchair mount, or a fixed workstation mount. The arm can be specialized for these workspaces by adjusting link lengths. Longer lengths can be specified for a rear mount on a power chair, but this reduces payload and reduces manipulability in front of the chair. Reconfigurable arm lengths allow greater leverage on the engineering input, as a single basic design may be adapted to numerous applications. This is only practical with electric drive and actuator placement directly at each joint.

9.2.1.6. Power Supply and Consumption

In the power wheelchair industry, a 24-volt lead-acid battery pack is standard, and is the natural choice for the power supply of a WMRA. All motors, controllers, input devices, sensors, etc. must be able to work with 24vdc. Energy consumption is important, as users would reject a device that worked well but left them stranded without the wheelchair power! Therefore, efficient components were chosen to keep power consumption low.

9.2.1.7. Cost Constraints

Reasonable cost is important to widespread adoption of these devices, but is not a major hurdle such as poor utility and difficulty of use. The target was to be in the mid-range of commercially available systems in terms of cost. The usability of the WMRA

system will be far more important than the cost when the user decides on which device should be used.

9.2.1.8. User Interface

People want a useful payload, and a simple intuitive control. Raptor lacks encoders and therefore control is manual, one joint at a time. Quadrature encoders are a cost-effective way to provide coordinated Cartesian control. The controllers of the new WMRA have PWM voltage regulation, and have built-in support for acceleration limits. The system easily scales to control grippers or even the base wheelchair, all through one standard control system.

9.2.1.9. Degrees of Freedom

Extra degrees of freedom help improve manipulability. This is evidenced by the considerable increase going from Raptor's 4 DOF to the 6 DOF of MANUS. Our new design incorporates 7 joints, allowing full pose control even in difficult regions of the workspace, such as reaching around the wheelchair, reaching up to a high shelf, manoeuvring around objects, or opening a door.

9.2.1.10. Actuation and Transmission Systems

Most actuation alternatives were restricted due to the requirement for reconfigurability. Changing the length of an arm that is driven through linkages or flexible cables from motors in the base would require many parts to change, which would require a new design. The option of pneumatics was eliminated due to positioning

difficulty and compressor noise. The best option was to drive the joints electrically through harmonic gearheads that carry large gear ratio, with the entire actuator positioned at each joint.

9.2.1.11. DC Motors as Actuators

The only serious choice was whether to use stepper or servo motors [52]. Due to recent improvements in servo controllers, the cost of this option is not much higher than that for stepper motors. Brush DC servomotors allow closed-loop control, and are much quieter, lighter and more efficient than steppers. For these reasons, DC Servo drive was selected. Quadrature encoders, mounted on the motors, were selected for their accuracy, simplicity and low cost. Optical limit switches ease initialization at power-up.

9.2.2. Kinematic Arrangements and Component Selection

The arm is a 7-DOF design, using 7 revolute joints [52]. It is anthropomorphic, with joints 1, 2 and 3 acting as a shoulder, joint 4 as an elbow, and joints 5, 6 and 7 as a wrist as shown in figure 9.1. The 3 DOF shoulder allows the elbow to be positioned anywhere along a spherical surface, whereas with the Raptor arm, elbow movement is limited to a circle. Throughout the arm, adjacent joint axes are oriented at 90 degrees as shown in figure 9.2. This helps to meet two goals: mechanical design simplicity and kinematic simplicity. Machining parts on a conventional milling machine is easier with right angles, and the coordinate transform equations simplify greatly resulting in low computational cost. All adjacent joint axes intersect, also simplifying the kinematics.

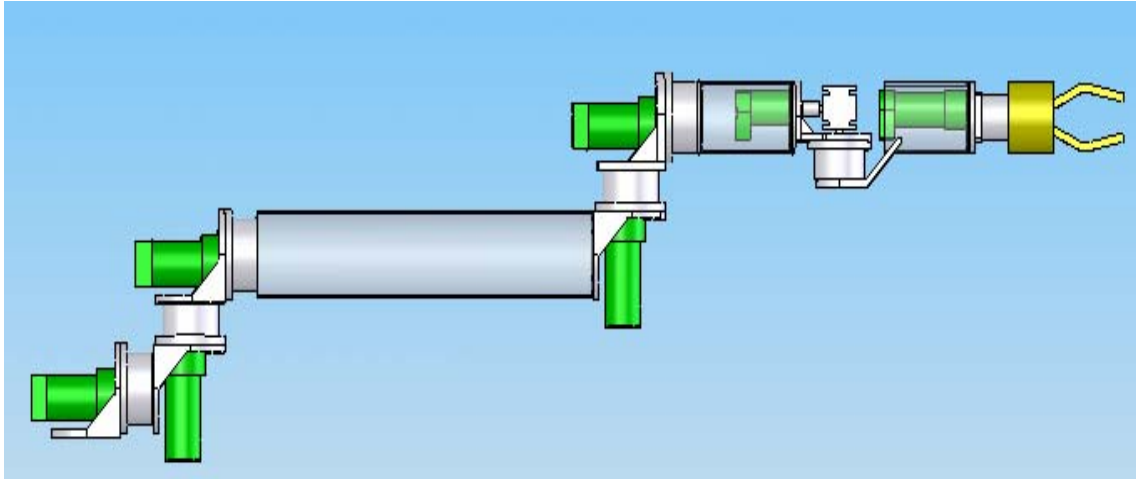


Figure 9.1: Complete SolidWorks Model of the WMRA.

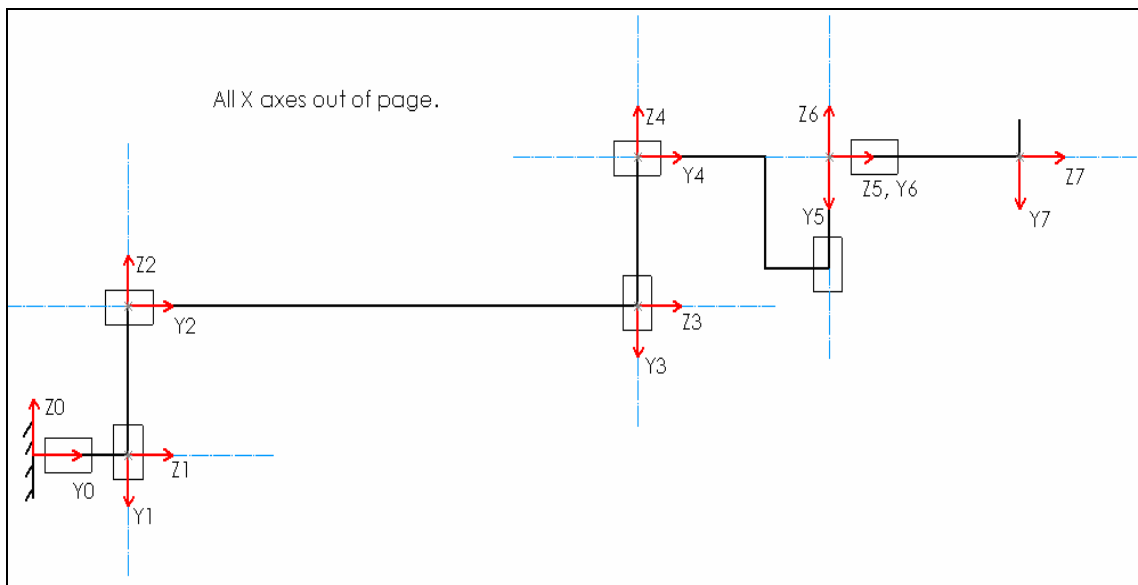


Figure 9.2: Kinematic Diagram with Link Frame Assignments.

Emphasis was placed on using off-the-shelf parts wherever possible. The basic arrangement for each joint is a high-reduction gearhead, a motor with encoder and spur-gear reduction, and a bracket that holds these two parts and attaches to the two neighbouring links. The hardware components were selected to meet the design requirements, as follows:

9.2.2.1. Gearhead Selection

Harmonic drive gearheads were chosen because they can achieve 100:1 reduction in a single stage, with only 64mm axial length [52]. In addition, they have bearings suitable for supporting overhung loads, enabling the next arm segment to be bolted directly to the output flange of the gearhead. This greatly simplifies the design, reducing weight and cost through lower part count. Gearheads were chosen based on required overhung loads and torques, with the size of the gearhead gradually reducing at consecutive distal joint. Once the basic type of gearhead was selected, information on the available sizes was collected, namely the mass and recommended maximum torque. Maximum recommended torque here was taken to be the lesser of two specifications from the manufacturer: maximum output torque and maximum overhung torque. A simple spreadsheet model of a horizontally outstretched arm was made, which accounted for link lengths and self-weight. The target payload (6 kg) was also applied to the end of the arm, and eventually the gearheads were chosen as shown in Table 9.1. One of the harmonic drive gearheads selected for the first two joints in the shoulder of the robotic arm is shown in figure 9.3.

Table 9.1: HD Systems Gearhead Selections for Each Joint.

Joint	Model Selected	Torque (N m)	Outside Diam (mm)	Mass (kg)
1	CSF-25	140	107	1.50
2	CSF-25	140	107	1.50
3	CSF-20	70	93	0.98
4	CSF-17	46	79	0.68
5	CSF-17	46	79	0.68
6	CSF-14	19.5	73	0.52
7	CSF-11	6.6	58	0.15

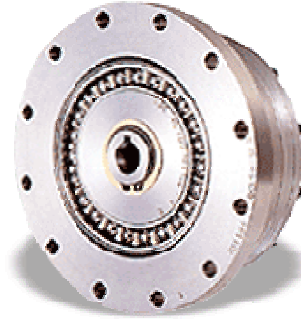


Figure 9.3: Harmonic Drive Gearhead.

9.2.2.2. Motor Selection

Brush DC motors were chosen since they are the least expensive way to achieve servo control [52]. While brushless motors are a future possibility, performance gains are dubious, and would significantly increase the cost of the robot. The marginal increase in efficiency is relatively unimportant, and gear train noise is already greater than commutator noise. The main benefits for brushless motors are increased service life before maintenance, and possibly better packaging. Brush DC servo drive is the best overall compromise for a WMRA. Brush DC, 24v Pittman motors were selected that meet all performance criteria, and have integrated gearboxes and encoders as shown in figure 9.4. These relative encoders are initialized with optical limit switches.

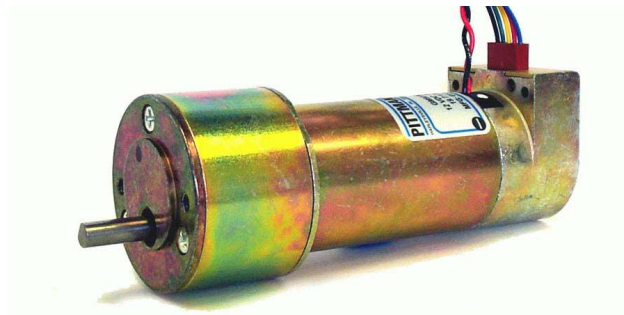


Figure 9.4: Pittman Servo Brush Motors with Gearbox and Encoder.

9.2.2.3. Material Selection

6061 Aluminum was chosen for the joint brackets because of machinability, weldability, lower cost, good strength-to-weight ratio, and availability [52]. This material was also chosen for the link tubes, for the same reasons. Steel was rejected due to its high density. In many places, the thickness of a bracket is not determined by strength or stiffness, but by simple packaging constraints. Steel would unacceptably increase mass in these areas. Carbon fiber/epoxy were considered for the link tubes due to the increase in stiffness and reduction in weight possible. It was rejected for the prototype because of its cost. Carbon fiber becomes especially attractive for a long-reach option, and may make a rear-mount arm more feasible.

9.2.2.4. Joint Design

Once all components were selected, design of each joint was rather straightforward [52]. The typical arrangement for a joint is to have a gearhead and motor held together by an angle bracket. This bracket mounts to the previous joint or link. The output flange of the gearhead attaches to the next link. Billet 6061 aluminum was chosen for its high strength and dimensional accuracy.

9.2.2.5. Wrist Design

There were two basic choices for a 3-DOF wrist: Orthogonal and nonorthogonal as shown in Figures 9.5. The conventional orthogonal arrangement was selected due to better packaging [52]. All three axes are mutually orthogonal and all axes of rotation converge at a single point. This is common in industrial manipulators, such as the Puma

560. It is done to simplify the kinematic equations and provide inverse kinematic solution with the least amount of computations. Joint 6 has a design unlike the others in this manipulator. A right angle gearbox between the motor and gearhead greatly improves packaging, but increases complexity. A single bracket was designed to hold all 3 parts in proper alignment, and to carry the load to joint 5. Joint 7 is coaxial with the last link, so irrespective of the pose of the arm, rotation about this axis is assured. The gearhead mounts to a flange welded to the end of the link tube, and the motor is hidden inside this tube. Wires to the motors and encoders can run inside the tubes or be clipped on the side of the tubes.

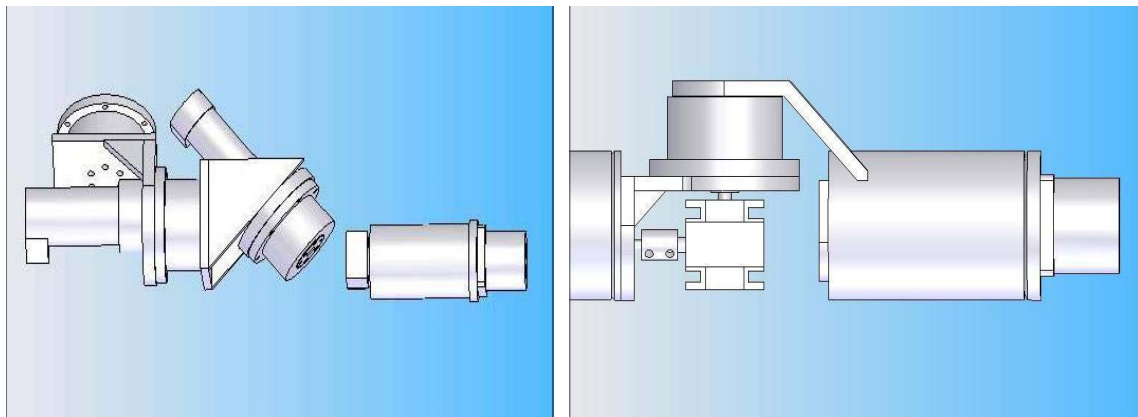


Figure 9.5: Wrist Design: 3-Roll Wrist (Left), Orthogonal Wrist (Right).

9.2.3. Final Design Testing and Specifications

Good stiffness leads to quality construction and accurate control. Stiffness was tested by extending the arm straight out in front of the wheelchair [52]. A dial indicator was set to measure vertical deflection, and then a known mass was applied to the wrist plate. Deflections were measured at the wrist plate (100.3 cm from axis 1), joint 4 (50.8

cm from joint 1) and directly on the joint 1 gearhead. Table 9.2 shows the arm deflection due to the applied load.

Table 9.2: Arm Deflections vs. Applied Load.

Load (kg)	Wrist Deflection (mm)	Elbow Deflection (mm)	Joint 1 Deflection (mm)
2	4.4	1.8	0.2
4	8.7	3.7	0.4
6	13.3	5.5	0.7

Each joint was individually tested for the maximum load it could lift. This was done by placing the arm in a pose most adverse for the joint in question. The arm was placed fully outstretched, pointing forward parallel with the ground. Weights were progressively added, and the joint was given full power to try to raise them. All joints were tested up to the design load. Testing shows that joints three and four are overpowered, and smaller motors could be substituted.

The maximum, unloaded speeds of each joint were measured using a known arc (90, 180, or 360 degrees as geometry permitted). Time to traverse this arc was measured with a stopwatch and joint velocities in RPM were calculated. Speeds range from 5 RPM in proximal joints to 16 RPM in Joint 7. With any battery-operated device, energy use is very important. A digital multi-meter was connected inline with the power feed from the wheelchair battery, and power consumption was recorded as shown in Table 9.3.

Table 9.3: Power Usage.

Condition	Current (A)
Idle - all motors off, controller only	0.36
Holding self-weight outstretched	0.58
Holding 6kg fully outstretched	1.70
Lifting 6kg with joint 1	3.30

While more testing will be instructive, a reasonable estimate is that typical household and office tasks will lead to an average current of 2 Amperes. Therefore, six continuous hours of arm use would consume 12 Ah. This would leave a 73 Ah battery (group 24 gel cell) with 61 Ah for propulsion, or 84% of capacity. Thus, driving range would be reduced, from about 30 km to 25 km. This should be acceptable for most users, and daytime charging can help restore range. Figure 9.6 shows the completed robotic arm in different positions using the SolidWorks model of the arm as well as the actual arm in two different positions. The photos show the arm with a Barrett hand installed at the end-effector. A summary of the specifications of the robotic arm built are shown in Table 9.4.

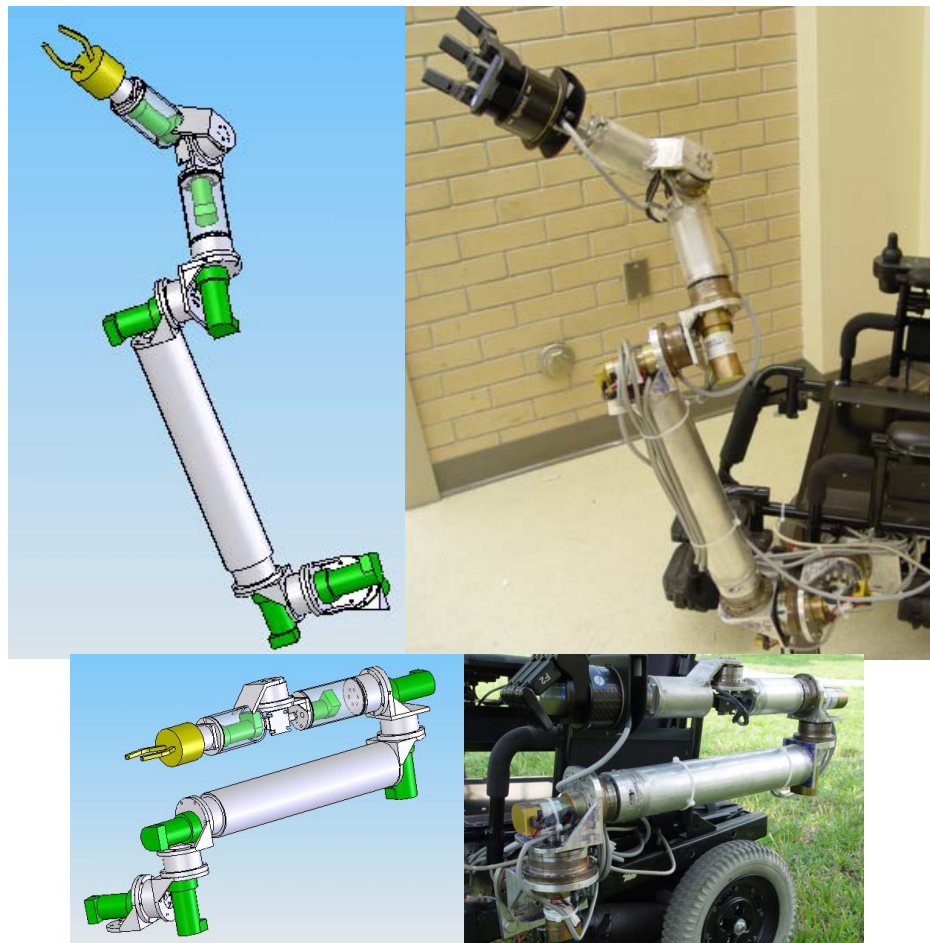


Figure 9.6: WMRA SolidWorks Models and the Corresponding Positions of the Built Device.

Table 9.4: Summary of the Robotic Arm Specifications.

Arm Mass	12.5 kg
Max reachable height above floor	1.37 m
Chair width increase with side mount	7.5 cm
Average Current Draw	2 A
Design Payload (including gripper)	6 kg
Deflection at design payload	13.3 mm
Degrees of Freedom	7
Actuator Type	Brush DC Servo
Transmission	Harmonic Drive
Controller Type	Pic-Servo SC

9.3. The New 2-Claw Ergonomic Gripper Design and Development

A new robotic gripper was designed and constructed for Activities of Daily Living (ADL) to be used with the new Wheelchair-Mounted Robotic Arm developed. Two aspects of the new gripper made it unique: one is the design of the paddles, and the other is the design of the actuation mechanism that produces parallel motion for effective gripping. The paddles of the gripper were designed to grasp a wide variety of objects with different shapes and sizes that are used in every day life. The driving mechanism was designed to be simple, light, effective, safe, self content, and independent of the robotic arm attached to it.

In designing a gripper, functionality is very important, and it remains one of the main factors considered in most robotics applications. If the design has good functionality, minimal cost, high durability, and the aesthetic characteristics are met, a good product is likely to be produced. In order to decide on a good design for a gripper, several aspects have to be inspected, such as the tasks required by the mechanism, size and weight limitations, environment to be used in as well as material selection. Some of the ADL tasks that will be performed using the gripper are opening doors, grasping a

glass to drink from, flipping on a light switch, pushing and turning buttons and knobs, holding books and similar objects, handling tiny objects such as a CD or loose sheets of paper, or holding a small ball.

9.3.1. Paddle Ergonomic Design

Specific considerations were taken in the attempt to optimize the functionality of the gripper. It was decided early on that the gripper would utilize parallel motion generated from a dual four bar mechanism attached to each side of the two fingers creating 8 links between the gripper surfaces and the driving mechanism itself. As a start, the gripper's fingers (paddles) were first put into consideration. Through the required tasks expected out of the overall device the gripper's surfaces were designed to be varied for the adequate handling and use of household objects mentioned. A rounded surface was implemented as shown in figure 9.7, which would give the gripper a soft look as well as good function while grasping objects.

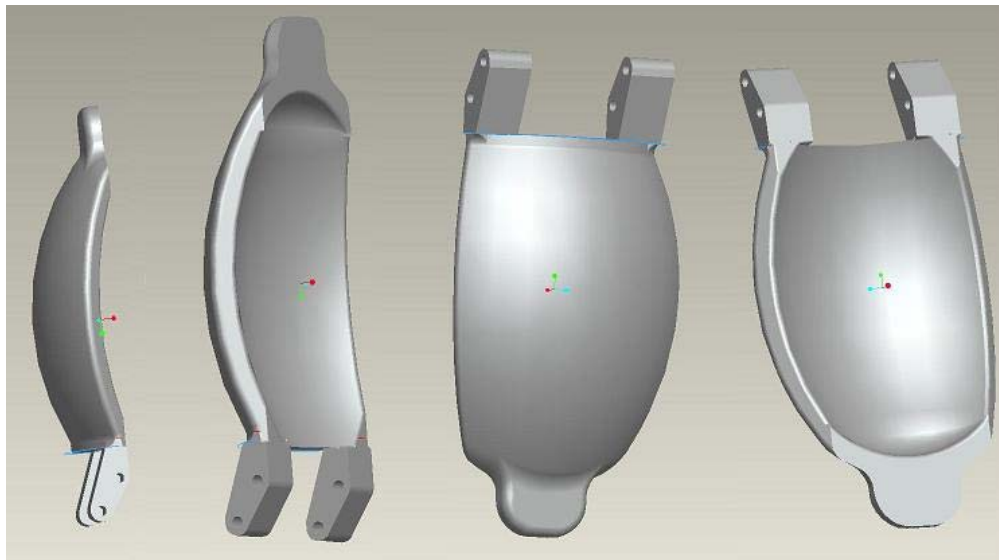


Figure 9.7: The New Gripper's Ergonomic Surfaces.

A spherically channeled surface was placed in the center of the paddle surface with the intention to contour to spherical door knobs. Small protrusions were added to the end of each paddle at the tip of the gripper for grasping smaller objects allowing added dexterity and the operations of press buttons and toggle switches. The tips were specifically made narrow for precision operations and rounded off to prevent the marring of surfaces that they would come in contact with. Optional protrusions extending toward the center of the grip at the tip of one of the paddles was added to allow objects such as door handles and door knobs to be pulled open with more security, rather than relying on friction and the locking of the mechanisms grip alone. The other paddle would have a small opening for the protrusions to go through when closing the gripper is required as seen in Figure 9.8.

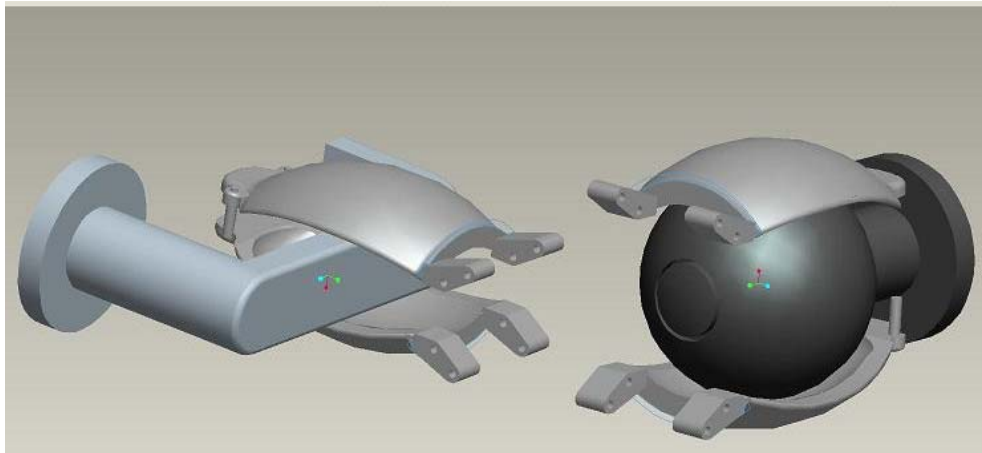


Figure 9.8: The Gripper Design in Application Reference.

It was later decided that an extra flat surface placed closer to the driver mechanism would be beneficial in grasping larger rectangular objects such as boxes or books. By relying on the finger tips of the gripper alone to grasp larger objects, a greater moment would be generated on the driving mechanism and higher stresses induced in the

links to achieve the same amount of gripping force attainable from a location closer to the driving mechanism itself. Figure 9.9 shows these changes to the paddles.

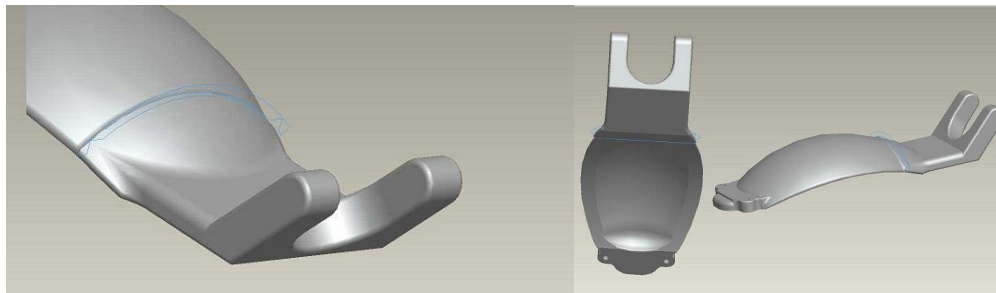


Figure 9.9: Extended Interior Surface Added to the Gripper.

As a final modification to the paddles, a spring hinge was added to the back of the flat paddle surface, near the hinge location, to allow for a small amount of torsional rotation. The thought behind this modification was for an added degree of freedom in the paddles to allow for a better grasp on tapered objects such as cups and for self-adjustment. Four main contact surfaces were intended for this gripper: The spherical area at the center of the paddles for spherical objects, the two round surfaces on both sides of the paddle for handling cylindrical and tapered objects, the two flat surfaces at the bottom and top of the paddles for handling rectangular and large objects, and the paddles' tips for handling small objects, switches, knobs and sheets of paper.

9.3.2. Actuation Mechanism

The driving mechanism was the next step in creating the gripper. As noted previously, the design was going to utilize four bar linkages to allow the paddles to open and close in a parallel motion. The main reasons for this were to increase the contact surfaces between the gripper and the handled object, and to prevent these objects from slipping out of the grasp of the paddles due to the angular change in the contact surfaces

caused by simpler pin joint gripper designs. By keeping the paddles parallel, a more predictable surface contact angle could be controlled which would allow larger objects to be grasped safely without the risk of being dropped. This modification was done and tested.

The first requirement for the gripper was for it to have a minimum gripping force of ten pounds and be capable of traveling from a full open position of four inches to a closed position within approximately four seconds. The gripper was also required to have an onboard motor for modularity reasons. The idea of utilizing an acme screw and a pull nut setup would be adequate for power transmission, and its compact size, relatively high variability in gear ratios, and its ability to lock the position without the use a mechanical brake mechanism made it a good choice for the purpose of this gripper. For this design a stainless steel 1/4-20 acme screw with a plastic nut was selected and thought to be the best design for space conservation and overall weight conservation as well. The selected motor carried relatively high torque to size ratio, and as a result, minimized the overall weight of the gripper dramatically. For the safety of the user, the handled object, and the mechanism, an adjustable slip-clutch was attached to the acme screw to build up the gripping force based on how delicate the object is, and to prevent the torque in the motor to rise above the designed limit of the mechanism.

9.3.3. Component Selection

The selected components are as follows:

- 1- The Motor: A 24 volt DC coreless gearhead servo motor was selected since the wheelchair can supply that voltage from its batteries. The diameter of the

selected motor was 0.67 inches having a length of 1.77 inches. This motor, made by Faulhaber, puts out a stall torque of 11.5 mNm with a maximum current of 190 mA and a maximum speed of 8000 rpm. This motor uses a 14-1 planetary gear ratio, and an optical encoder with 512 counts per revolution for the use of feedback control. Figure 9.10 shows the motor assembly with the gearhead and the encoder.



Figure 9.10: The Selected Coreless Gearhead Servo Motor.

- 2- Acme Screw and Pull Nut: A Stainless Steel 20 thread-per-inch acme screw was selected with a diameter of 0.25 inches to transmit the motion from the motor to the linkages through a Delrin plastic pull nut. This helps in locking the mechanism when the motor is stopped, and it gives a proper conversion of the motor speed to the required torque for driving the system.
- 3- Slip Clutch: An adjustable 0 to 50 oz-in slip clutch was selected to build up the grip force and slip in case the motor is still running while the required torque is reached. Figure 9.11 shows a drawing of the slip clutch
- 4- Spur Gears and Flange Ball Bearings: Two spur gears made out of anodized aluminum were selected with a pitch of 0.25 inch to transmit the motion from the motor shaft to the acme screw. A gear ration of 2:1 is used with 36 teeth,

9.5 mm diameter gear on the motor shaft and 72 teeth, 18.5 mm diameter gear on the acme screw.

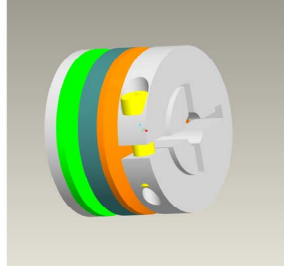


Figure 9.11: The Selected Slip Clutch.

Figure 9.12 shows the actual driving mechanism after assembly. Aluminum was the main component used in building the housing and shield of the mechanism and the links. Other components, including plastics and Teflon, were used as sleeves in the joints of the driving four-bar linkage mechanism.

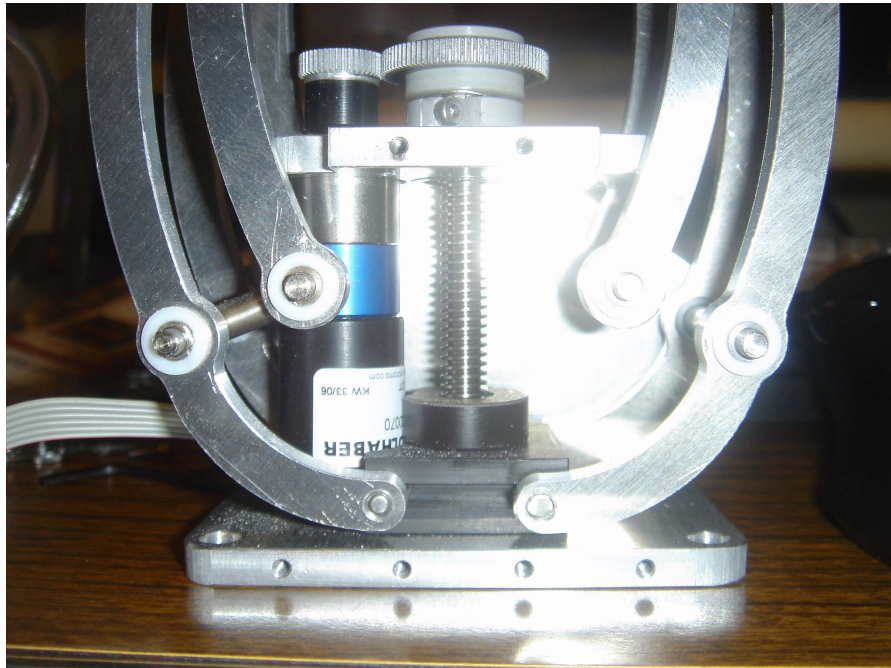


Figure 9.12: The Assembled Actuation Mechanism.

When all the side panels are in place and the top cap of the housing seal the compartment of the spur gears, safety of the operator is ensured in terms of getting any

external object caught in the driving mechanism. It also ensures proper protection of the motor and the small components driving the gripper from external dust and debris. Extensions on both sides of the gripper's base with extra holes were added for expandability in case other devices such as a camera and a laser range finder need to be mounted to the gripper's base plate. Figures 9.13 and 9.14 show the final Pro/E drawing and the physical gripper with the involved components after assembly, respectively.

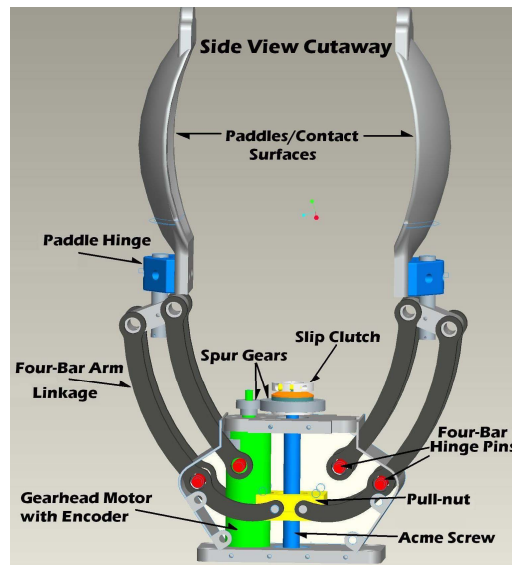


Figure 9.13: The New Gripper and the Actuation Mechanism Drawing.



Figure 9.14: The New Gripper and the Actuation Mechanism.

9.3.4. Final Design and Testing

Force analysis of the mechanism was accomplished by working from the paddles' contact surfaces through the mechanism linkages until reaching the electronic motor. The force considered in the design was 10 pounds of gripping force at the contact surfaces of the gripper. The force from the paddle surfaces was then translated through the parallel four-bar linkages to the pull-nut using static analysis. Teflon bushings were utilized in the hinges at this joint to reduce friction but accounted for while calculating the forces. The pull-nut static calculations were used to determine the required torque on the acme screw to generate the force needed at the pull-nut. This was accomplished relatively accurately by using the offered specifications by the manufacturer of the acme screw.

Input torque per output force measurements were utilized when calculating the torque required within the acme screw. Ball bearings were used to support the acme screw for maximum efficiency. After calculating the torque needed in the acme screw, forces were determined at the teeth of the spur gears used in the mechanism. The required torque and speed of the motor were calculated by assuming a required minimum opening and closing time of 4 seconds with the given force at the gripper. A safety factor of 2 was used in selecting a motor for the required torque.

Figure 9.15 shows a close-up view of the gripper, attached to the newly designed 9-DoF WMRA system on a power wheelchair, holding a 2.5 inch diameter ball. Several tests were conducted using the rapid prototype models and test objects to ensure proper application before the final design was reached. When the gripper machining was

completed and the gripper was assembled, actual grasping tasks commonly used in ADL were conducted.



Figure 9.15: The New Gripper When Holding a Spherical Object.

Another application tested show the adjustability of the paddles to the grasped object, as shown in figure 9.16. A standard cup was the test object to show adjustability of the paddles due to the added hinges that give them an extra degree of freedom for adjustment to the tapered object.



Figure 9.16: The New Gripper When Holding a Tapered Cup.

One of the main objectives intended for this gripper is the ability to handle different door handles. Figure 9.17 shows both kinds of handles, the lever handle and the

knob handle, commonly used in doors. These handles were used in this test to ensure proper application.



Figure 917: The Gripper When Opening a Door with a Lever Handle (Left) and a Knob Handle (Right).

Another test for handling small objects and sheets of paper were conducted. Figure 9.18 shows the gripper holding a business card using the tips of the paddles without the need to fully close the other end of the gripper.

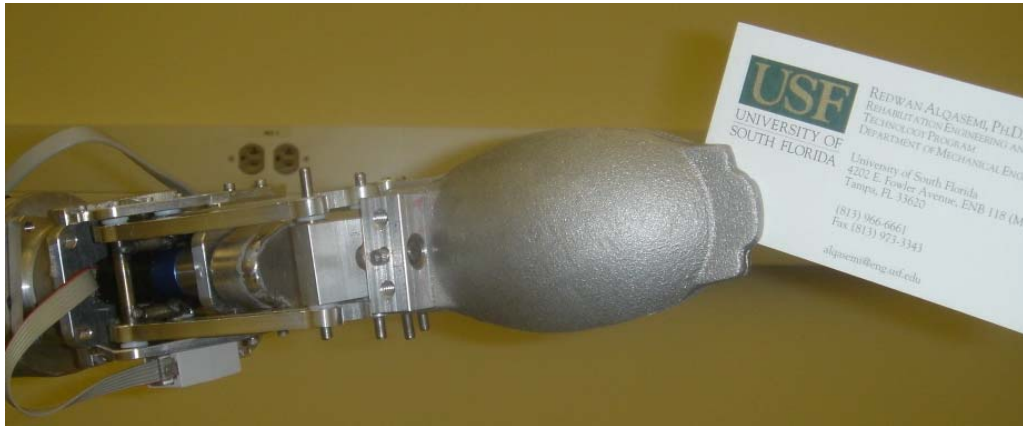


Figure 9.18: The New Gripper When Handling Small Objects.

Handling large objects can be challenging based on the geometrical complexity of that object. Figure 9.19 shows the gripper holding the box of heavy tools while moving it from one place to another. The two side-curved surfaces and the middle spherical surfaces help in supporting odd objects in case complex shapes are handled.



Figure 9.19: The New Gripper When Handling Large and Heavy Objects.

9.4. Modification of a Standard Power Wheelchair

To install the newly designed components on a power wheelchair for a complete WMRA system, modifications had to be made to a standard power wheelchair both in hardware and control. The selected power wheelchair was the “Action Ranger X Storm Series”. The wheelchair has been modified by adding an incremental encoder on each one of the wheels. The controller module of the wheelchair has also been modified using TTL compatible signal conditioner and a DA converter so that the signal going to the wheels can be controlled using the same PIC-Servo SC controllers used in the arm. The only difference is that the output from this control board used for the wheelchair is the PWM signal rather than the amplified analogue signal.

Since the wheelchair controller was sealed, and the manufacturers treat these controllers as proprietary components, we had to find a way to take over the control of the wheelchair. The best way to do this was by opening the joystick module and interfacing our control system with the joystick signal. The joystick sends two

independent analogue voltages to the wheelchair controller, one controls the forward speed of wheelchair (i.e. both wheels at the same speed and direction) and the other controls the rotation of the wheelchair (i.e. both wheels at the same speed but opposite directions). The voltage sent is as follows:

- 1- A voltage of 0.4 volts corresponds to a full positive speed.
- 2- A voltage of 2.6 volts corresponds to a stop and applies breaks.
- 3- A voltage of 4.0 volts corresponds to a full negative speed.

Any voltage between these values corresponds to slower motion of the wheels.

The controllers used in our WMRA system are capable of supplying pulse-width modulation (PWM) signal at 20 KHz. Changing the duty cycle means changing the average of the signal. A circuit that converts a constant-frequency PWM signal is shown in figure 9.20. Two independent circuits like these can be connected between the WMRA controller and the joystick of the wheelchair so that the wheelchair can be controlled using the arm controller.

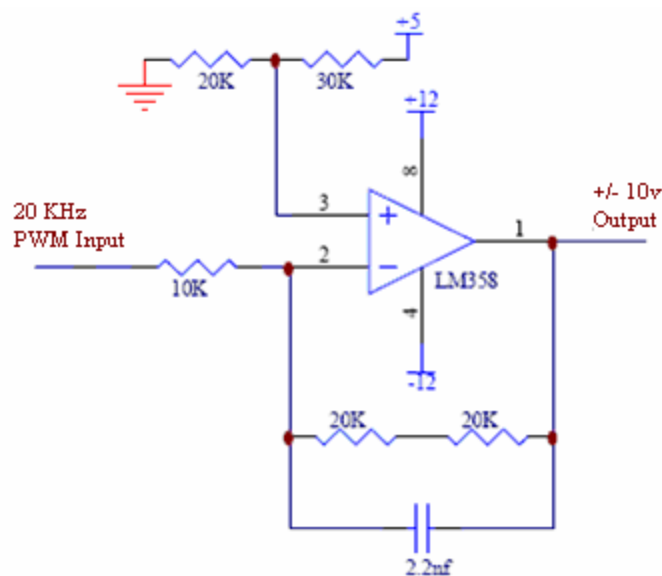


Figure 9.20: A Circuit Designed to Convert Digital PWM Duty-Cycle Control Signal to Analogue Signal.

A new controller box was designed to fit 12 controller boards, two power adapters, one converter, a cooling fan, and the connecting cables. Figure 9.21 shows the box before attaching it to the power wheelchair. This box was built so that it is easy to take off and put on the wheelchair with quick connectors that can be disconnected from the arm and the power supply to the battery.

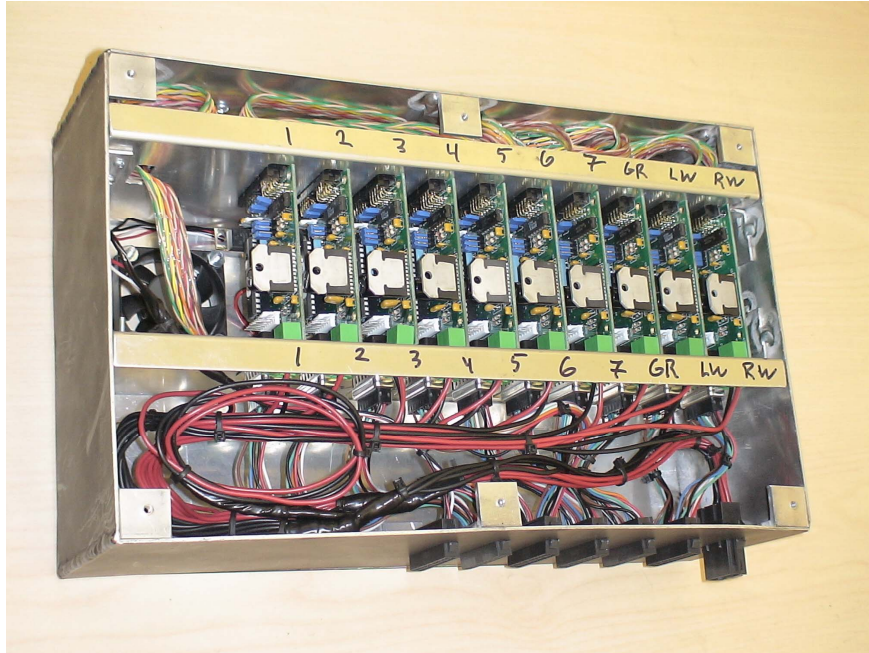


Figure 9.21: The Designed Controller Box Installed on the Modified Wheelchair.

Another item attached to the power wheelchair was the quick-release mechanism, shown in figure 9.22, that is permanently attached to the power wheelchair and can quickly mount or dismount the designed robotic arm into or out of the wheelchair. This mechanism allows the user to quickly detach the arm if the wheelchair needs to be transported in a small container or a minivan that does not fit the WMRA system. Cable connectors extending from the robotic arm to the controller box are designed to quickly disengage the power and logic to and from the arm and the controller box. This also allows for easier portability that can be done by an average person.

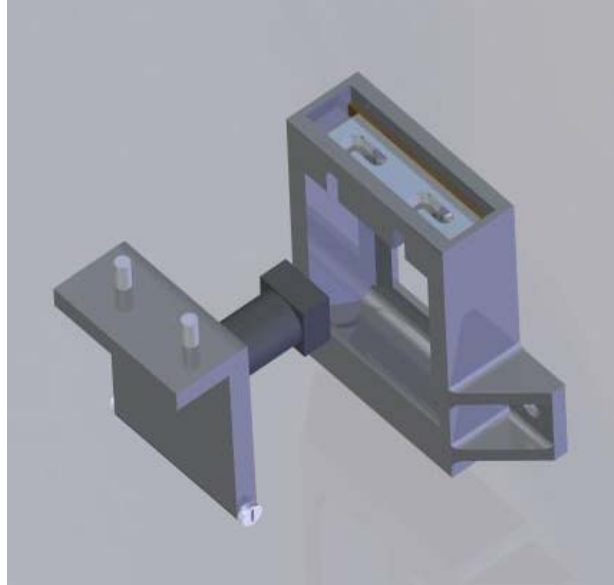


Figure 9.22: The Quick-Release Mechanism that Mounts the Robotic Arm on the Wheelchair.

9.5. Controller Hardware

The controller hardware was designed to control all joint servo motors simultaneously through an amplified analogue signal, and to control the motors of the differential drive of the wheelchair through PWM signal. Wiring of the boards to the individual motors was done using quick-release sockets.

9.5.1. Controller Boards

PIC-SERVO SC controllers that support the DC servo actuators were chosen as shown in figure 9.23. At 5cm x 7.5cm, this unit has a microprocessor that drives the built-in amplifier with a PWM signal, handles PID position and velocity control, communicates with RS-485, and can be daisy-chained with up to 32 units. It also reads encoders, limit switches, an 8 bit analogue input, and supports coordinated motion control. Each joint controller is individually addressable, and can be controlled in

position, velocity, or current (torque) mode. In position mode, velocity and acceleration limits may be specified for smooth operation. Data for the entire arm is interfaced to the main computer using a single serial link. The PIC-Servo SC controllers use RS-485, and a hardware converter interfaces this with the RS-232 port or a USB port on the host PC. The current host PC is an IBM laptop, running Windows XP. However, the communications protocol is simple and open, and could be adapted to virtually any hardware/software platform with an RS-232 or a USB port. These controller boards were all connected to the computer using a single cable.



Figure 9.23: JRKERR PIC Servo SC Controller Boards.

9.5.2. Communication and Wiring

As shown in figure 9.24, PIC-SERVO SC controllers (C1 through C7) that support the DC servo actuators (J1 through J7) were integrated in the control box. The logic of the boards run through 12v DC power converted from the wheelchair's batteries.

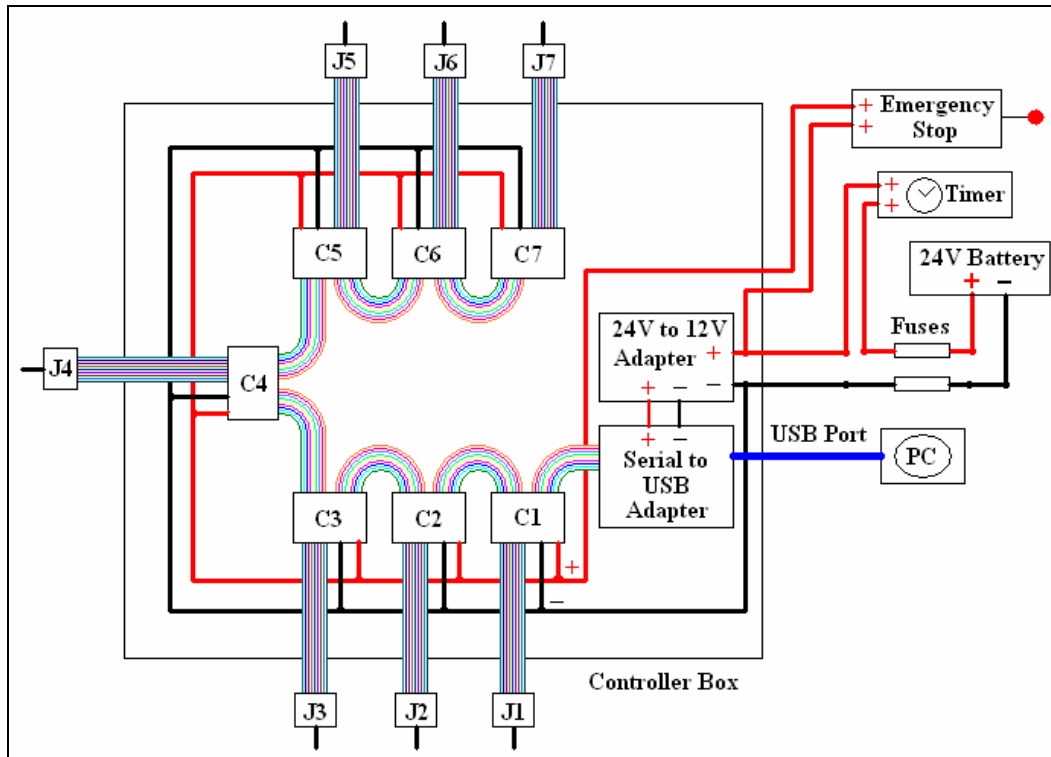


Figure 9.24: Control System Circuitry.

The seven motors used to actuate the power wheelchair were connected via a serial port as shown in figure 9.25 (left), and the single servo motor used for the gripper was connected to the controller boards using a different serial port as shown in figure 9.25 (right). In figure 9.26, the circuit that connects the wheelchair encoders to the controller boards is shown. A toggle switch will be added to the joystick supplied with the wheelchair so that it can still be used if the user wants to run the wheelchair regularly.

9.5.3. Safety Measures

Two safety measures were added to the hardware of the WMRA system. The first is a panic stop button that is connected and situated under the right elbow of the user to stop the motor power supply from its battery source without shutting off the logic power.

This way, the system can run back up, or a diagnostic procedure can detect any problems that may have happened. Another safety feature is the use of a timer that cuts off the power to the motors and to the logic circuits so that the batteries of the power wheelchair can conserve energy in case the user forgot to shut the system off.

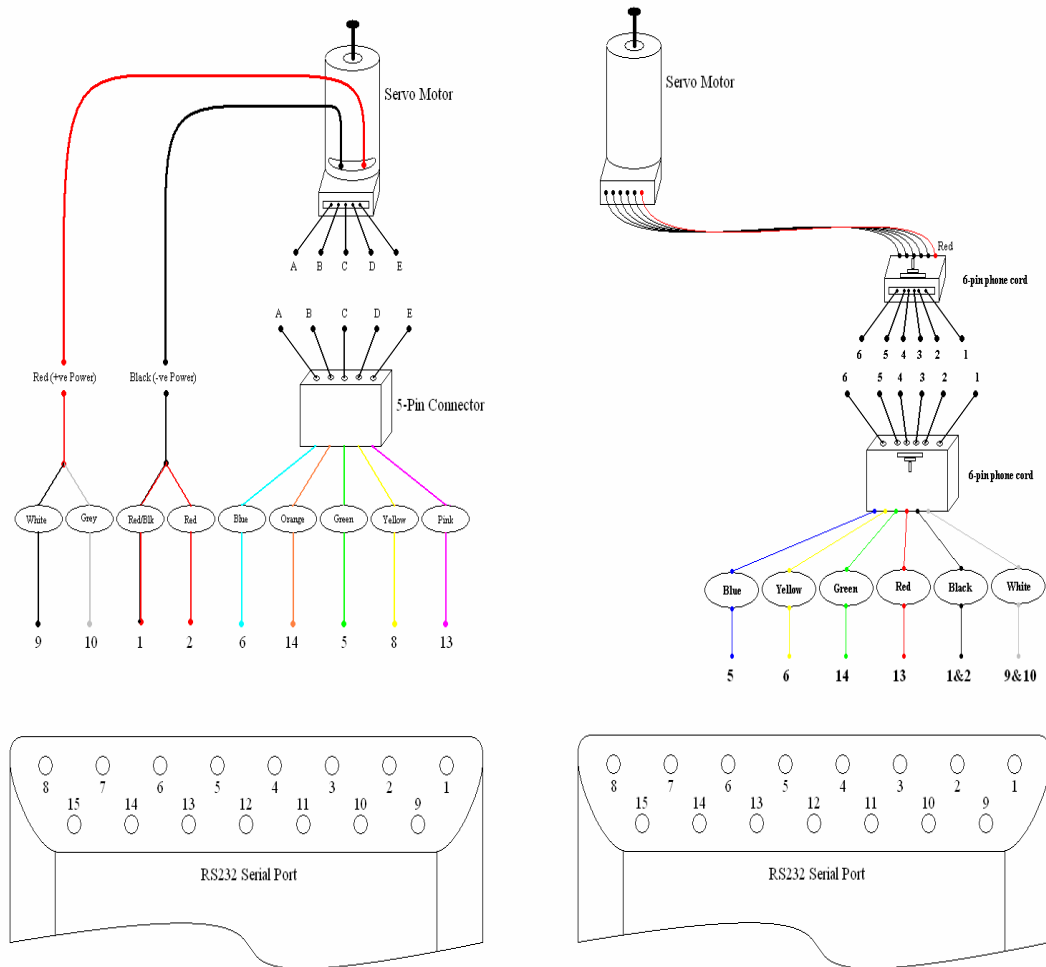


Figure 9.25: Serial Port Connection of the Joint Motors (Left) and the Gripper (Right).

9.6. Experimental Testing

The newly designed WMRA was put to test in its early stages when the robotic arm was ready for testing. Even though wheelchair modification is still undergoing, we

were able to run the control algorithm to move the arm only as we had this in one of the user options in the control software. Figure 9.27 shows the WMRA system with the Barrette hand installed and a video camera used by a person affected by Guillain-Barre Syndrome. In her case, she was able to use both the computer interfaces.

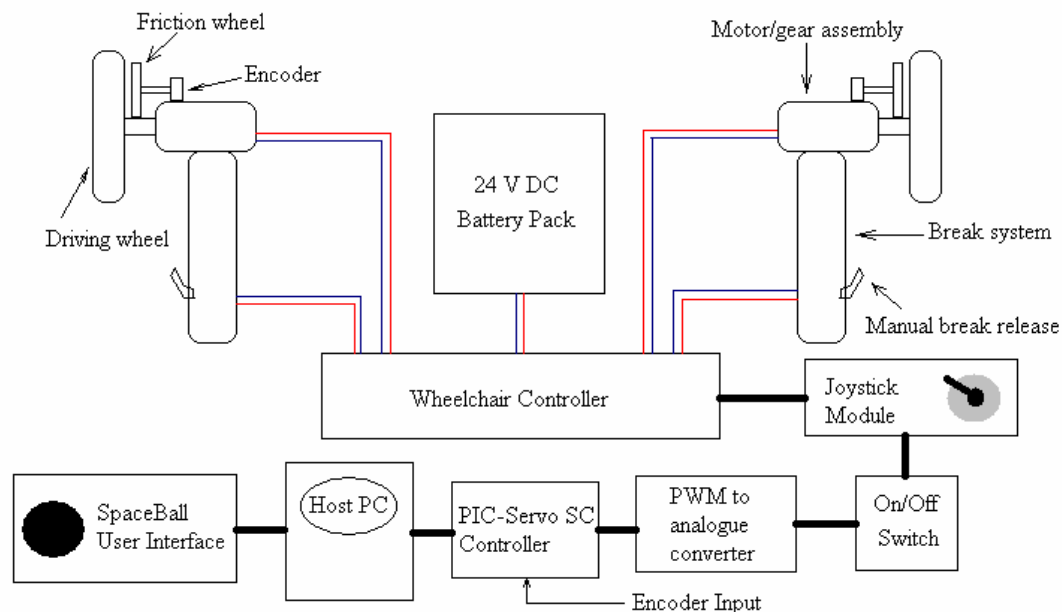


Figure 9.26: Wheelchair Encoders and Control Communications.

The robotic arm was also tested with an able bodied human subject using the Brain-Computer Interface (BCI-2000) with the newly designed gripper as shown in figure 9.28. The user was able to move the robotic arm without touching any of the controls by looking at the feedback screen and counting the number of flashes of the particular direction or choice displayed on the screen. This was a successful test of this interface that encountered some unanticipated problems. When the user sits on the wheelchair, which is within the electromagnetic field of all the running wires inside the WMRA system, the BCI sensors were picking up a lot of noise and magnifying them along with the brain signal. This reduced the accuracy of the user's choice recognition,

but it was good enough for him to execute the task he was trying to do without the need to step off the wheelchair. This noise might be reduced if the BCI-2000 gains were trained on the user while sitting on the wheelchair..



Figure 9.27: A Person with Guillain-Barre Syndrome Driving the New WMRA System.

Another user interface tested with this WMRA system was the touch screen. It was one of the most convenient control interfaces that we tested if the user is able to hold on to the stylus pen and touch the screen icons with it. In autonomous mode, the arm was also tested by commanding the controller to drive it from one point to another with a specific trajectory, and it moved the arm at that trajectory and returned back to the ready position as needed. Other physical tests conducted include the use of end-effector Cartesian velocity profile inputs to move the arm for a specified period of time. It is noteworthy at this point to mention that the electronics used to control the WMRA system stop responding occasionally with no apparent reason and at random without specific conditions. Overall, the system was functioning as designed, and it was able to execute ADL tasks with different user interfaces. More field testing will be conducted later, and the results will be published.



Figure 9.28: A Human Subject Testing of the BCI-2000 Interface with the WMRA System.

9.7. Summary

In this chapter, the design of a new 7-DoF robotic arm was presented. The component selection was discussed, and the final product testing was described along with the specifications of the device. A new gripper that was designed specifically for activities of daily living was presented. Special claw design procedure and features were presented. Each feature represents a specific use for task execution and grasping. The actuation mechanism of the gripper was designed, and proper components were selected to provide sufficient power to grasp the desired objects.

A standard power wheelchair was modified to hold the robotic arm and the controller box and the associated hardware. Two optical encoders were added to the wheels of the power wheelchair to give feedback to the controller when moving the wheelchair for a closed loop control. The controller hardware that controls the seven joint motors of the robotic arm, the two wheels of the power wheelchair, and the motor of the gripper was shown. This controller is capable of running up to 32 controller boards that are daisy-chained to form a single interface to the computer. Operator safety was addressed by adding panic stop button and a timer to turn the system power off.

Testing of the WMRA with human subjects was conducted to ensure proper operation of the system. Several user interface options were tested as well, and the results were satisfactory.

Chapter 10: Conclusions and Recommendations

10.1. Overview

Extensive analysis was conducted to combine the WMRA's 7-DoF and the wheelchair's mobility in the new redundant 9-DoF system. This redundancy was used and optimized to improve manipulation capabilities for activities of daily living (ADLs) and avoid obstacles, joint limits and singularities. The new system was capable of executing pre-set tasks autonomously as well as in teleoperation mode. A real-time controller was developed and implemented to provide high frequency inverse kinematics update rates and real-time sensory feedback for effective closed-loop control of the WMRA system.

The control algorithm was implemented in Virtual Reality simulation to test its ability to provide a good and comprehensive control structure that can be used by persons with disabilities.

A newly built modular WMRA was used. It was developed based on manipulation trajectories needed for activities of daily living. This WMRA utilized an optimized controller for both WMRA and the power wheelchair. A standard power wheelchair was modified to include PC based control and sensory feedback.

A keypad, a Spaceball, a touch screen, and a Brain-Computer Interface (BCI) were used as modular user interfaces with different capabilities for each input device to fit the individual user needs and capabilities. Future testing will determine the appropriate interface needed for a specific disability. Higher level control algorithms were developed to interface the sensory data and the user input for an easy control of the system.

Testing procedures were developed for both simulation and experimental testing on the developed testbed. That testbed was created to conduct the necessary testing of the system in realistic environments (Home, Office, etc.). Several US patents were planned for many parts of this work.

10.2. General Discussion

A 7-DoF robotic arm and a 2-DoF non-holonomic wheelchair were mathematically modeled for kinematic control. A combination of the two mathematical models created a new 9-DoF redundant manipulator that combined the mobility and manipulation. The control system was designed for coordinated Cartesian control with singularity robustness and task-optimized combined mobility and manipulation. Weighted Least Norm solution was implemented among others for preference control of each of the joints and the wheelchairs' position and orientation.

The control algorithm utilized redundancy to optimize the motion based on different criteria functions or user-defined weights of preference. It was noticed that the use of conventional optimization methods resulted in unintended motion that may turn into undesirable move potentially harming the human user or the WMRA hardware. These methods add an optimization term that can still be active even when the user is not

commanding the arm to move. Even though these conventional methods were kept in the control algorithm for the user to choose, it was found that the Weighted Least Norm solution with the new modifications added to it gave the most predictable and robust control algorithm that resulted in a smooth motion with joint limit avoidance and user-preferred motion weights.

A wheelchair-mounted robotic arm (WMRA) was designed and built to meet the needs of mobility-impaired persons, and to exceed the capabilities of current devices of this type. The mechanical design incorporates DC servo drive with actuators at each joint, allowing reconfigurable link lengths and thus greater adaptability to a range of workspaces. Nine principal degrees of freedom allow full pose control of both the wheelchair and the arm. The used control electronics are capable of controlling up to 32 devices when daisy-chained together, and it is capable of reading different sensory feedback and supplying it to the control software. Reliability of these electronics proved unpredictable since it showed some failure with no specific reason or pattern for diagnostics.

A new gripper was designed specifically to be used for activities of daily living. The design includes two ergonomic claws with designated surfaces for handling specific shapes and objects. The actuation mechanism was designed to be light, effective and safe at the same time. Interfacing the gripper with the robotic arm and controlling it using the same controllers used to control the arm were some of the features included in the design and component selection process.

Modularity in both the hardware and software levels allows multiple input devices to be used to control the system. User interfaces include the SpaceBall, the keyboard and

mouse, a touch screen, or the Brain-Computer Interface (BCI) used for people with disabilities. Any other preferred device can be used easily since the control software is flexible enough to allow any other user-interface hardware to be added.

Simulation testing of the new control algorithm was conducted using Matlab and Virtual Reality toolbox among other C++ programs. Modular functions with proper interfaces were designed to ease the addition to any future developments that might be needed. The results showed a powerful method of controlling this 9-DoF combined WMRA system. Simulation results were also shown to emphasize on the effectiveness of the methods.

The use of simulation and hardware testing showed successful integration between the mobility and manipulation mathematically and in the real application. When tested with the actual robotic arm, there were some unpredictable moments of unreliability between Matlab functions and C++ functions that resulted in the loss of motion in few occasions. This may be due to the lack of good compatibility between Matlab program and the DLL library that was compiled using C++. This problem can very likely be taken care of if the control software of the actual WMRA is done separately on a similar program done in C++.

The following is a list of the major contributions made in this dissertation:

- 1- Design and development of a 9-DoF wheelchair-mounted modular robotic arm system.
- 2- Design and development of an ergonomic gripper to be used for ADL tasks.
- 3- Development of a complex inverse-kinematics algorithm that combines the mobility of non-holonomic motion and the manipulation of redundant

manipulators for a complex 9-DoF control system with all the associated details.

- 4- Expand the WLN method with the S-R inverse for a new control method that is robust and reliable in real applications.
- 5- Utilization of redundancy for joint limit avoidance of such complex systems through optimization.
- 6- Development of a powerful and modular simulation tool using Virtual Reality and friendly graphical user interface to model and simulate the 9-DoF WMRA system with theory implementation.
- 7- The implementation of the theory on the actual WMRA hardware, and the resolution of all communication and interface challenges.
- 8- The use of the BCI system to control WMRA for people with severe disabilities.

10.3. Recommendations

Going back to the control method, it would add more enhancements to the control algorithm if an accurate mathematical model of the human subject along with the wheelchair and the surrounding obstacles are available. This way, the WMRA motion would change the configuration to avoid these objects rather than stop the system. The implemented safety measures and conditions can still be kept in case the system comes close to singularity or goes out of control.

In the hardware part, better control boards may eliminate the occasional unpredictable failure of the WMRA system to respond. While the daisy-chaining

capabilities of these control boards give the system better connection to the computer through a single wire, performance was greatly affected and was noticeable. When the WMRA system was commanded to execute a task, the commanded joint positions and velocities were sent to the boards for execution. The problem with that was the fact that the command goes from the first board to move its joint, to the last board, and by the time the last joint moves, the first joint would already have finished its motion. This introduced some uneven periods of motion among the joints. A single and more compact board to control all ten motors may replace the ten boards in use currently whenever they are commercially available. It is also recommended to change the flexible coupling in joint 6 to a rigid coupling to avoid slippage.

In the simulation side of this work, a separation of the physical WMRA system control and the simulation control may be done to convert the control software to C++ only rather than the combination of Matlab and C++ together. While Matlab's Virtual Reality simulation is impressive, using it to control the robotic arm in a conventional operating system introduced undesirable delays. Separating the two softwares would allow the user to use the actual WMRA system more efficiently, and at the same time allow the system to be work in powerful virtual environment for testing and development of new implementations to the system.

User interfaces were also using C++ applications under Windows operating system. When used with Matlab, it was necessary to interface these two programming packages together either by intermediate programs that link the information between the two packages, or by assigning virtual ports and sending the data and information through

these ports. In both cases delays were introduced in the system, and some times, it even froze the computer.

The current control system is sufficient for low velocities, which is what persons with disabilities need to perform their ADL tasks. Expanding the mathematical representation of the WMRA system to include a full dynamic model and gravity compensation can help the user to perform other tasks that require high velocities such as sports and recreational activities.

Chapter 11:

Future Work

11.1. Introduction

This WMRA system has the potential to be one of the leading assistive device projects in the country. Several patents were planned for many aspects of this work. Commercializing this WMRA system would benefit many people with disabilities who find themselves physically dependent on other that may or may not be willing to provide the best help possible. Several steps can be taken to ensure an effective system that can be widely used with many wheelchair-bound individuals. This chapter gives a glimpse of what can be done in the future to add more capabilities and ease of use to this WMRA system.

11.2. Quick Attach-Detach Mechanism

This is an ongoing work that is aiming to have the robotic arm and the controller hardware detached and attached quickly with minimum efforts. The idea is to allow a single individual the ability to attaching or detaching the robotic arm and the controller box with all the wires and cables to and from the wheelchair. It was noticed with both Manus and Raptor that when they are attached to a power wheelchair, transportation of the wheelchair becomes a problem, even when a power lift is used with a custom-

designed wheelchair transportation van. Manus already has a quick-release mechanism to detach the arm from its hosting wheelchair. The ongoing efforts in this project will employ a mechanism to attach the arm to the wheelchair using the weight of the arm to slide it into place and lock the system solid. Quick connection cables are also designed to remove the cables from the controller box and detach the whole control system from the wheelchair. This will allow the user to transport both the wheelchair and the robotic arm independently and with minimum effort.

11.3. A Single Compact Controller

In the current design, each motor to be controlled uses a dedicated controller board to send the commanded position and velocity to that motor. This resulted in ten controller boards so far, seven for the seven robotic joints, two for the wheelchair and one for the gripper. An additional board was used to take the signals from all ten daisy-chained controller boards to the computer's serial port or USB port on a single cable.

Currently, we are seeing huge advancements in microelectronics and micro-processing. Having a single board that is capable of simultaneously controlling all ten motors without the serial connection delays will give a better performance to the whole system. This will also affect the size of the controller box. The current controller-boards box is significant in its size housing the ten control boards, a converter board and power adapter, and was mounted under the seat cushion of the power wheelchair. Having a significantly smaller box may eliminate the need for having it mounted on the wheelchair and keep it on the robotic arm itself. This will reduce the need for many connectors and

cables between the wheelchair and the robotic arm, and it will certainly make it lighter and more self-sufficient.

11.4. Sensory Suite

It is essential for an intelligent robotic system to carry in its sensory suite many different sensors. In this WMRA system, ten optical encoders were installed for joint angle measurements. A laser range finder and a digital camera are two other sensors that are to be added. The laser range finder will be used for object-following or for determining object coordinates based on the read distance and the current orientation of the laser range finder. The camera will be used for navigation feedback to the user as well as for object recognition and tracking in the plane.

Other sensors, such as proximity sensors, will be used for on-line obstacle avoidance and for guidance through narrow pathways. Force-torque sensors will be added at the gripper's base to provide force feedback to enhance the manipulation of different objects.

11.5. Real-Time Control

When operating the WMRA system using Windows XP operating system, time and priority assignments are uncontrollable by the programmer. Doing the control under a real-time operating system such as QNX allows the programmer to control the priorities and set priority rules to operate the WMRA as well as any other software or hardware used by the computer. This will enhance the response time of the system and make the

programmer run the system at higher frequencies without compromising the accuracy of the WMRA system or the operating system.

11.6. Bluetooth Wireless Technology for Remote Wireless Teleoperation

Bluetooth wireless is being integrated to the system to add remote wireless teleoperation so that the user can perform some ADL tasks while not seated on the wheelchair. The current USB and serial connections between the WMRA system and the control software on the tablet PC will be made wireless through special adaptors that will convert the signal from its current protocol to Bluetooth protocol and back at the computer terminal. This will allow the user to detach the tablet PC from the wheelchair when he/she is not using it. For instance, if a user with severe disabilities wakes up in the morning in need of a drink or a snack, that person will usually wait until the designated aid or family member comes to the room for assistance. Having the ability to control the WMRA system remotely allows him/her to drive the WMRA around the house by operating the system through a selected user interface and looking at the camera view through the tablet PC monitor. When the fridge is reached, the operator would be able to use the arm to open the door, get the desired drink and come back to the room with no need to wait for a human aid.

11.7. Sensor Assist Functions (SAFs)

Sensor Assist Functions (SAFs) will be used to assist or resist user's motion based on the trajectory generated to execute the intended task and the motion input coming from the user interface. Velocity scaling teleoperation, force reflection, varying

impedance parameters, and visual servoing for object grasping will be used to enhance the manipulation capabilities of persons with disabilities. When using a haptic device, the user can be trained to perform better in ADL tasks by starting with the assist functions and slowly releasing them with time as the user gets used to proper control.

11.8. Pre-Set ADL Tasks

Programming several tasks to the current WMRA system is quite straight forward. The plan is to program commonly used tasks for each individual using the customized WMRA system so that these tasks can be done autonomously as the user selects them. Several tasks will be included in this WMRA system as follows:

- 1- Turning switches on and off.
- 2- Operating an oven, washer, dryer, microwave, dishwasher, etc.
- 3- Opening and going through spring-loaded doors when the door dimensions are according to common standards
- 4- Object-following task as the camera and/or the laser range finder guide the WMRA system to autonomously follow that object or human.
- 5- Inserting CDs/diskettes into the computer or the CD player.

These pre-programmed ADL tasks are among many others that can be programmed to execute at the user's request. A good scenario of such tasks is when a user is in a hallway or a room and would like to go outside that room through a spring-loaded door. The user can point the attached laser range finder to the door handle and press the assigned button. The control system will start the autonomous mode while keeping the teleoperation mode running. The autonomous operation will start by

calculating the coordinates of the door knob from the given distance from the laser and the given laser orientation from the optical encoders and forward kinematics of the WMRA system. Once the door knob location is fully defined, the WMRA control system moves the wheelchair to a close proximity from the door at certain pre-calculated angle, reach to the door knob using the arm, grasp it using the end-effector, open the door and backup the wheelchair from the door way with a resultant circular motion at the gripper to match the door handle trajectory while opening. The system can then advance the wheelchair while holding the door using the robotic arm, and drive the wheelchair through the door until it is clear, and then release the door. The autonomous mode will then stop until the next pre-set operation is requested.

References

- [1] US Census Bureau (1997), "Disabilities Affect One-Fifth of All Americans", Census Brief, CENBR/97-5, <http://www.census.gov/prod/3/97pubs/cenbr975.pdf>, 1997.
- [2] J. Reswick, "The Moon Over Dubrovnik - A Tale of Worldwide Impact on Persons with Disabilities", Advances in External Control of Human Extremities, 1990.
- [3] R. Murphy, "Introduction to AI Robotics", MIT Press, 2nd edition, 2002.
- [4] J. Allen, A. Karchak, and E. Bontrager, "Design and Fabrication of a Pair of Rancho Anthropomorphic Arms", Technical Report of the Attending Staff Association of the Rancho Los Amigos Hospital Inc, 1972.
- [5] T. Rahman, S. Stroud, R. Ramanathan, M. Alexander, R. Alexander, R. Seliktar, and W. Harwin, "Consumer Criteria for an Arm Orthosis", Applied Science and Engineering Laboratories, www95.homepage.villanova.edu/rungun.ramanathan/publications/t_and_d.pdf, 2000.
- [6] H.F.M. Van der Loos, VA/Stanford Rehabilitation Robotics Research and Development Program, "Lessons Learned in the Application of Robotics Technology to the Field of Rehabilitation", IEEE Transactions on Rehabilitation Engineering, V. 3, N. 1, pp. 46-55, 1995.
- [7] M. Topping, "An Overview of the Development of Handy 1, a Rehabilitation Robot to Assist the Severely Disabled", Journal of Intelligent and Robotic Systems, V. 34, N. 3, pp. 253-263, 2002.
- [8] M. Topping, H. Heck, G. Bolmsjo, and D. Weightman, "The Development of RAIL (Robotic Aid to Independent Living)", Proceedings of the third TIDE Congress, 1998.
- [9] J. Dallaway, and R. Jackson, "RAID - A Vocational Robotic Workstation", Proceedings of the IEEE International Conference on Rehabilitation Robotics (ICORR), 1992.

- [10] Robotic Assistive Device, Neil Squire Foundation, <http://www.neilsquire.ca/rd/projects/RobotApp.htm>.
- [11] N. Katevas, "Mobile Robotics in Health Care Services", IOS Press Amsterdam, pp. 227-251, 2000.
- [12] H. Yanco, "Integrating Robotic Research: A Survey of Robotic Wheelchair Development", AAAI Spring Symposium on Integrating Robotic Research, Stanford, California, 1998.
- [13] P. Warner, and S. Prior, "Investigations into the Design of a Wheelchair-Mounted Rehabilitation Robotic Manipulator", Proceedings of the 3rd Cambridge Workshop on Rehabilitation Robotics, Cambridge University, England, 1994.
- [14] S. Sheredos, B. Taylor, C. Cobb, and E. Dann, "The Helping Hand Electro-Mechanical Arm", Proceedings of RESNA, pp. 493-495, 1995.
- [15] M. Hamilton, "The Weston Wheelchair Mounted Assistive Robot - The Design Story", Robotica, V. 20, pp. 125-132, 2002.
- [16] G. Bolmsjö, M. Olsson, P. Hedenborn, U. Lorentzon, F. Charnier, H. Nasri, "Modular Robotics Design - System Integration of a Robot for Disabled People", EURISCON, 1998.
- [17] B. Borgerding, O. Ivlev, C. Martens, N. Ruchel, and A. Gräser, "FRIEND - Functional Robot Arm With User Friendly Interface For Disabled People", The 5th European Conference for the Advancement of Assistive Technology, Düsseldorf, Germany, 1999.
- [18] H.F.M. Van der Loos, "Lessons Learned in the Application of Robotics Technology to the Field of Rehabilitation", IEEE Transactions on Rehabilitation Engineering, V. 3, N. 1, pp. 46-55, 1995.
- [19] M. Pauly, "TAURO - Teleautomated Service Robot for the Disabled", Automated Mobile Systems, pp. 30-39, 1995.
- [20] N. Hogan, H. Krebs, J. Charnnarong, P. Srikrishna, and A. Sharon, "MIT-MANUS: A Workstation for Manual Therapy and Training", Proceedings of the 1992 IEEE International Workshop on Robot and Human Communication, pp. 161-165, Tokyo, Japan, 1992.
- [21] H. Takanobu, A. Takanishi, D. Ozawa, K. Ohtsuki, M. Ohnishi, and A. Okino, "Integrated Dental Robot System for Mouth Opening and Closing Training", Proceedings of the 2002 IEEE International Conference on Robotics and Automation (ICRA), V. 2, pp. 1428-1433, 2002.

- [22] D. Tougaw, C. Polito, P. Weiss, and J. Will, "Sponsoring a FIRST Robotics Team", Proceedings of the 2003 ASEE IL/IN Section Conference. pp. 60-62. 2003.
- [23] H. Efring, and K. Boschian, "Technical Results from Manus User Trials", Proceedings of the 1999 IEEE International Conference on Rehabilitation Robotics (ICORR), pp. 136-141, 1999.
- [24] M. Hillman, and A. Gammie, "The Bath Institute of Medical Engineering Assistive Robot", Proceedings of the 1994 IEEE International Conference on Rehabilitation Robotics (ICORR), pp. 211-212, 1994.
- [25] P. Chang, "A Closed-Form Solution for Inverse Kinematics of Robot Manipulators with Redundancy", IEEE International Journal of Robotics and Automation, V.3, N. 5, 1987.
- [26] S. Khadem, and R. Dubey, "Global Redundant Robot Control Scheme for Obstacle Avoidance", Proceedings of the 1988 IEEE Southeast Conference. pp. 397-402, Knoxville, Tennessee, 1988.
- [27] T. Chan, and R. Dubey, "A Weighted Least-Norm Solution Based Scheme for Avoiding Joint Limits for Redundant Joint Manipulators", IEEE Robotics and Automation Transactions (R&A Transactions). V. 11, N. 2, pp. 286-292, 1995.
- [28] S. McGhee, T. Chan, R. Dubey, and R. Kress, "Probability-Based Weighting of Performance Criteria for a Redundant Manipulator", Proceedings of the 1994 IEEE International Conference on Robotics and Automation (ICRA). V. 3, pp. 1887-1894, San Diego, California, 1994.
- [29] L. Beiner and J. Mattila, "An Improved Pseudoinverse Solution for Redundant Hydraulic Manipulators", Robotica, V. 17, pp. 173-179, 1999.
- [30] E. Zergeroglu, D. Dawson, I. Walker, and P. Setlur, "Nonlinear Tracking Control of Kinematically Redundant Robot Manipulators", IEEE/ASME Transactions on Mechatronics, V. 9, N. 1, pp. 129-132, 2004.
- [31] W. Kwon, B. Lee, and M. Choi, "Resolving Kinematic Redundancy of a Robot Using a Quadratically Constrained Optimization Technique", Robotica, V. 17, pp. 503-511, 1999.
- [32] L. Ellekilde, P. Favrholdt, M. Paulin, and H. Petersen, "Robust Inverse Jacobian Control with Joint Limit and Singularity Handling for Visual Servoing Applications", The International Journal of Robotics Research, 2006.

- [33] C. Perrier, P. Dauchez, and F. Pierrot, "A Global Approach for Motion Generation of Non-Holonomic Mobile Manipulators", Proceedings of the 1998 IEEE International Conference on Robotics and Automation (ICRA), V. 4, pp. 2971-2976, Leuven, Belgium, 1998.
- [34] H. Osumi, M. Terasawa, and H. Nojiri, "Cooperative Control of Multiple Mobile Manipulators on Uneven Ground", Proceedings of the 1998 IEEE International Conference on Robotics & Automation (ICRA), 1998.
- [35] Q. Huang, S. Sugano, and K. Tanie, "Motion Planning for a Mobile Manipulator Considering Stability and Task Constraints", Proceedings of the 1998 IEEE International Conference on Robotics and Automation (ICRA), 1998.
- [36] Y. Yamamoto, and X. Yun, "Unified Analysis on Mobility and Manipulability of Mobile Manipulators", Proceedings of the 1999 IEEE International Conference on Robotics & Automation (ICRA), pp. 1200-1206, Detroit, Michigan, 1999.
- [37] M. Egerstedt, and X. Hu, "Coordinated Trajectory Following for Mobile Manipulation", Proceedings of the 2000 IEEE International Conference on Robotics & Automation (ICRA), 2000.
- [38] A. Mohri, S. Furuno, M. Iwamura, and M. Yamamoto, "Sub-Optimal Trajectory Planning of Mobile Manipulator", Proceedings of the 2001 IEEE International Conference on Robotics & Automation (ICRA), 2001.
- [39] B. Bayle, J. Fourquet, and M. Renaud, "Manipulability Analysis for Mobile Manipulators", Proceedings of the 2001 IEEE International Conference on Robotics & Automation (ICRA), 2001.
- [40] B. Bayle, J. Fourquet, F. Lamiroux, and M. Renaud, "Kinematic Control of Wheeled Mobile Manipulators", Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2002.
- [41] K. Nagatani, T. Hirayama, A. Gofuku, and Y. Tanaka "Motion Planning for Mobile Manipulator with Keeping Manipulability", Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2002.
- [42] H. Seraji, "A Unified Approach to Motion Control of Mobile Manipulators", The International Journal of Robotics Research, V. 17, N. 2, pp. 107-118, 1998.
- [43] J. Chung, S. Velinsky, and R. Hess, "Interaction Control of a Redundant Mobile Manipulator", The International Journal of Robotics Research, V. 17, N. 12, pp. 1-8, 1998.
- [44] J. Gardner, and S. Velinsky, "Kinematics of Mobile Manipulators and Implications for Design", Journal of Robotic Systems, V. 17, N. 6, pp. 309-320, 2000.

- [45] B. Bayle, J. Fourquet, and M. Renaud, “Manipulability of Wheeled Mobile Manipulators: Application to Motion Generation”, *The International Journal of Robotics Research*, V. 22, N. 7–8, pp. 565-581, 2003.
- [46] D. Xu, H. Hu, C. Calderon, and M. Tan, “Motion Planning for a Mobile Manipulator with Redundant DOFs”, *Proceedings of the International Conference on Intelligent Computing (ICIC)*, Hefei, China, 2005.
- [47] A. Luca, G. Oriolo, and P. Giordano, “Kinematic Modeling and Redundancy Resolution for Nonholonomic Mobile Manipulators”, *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1867-1873, 2006.
- [48] E. Papadopoulos, and J. Poulakakis, “Planning and Model-Based Control for Mobile Manipulators”, *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1810-1815, 2000.
- [49] J. Craig, “Introduction to Robotics Mechanics and Control”, Third Edition, Addison- Wesley Publishing, 2003.
- [50] Y. Nakamura, “Advanced Robotics: Redundancy and Optimization”, Addison-Wesley Publishing, 1991.
- [51] R. Paul, “Robot Manipulators: Mathematics, Programming, and Control”, MIT Pres, 1981.
- [52] K. Edwards, R. Alqasemi, R. Dubey, “Design, Construction and Testing of a Wheelchair-Mounted Robotic Arm”, *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3165-3170, 2006.
- [53] T. Yoshikawa, “Manipulability and Redundancy Control of Robotic Mechanisms”, *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA)*, V. 2, pp. 1004-1009, 1985.
- [54] T. Yoshikawa, “Foundations of Robotics: Analysis and Control”. MIT Press, 1990.
- [55] W. Koepf, “The Algebra of Holonomic Equations”, *Mathematica*, V. 44, pp. 173–194, 1997.
- [56] O. Krupkova, and P. Volny, “Euler-Lagrange and Hamilton Equations for Non-Holonomic Systems in Field Theory”, *Journal of Physics*, V. 38, pp. 8715-8745, 2005.

- [57] E. McCaffrey, R. Alqasemi, and R. Dubey, “Kinematic Analysis and Evaluation of Wheelchair Mounted Robotic Arms”, Proceedings of the 2004 IMECE International Mechanical Engineering Congress & Exposition (IMECE), Anaheim, California, 2004.
- [58] G. Schalk, D. McFarland, T. Hinterberger, N. Birbaumer, and J. Wolpaw, “BCI2000: A General-Purpose Brain-Computer Interface (BCI) System”, IEEE Transactions on Biomedical Engineering, V. 51, N. 6, pp. 1034-1043, 2004.
- [59] S. Sutton, M. Braren, J. Zublin, and E. John, “Evoked Potential Correlates of Stimulus Uncertainty”, Science, V. 150, pp. 1187–1188, 1965.
- [60] Sensable Technologies website: <http://www.sensable.com>, 2007.
- [61] D. Xu, M. Tan, G. Chen, “An Improved Dead Reckoning Method for Mobile Robot with Redundant Odometry Information”, Proceedings of the 7th International Conference on Control, Automation, Robotics and Vision (ICARCV), pp. 631-636, Singapore, 2002.

Appendices

Appendix A. Hardware Components

A.1. Robotic Arm Gear Motors with Encoders



GM8724S009

Lo-Cog® DC Servo Gearmotor



Assembly Data	Symbol	Units	Value	
Reference Voltage	E	V	12	
No-Load Speed	S _{NL}	rpm (rad/s)	720	(75.4)
Continuous Torque (Max.) ¹	T _C	oz-in (N-m)	15	(1.0E-01)
Peak Torque (Stall) ²	T _{PK}	oz-in (N-m)	42	(3.0E-01)
Weight	W _M	oz (g)	11.2	(316)
Motor Data				
Torque Constant	K _T	oz-in/A (N-m/A)	3.09	(2.18E-02)
Back-EMF Constant	K _E	V/krpm (V/rad/s)	2.29	(2.18E-02)
Resistance	R _T	Ω	4.33	
Inductance	L	mH	2.34	
No-Load Current	I _{NL}	A	0.18	
Peak Current (Stall) ²	I _P	A	2.77	
Motor Constant	K _M	oz-in/√W (N-m/√W)	1.49	(1.05E-02)
Friction Torque	T _F	oz-in (N-m)	0.35	(2.5E-03)
Rotor Inertia	J _M	oz-in-s ² (kg-m ²)	2.3E-04	(1.6E-06)
Electrical Time Constant	τ _E	ms	0.54	
Mechanical Time Constant	τ _M	ms	14.7	
Viscous Damping	D	oz-in/krpm (N-m-s)	0.020	(1.4E-06)
Damping Constant	K _D	oz-in/krpm (N-m-s)	1.6	(1.1E-04)
Maximum Winding Temperature	θ _{MAX}	°F (°C)	311	(155)
Thermal Impedance	R _{TH}	°F/watt (°C/watt)	70.5	(21.4)
Thermal Time Constant	τ _{TH}	min	10.7	
Gearbox Data				
Reduction Ratio			6.3	
Efficiency ³			0.95	
Maximum Allowable Torque		oz-in (N-m)	100	(0.71)
Encoder Data				
Channels			3	
Resolution		CPR	500	

1 - Specified at max. winding temperature at 25°C ambient without heat sink. 2 - Theoretical values supplied for reference only.
3 - Effective gearbox efficiency for this unit improved by use of ball bearings.

Included Features

- 2-Pole Stator
- Ceramic Magnets
- Heavy-Gauge Steel Housing
- 7-Slot Armature
- Silicon Steel Laminations
- Stainless Steel Shaft
- Copper-Graphite Brushes
- Diamond Turned Commutator
- Motor Ball Bearings
- Output Ball Bearing
- Standard Gears

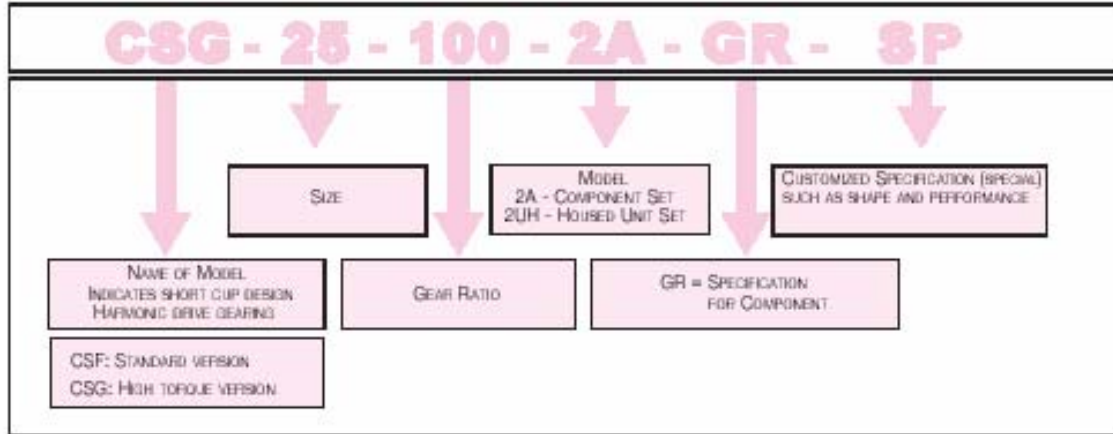
Customization Options

- Alternate Winding
- Sleeve or Ball Bearings
- Modified Output Shaft
- Custom Cable Assembly
- Special Brushes
- EMI/RFI Suppression
- Alternate Gear Material
- Special Lubricant
- Optional Encoder
- Fail-Safe Brake

Appendix A. (Continued)

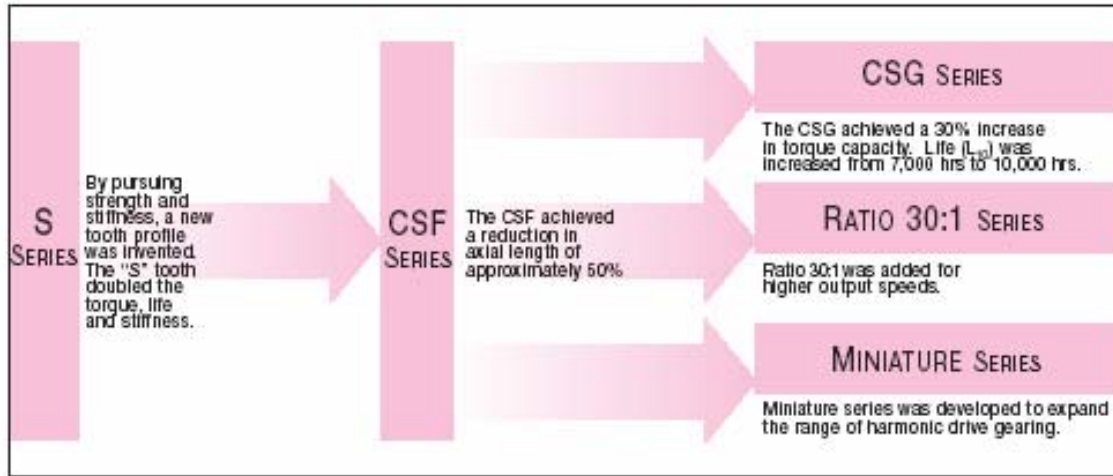
A.2. Harmonic Drive Gearheads

Ordering Information



Evolution of Harmonic Drive Gearing

Harmonic drive gearing continues to evolve by improving performance and functionality.

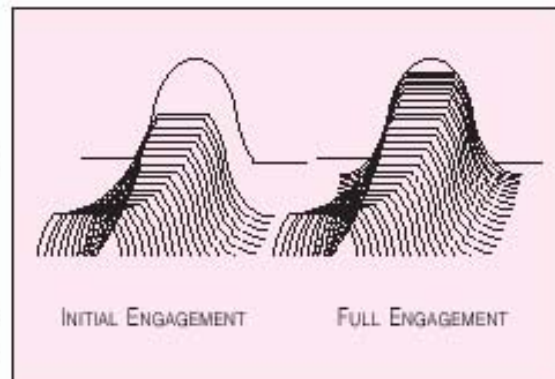


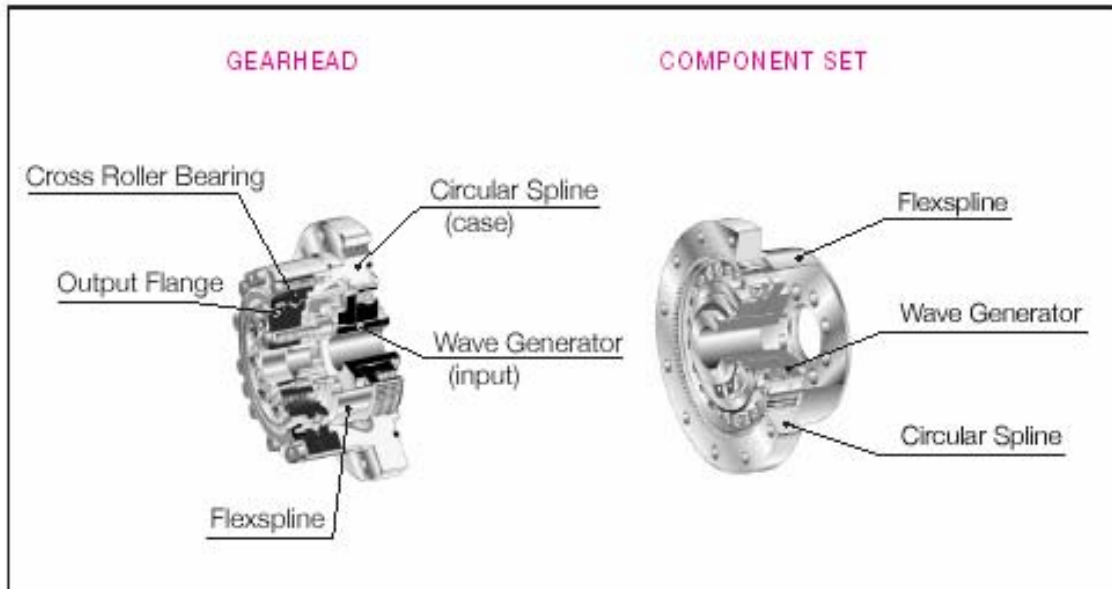
Tooth Profiles

The harmonic drive component sets and housed units presented in this catalog incorporate the "S" gear tooth profile. This patented tooth profile provides a significant improvement in gear operating characteristics and performance.

The new "S" tooth profile significantly increases the region of tooth engagement. For the traditional tooth profile 15% of the total number of teeth are in contact, while for the new profile up to 30% of the teeth are in contact. The increased number of teeth in engagement results in a 100% increase in torsional stiffness in the low & mid torque ranges.

The new tooth profile also features an enlarged tooth root radius, which results in a higher allowable stress and a corresponding increase in torque capacity. Furthermore, the enlarged region of tooth engagement leads to a more even loading of the Wave Generator bearing, resulting in more than double the life expectancy for the gear.



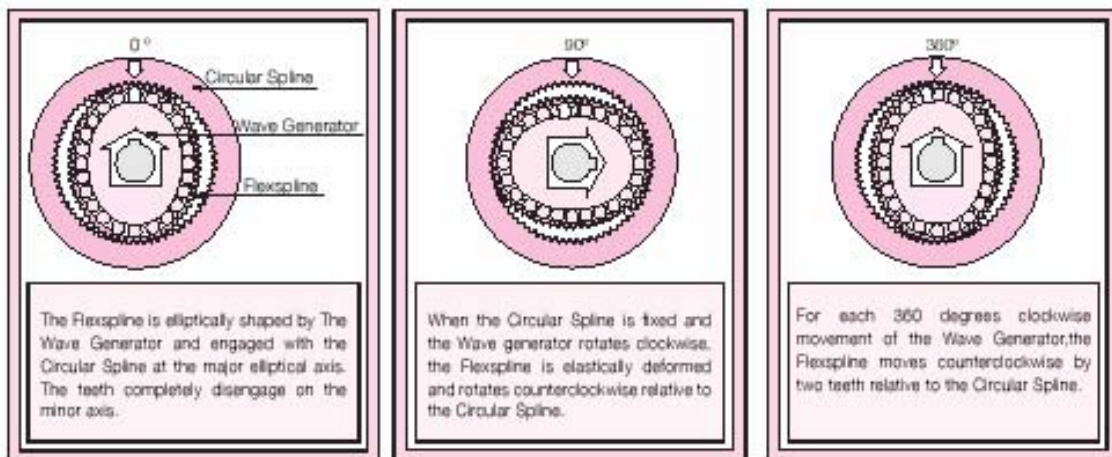


System Components

The FLEXSPLINE is a non-rigid, thin cylindrical cup with external teeth on a slightly smaller pitch diameter than the Circular Spline. It fits over and is held in an elliptical shape by the Wave Generator.

The WAVE GENERATOR is a thin faced ball bearing fitted onto an elliptical plug serving as a high efficiency torque converter.

The CIRCULAR SPLINE is a rigid ring with internal teeth, engaging the teeth of the Flexspline across the major axis of the Wave Generator.

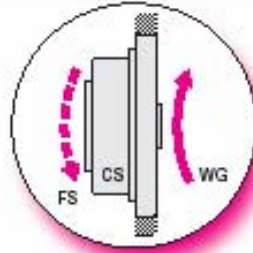


Appendix A. (Continued)

Driving Configurations

Driving Configurations

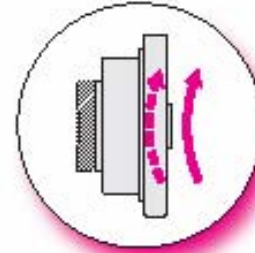
A variety of different driving configurations are possible, as shown below. The reduction ratio given in the tables on page 10 and 11 correspond to arrangement 1, in which the Wave Generator acts as the input element, the Circular Spine is fixed and the Flexspline acts as the output element.



1. Reduction Gearing
CS Fixed
WG Input
FS Output

$$\text{Ratio} = -\frac{R}{1} \quad [\text{Equation 1}]$$

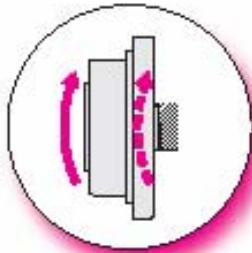
Input and output in opposite direction.



2. Reduction Gearing
FS Fixed
WG Input
CS Output

$$\text{Ratio} = \frac{R+1}{1} \quad [\text{Equation 2}]$$

Input and output in same direction.



3. Reduction Gearing
WG Fixed
FS Input
CS Output

$$\text{Ratio} = \frac{R+1}{R} \quad [\text{Equation 3}]$$

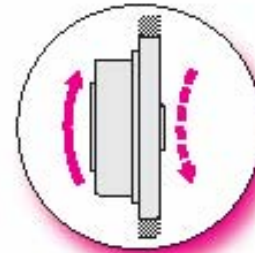
Input and output in same direction.



4. Speed Increaser Gearing
WG Fixed
CS Input
FS Output

$$\text{Ratio} = \frac{R}{R+1} \quad [\text{Equation 4}]$$

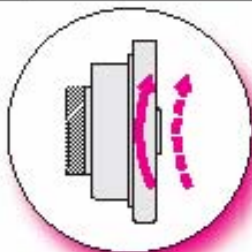
Input and output in same direction.



5. Speed Increaser Gearing
CS Fixed
FS Input
WG Output

$$\text{Ratio} = -\frac{1}{R} \quad [\text{Equation 5}]$$

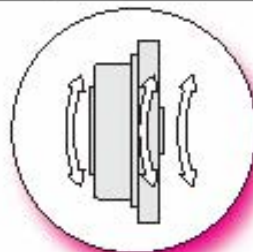
Input and output in opposite direction.



6. Speed Increaser Gearing
FS Fixed
CS Input
WG Output

$$\text{Ratio} = \frac{1}{R+1} \quad [\text{Equation 6}]$$

Input and output in same direction.



7. Differential Gearing
WG Control Input
CS Main Drive-Input
FS Main Drive-Output

Numerous differential functions can be obtained by combinations of the speed and rotational direction of the three basic elements.

Appendix A. (Continued)

CSF Rating Table

Table 1

Size	Ratio	Rated Torque at 2000 T_r rpm		Limit for Repeated Peak Torque		Limit for Average Torque		Limit for Momentary Peak Torque		Maximum Input Speed		Limit for Average Input Speed		Moment of Inertia	
		Nm	in-lb	Nm	in-lb	Nm	in-lb	Nm	in-lb	rpm		rpm		$\times 10^{-4} \text{kgm}^2$	$\times 10^{-4} \text{kgLms}^2$
										Oil	Grease	Oil	Grease		
8	30	0.9	8	1.8	16	1.4	12	3.3	29	14000	8600	6500	3500	0.003	0.0031
	50	1.8	16	3.3	29	2.3	20	6.6	58						
	100	2.4	18	4.8	42	3.3	29	9.0	80						
11	30	2.2	19	4.5	40	3.4	30	8.5	75	14000	8600	6500	3500	0.012	0.012
	50	3.5	31	8.3	73	5.5	49	17	150						
	100	5.0	44	11	97	8.9	79	25	221						
14	30	4.0	35	9.0	80	6.8	60	17	150	14000	8600	6500	3500	0.033	0.034
	50	5.4	48	18	159	8.9	81	35	310						
	80	7.8	69	23	204	11	97	47	418						
	100	7.8	69	28	248	11	97	54	478						
17	30	8.8	78	16	142	12	108	30	268	10000	7300	6500	3500	0.079	0.081
	50	16	142	34	301	26	230	70	620						
	80	22	195	43	381	27	239	87	770						
	100	24	212	54	478	39	345	108	958						
	120	24	212	54	478	39	345	86	761						
20	30	15	133	27	239	20	177	50	443	10000	6600	6500	3500	0.193	0.197
	50	25	221	56	496	34	301	95	867						
	80	34	301	74	655	47	411	127	1124						
	100	40	354	82	728	49	434	147	1301						
	120	40	354	87	770	49	434	147	1301						
	160	40	354	92	814	49	434	147	1301						
25	30	27	239	50	443	38	338	95	841	7500	5800	5800	3500	0.413	0.421
	50	39	345	99	888	55	487	188	1646						
	80	83	558	137	1212	87	770	295	2257						
	100	87	593	157	1389	108	958	284	2513						
	120	87	593	167	1478	108	958	304	2690						
	160	87	593	178	1598	108	958	314	2779						
32	30	54	478	100	885	75	664	200	1770	7000	4800	4800	3500	1.69	1.72
	50	76	673	218	1912	108	958	382	3381						
	80	118	1044	304	2690	167	1478	588	5027						
	100	137	1212	383	2947	218	1912	647	5728						
	120	137	1212	383	3124	218	1912	686	6071						
	160	137	1212	372	3292	218	1912	686	6071						
40	30	137	1212	402	3598	196	1735	686	6071	5600	4000	3800	3000	4.50	4.59
	50	208	1823	519	4593	264	2513	960	8673						
	100	285	2345	588	5027	372	3292	1080	9558						
	120	294	2602	617	5480	451	3991	1180	10443						
	160	294	2602	647	5728	451	3991	1180	10443						
	160	294	2602	647	5728	451	3991	1180	10443						
45	30	178	1598	500	4425	265	2345	960	8408	5000	3800	3300	3000	8.88	8.88
	50	313	2770	708	6248	390	3452	1270	11240						
	100	383	3124	755	6662	500	4425	1570	13895						
	120	402	3598	823	7284	620	5487	1760	15578						
	160	402	3598	882	7808	630	5576	1910	16904						
50	30	245	2188	715	6328	350	3098	1430	12658	4500	3600	3000	2500	12.5	12.8
	50	372	3292	941	8328	519	4593	1860	16461						
	100	470	4180	960	8673	686	5894	2060	18291						
	120	529	4682	1050	9558	813	7195	2060	18291						
	160	529	4682	1180	10443	843	7481	2450	21663						
58	30	353	3124	1020	9027	520	4602	1960	17348	4000	3000	2700	2200	27.3	27.9
	50	549	4859	1450	13098	770	6815	2450	21663						
	100	698	6180	1590	14072	1060	9381	3180	28143						
	120	745	6593	1720	15222	1190	10532	3330	29471						
160	745	6593	1840	16284	1210	10709	3430	30358							

Appendix A. (Continued)

Rating Table

Table 2

Size	Ratio	Rated Torque at 2000 Tr rpm		Limit for Repeated Peak Torque		Limit for Average Torque		Limit for Momentary Peak Torque		Maximum Input Speed rpm		Limit for Average Input Speed rpm		Moment of Inertia	
		Nm	in-lb	Nm	in-lb	Nm	in-lb	Nm	in-lb	Oil	Grease	Oil	Grease	$\times 10^4 \text{kgm}^2$	$\times 10^4 \text{kgfms}^2$
65	50	490	4337	1420	12567	720	6372	2690	25046	3500	2600	2400	1900	48.8	47.8
	60	745	6593	2110	187	1040	9204	3720	32922						
	100	951	8418	2300	20355	1520	13452	4750	42098						
	120	951	8418	2510	22214	1570	13895	4750	42098						
	160	951	8418	2630	23276	1570	13895	4750	42098						
80	50	672	7717	2440	21594	1260	11151	4670	43100	2900	2300	2200	1500	122	124
	60	1320	11892	3430	30356	1830	16196	6590	59322						
	100	1700	15045	4220	37347	2360	20686	7910	70004						
	120	1990	17612	4590	40622	3130	27701	7910	70004						
	160	1990	17612	4910	43454	3130	27701	7910	70004						
90	50	1180	10443	3530	31241	1720	15222	6660	59941	2700	2000	2100	1300	214	218
	60	1550	13718	3990	35312	2510	22214	7250	64183						
	100	2270	20090	5680	50268	3360	29736	9020	79827						
	120	2570	22745	6160	54516	4300	38055	9600	86730						
	160	2700	23695	6940	60534	4300	38055	11300	100005						
100	50	1580	13863	4450	39363	2260	20178	8900	78785	2500	1800	2000	1200	366	363
	60	2360	21063	6060	53631	3310	29294	11600	102660						
	100	2940	26019	7350	65048	4630	40976	14100	124765						
	120	3190	28143	7960	70446	5720	50622	15300	135405						
	160	3550	31418	9180	81243	5720	50622	15500	137175						

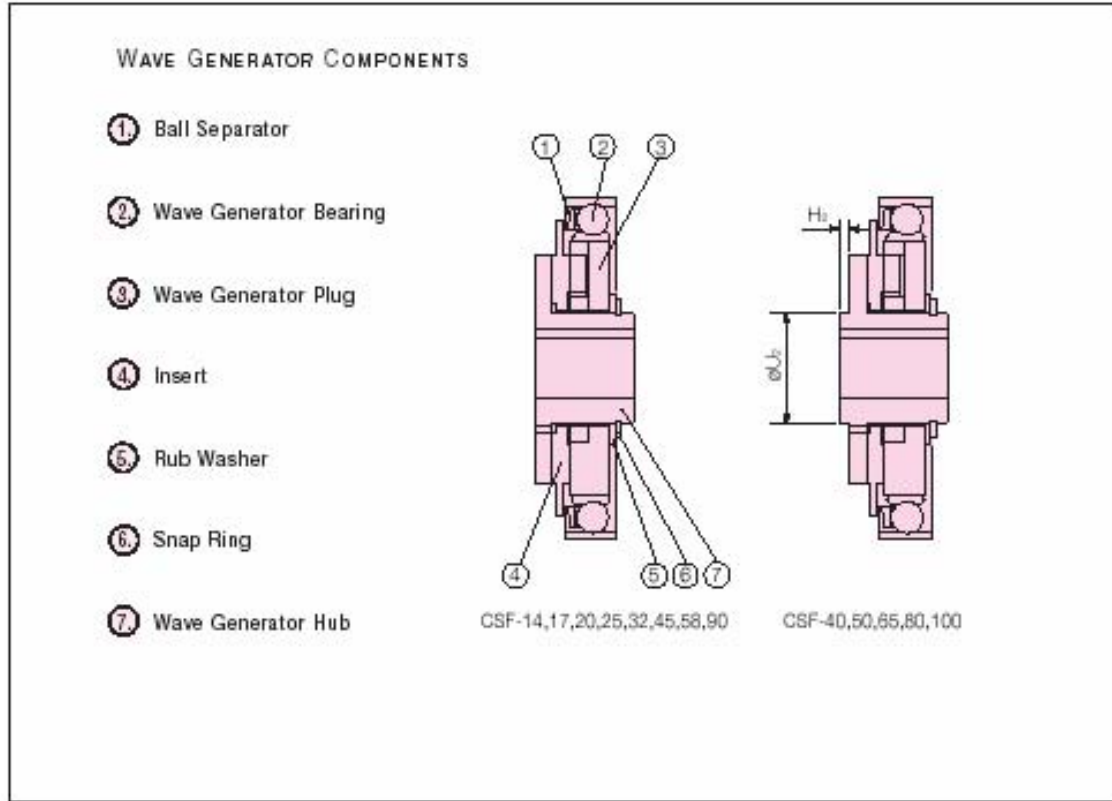
CSG Rating Table

Table 3

Size	Ratio	Rated Torque at 2000 Tr rpm		Limit for Repeated Peak Torque		Limit for Average Torque		Limit for Momentary Peak Torque		Maximum Input Speed rpm		Limit for Average Input Speed rpm		Moment of Inertia	
		Nm	in-lb	Nm	in-lb	Nm	in-lb	Nm	in-lb	Oil	Grease	Oil	Grease	$\times 10^4 \text{kgm}^2$	$\times 10^4 \text{kgfms}^2$
14	50	7.0	62	23	204	9	80	46	407	14000	8500	6500	3500	0.033	0.0094
	60	10	89	30	266	14	124	61	540						
	100	10	89	36	319	14	124	70	620						
17	50	21	186	44	390	34	301	91	805	10000	7300	6500	3500	0.079	0.061
	60	29	257	56	496	35	310	113	1000						
	100	31	274	70	620	51	451	143	1266						
	120	31	274	70	620	51	451	112	991						
20	50	33	292	73	646	44	389	127	1124	10000	8500	6500	3500	0.196	0.197
	60	44	389	96	850	61	540	165	1460						
	100	52	460	107	947	64	566	191	1690						
	120	52	460	113	1000	64	566	191	1690						
	160	52	460	120	1062	64	566	191	1690						
25	50	51	451	127	1124	72	637	242	2142	7500	5600	5600	3500	0.413	0.421
	60	62	726	178	1575	113	1000	332	2938						
	100	67	770	204	1805	140	1239	369	3266						
	120	67	770	217	1920	140	1239	395	3496						
	160	67	770	229	2027	140	1239	408	3611						
32	50	99	876	261	2487	140	1239	407	4399	7000	4800	4600	3500	1.69	1.72
	60	153	1354	395	3496	217	1920	738	6631						
	100	178	1575	433	3832	261	2487	841	7443						
	120	178	1575	459	4062	261	2487	892	7894						
	160	178	1575	464	4263	261	2487	892	7894						
40	50	178	1575	523	4629	255	2257	892	7894	5900	4000	3600	5000	4.50	4.59
	60	266	2372	675	5974	369	3266	1270	11240						
	100	345	3053	738	6531	464	4263	1400	12390						
	120	352	3381	802	7098	566	5186	1530	13541						
	160	352	3381	841	7443	566	5186	1530	13541						

Appendix A. (Continued)

External Dimension & Shape



There is a difference in appearance of the the ball separator between CSF and CSG.
(CSG size 14 and 17 use the same ball separator as CSF.



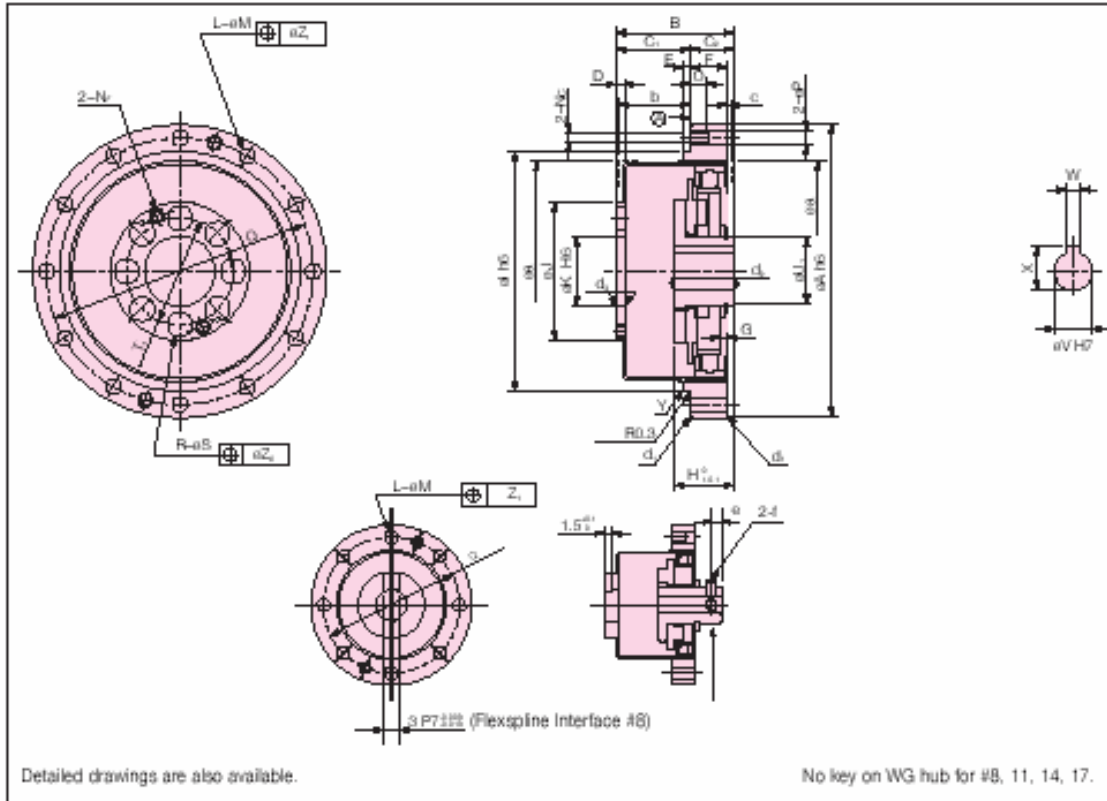
CSF ALL SIZES



CSG-20 AND ABOVE

Appendix A. (Continued)

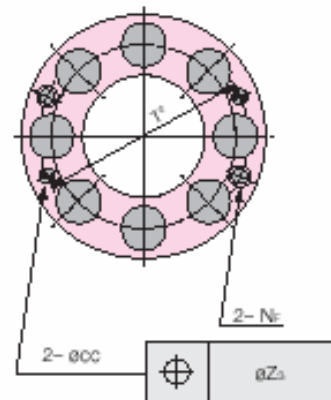
External Dimension & Shape



Flexspine Dowel Pin Hole

Dowel Pin Option

In cases where the gear will see loads near the Momentary Peak Torque level, the use of additional dowel pins in addition to the screws is recommended. Dowel pin holes are manufactured by reamer and the dimensions are shown. In addition, the CSF has a different number of dowel pin holes than the CSG.



Appendix A. (Continued)

External Dimensions

Table 8		(mm)							
		8	11	14	17	20	25	32	40
B	ϕA h5	30	40	50	60	70	85	110	135
	CSF	22.1 _{±0.1}	25.8 _{±0.1}	28.5 _{±0.1}	32.5 _{±0.1}	33.5 _{±0.1}	37 _{±0.1}	44 _{±0.1}	53 _{±0.1}
	CSG	-	-	28.5 _{±0.1}	32.5 _{±0.1}	33.5 _{±0.1}	37 _{±0.1}	44 _{±0.1}	53 _{±0.1}
	C ₁	12.5 ^{±0.05}	14.5 ^{±0.05}	17.5 ^{±0.05}	20 ^{±0.05}	21.5 ^{±0.05}	24 ^{±0.05}	28 ^{±0.05}	34 ^{±0.05}
	C ₂	9.5	11.3	11	12.5	12	13	16	19
	D	2.7	2	2.4	3	3	3	3.2	4
	E	-	2	2	2.5	3	3	3	4
	F	4.5	5	6	6.5	7.5	10	14	17
G	CSF	-	-	0.4	0.3	0.1	2.1	2.5	3.3
	CSG	-	-	1.4	1.6	1.5	3.5	4.2	5.6
H ₁ \pm 0.1	CSF	12	16	17.6	19.5	20.1	20.2	22	27.5
	CSG	-	-	18.5	20.7	21.5	21.6	23.6	29.7
	H ₂	-	-	-	-	-	-	-	0.4
a1 h5	Ratio*30	-	31	38	48	54	67	90	110
	Ratio*30	-	31	38	48	55	68	90	-
	ϕJ	12.3	17.8	23	27.2	32	40	52	64
	ϕK h6	6	6	11	10	16	20	26	32
L	CSF	8	8	6	12	12	12	12	12
	CSG	-	-	8	16	16	16	16	16
	ϕM	2.2	2.9	3.5	3.4	3.5	4.5	5.5	6.6
	N ₁	M2	M2.5	M3	M3	M3	M4	M5	M6
	N ₂	-	-	M3	M3	M3	M4	M5	M6
	O	3	3	6	6.5	4	6	7	9
	ϕP	2.2	2.9	-	-	3.5	4.5	5.5	6.6
	Q (PCD)	25.5	35	44	54	62	75	100	120
	R	-	6	6	6	8	8	8	8
	ϕS	-	3.4	4.5	5.5	5.5	6.6	9	11
	T ₁ (PCD)	-	12	17	19	24	30	40	50
	T ₂ (PCD)	-	15.2	18.5	21.5	27	34	45	56
	ϕU_1	7	11	14	18	21	26	26	32
	ϕU_2	-	-	-	-	-	-	-	32
ay	(HT)minimun	3	5	6	8	9	11	14	14
	maximum	-	-	8	10	13	15	15	20
	VWj30	-	-	-	-	3	4	5	5
	X	-	-	-	-	10.4 ^{±0.1}	12.8 ^{±0.1}	16.3 ^{±0.1}	16.3 ^{±0.1}
	Y	-	0.2	0.3	0.4	0.4	0.4	0.4	0.4
	ϕZ_1	0.1	0.2	0.25	0.20	0.25	0.25	0.25	0.3
	ϕZ_2	-	0.2	0.25	0.25	0.25	0.3	0.5	0.5
	ϕZ_3	-	0.02	0.02	0.02	0.02	0.02	0.02	0.02
a _g		21.5	30	38	45	53	66	86	106
b	Minimum housing distance	11.34	14	17.1	19	20.5	23	26.8	33
c		-	-	1	1	1.5	1.5	1.5	2
B _{CC} HT	CSF	-	2	3	3	3	4	5	6
	CSG	-	-	2.5	3	3	4	5	5
	d ₁	0.3	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	d ₂	0.3	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	d ₃	0.3	0.3	0.5	0.5	0.5	0.5	0.5	0.5
	e	2	3	2.5	3	-	-	-	-
	f	M2K3	M3K4	M3K4	M3K5	-	-	-	-
	Weight (kg)	0.026	0.05	0.09	0.15	0.28	0.42	0.89	1.7

Appendix A. (Continued)

External Dimensions

Table 9 (mm)

	45	50	58	65	80	90	100
RA HS	155	170	195	215	265	300	330
B	58.5 ^{+0.1}	64 ^{+0.1}	75.5 ^{+0.1}	83 ^{+0.1}	101 ^{+0.1}	112.5 ^{+0.1}	125 ^{+0.1}
C	38 ^{+0.04}	41 ^{+0.04}	48 ^{+0.04}	52.5 ^{+0.04}	64 ^{+0.04}	71.5 ^{+0.04}	79 ^{+0.04}
C ₀	20.5	23	27.5	30.5	37	41	46
D	4.5	5	5.8	6.5	8	9	10
E	4	4	5	5	6	6	6
F	19	22	25	29	36	41	46
G	3.7	4.2	4.8	5.8	6.6	7.5	8.3
H	27.9	32	34.9	40.9	49.1	48.2	56.7
H ₀	-	0.8	-	2.2	3.1	-	4.5
eJ HS	1/30 except	124	135	156	177	218	245
	1/30	-	-	-	-	-	-
eJ	72	80	92.8	104	128	144	160
eK HS	36	40	46	52	65	72	80
L	12	12	12	12	16	16	16
eM	9	9	11	11	11	14	14
N _L	M8	M8	M10	M10	M10	M12	M12
N _r	M6	M8	M8	M8	M8	M12	M10
O	12	13	15	15	15	18	20
eP	9	9	11	11	11	14	14
Q(PCD)	140	150	175	195	240	270	300
R	8	8	8	8	10	8	12
eS	3.5	15.5	15.5	18	18	22	22
T ₁ (PCD)	54	60	70	80	100	110	130
T ₂ (PCD)	61	68	79	90	114	120	142
eU ₁	32	32	40	48	55	60	65
U ₁	-	32	-	48	55	-	65
eV	[H7]Standard	19	19	22	24	28	28
	Maximum	20	20	25	30	35	40
	WUsØ	6	6	6	8	8	8
X	21.8 ^{+0.1}	21.8 ^{+0.1}	24.8 ^{+0.1}	27.3 ^{+0.04}	31.3 ^{+0.04}	31.3 ^{+0.04}	31.3 ^{+0.04}
Y	Ø0.4	Ø0.8	Ø0.8	Ø0.8	Ø0.8	Ø0.8	Ø0.8
eZ ₁	0.5	0.5	0.5	0.5	0.5	1.0	1.0
eZ ₂	0.75	0.75	0.75	1.0	1.0	1.0	1.0
eZ ₃	0.02	0.02	0.02	0.02	0.02	0.02	0.02
eQ	119	133	154	172	212	239	265
b Minimum housing clearance	36.5	39	46.2	50	61	68.5	76
c	2	2	2.5	2.5	3	3	3
eOC H7	6	8	8	8	8	12	10
d ₁	Ø0.4	Ø0.4	Ø0.4	Ø0.4	Ø0.4	Ø0.4	Ø0.4
d ₂	Ø0.4	Ø0.4	Ø0.4	Ø0.4	Ø0.4	Ø0.4	Ø0.4
d ₃	Ø0.5	Ø0.5	Ø0.5	Ø0.5	Ø0.5	Ø0.5	Ø0.5
e	-	-	-	-	-	-	-
f	-	-	-	-	-	-	-
Weight (kg)	2.3	3.2	4.7	6.7	12.4	17.6	23.5

The pilot diameter for the Circular spline can be either ØI or ØA. Surface  is the recommended mounting surface.

The following parameters can be modified to accommodate customer-specific requirements.

Wave Generator: ØV, X, W

Recospline: R, ØS

Circular Spline: ØM, L

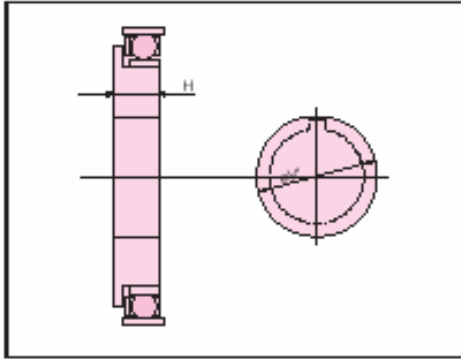
Appendix A. (Continued)

Diameters

Hole Diameter of Wave Generator Hub Table 18

Size	8	11	14	17	20	25	32	40	45	50	58	65	80	90	100
Standard Dimension	3	5	6	8	9	11	14	14	19	19	22	24	28	28	28
Minimum Hole Dimension	-	-	3	4	5	6	6	10	10	10	13	16	16	19	22
Maximum Hole Dimension	-	-	8	10	13	15	15	20	20	20	25	30	35	37	40

Hole Diameter of Wave Generator



Installation of Three Basic Elements

Installation for Wave Generator and the maximum hole dimensions. Shown above is the standard hole dimension of the Wave Generator for each size. The dimension can be changed within a range up to the maximum hole dimension shown in table 18. We recommend the dimension of keyway based on JIS standard. It is necessary that the dimension of keyways should sustain the transmission torque.

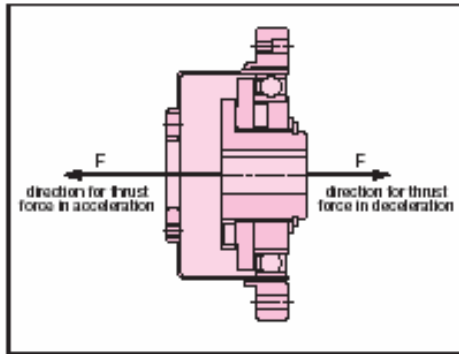
Please note: Tapered holes are also available.

In cases where a larger hole is required, use the Wave Generator without the Oldham coupling. The maximum diameter of the hole should be considered to prevent deformation of the Wave Generator plug by load torque. The dimension is shown in table 19 include the dimension of depth of keyway.

Maximum Diameter of Hole without Oldham Coupling Table 19

Size	8	11	14	17	20	25	32	40	45	50	58	65	80	90	100
Maximum Diameter of H	10	14	17	20	23	28	35	42	47	52	60	67	72	84	95
Max. thickness of plug H _{0.1}	5.7	6.7	7.2	7.6	11.3	11.3	13.7	15.9	17.8	19	21.4	13.5	28.5	31.3	34.9

Direction for Thrust Force of Wave Generator



Axial Force of Wave Generator

When a harmonic drive gear is used to accelerate a load, the deflection of the Flexspline leads to an axial force acting on the Wave Generator. This axial force, which acts in the direction of the closed end of the Flexspline, must be supported by the bearings of the input shaft (motor shaft).

When a harmonic drive gear is used to decelerate a load, an axial force acts to push the Wave Generator out of the Flexspline cup. Maximum axial force of the Wave Generator can be calculated by the equation shown below. The axial force may vary depending on its operating condition. The value of axial force tends to be a larger number when using high torque, extreme low speed and constant operation. The force is calculated (approximately) by the equation. In all cases, the Wave Generator must be axially (in both directions), as well as torsionally, fixed to the input shaft.

(note) Please contact us when you fix the Wave Generator hub and input shaft using bolts.

Equation for axial force

Gear Ratio	equation
i=1/30	$F=2x \frac{T}{D} x 0.07 x \tan 32^\circ$
i=1/50	$F=2x \frac{T}{D} x 0.07 x \tan 30^\circ$
i=1/80 and up	$F=2x \frac{T}{D} x 0.07 x \tan 20^\circ$

Symbols for equation

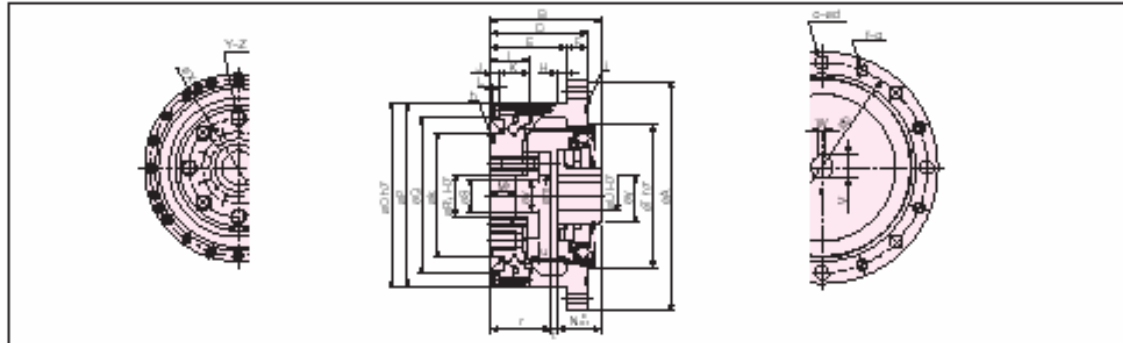
F	axial force	N
D	HD Size x 0.00254	m
T	output torque	Nm

Calculation Example

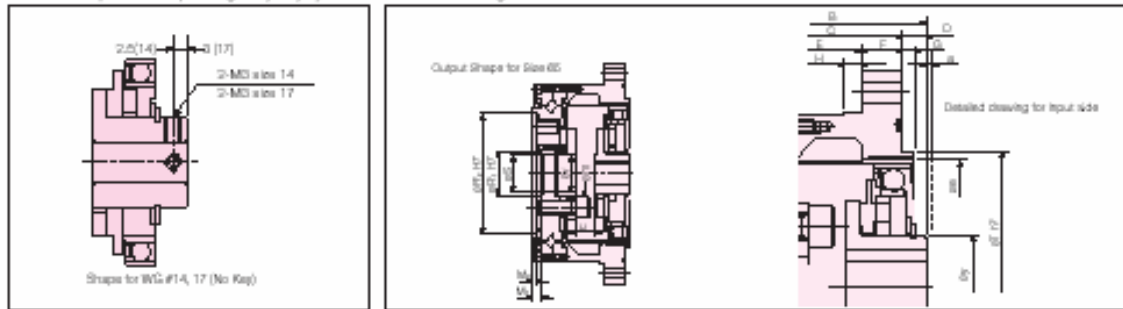
size	:	32
Ratio	:	i=1/50
Output Torque	:	300Nm
$F=2x \frac{300}{(32x0.00254)} x 0.07 x \tan 30^\circ$		
F=298N		

Appendix A. (Continued)

External Dimensions of Housed Unit



Note: Please note that the engagement length of bolt is within the length of threaded hole. Bolts that are too long may cause damage to Respline. The shape of the output flange may vary by size. Please contact our engineers for more detailed information.



Dimensions		34	37	20	25	32	40	45	50	58	65
	#A	73	79	93	107	138	160	180	190	226	260
	B	41 $\pm_{0.1}$	45 $\pm_{0.1}$	45.5 $\pm_{0.1}$	52 $\pm_{0.1}$	62 $\pm_{0.1}$	72.5 $\pm_{0.1}$	79.5 $\pm_{0.1}$	90 $\pm_{0.1}$	104.5 $\pm_{0.1}$	115 $\pm_{0.1}$
	C	34	37	38	46	57	66.5	74	85	97	108.5
D	CSF	7	8	7.5	6	5	6	5.5	5	7.5	6.5
	CSG	7 $\pm_{0.1}$	8 $\pm_{0.1}$	7.5 $\pm_{0.1}$	6 $\pm_{0.1}$	5 $\pm_{0.1}$	6 $\pm_{0.1}$	-	-	-	-
	E	27	29	28	36	45	50.5	58	69	77	84.5
	F	7	8	10	10	12	16	16	16	20	24
	G	2	2	3	3	3	4	4	4	5	5
	H	3.5	4	5	5	5	5	6	6	6	6
	I	16.5	16.5	16.5	18.5	22.5	24	27	31	36	39
	J	4.5	4.5	4	4.5	5.5	7.5	7	8	8.5	8.5
	K	12	12	12.5	14	17	16.5	20	23	26.5	30.5
	L	0.5	0.5	0.5	0.5	1	1.5	1	1	1.5	2
	M	9.4	9.5	9	12	15	5	6	8	10	10
	M ₁	-	-	-	-	-	-	-	-	-	4
N $\pm_{0.1}$	CSF	17.6	19.5	20.1	20.2	22	27.5	27.9	32	34.9	40.9
	CSG	18.5	20.7	21.5	21.6	23.6	29.7	-	-	-	-
	#Oh7	56	63	72	86	113	127	148	158	186	212
	#P	55	62	70	85	112	126	147	157	185	210
	#Q	42.5	49.5	58	73	96	109	127	137	161	186
	#R.H7	11	10	14	20	26	32	32	40	46	52
	#R.H7	-	-	-	-	-	-	-	-	-	142
	#S	11	10	14	20	26	24	25	32	38	44
	#T h7	38	48	56	67 (68)*	90	110	124	135	156	177
	#U H7	6	8	12	14	14	14	19	19	22	24
	V	-	-	13.8 $\pm_{0.1}$	16.3 $\pm_{0.1}$	16.3 $\pm_{0.1}$	16.3 $\pm_{0.1}$	21.8 $\pm_{0.1}$	21.8 $\pm_{0.1}$	24.8 $\pm_{0.1}$	27.3 $\pm_{0.1}$
	W Jd9	-	-	4	5	5	5	6	6	6	8
	#X	23	27	32	42	55	68	82	84	100	110

* Dimensions in parentheses indicates ratio 30:1

Appendix A. (Continued)

External Dimensions of Housed Unit

Dimensions (mm)

Table 29

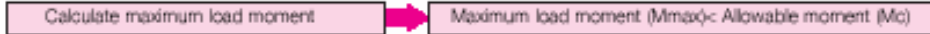
	14	17	20	25	32	40	45	50	58	65
Y	6	6	8	8	8	8	8	8	8	8
Z	M4X8	M5X10	M5X9	M5X12	M10X15	M10X15	M12X18	M14X21	M16X24	M16X24
a	1	1	1.5	1.5	1.5	2	2	2	2.5	2.5
ab	65	71	82	96	125	144	164	174	206	236
c	6	6	6	8	12	8	12	12	12	8
cd	4.5	4.5	5.5	5.5	6.6	9	9	9	11	14
ee	38	45	53	66	86	106	119	133	154	172
f	6	6	6	8	12	8	12	12	12	8
g	M4	M4	M5	M5	M5	M5	M8	M8	M10	M12
h	29.0X0.50	34.5X0.80	40.64X1.14	53.28X0.99	S71	AS568-042	S100	S106	S125	S136
i	550	556	567	590	S106	S125	S145	S156	S180	S206
ok	31	35	45	58	78	90	107	112	135	155
om	10	10.5	15.5	20	27	34	36	39	46	56
r	21.4	23.6	23	29	37	39.5	45.5	53	62.8	66.5
CSF	2	2	2.4	2.8	3	5.5	6.1	5	6.8	7.6
CSG	1.1	0.8	1	1.4	1.4	3.3	-	-	-	-
C3F	4	5	5	6	8	10	12	14	16	16
C3G	3.1	3.8	3.6	4.6	6.4	7.8	-	-	-	-
ev	8	7	10	15	20	24	25	32	38	44
ey	14	18	21	26	26	32	32	32	40	48
Weight (Kg)	0.52	0.68	0.98	1.5	3.2	5.0	7.0	8.9	14.6	20.9

Specification for Cross Roller Bearing

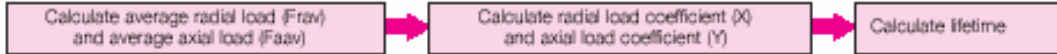
Housed units incorporate a precise cross roller bearing to directly support a load. The inner race of the bearing forms the output flange. Please calculate maximum load moment, life of cross roller bearing, and static safety factor to fully maximize the performance of the housed unit (gearhead).

Calculation Procedure

1. Maximum Load Moment (Mmax)



2. Output Bearing Life



3. Static Safety Factor



Specification for cross roller bearing

Specification for cross roller bearing is shown on figure.

Table 30

Size	Pitch Circle dp m	Offset R mm	Basic Dynamic Rated Load C		Basic Static Rated Load C ₀		Allowable Moment Load Mc		Moment Rigidity Km x10 ⁴ Nm/rad	Kin in-lb/ arc-min
			X10N	in-lb	X10N	in-lb	Nm	in-lb		
14	0.035	0.0095	47	416	60.7	537	41	363	4.38	113
17	0.0425	0.0095	52.9	468	75.5	668	64	566	7.75	200
20	0.050	0.0095	57.8	51	90.0	796	91	806	12.8	330
25	0.062	0.0115	96.0	850	151	1336	156	1381	24.2	623
32	0.080	0.013	150	1328	250	2212	313	2770	53.9	1380
40	0.096	0.0145	213	1885	365	3230	450	3983	91.0	2340
45	0.111	0.0155	230	2036	426	3770	686	6071	141	3630
50	0.119	0.018	348	3080	602	5328	759	6717	171	4400
58	0.141	0.0205	518	4584	904	8000	1180	10443	283	7290
65	0.160	0.0225	556	4921	1030	9116	1860	1646	404	10400

Basic dynamic rated load is a constant radial load where the basic dynamic rated life of CRB is 1×10^6 rotations.

Basic static rated load is a static load where the value of moment rigidity is the average value.

Appendix A. (Continued)

A.3. Wheelchair Selected Encoders

H5

Ball Bearing Optical Shaft Encoder

Description:
The H5 series ball bearing optical shaft encoder has either a molded polycarbonate or a machined aluminum enclosure, which utilizes either a 5-pin or 10-pin finger-latching connector. This non-contacting rotary to digital converter is designed to provide digital feedback information.

The H5 is fully assembled with a shaft, two 1/4" ID by 1/2" OD heavy duty ball bearings and a mounting plate. The shaft is either made of brass (polycarbonate version) or stainless steel (metal version). This design allows for an optional rear shaft extension (polycarbonate versions only). The mounting plate comes with 2 mounting holes for screws #4 or smaller.

A secure connection to the H5 series encoder is made through a 5-pin (single-ended versions) or 10-pin (differential versions) finger-latching connector (sold separately). The mating connectors are available from US Digital with several cable options and lengths.

For differential versions: the internal differential line driver (26C31) can source and sink 20mA at TTL levels. The recommended receiver is industry standard 26C32. Maximum noise immunity is achieved when the differential receiver is terminated with a 110 ohm resistor in series with a .0047mf capacitor placed across each differential pair. The capacitor simply conserves power; otherwise power consumption would increase by approximately 20mA per pair, or 60mA for 3 pairs.

Features:

- > Heavy duty ball bearings track up to 10,000 RPM
- > Low cost
- > 2-channel quadrature, TTL squarewave outputs
- > Optional index (3rd channel)
- > Differential outputs available
- > Optional Agilent compatible pin-out
- > Positive finger-latching connector
- > 32 to 1250 cycles per revolution (CPR)
- > 128 to 5000 pulses per revolution (PPR)
- > Tracks from 0 to 100,000 cycles/sec
- > -40 to +100°C operating temperature
- > Single +5VDC supply
- > US Digital warrants its products against defects in materials and workmanship for two years. See complete warranty for details.

Mechanical Specifications:

Parameter	Dimension / Units
Shaft Speed	10,000 RPM max. continuous
Acceleration	10,000 rad/sec ²
Shaft Torque	0.05 in. oz. max.
Shaft Loading	2 lbs. max.
Bearing Life	(90/PP) - life in millions of revs. where P = radial load in lbs.

Weight

Polycarbonate Single-ended (H5S)	1.79 oz.
Polycarbonate Differential (H5D)	1.89 oz.
Metal Single-ended (H5MS)	2.26 oz.
Metal Differential (H5MD)	2.32 oz.

Shaft Runout 0.0006 T.I.R. max.
Moment of Inertia 0.0001 oz. in. s²
Vibration 20 g. 5 to 20Hz

Single-ended Electrical Specifications:
For complete details see the EMI / HEDS data sheet.

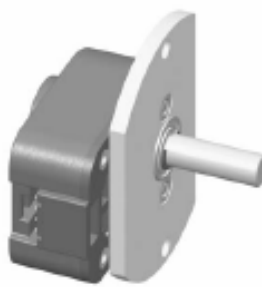
Phase Relationship:
B leads A for clockwise shaft rotation, and A leads B for counterclockwise rotation viewed from the shaft side of the encoder (see the EMI / HEDS data sheet).

Differential Electrical Specifications:


Specification	Min.	Typ.	Max.	Units	Notes
Supply	4.5	5.0	5.5	Volts	
Current Consumption					
Index: 64 CPR	-	28	53	mA	No load
Index: 1800, 2500 CPR	-	56	59	mA	No load
Index: All Other Resolutions	-	58	88	mA	No load
Non-index: <2000 CPR	-	18	43	mA	No load
Non-index: >=2000 CPR	-	58	88	mA	No load
Output Voltage					
Sourcing to +5	2.4	3.4	-	Volts	@ -20mA
Sinking to Ground	-	0.2	0.4	Volts	@ 20mA

> For complete details see the EMI / HEDS data sheet.

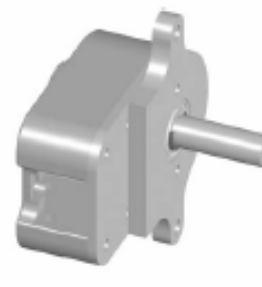
Polycarbonate Single-ended (H5S)



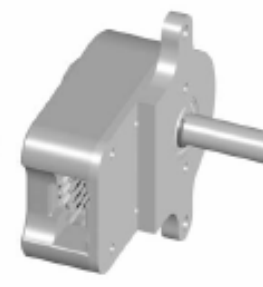
Polycarbonate Differential (H5D)



Metal Single-ended (H5MS)



Metal Differential (H5MD)



US DIGITAL

info@usdigital.com • www.usdigital.com
 Local: 360.260.2468 • Sales: 800.736.0194
 Support: 360.397.9999 • Fax: 360.260.2469
 1480 NE 136th Ave. • Vancouver, Washington • 98684 • USA

page 1

264

المنارة للاستشارات

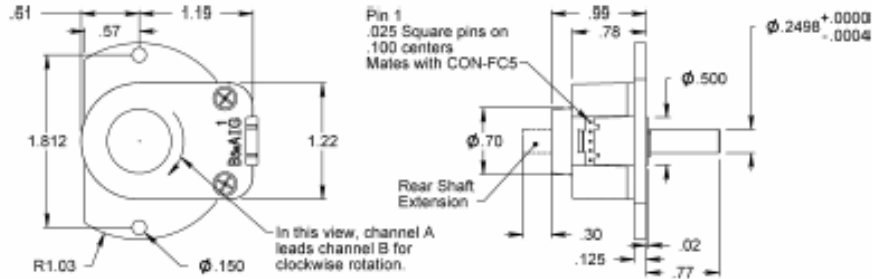
www.manaraa.com

Appendix A. (Continued)

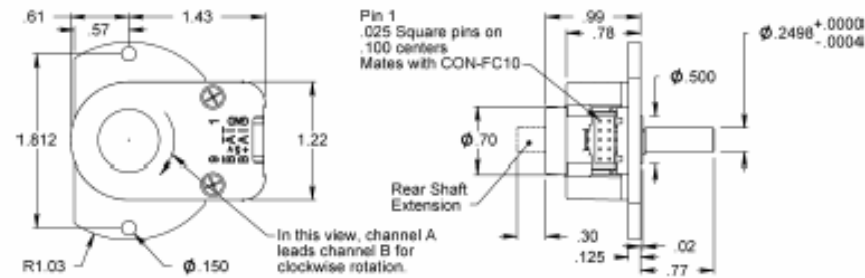
H5

Ball Bearing Optical Shaft Encoder

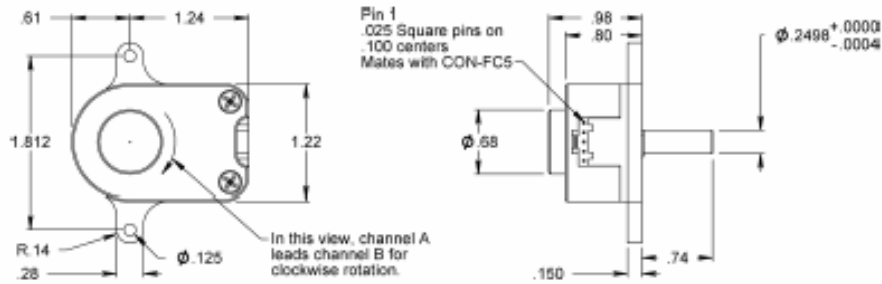
Polycarbonate Single-ended Mechanical Drawing (H5S):



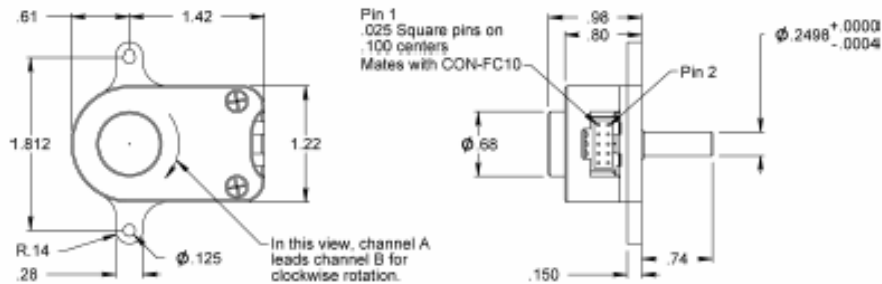
Polycarbonate Differential Mechanical Drawing (H5D):



Metal Single-ended Mechanical Drawing (H5MS):



Metal Differential Mechanical Drawing (H5MD):



US DIGITAL info@usdigital.com • www.usdigital.com
 Local: 360.260.2468 • Sales: 800.736.0194
 Support: 360.397.9999 • Fax: 360.260.2469
 1400 NE 136th Ave. • Vancouver, Washington • 98684 • USA

Appendix A. (Continued)

H5 Ball Bearing Optical Shaft Encoder

Compatible Cables & Connectors:

Finger-latching:		
5-pin	10-pin	Description
CON-FC5-22*	CON-FC10	Connector
CA-3133-1FT		Connector on one end with 4 12" wires
CA-3132-1FT		Connector on one end with 5 12" wires
CA-3131-8FT	CA-4217-8FT	Connector on one end of a 6' shielded round cable
	CA-4174-8FT**	Same as CA-4217, but for L-option only
CA-3620-8FT	CA-3619-8FT	Connectors on both ends of a 6' shielded round cable
	CA-3807-FT**	Same as CA-3619, but for L-option only

* 22 AWG is standard. 24, 26 and 28 AWG are also available.
 ** Agilent compatible cable assembly.

Attention:
 > Specify cable length when ordering.
 > Custom cable lengths are available. See the Cables / Connectors data sheet for more information.

Pin-outs:

Pin	5-pin Single-ended	10-pin Differential Standard	10-pin Differential Agilent (L-option)
1	Ground	Ground	No connection
2	Index	Ground	+5VDC power
3	A channel	Index-	Ground
4	+5VDC power	Index+	No connection
5	B channel	A- channel	A- channel
6		A+ channel	A+ channel
7		+5VDC power	B- channel
8		+5VDC power	B+ channel
9		B- channel	Index-
10		B+ channel	Index+

Ordering Information:

<table border="0"> <tr> <td>H5S</td> <td>H5S</td> </tr> <tr> <td>Standard:</td> <td>Index/HiRes:</td> </tr> <tr> <td></td> <td>(Hi Res: >=1000 CPR)</td> </tr> <tr> <td>\$59.85 / 1</td> <td>\$68.83 / 1</td> </tr> <tr> <td>\$55.65 / 10</td> <td>\$64.00 / 10</td> </tr> <tr> <td>\$51.45 / 50</td> <td>\$59.17 / 50</td> </tr> <tr> <td>\$49.35 / 100</td> <td>\$56.75 / 100</td> </tr> </table>	H5S	H5S	Standard:	Index/HiRes:		(Hi Res: >=1000 CPR)	\$59.85 / 1	\$68.83 / 1	\$55.65 / 10	\$64.00 / 10	\$51.45 / 50	\$59.17 / 50	\$49.35 / 100	\$56.75 / 100	<table border="0"> <tr> <td>H5D</td> <td>H5D</td> </tr> <tr> <td>Standard:</td> <td>Index/HiRes:</td> </tr> <tr> <td></td> <td>(Hi Res: >=1000 CPR)</td> </tr> <tr> <td>\$73.50 / 1</td> <td>\$83.06 / 1</td> </tr> <tr> <td>\$69.30 / 10</td> <td>\$78.31 / 10</td> </tr> <tr> <td>\$65.10 / 50</td> <td>\$73.56 / 50</td> </tr> <tr> <td>\$61.95 / 100</td> <td>\$70.00 / 100</td> </tr> </table>	H5D	H5D	Standard:	Index/HiRes:		(Hi Res: >=1000 CPR)	\$73.50 / 1	\$83.06 / 1	\$69.30 / 10	\$78.31 / 10	\$65.10 / 50	\$73.56 / 50	\$61.95 / 100	\$70.00 / 100	<p>Cost Modifiers: > Add \$2 for E-option.</p>
H5S	H5S																													
Standard:	Index/HiRes:																													
	(Hi Res: >=1000 CPR)																													
\$59.85 / 1	\$68.83 / 1																													
\$55.65 / 10	\$64.00 / 10																													
\$51.45 / 50	\$59.17 / 50																													
\$49.35 / 100	\$56.75 / 100																													
H5D	H5D																													
Standard:	Index/HiRes:																													
	(Hi Res: >=1000 CPR)																													
\$73.50 / 1	\$83.06 / 1																													
\$69.30 / 10	\$78.31 / 10																													
\$65.10 / 50	\$73.56 / 50																													
\$61.95 / 100	\$70.00 / 100																													
<table border="0"> <tr> <td>H5MS</td> <td>H5MS</td> </tr> <tr> <td>Standard:</td> <td>Index/HiRes:</td> </tr> <tr> <td></td> <td>(Hi Res: >=1000 CPR)</td> </tr> <tr> <td>\$80.85 / 1</td> <td>\$92.98 / 1</td> </tr> <tr> <td>\$77.70 / 10</td> <td>\$89.36 / 10</td> </tr> <tr> <td>\$71.40 / 50</td> <td>\$82.11 / 50</td> </tr> <tr> <td>\$68.25 / 100</td> <td>\$78.49 / 100</td> </tr> </table>	H5MS	H5MS	Standard:	Index/HiRes:		(Hi Res: >=1000 CPR)	\$80.85 / 1	\$92.98 / 1	\$77.70 / 10	\$89.36 / 10	\$71.40 / 50	\$82.11 / 50	\$68.25 / 100	\$78.49 / 100	<table border="0"> <tr> <td>H5MD</td> <td>H5MD</td> </tr> <tr> <td>Standard:</td> <td>Index/HiRes:</td> </tr> <tr> <td></td> <td>(Hi Res: >=1000 CPR)</td> </tr> <tr> <td>\$94.50 / 1</td> <td>\$106.79 / 1</td> </tr> <tr> <td>\$89.25 / 10</td> <td>\$102.64 / 10</td> </tr> <tr> <td>\$85.05 / 50</td> <td>\$97.81 / 50</td> </tr> <tr> <td>\$80.85 / 100</td> <td>\$92.98 / 100</td> </tr> </table>	H5MD	H5MD	Standard:	Index/HiRes:		(Hi Res: >=1000 CPR)	\$94.50 / 1	\$106.79 / 1	\$89.25 / 10	\$102.64 / 10	\$85.05 / 50	\$97.81 / 50	\$80.85 / 100	\$92.98 / 100	
H5MS	H5MS																													
Standard:	Index/HiRes:																													
	(Hi Res: >=1000 CPR)																													
\$80.85 / 1	\$92.98 / 1																													
\$77.70 / 10	\$89.36 / 10																													
\$71.40 / 50	\$82.11 / 50																													
\$68.25 / 100	\$78.49 / 100																													
H5MD	H5MD																													
Standard:	Index/HiRes:																													
	(Hi Res: >=1000 CPR)																													
\$94.50 / 1	\$106.79 / 1																													
\$89.25 / 10	\$102.64 / 10																													
\$85.05 / 50	\$97.81 / 50																													
\$80.85 / 100	\$92.98 / 100																													

H5 - -

Version:
 S = Polycarbonate single-ended
 D = Polycarbonate differential
 MS = Metal single-ended
 MD = Metal differential.

CPR:
 32**
 50
 96
 100
 110*
 120*
 132
 200
 250
 256
 360
 400
 500
 540**
 720**
 900**
 1000
 1016*
 1024
 1250**

Options: (specify in ordersheet)
 I = Index (3rd channel)
 L = Agilent compatible pin-out.*
 E = Rear shaft extension.**

CPR Notes:
 * Index option not available.
 ** 32, 720, 900, 1250 CPR only available with index.

Options Notes:
 * Only available with differential versions (H5D and H5MD).
 ** Only available with polycarbonate versions (H5S and H5D).

Technical Data, Rev. 08.30.06, August 2006
 All information subject to change without notice.

US DIGITAL


info@usdigital.com • www.usdigital.com
 Local: 360.260.2468 • Sales: 800.736.0194
 Support: 360.397.9999 • Fax: 360.260.2469
 1400 NE 136th Ave. • Vancouver, Washington • 98684 • USA

page 3

Appendix A. (Continued)

A.4. Wheelchair Selected Friction Wheels

Bulletin 842 Encoder Accessories Measuring Wheels/Servo Clamps



Selection Guide

845 — M W — A — 1
a

Contact Material	
Code	Description
1	Rubber O-Ring
2	Polyurethane

Specifications

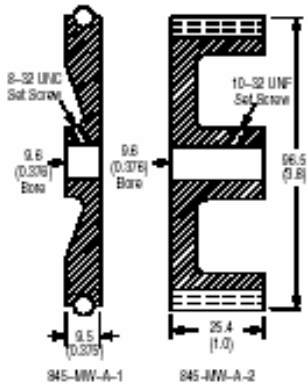
Circumference	304.8mm (12.00in)
Bore Hole Diameter	6.8mm (0.378in) Diameter
Durometer	70 D Shore
Material	Cast Aluminum


Description

Measuring wheels are used to convert a fixed amount of linear motion to a corresponding amount of rotary motion. Rubber O-Ring type contact material is used on metal, paper, foil, film and hard plastics. Polyurethane type contact material is used on soft smooth materials, such as soft paper, cardboard and fine weave textiles.

Dimensions—mm (inches)

Measuring Wheels





Selection Guide

845 — SC

Dimension Code	Approximate Dimension
A	2.38 ±0.003 (0.093)
B	1.02±0.003 (0.040)
C	3.38 (0.133)
D	9.5±0.000, -0.032 (0.375)
E	3.43 (0.135)
F	9.90 (0.390)
G	3.18 (0.125)
H	1.73 (0.068) max.
J	#4-40
K	0.25 (0.010)

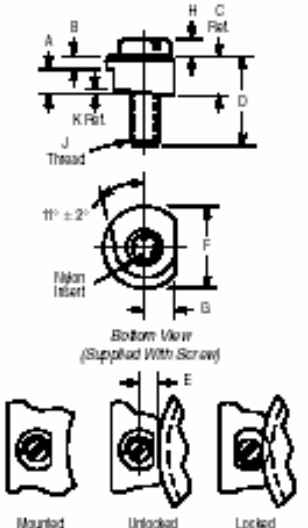
Description

Servo clamps are used for the mounting of all encoders with the servo type mounting option. For the size 15 device, 3 clamps are arrayed on a 48.1mm (1.89in) diameter bolt circle. For the size 20 device, 3 or 4 clamps are arrayed on a 57.7mm (2.27in) diameter bolt circle. For 60mm encoders (i.e., the 842A), 3 or 4 clamps are arrayed on a 66.3mm (2.61in) diameter bolt circle. For the size 25 device, 3 or 4 clamps are arrayed on a 70.4mm (2.77in) diameter bolt circle. Servo clamps are sold as a kit (set of 4 clamps).

Material: Stainless Steel 316

Dimensions—mm (inches)

Servo Clamps



Appendix A. (Continued)

A.6. Gripper's Planetary Gearhead



Planetary Gearheads

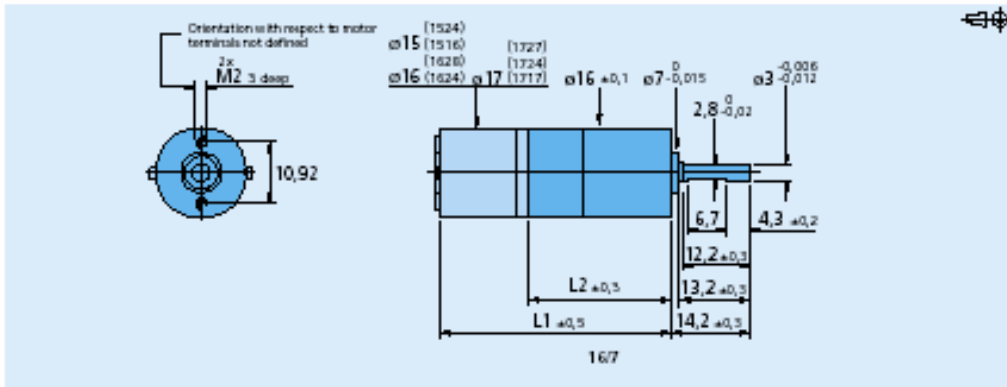
0,3 Nm

For combination with
DC-Motors:
1516, 1524, 1624, 1717, 1724, 1727
Brushless DC-Servomotors:
1628
DC-Motor-Tacho Combinations:
1841

Series 16/7

	16/7
Housing material	metal
Geartrain material	all steel
Recommended max. input speed for:	
- continuous operation	5 000 rpm
Backlash, at no-load	$\leq 1^\circ$
Bearings on output shaft	preloaded ball bearings
Shaft load, max.:	
- radial (6,5 mm from mounting face)	≤ 30 N
- axial	≤ 5 N
Shaft press fit force, max.	≤ 5 N
Shaft play (on bearing output):	
- radial	$\leq 0,02$ mm
- axial	$= 0$ mm
Operating temperature range	$-30 \dots +100$ °C

reduction ratio (nominal)	weight without motor	length without motor L2	length with motor						output torque		direction of rotation (reversible)	efficiency
			1516 T	1524 T	1628 T	1717 T	1724 T	1727 U	continuous operation	intermittent operation		
	g	mm	L1	L1	L1	L1	L1	L1	M max. mNm	M max. mNm	=	%
3,71:1	18	17,0	32,8	40,8	45,0	34,0	41,0	44,2	200	300	=	90
14 :1	23	21,2	36,9	44,9	49,1	38,1	45,1	48,3	300	450	=	80
43 :1	28	25,3	41,1	49,1	53,3	42,3	49,3	52,5	300	450	=	70
66 :1	28	25,3	41,1	49,1	53,3	42,3	49,3	52,5	300	450	=	70
134 :1	33	29,4	45,2	53,2	57,4	46,4	53,4	56,6	300	450	=	60
159 :1	33	29,4	45,2	53,2	57,4	46,4	53,4	56,6	300	450	=	60
246 :1	33	29,4	45,2	53,2	57,4	46,4	53,4	56,6	300	450	=	60
415 :1	38	33,5	49,3	57,3	61,5	50,5	57,5	60,7	300	450	=	55
592 :1	38	33,5	49,3	57,3	61,5	50,5	57,5	60,7	300	450	=	55
989 :1	38	33,5	49,3	57,3	61,5	50,5	57,5	60,7	300	450	=	55
1 526 :1	38	33,5	49,3	57,3	61,5	50,5	57,5	60,7	300	450	=	55
2 608 :1	43	37,6	53,4	61,4	65,6	54,6	61,6	64,8	300	450	=	50
4 365 :1	43	37,6	53,4	61,4	65,6	54,6	61,6	64,8	300	450	=	50
5 647 :1	43	37,6	53,4	61,4	65,6	54,6	61,6	64,8	300	450	=	50



For notes on technical data and lifetime performance refer to "Technical Information".

Specifications subject to change without notice.

Appendix A. (Continued)

A.7. Gripper's Optical Encoder



Encoders

Magnetic Encoders

Features:
64 to 512 Pulses per revolution
2 Channels
Digital output

Series IE2 – 512

		IE2 – 64	IE2 – 128	IE2 – 256	IE2 – 512	
Pulses per revolution	N	64	128	256	512	
Signal output (quadrature)		2				channels
Supply voltage	V _{CC}	4.5 to 5.5				V DC
Current consumption, typical (V _{CC} = 5 V DC)	I _{CC}	typ. 6, max. 12				mA
Output current, max. ¹⁾	I _{OUT}	5				mA
Pulse width	P	180 ± 45				°e
Phase shift, channel A to B	φ	90 ± 40				°e
Signal rise/fall time, max. (C _L = 50 pF)	t _{trf}	0.1 / 0.1				µs
Frequency range ²⁾ , up to	f	20	40	80	160	kHz
Inertia of code disc	J	1.275 · 10 ⁻⁶				oz·in·sec ²
Operating temperature range		– 25 to +85 (– 13 to +185)				°C (°F)

¹⁾ V_{CC} = 5 V DC. Low logic level < 0.5 V, high logic level > 4.5 V; CMOS and TTL compatible

²⁾ Velocity (rpm) = f (Hz) x 60/N

Ordering Information

Encoder	number of channels	pulses per revolution	In combination with
IE2 – 64	2	64	DC Micromotors series 1336...C, 1516...SR, 1524...SR, 1717...SR, 1724...SR, 1727...C 2224...SR, 2342...CR, 2642...CR, 2657...CR, 3242...CR, 3257...CR, 3863...C
IE2 – 64	2	64	
IE2 – 64	2	64	
IE2 – 128	2	128	
IE2 – 128	2	128	
IE2 – 128	2	128	
IE2 – 256	2	256	Brushless DC-Servomotors series 1628...B, 2036...B, 2444...B
IE2 – 256	2	256	
IE2 – 256	2	256	
IE2 – 512	2	512	
IE2 – 512	2	512	
IE2 – 512	2	512	

Features

These incremental shaft encoders in combination with the FAULHABER DC-Micromotors and brushless DC-Servomotors are used for indication and control of both shaft velocity and direction of rotation as well as for positioning.

The encoder is integrated in the DC-Micromotors SR-Series and extends the overall length by only 1.4 mm (0.06 in.) and built-up option for DC-Micromotors and brushless DC-Servomotors.

Hybrid circuits with sensors and a low inertia magnetic disc provide two channels with 90° phase shift.

The supply voltage for the encoder and the DC-Micromotor as well as the two channel output signals are interfaced through a ribbon cable with connector.

Details for the DC-Micromotors and suitable reduction gearheads are on separate catalog pages.

Output signals / Circuit diagram / Connector information

EDG Connector
01-1001
M12M (1M)

Output signal
with absolute reference zero
from manufacturer

Output circuit
Basic order: 1001000000000000
Access by option: M42

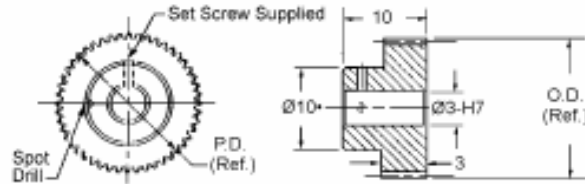
MikroMo Electronics, Inc. · 14881 Evergreen Avenue · Clearwater · FL 33762-3008 · Toll-Free: (800) 807-9166 · Fax: (727) 573-5918 · Info@mikromo.com · www.mikromo.com

Appendix A. (Continued)

A.8. Gripper's Spur Gears

SPUR GEARS

MODULE	STYLE	BORE	FACE	MATERIAL	QUALITY	PRESSURE ANGLE
0.25	PIN HUB	3	3	STAINLESS STEEL DIN 1.4305 OR ALUMINUM DIN 3.1356 ANODIZED	DIN 7/AGMA 10	20°



STAINLESS STEEL STOCK NO.	ALUMINUM STOCK NO.	NO. OF TEETH	PITCH DIA.	OUTSIDE DIA.
PJS20-32	PJA80-32	32*	8.00	8.50
PJS20-34	PJA80-34	34*	8.50	9.00
PJS20-35	PJA80-35	35*	8.75	9.25
PJS20-36	PJA80-36	36*	9.00	9.50
PJS20-38	PJA80-38	38*	9.50	10.00
PJS20-39	PJA80-39	39*	9.75	10.25
PJS20-40	PJA80-40	40*	10.00	10.50
PJS20-42	PJA80-42	42*	10.50	11.00
PJS20-44	PJA80-44	44	11.00	11.50
PJS20-46	PJA80-46	46	11.50	12.00
PJS20-47	PJA80-47	47	11.75	12.25
PJS20-48	PJA80-48	48	12.00	12.50
PJS20-50	PJA80-50	50	12.50	13.00
PJS20-52	PJA80-52	52	13.00	13.50
PJS20-54	PJA80-54	54	13.50	14.00
PJS20-56	PJA80-56	56	14.00	14.50
PJS20-58	PJA80-58	58	14.50	15.00
PJS20-60	PJA80-60	60	15.00	15.50
PJS20-62	PJA80-62	62	15.50	16.00
PJS20-64	PJA80-64	64	16.00	16.50
PJS20-65	PJA80-65	65	16.25	16.75
PJS20-67	PJA80-67	67	16.75	17.25
PJS20-70	PJA80-70	70	17.50	18.00
PJS20-72	PJA80-72	72	18.00	18.50
PJS20-74	PJA80-74	74	18.50	19.00
PJS20-75	PJA80-75	75	18.75	19.25
PJS20-80	PJA80-80	80	20.00	20.50
PJS20-84	PJA80-84	84	21.00	21.50
PJS20-88	PJA80-88	88	22.00	22.50
PJS20-90	PJA80-90	90	22.50	23.00
PJS20-96	PJA80-96	96	24.00	24.50
PJS20-100	PJA80-100	100	25.00	25.50
PJS20-105	PJA80-105	105	26.25	26.75
PJS20-127	PJA80-127	127	31.75	32.25
PJS20-144	PJA80-144	144	36.00	36.50
PJS20-156	PJA80-156	156	39.00	39.50
PJS20-168	PJA80-168	168	42.00	42.50

Available on request:
Other numbers of teeth:
14 - 1/2° pressure angle;
Quality Class AGMA 12 and 14 / DIN 5 and 3

* For 25-42 teeth hub diameter equals 5.5

Other bore sizes, styles and materials available.
Check our website or call us.

For Couplings See Section MG.
For Shafts, Bearings and Collars See Section MI.

1-800-232-BERG (USA ONLY)

MB 10

WWW.WMBERG.COM

Appendix A. (Continued)

A.9. Gripper's Slip Clutch

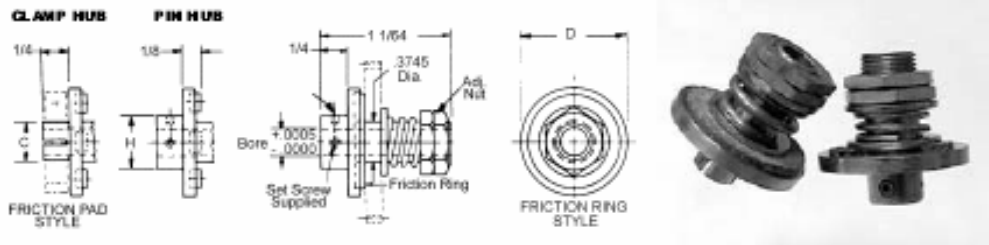
SLIP CLUTCH

BORE	STYLE	TORQUE	MATERIAL
1/8" TO 1/4"	PIN HUB	ADJUSTABLE 0 OZ.IN. TO 50 OZ.IN	303 STAINLESS STEEL

STOCK NO.	BORE SIZE	H	D	FRIC TION FACE	ADJUSTABLE SLIP TORQUE
JC-10	.1248	5/16			
JC-11	.1873	3/8	5/8	FRIC TION RING	0 TO 10 OZ. IN.
JC-12	.2498	1/2			
JC-10-50	.1248	5/16			
JC-11-50	.1873	3/8	5/8	FRIC TION RING	10 TO 50 OZ. IN.
JC-12-50	.2498	1/2			
JA-1	.1248	5/16			
JA-2	.1873	3/8	1	FRIC TION RING	10 TO 50 OZ. IN.
JA-3	.2498	1/2			
JC-1	.1248	5/16			
JC-2	.1873	3/8	1	FRIC TION PADS	0 TO 10 OZ. IN.
JC-3	.2498	1/2			
JC-1-50	.1248	5/16			
JC-2-50	.1873	3/8	1	FRIC TION PADS	10 TO 50 OZ. IN.
JC-3-50	.2498	1/2			

BORE	STYLE	TORQUE	MATERIAL
1/8" TO 1/4"	CLAMP HUB	ADJUSTABLE 10 OZ.IN. TO 50 OZ.IN	303 STAINLESS STEEL

STOCK NO.	BORE SIZE	C	D	FRIC TION FACE	ADJUSTABLE SLIP TORQUE
JA-1C	.1248	3/16			
JA-2C	.1873	1/4	1	FRIC TION RING	10 TO 50 OZ. IN.
JA-3C	.2498	5/16			
JC-1-C	.1248	3/16			
JC-2-C	.1873	1/4	1	FRIC TION PADS	0 TO 10 OZ. IN.
JC-3-C	.2498	5/16			
JC-1-50C	.1248	3/16			
JC-2-50C	.1873	1/4	1	FRIC TION PADS	10 TO 50 OZ. IN.
JC-3-50C	.2498	5/16			



Order gears and clamps separately.

1-800-232-BERG

G 80

WWW.WMBERG.COM

Appendix A. (Continued)

A.10. PIC Servo SC Motion Controller Board

PIC-SERVO SC Motion Control Board

For Brush-type DC Motors (P/N: KAE-T0V10-BDV1)

The **PIC-SERVO SC** Motion Control board is a complete servo control system with the following features:

- **PIC-SERVO SC** motion control chip providing servo control of DC motors with incremental encoders, including trapezoidal profiling, velocity profiling and support for coordinated multi-axis motions.
- LMD18200 amplifier capable of driving 3 amps continuously, 6 amps peak at up to 48vdc. Built-in thermal, overcurrent and undervoltage protection.
- Current sensing, active current limiting, and overvoltage protection.
- PWM and DIR signals are provided for use with other external amplifiers.
- RS485 serial interface allows up to 32 **PIC-SERVO** controllers to be controlled from a single serial port. Connects to an RS232 port through commonly available full-duplex adapters or using the **Z232-485** converter board.
- Step & Direction inputs for control from stepper based indexing systems.
- Two limit switch inputs for overtravel protection.
- Its small size (2" x 3") allows it to be mounted near motors, reducing noise and simplifying wiring.
- Windows test software provided including 32 bit Windows DLL and C source code. DOS based C code and Basic code are also available.

1. Quick Start

What you will need:

PIC-SERVO SC Motion Control Board
Z232-485 Converter Board, **USB-NMC** Adapter, or equivalent
DC Motor (48v max., 3 amp continuous current max.) with TTL compatible encoder
Motor power supply (12v min. - 48vdc max.)
Logic power supply (7.5 - 12vdc, 500 ma)
Motor/encoder cable (DB15 male connects to your motor)
10 pin flat ribbon cable with standard IDC socket connectors at both ends
Straight DB9 male / DB9 female cable to PC COM port
PC compatible computer running Windows
Test software - **NMCTest** for Windows95/98/2000/NT/XP
(available for download from www.jrkerr.com)

Most of the cables are available from computer or electronics stores. However, you will probably have to make your own motor/encoder cable to connect to your particular motor. Refer to *Section 2.1* for the connector pin definitions. To start off, you only need to connect M+, M-, Encoder A, Encoder B, Encoder +5v and Encoder GND. Other connections can be made as

CAUTION

The **PIC-SERVO SC** Motion Control Board does not incorporate safeguards for fail-safe operation. As such, this board should not be used in any device which could cause injury, loss of life, or property damage. JEFFREY KERR, LLC makes no warranties whatsoever regarding the performance, operation, or fitness of this board for any particular purpose.

JEFFREY KERR, LLC • www.jrkerr.com

Appendix A. (Continued)

needed. Note that when testing, you may have to swap the M+ and M- leads to correct for the polarity of your motor.

Interconnections and Jumpers:

Basic interconnections and jumpers are shown in *Figure 1* for both a single controller and for a multiple controller configuration. On the **Z232-485** converter, jumpers JP3 and JP4 are installed in the 1-2 position for use as a simple converter. (Please refer to the **Z232-485** documentation for use with the optional standalone processor cards.) Jumper JP5 is installed to distribute logic power to the controller boards over the communications cable. Logic power is supplied on connector JP6. If you are using a different type of serial port adapter, you may attach power to the pins of JP8 on the **PIC-SERVO SC** board.

On the **PIC-SERVO SC** controller board, jumpers JP6 and JP7 are installed to connect logic power supplied by the communications cable to the board's logic supply. In the *single* controller configuration, the three jumpers labeled JP3, JP4 and JP5 should be installed as shown. In the *multiple* controller configuration, these jumpers should only be installed on *last* controller, furthest from the PC host. On all intermediate controllers, jumpers at JP3, JP4 and JP5 should be left *uninstalled*.

Motor power should be connected to the two screw terminals, with 12 - 48vdc connected to the terminal towards the edge of the board and GND connected to the terminal towards the center as shown in *Figure 1*.

We recommend that both your logic power supply and your motor power supply have floating outputs. The **PIC-SERVO SC** board creates a common ground between the (-) side of the logic supply, the (-) side of the motor supply and the communications ground. With your power supply outputs floating, you will have a single ground referenced to your PC's ground, thus avoiding ground loops.

Loading and Running Software:

First unzip NMCTEST.ZIP into a single directory. Before starting up the test code, make sure all of your jumpers and interconnections are as shown in *Figures 1*. Also make sure you have logic power supplied to the **Z232-485** converter.

Run the program NMCTest.exe. Select the correct COM port when prompted (leaving the default baud rate at 19200 for now). If you are using a different COM port, you will get an error message saying no modules were found. If this is the case, click on the Reset Network button and set the COM port to the correct value. The program will attempt to locate controllers on the RS485 network and will respond with the number of controllers found. If the number of controllers reported does not match the number connected, re-check the interconnections, jumpers and power, and then try again.

The list box on the left side of the window will display the list of motors found. **PIC-SERVO** module 1 will be the last controller which is furthest from the host PC. Clicking on different controllers will display the status and controls for that particular motor. Click on **PIC-SERVO 1** and spin the motor shaft by hand. See that the position changes accordingly in the status panel.

Before enabling the motor servo make sure that the motor is disconnected from any mechanism which might be damaged. To test the motor, first turn on the motor power. You should see the

Appendix A. (Continued)

Motor Power box checked in the status panel. Next, click on the Enable Amplifier box in the Motion Command panel. Now click on the STOP! button. Try turning the motor shaft by hand. If the motor jerks and stops, or spins out of control, turn off the motor power and try swapping the M+ and M- leads on the motor. Turn the power back on, click on STOP! again. The motor should attempt to hold a fixed position. If it does, click on Pos mode, type in a position value of 1000, and then click on "GO". The motor should move to position 1000 (or close to it, depending on how the gains are set). Try moving to a bunch of different positions until you are satisfied that the motor is moving as it should. (Note that if your motor has a gearhead, the motion of 1000 counts may produce an imperceptibly small motion, and you should use a larger number instead.)

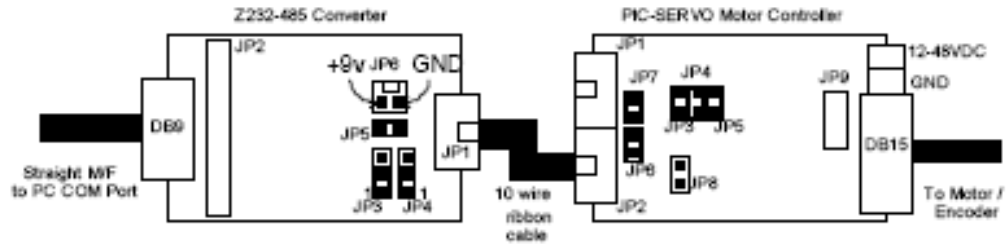
If the motor seems to buzz when it is stationary, try setting the Deadband Compensation (DC) value to 1 or 2 to compensate for non-linearity in the amplifier at close to zero output.

Otherwise, the control gains, and maximum velocities and accelerations are set to default values which are reasonable for most small motors. Please refer to the *PIC-SERVO SC* chipset data sheet for details on the values for the gains, velocities and accelerations. The NMCTest online help also has a great deal of information about the *PIC-SERVO* controller.

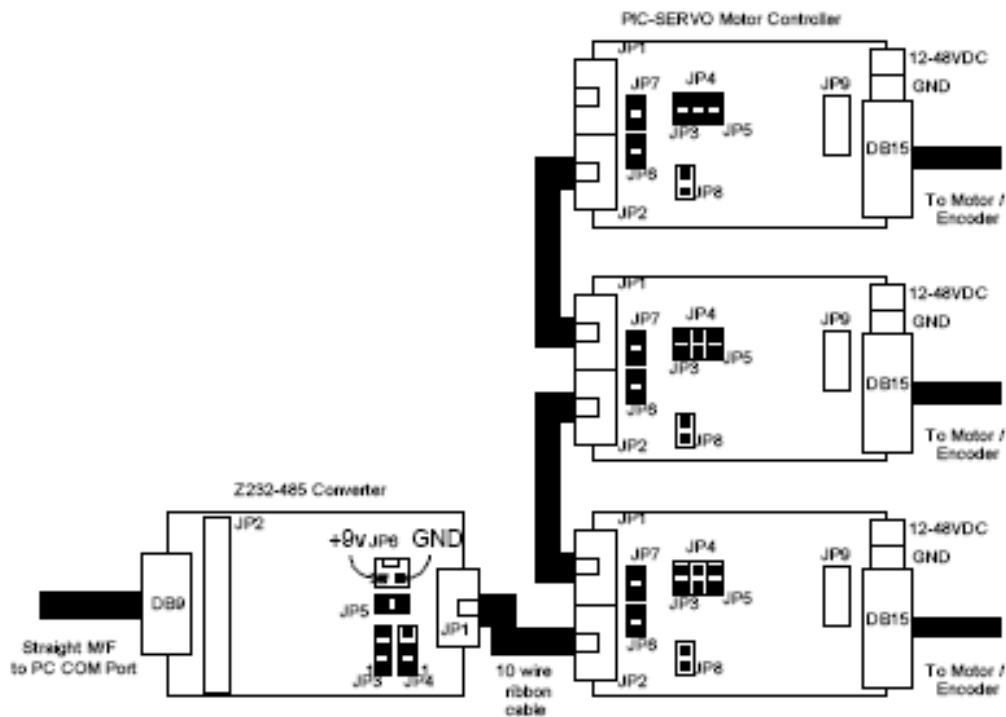
At this point, you will want to read the *PIC-SERVO SC* controller chip data sheet in detail to discover all of the features of your *PIC-SERVO SC* motion controller and for information on how to start developing your application.

Appendix A. (Continued)

Single Controller Configuration



Multiple Controller Configuration



CAUTION: Connecting communications cables incorrectly, or installing jumpers JP3, JP4 and JP5 (on the PIC-SERVO board) in the wrong location may damage the PIC-SERVO SC or other NMC controller chip!

Figure 1 - Basic Interconnections.

Appendix A. (Continued)

2. Connectors and Jumpers

2.1 Pinouts

Motor Connector P1 - DB15 Male

<i>Pin</i>	<i>Definition</i>
1	Motor Output (M+)
2	Motor Output (M+)
3	LED power - pulled up to 5v with a 330 ohm resistor (for use with opto-interrupt type switches)
4	Limit Switch 1 (pulled up to 5v with a 4.7k resistor)
5	Encoder Channel A (pulled up to 5v with a 4.7k resistor)
6	Encoder Channel B (pulled up to 5v with a 4.7k resistor)
7	Limit Switch 2 (pulled up to 5v with a 4.7k resistor)
8	Encoder Index (pulled up to 5v with a 4.7k resistor)
9	Motor Output (M-)
10	Motor Output (M-)
11	GND
12	GND
13	GND (supplied to encoder)
14	+5v (supplied to encoder)
15	GND

Network Connectors JP1, JP2 - 10 Pin Flat Ribbon IDC Sockets

<i>Pin</i>	<i>Definition</i>
1	PIC-SERVO RCV+
2	PIC-SERVO RCV-
3	PIC-SERVO XMT+
4	PIC-SERVO XMT-
5	PIC-SERVO ADDR_IN on JP1, ADDR_OUT on JP2
6	GND
7	Logic power (7.5 - 12vdc)
8	GND
9	Logic power (7.5 - 12vdc)
10	GND

External Amplifier Connector JP9 - 8 Pin Single Row Locking Header

<i>Pin</i>	<i>Definition</i>
1	PWM output - 20 KHz square wave magnitude signal or Antiphase signal
2	Direction output (not needed if in Antiphase mode)
3	Amplifier Enable output (active high)
4	Limit2 input (same as P1, pin 7) or Step input if in Step & Direction mode
5	Limit1 input (same as P1, pin 4) or Direction input if in Step & Direction mode
6	MCLR input - can be used to enable/disable controller if in Step & Direction mode - pull low to disable controller (pulled up to 5v with a 4.7k resistor)
7	ADDR_OUT output - can be used as a servo fault indicator if in Step & Direction mode (signal goes high when servo is disabled)
8	GND

Motor Power Connector P2 - Screw Terminals

<i>Pin</i>	<i>Definition</i>
1	Motor Power 12 - 48vdc (at edge of board)
2	Motor Power Ground (connected internally to logic ground)

JEFFREY KERR, LLC • www.jrkerr.com

Appendix A. (Continued)

Logic Power Connector JP8 - 2 Pin Locking Header

(Use only if logic power is **not** supplied via the network communications cable.)

Pin	Definition
1	7.5 - 12vdc (pin towards the lower edge of the board)
2	Ground (pin towards the center of the board)

2.2 Jumpers

PIC-SERVO Motor Control Board Jumpers:

Jumper	Description
JP3	Connects ADDR_IN to GND. Insert jumper for the last <i>PIC-SERVO</i> on the network (or if only 1 <i>PIC-SERVO</i> is used)
JP4, JP5	Enables termination resistors on RX and TX. Insert these jumpers for the last <i>PIC-SERVO</i> on the network (or if only 1 <i>PIC-SERVO</i> is used).
JP6,JP7	Logic power interconnection. Inserting JP6 connects logic power to network connector JP2. Inserting JP7 connects logic power to JP1. These are used to control the distribution of logic power over the network cables. Normally both these jumpers are installed.

3. *PIC-SERVO* SC Motion Control Board Description

The *PIC-SERVO* SC Motion Control board is a complete motor servo control system including a servo controller, amplifier, serial communications interface, optical encoder interface, and limit switch inputs. The board is designed so that up to 32 controllers can be connected directly to a single standard serial port (using an RS232-RS485/RS422 converter if necessary).

3.1 *PIC-SERVO* SC Motion Control Chip

The *PIC-SERVO* SC motion control chip forms the core of the controller, performing all of the tasks of servo control, motion profiling, communications and amplifier control. Please refer to the *PIC-SERVO* SC chip data sheet for complete details on the theory of operation of the servo control and motion profiling algorithms. Please see our web pages www.jrkerr.com/docs.html and www.jrkerr.com/software.html for a wide variety of information and software useful for developing your application.

Note that earlier versions of the *PIC-SERVO* chipset (v.5 and earlier) cannot be used in the *PIC-SERVO* SC board.

3.2 Communications Interface

The *PIC-SERVO* SC uses an RS485 multi-drop interface for allowing multiple control modules to communicate over the same RS485 communication port. The host computer sends commands out over a dedicated pair of transmit wires, and all status data comes back over a shared pair of receive wires. Because the host has a dedicated transmit line, a standard RS232 serial port can be used with simple RS242-RS485 converter.

With multiple controllers on a single network, each controller must have a unique address for sending commands. Rather than using dip switches or jumpers to assign addresses, the *PIC-SERVO* SC uses a method of daisy-chaining an ADDR_IN signal and an ADDR_OUT signal for dynamically assigning addresses. With the controllers interconnected as shown in *Figure 1*, the ADDR_OUT signal of one board is connected to ADDR_IN of the next board. The very last board has ADDR_IN jumpered to GND. On power-up, all boards with ADDR_IN held high will

Appendix A. (Continued)

have their communications disabled. Therefore, only the last board will be able to communicate with a default address of 0.

To initialize the network, a command is sent to the last controller (with address 0) to change its address to a value of 1. This has the side effect of causing its ADDR_OUT to lower, enabling communications with the next controller. The next command sent to address 0 will now be sent to the second-to-last controller. This process of assigning addresses is repeated until all controllers have been given unique addresses.

3.3 Amplifier

The *PIC-SERVO* Motor Control board uses an LMD18200 H-bridge amplifier to drive DC brush-type motors with up to 3 amps continuously, 6 amps peak, with a supply voltage of 12 to 48vdc. This amplifier has overcurrent, overtemperature and undervoltage protection. The *PIC-SERVO SC* chip provides additional overvoltage protection for the amplifier. Note that if you are driving more than 500 ma, you will likely need to mount a heat sink to the tab of the amplifier. This tab is connected to GND, so it may also be bolted directly to a metal enclosure if your case ground is connected to your power ground.

The LMD18200 provides a current sense output which can be read by the host, and may be also used for current limiting by the *PIC-SERVO SC* chip. The board is configured so that the *PIC-SERVO SC* will produce an A/D reading of about 39 counts per amp of current. To enable current limiting, you should set the current limit value to an odd, non-zero value (ie, 39, 51, etc.). For example to limit the current to about 2 amps, you should set the current limit value to approximately 2×39 , or 79.

The *PIC-SERVO SC* controller chip provides additional overvoltage and undervoltage protection for the amplifier. As configured on the *PIC-SERVO SC* controller board, if the motor supply voltage drops below 10.8v, the servo will be disabled, and the amplifier will be temporarily disabled. If the voltage rises back above 10.8v, the amplifier will be re-enabled (if it had been enabled before the voltage drop), but you must send a command to the board if you want to re-enable the servo.

If the motor supply voltage rises above 54v, the amplifier will be temporarily disabled until the voltage drops back below 54v. The P.I.D. servo, however, will remain enabled. This is primarily to prevent a decelerating motor from acting as a generator and driving the voltage above a safe value.

If greater than 3 amps is required, the *PIC-SERVO SC* Motion Control board can be used with an external amplifier. External amplifiers may be for brush or *brushless* motors. PWM, Direction and Enable signals are provided on connector JP9. The *PIC-SERVO SC*'s 20 KHz PWM output can be configured as either a PWM magnitude signal with a separate Direction signal, or as a single Antiphase PWM signal. In Antiphase output mode, a 50% duty cycle output will correspond to a *zero* drive output, a 100% duty cycle will correspond to a full *forward* output, and a 0% duty cycle will correspond to a full *reverse* output. This Antiphase output mode is more convenient for connecting to external amplifiers requiring an analog drive input.

The NMCTest program may be used for enabling the Antiphase mode, and also for permanently storing the Antiphase output mode option in EEPROM. On powerup, the Antiphase output mode will be automatically restored.

Appendix A. (Continued)

3.4 Limit Switch Inputs

The *PIC-SERVO SC* board has two limit switch inputs which can be used for overtravel protection. A normally closed limit switch should be connected between the limit switch input and ground. (The limit switch inputs are pulled-up internally to +5v.) When limit switch 1 is hit (*i.e.*, the switch is opened), forward motion of the motor will be inhibited. When limit switch 2 is hit, reverse motion will be inhibited. Note that the limit switch overtravel protection must be enabled with a command to the *PIC-SERVO SC*.

3.5 Step & Direction Inputs

The *PIC-SERVO SC* has the option of configuring the limit switch inputs as Step & Direction inputs instead. This is useful when using the *PIC-SERVO SC* controller and servo motor as a high performance replacement for a stepper motor & driver. If the Step and Direction inputs are being used, the limit switch inputs are unavailable, and you should connect your limit switches to your stepper indexing system instead.

When Step & Direction mode is enabled, the ADDR_OUT signal may be used to detect a servo fault condition. During normal operation, the ADDR_OUT output will be low. However, if in Step & Direction mode a servo fault condition is detected and the servo is disabled, the ADDR_OUT signal will go HI.

Also when in Step & Direction mode, it is useful for your stepper indexer system to be able to disable the *PIC-SERVO SC* through a TTL level signal. The MCLR input (internally pulled high to +5v) can be pulled low to disable the *PIC-SERVO SC*.

The Step, Direction, ADDR_OUT and MCLR signals all appear on connector JP9 for connecting to your stepper indexing system.

Lastly, the *PIC-SERVO SC* has an internal EEPROM which can be configured to make the board power-up ready to accept Step & Direction signals. The NMCTest program has a "Configure EEPROM" button which will allow you to store the proper startup parameters in EEPROM. When configuring the EEPROM, you should click on the "Restore Current Address", "Enable Amplifier", "Enable Servo" and "Enable step/direction inputs" options and then save the parameters. When the board is next powered up, the servo will be enabled and ready to accept step and direction signals.

3.6 Physical Dimensions

The *PIC-SERVO* Motor Control board is 2.1" x 3.1" with four 0.156" dia. mounting holes at 1.8" x 2.45". The overall dimensions of the *PIC-SERVO SC* board and the placement of the connectors and amplifier chip are identical to earlier *PIC-SERVO* boards.

Appendix A. (Continued)

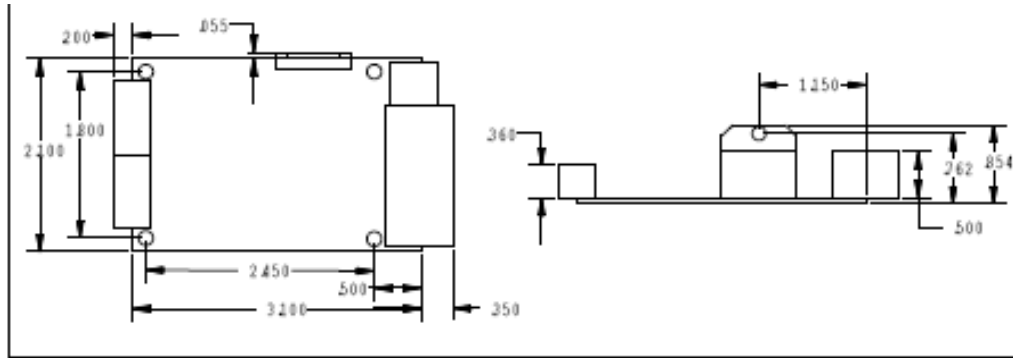


Figure 2 - PIC-SERVO SC board dimensions.

4. Contact Information

Additional information may be found from these sources:

JEFFREY KERR, LLC Web Site

www.jrkerr.com

Your source for data sheets, application notes and test code. Technical support is provided via e-mail. Please see jrkerr.com/contact.html for contact information.

Microchip

www.microchip.com

The PIC-SERVO SC is based on the Microchip PIC18F2331 microcontroller. Please refer to the Microchip data sheet for this devices for complete electrical, timing, dimensional and environmental specifications.

National Semiconductor

www.national.com

Datasheet for the LMD18200 PWM amplifiers.

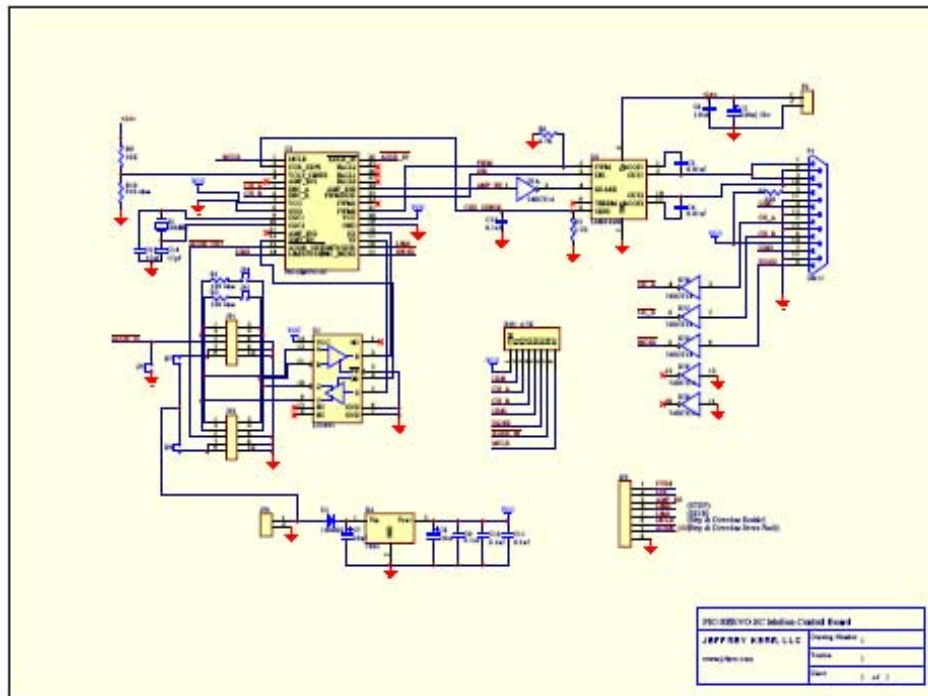


Figure 3 - PIC-SERVO SC Motion Control Board Schematic.

Appendix A. (Continued)

A.11. SSA-485 Smart Serial Adapter

SSA-485 Smart Serial Adapter

Serial Converter / Stand-alone host controller

The **SSA-485** Smart Serial Adapter is a USB/RS232 to RS485 converter, which allows you to communicate with NMC control modules such as the **PIC-SERVO**, **PIC-STEP**, and **PIC-I/O** through your host computer's USB or RS232 COM port. The **SSA-485** may also be used as a stand-alone host by adding either our **Simple Sequencer** chip or a PIC18F2620 microcontroller. The **SSA-485** includes the following features:

- Supports USB or RS232 serial interface with host computer. USB interface is ideal for running diagnostics on a laptop.
- Up to 32 NMC **PIC-SERVO**, **PIC-STEP**, or **PIC-I/O** modules can be controlled from a single **SSA-485** Smart Serial Adapter.
- RS232 operation support baud rates from 19,200 up to 115,200.
- USB driver support for Windows, Linux, Apple OS, and other operating systems.
- May be used as a stand-alone host with either our **Simple Sequencer** processor chip or a PIC18F2620 microcontroller.
- On-board connector for the Microchip MPLAB ICD2 In-Circuit Debugger supports optional PIC18F2620 code development in C or assembly language.
- If used with a PIC18F2620 microcontroller, 10 pin header provides additional I/O capabilities.

1. Use as a Basic Converter

In normal operation, the **SSA-485** is a USB/RS232 to RS485 (4-wire, full-duplex) converter, allowing you to communicate with NMC control modules such as the **PIC-SERVO**, **PIC-STEP**, and **PIC-I/O** directly through your host computer's USB or RS232 COM port. Up to 32 NMC modules may be controlled using a single **SSA-485**. Logic power is supplied to the NMC modules and **SSA-485** from a single point and distributed through the NMC communications cables (default) or logic power can be supplied to the boards individually.

The **SSA-485** may be configured for either RS232 or USB communication with the host PC. For communications settings (such as baud rate and serial port settings) with a particular controller module, please refer to the serial communications protocol description in the corresponding **PIC-SERVO**, **PIC-STEP**, or **PIC-I/O** data sheets.

Please see *Section 3* for jumper and pin definitions.

1.1 RS232 Converter Mode

For use as RS232 to RS485 converter, jumper JP8 should be set to the position marked '232' on the board, and jumpers JP3 and JP4 should be set to the position marked 'PT' on the board as indicated in *Figure 1*. A straight (*i.e.*, not a null modem) DB9 Male / DB9 Female extension cable should be used to connect the DB9 connector P1 to standard PC RS232 (COM) ports.

--- CAUTION ---

The **SSA-485** Smart Serial Adapter does not incorporate safeguards for fail-safe operation. As such, this board should not be used in any device which could cause injury, loss of life, or property damage. JEFFREY KERR, LLC makes no warranties whatsoever regarding the performance, operation, or fitness of this board for any particular purpose.

Appendix A. (Continued)

1.2 USB Converter Mode

For use as a USB to RS485 converter, jumper JP8 should be set to the position marked 'USB' on the board, and jumpers JP3 and JP4 should be set to the position marked 'PT' on the board. A USB Type A male / USB Type B male extension cable should be used to connect the USB connector JP7 to standard PC USB ports.

Note that when used in USB converter mode, your host computer creates a 'virtual' COM port, and your host's software will communicate with the NMC modules exactly as if you were using a standard RS232 COM port. See *Section 1.5* below for details on USB driver installation.

1.3 Logic Power

Logic power (7.5v to 12v D.C., 500 ma typ.) to the **SSA-485** and to NMC control modules can be supplied at a single point and distributed over the NMC network flat ribbon cables, or it can be supplied to the various boards individually. To distribute the **SSA-485**'s logic power to the network cables, insert a jumper on JP5 (default position); remove the jumper to power the board separately. Please see the schematic diagrams for your controller modules for more details on distributing logic power to the rest of your controllers over the NMC communications cables.

Typically, if all NMC control modules and the **SSA-485** converter share the same logic power, power is supplied via connector JP6 on the **SSA-485** board. If this is the case, make sure that jumper JP5 is installed. In this configuration, 500 ma power supply can drive one **SSA-485** converter and approximately 4 additional NMC control modules (the converter and modules each draw about 100 ma).

1.4 Connection to NMC Modules

Figure 1 below shows how to connect multiple NMC modules to you host computer via the **SSA-485**. Modules are daisy-chained with 10-wire ribbon cables connecting JP1 of one module to JP2 of the next module as shown. On the last controller in the chain, connector JP1 should be left open, and jumpers JP3, JP4, and JP5 should be installed. On intermediate modules, JP3, JP4, and JP5 should be removed.

The 10 pin header connector (JP1) can be used with 10 pin flat ribbon cables (with standard IDC type connectors) to interconnect to NMC control modules. For NMC network interconnect cables longer than 3 meters, it is recommended that you use twisted-pair flat ribbon cable.

1.5 USB Driver Installation

The USB interface of the **SSA-485** uses the FT232BM, a USB serial port converter chip from FTDI, Inc. Drivers for Windows, Linux, Apple OS, and other operating systems are available from FTDI's web site www.ftdichip.com. With the Windows drivers you can create a virtual COM port (COM1 – COM256) and communicate with NMC controllers with the same code written for use with your PC's standard RS232 ports.

Most versions of Windows XP have drivers for the FDTI interface chip pre-installed. When you connect the **SSA-485** to your PC's USB port and apply logic power, Windows should automatically recognize the adapter and create a virtual COM port. If Windows does not recognize the driver, or if you are using a different operating system, please visit www.ftdichip.com to find drivers and installation instructions.

Appendix A. (Continued)

Once the driver is installed, you should open the Control Panel, double click on System, and select the Hardware Device Manager. Under "Ports" you should see listed "USB Serial Port (COMx)". If the COM port is not set to COM5 or COM6, right-click on the port and select Properties. Under the Port Settings tab, click on the 'Advanced' button and set the port to COM5 or COM6. This will insure that you have no conflicts with existing serial ports or modems.

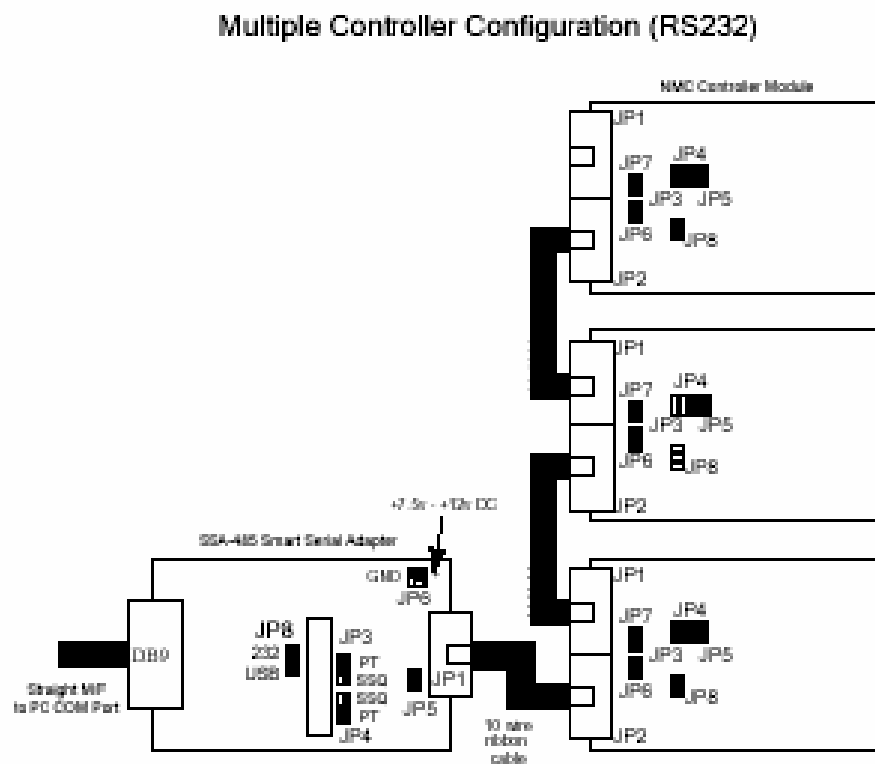
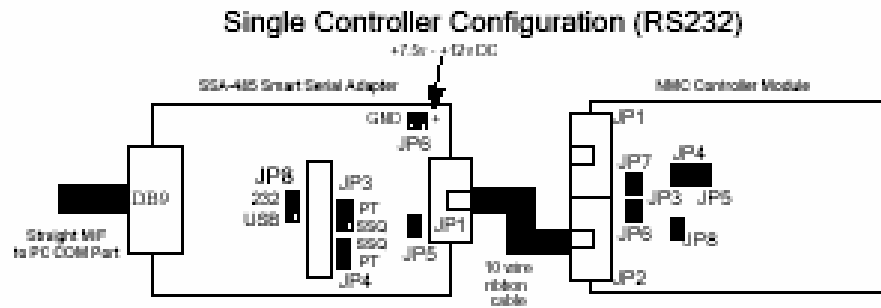
Please see the documentation included with the drivers for additional installation information.

Note that if you turn off the logic power to your NMC modules, the USB serial port will not appear in the list of ports. It appears and disappears dynamically depending on whether the **SSA-485** is plugged in and powered up.

You are now ready to run the NMCTest utility program to test your **SSA-485** and NMC controller modules. (Make sure you have downloaded the latest version of this program from www.jrker.com/software.html which supports COM5 and COM6). Run this program and use whichever COM port you selected for the converter in the Control Panel.

The **SSA-485** converter and our NMC modules have all been tested at baud rates up to 115,200. In your own applications, it may be possible to use faster baud rates, but these are untested and it is up to the user to verify the reliability of higher speed connections.

Appendix A. (Continued)



Caution: Connecting communications cables incorrectly, or installing jumpers JP3, JP4, JP5 (on the NMC Controller board) in the wrong location may damage the NMC Controller board or the controller chip!

Figure 1 – SSA-485 used as a basic converter.

Appendix A. (Continued)

2. Use as a Stand-alone Host

The socket U4 is designed to accept our **Simple Sequencer** processor chip or a PIC18F2620 microcontroller to create stand-alone motion control systems. If you are using the **Simple Sequencer**, you will also need to populate the socket U5 with a 24LC256 EEPROM chip.

The **Simple Sequencer** chip and the 24LC256 EEPROM chip can be purchased from JEFFREY KERR, LLC. PIC18F2620 microcontrollers can be obtained from Digikey (www.digikey.com), Mouser (www.mouser.com) or other electronics distributors.

A 20 Mhz crystal is included on the board for use with the **Simple Sequencer** or the PIC18F2620. The **Simple Sequencer** requires a 20 Mhz crystal, but the PIC18F2620 may be used with other frequencies. In particular, you can use the PIC18F2620 with a 10 Mhz crystal and the 4x PLL oscillator mode to give an effective clock frequency of up to 40 Mhz. If you need to use a different frequency, you will have to unsolder the crystal X2 and replace it.

2.1 Simple Sequencer Operation

The **Simple Sequencer** is a programmable controller for creating simple sequences of motion and other actions. It is programmed using a simple Windows programming environment which requires no formal programming skills. Please refer to the **Simple Sequencer** documentation (available at www.jrkerr.com/docs.html) for complete details.

To configure the **SSA-485** for Simple Sequencer operation, install the **Simple Sequence** chip (P/N **KAE-SSQV1-BDV1**) in the socket marked U4 and install the 24LC256 chip in the socket marked U5. Set jumpers JP3 and JP4 in the position marked 'SSQ'. Set JP8 for either USB or RS232 host communications. In this configuration, USB/RS232 signals from a host will connect to serial port 0 of the **Simple Sequencer**, and the RS485 signals from the NMC network will be connected to serial port 1 of the **Simple Sequencer**.

Logic power connections and connections to NMC Modules are described in *Section 1*.

2.2 PIC18F2620 Processor Operation

The **SSA-485** supports stand-alone host development using a PIC18F2620 microcontroller. In this configuration, users can insert the 28 pin PIC18F2620 into socket U4 and develop code to use the **SSA-485** as a stand-alone host. The PIC18F2620 is a general purpose microcontroller with a variety of standard peripheral features such as analog inputs, timers, etc.. If you need special processor features, Microchip (www.microchip.com) offers a wide variety of other 28 pin processor chips in the PIC18Fxxxx series which may also be compatible with the **SSA-485**. Please refer to the Microchip PIC18Fxxxx series data sheets for more information.

2.2.1 Configuration

To configure the **SSA-485** for PIC processor operation, install a PIC18F2620 processor in the socket marked U4, and set jumpers JP3 and JP4 in the position marked 'SSQ'. (Socket U5 may be left empty.) Set JP8 for either USB or RS232 host communications. In this configuration, the USB/RS232 signals will be connected to the PIC's hardware UART, and the RS485 signals from the NMC network will be connected to pins RC0 and RC5 which are to be used as a software UART.

Appendix A. (Continued)

2.2.2 Microchip MPLAB ICD 2 Interface

Connector JP9 connects directly to the Microchip MPLAB ICD2 In-Circuit Debugger. The MPLAB ICD2 In-Circuit Debugger, available from Microchip (www.microchip.com), is a complete assembly and C language programming development environment. The MPLAB ICD2 allows users to develop, download, and debug PIC18F2620 code on the **SSA-485**.

Note that when using the ICD2, you should configure it so that Vcc is supplied by the target. See the MPLAB ICD2 User's Guide for details on installing and using the MPLAB ICD2 In-Circuit Debugger.

2.2.3 Auxiliary I/O Connector

Connector JP2 on the **SSA-485** connects to 8 I/O pins on the PIC18F2620 processor. This provides your system with additional I/O capabilities in addition to whatever NMC control modules you may have connected. These pins can be used as general purpose digital I/O, but 3 of the pins can also be used for analog input, 2 can be used for PWM output, and 3 can be used for generating interrupts. See *Section 3* for connector pin definitions.

Note that these additional I/O capabilities are only available if programming your own PIC18F2620 processor. The **Simple Sequencer** does not make use of these I/O pins.

2.2.4 SSA-485 Motion Control Library

The **SSA-485** Motion Control Library is a library of C language functions for communicating with the **PIC-SERVO**, **PIC-STEP**, and **PIC-I/O** modules. This library is for use with Microchip's MCC18 C compiler. (A basic version of this compiler, along with the MPLAB development environment, can be downloaded free from www.microchip.com. An enhanced version of the compiler can be purchased from Microchip.)

Included in the library are high level routines for initializing, controlling, and managing the **PIC-SERVO**, **PIC-STEP**, and **PIC-I/O** modules. The library and example programs are available from www.jrkerr.com/software.html.

3. Connectors and Jumpers

3.1 Connector Pin Definitions

Logic Power Connector **JP6** (1x2 pin locking header – 0.100" spacing)

(Use only if logic power is *not* supplied via network communications cable)

Pin	Definition
1	7.50- 12vdc (pin marked +)
2	Ground (pin marked GND)

RS232 Serial Connector **P1** (female DB9 connector)

Pin	Definition
2	RS232 Transmit Data Output
3	RS232 Receive Data Input
5	Ground
1,4, 6-9	Not Used

Appendix A. (Continued)

USB Connector **JP7** (female USB Type B Socket)

<i>Pin</i>	<i>Definition</i>
1	5 vdc (from USB port)
2	-Data
3	+Data
4	Ground

Modular Jack **JP9** (female 6 pin modular jack)
Connector to Microchip MPLAB ICD2 In-Circuit Debugger

<i>Pin</i>	<i>Definition</i>
1	Vpp/-MCLR
2	5 vdc (configure ICD 2 so that Vcc is supplied by the target.)
3	Ground
4	Program Data
5	Program Clock
6	NC

Network Connector **JP1** – 10 Pin Flat Ribbon IDC Socket

<i>Pin</i>	<i>Definition</i>
1	SSA-485 XMT+
2	SSA-485 XMT-
3	SSA-485 RCV+
4	SSA-485 RCV-
5	N.C.
6	GND
7	Logic Power (7.5 – 12vdc)
8	GND
9	Logic Power (7.5 – 12vdc)
10	GND

Auxiliary I/O Connector **JP2** (1x10 pin locking header – 0.100" spacing)

<i>Pin</i>	<i>Definition</i>
1	5vdc (<i>rightmost pin</i>)
2	PIC Pin 2 (RA0/AN0) – digital I/O or analog input
3	PIC Pin 3 (RA1/AN1) – digital I/O or analog input
4	PIC Pin 4 (RA2/AN2) – digital I/O or analog input
5	PIC Pin 21 (RB0/INT0) – digital I/O or interrupt input
6	PIC Pin 22 (RB1/INT1) – digital I/O or interrupt input
7	PIC Pin 23 (RB2/INT2) – digital I/O or interrupt input
8	PIC Pin 12 (RC1/CCP2) – digital I/O or PWM output
9	PIC Pin 13 (RC2/CCP1) – digital I/O or PWM output
10	GND (<i>leftmost pin</i>)

Appendix A. (Continued)

3.2 Jumpers

SSA-485 Jumpers

Pin	Definition
JP3, JP4	Selects operating mode. Jumper pins marked PT (Pass through) for operation as a simple converter (default). Jumper pins marked SSQ (Simple Sequencer) for optional stand-alone operation.
JP5	Logic power interconnection. Inserting JP5 connects logic power to the network connector JP1. Normally this jumper is installed.
JP8	Used to select RS232 or USB communications with the host computer. Jumper pins marked 232 (default) or USB for RS232 or USB operation.

3.3 Physical Dimension

Figure 2 below gives overall board dimensions and the location of the 4 mounting holes. The RS232 connector and the RS485 NMC network connector are right-angle connectors mounted on the ends of the board. The USB and ICD2 connectors are vertical connectors mounted in the center of the board. The Auxiliary I/O connector and the logic power connector are vertical headers mounted along the top edge of the board.

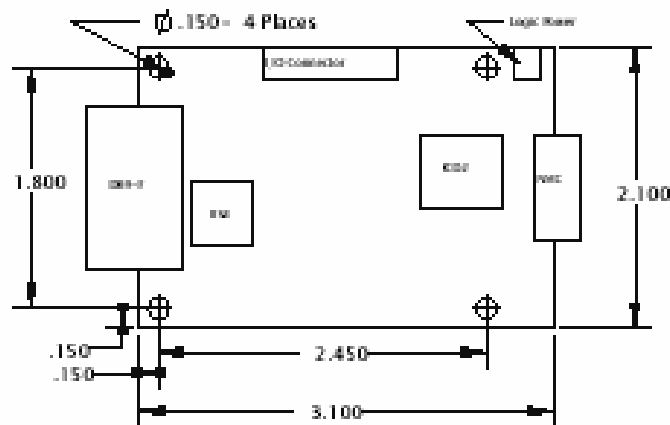


Figure 2 – SSA-485 board dimensions.

4.0 Contact Information

Additional information may be found from these sources:

JEFFREY KERR, LLC Web Site

www.jrkerr.com

Your source for data sheets, application notes and test code. Technical support is provided via e-mail.

Microchip

www.microchip.com

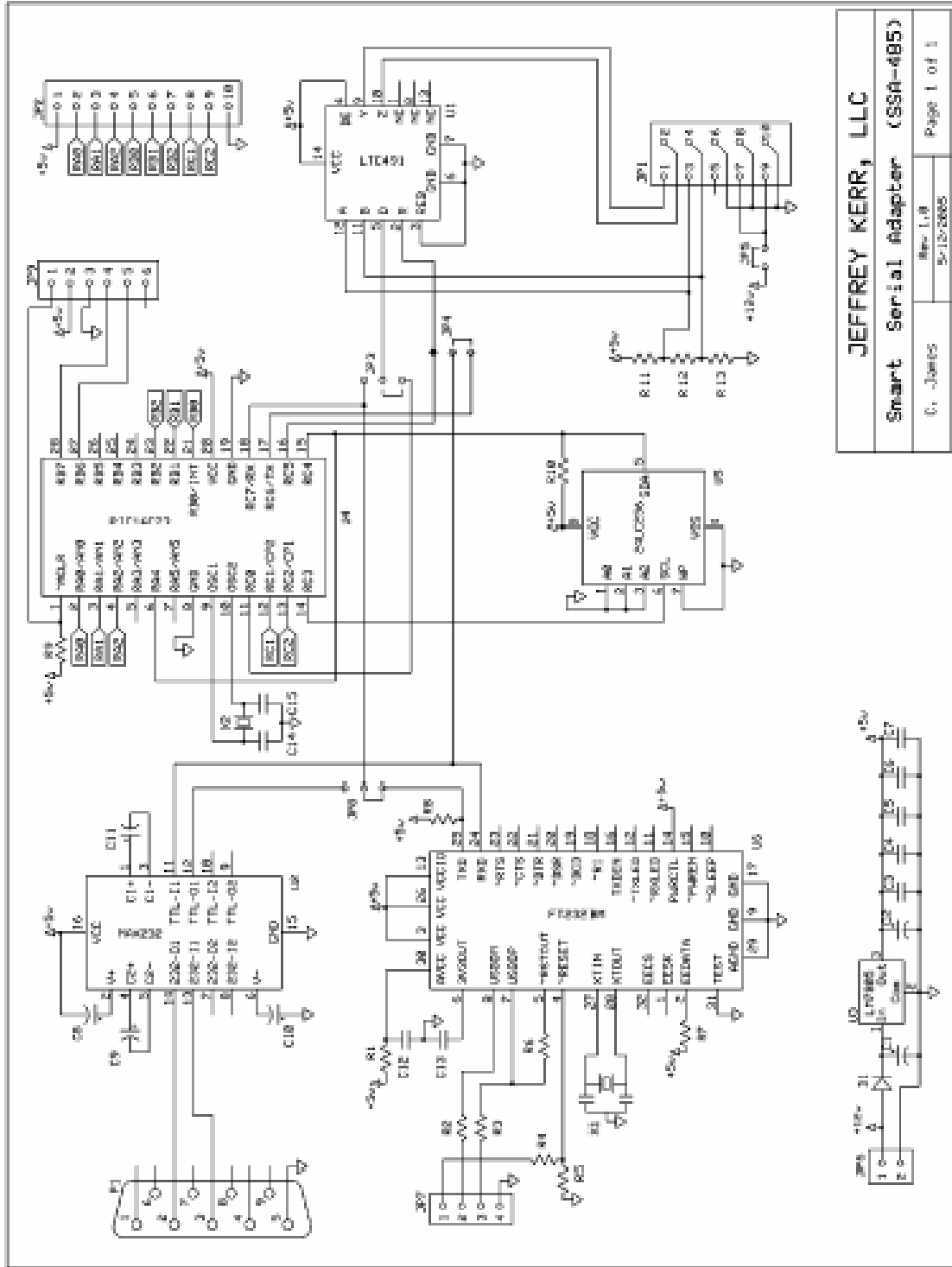
Visit Microchip's web site for information on their ICD2 development tools and their PIC18Fxxxx series microcontrollers.

FTDI Chip

www.ftdichip.com

Data sheet and drivers for the FTB232BM USB chip.

Appendix A. (Continued)



Appendix B. Matlab Programs and Functions

B.1. VRML File of the Virtual Reality Control Code

```
#VRML V2.0 utf8

#####
##### COPY RIGHTS RESERVED #####
#####

##### Developed By: Redwan M. Alqasemi #####
#####

##### April 2007 #####
#####

NavigationInfo { type "EXAMINE" speed 30 avatarSize [ 1, 0, 0 ] headlight TRUE }
DEF WMRAROBOT Group {
  children[
    Group {
      children [
        DEF EXT_SETTINGS Group {
          children [
            WorldInfo { title "Wheelchair Mounted Robotic Arm, By: Redwan Alqasemi, USF 2007"},
            NavigationInfo {
              type "EXAMINE"
              avatarSize 180
              visibilityLimit 200
              speed 1000
            },
          ],
        },

        Background {
          groundColor [ 0.8 0.7 0.1 , 0.8 0.7 0.1]
          groundAngle [1.57]
          skyColor [ 0 0 1 , 0 0.5 1 , 0 0.5 1 , 0.5 0.5 0.5 , 1 0.5 0]
          skyAngle [ 1 1.15 1.35 1.57]
          #topUrl "cloud.jpg"
        },

        DEF DynamicView Transform {
          rotation 0 1 0 0
          translation 0 0 0
          children [
            Viewpoint {
              description "a_start"
              position 2500 500 1800
              orientation 0 1 0 0.8
              jump FALSE
            },
            Viewpoint {
              description "a_far"
              position 900 6000 -200
              orientation -0.601 -0.547 -0.582 2.172
              jump FALSE
            },
            Viewpoint {
              description "a_bk-lt-up"
              position -1300 1600 -1600
              orientation -0.1 -1 -0.25 2.4
              jump FALSE
            },
            Viewpoint {
              description "a_bk-lt-dn"
              position -1400 400 -1800
              orientation 0.025 -1 0.037 2.4
              jump FALSE
            },
          ],
        },
      ],
    },
  ],
}
```

Appendix B. (Continued)

```
Viewpoint {
  description "a_ft-lt-up"
  position 1600 1800 -1400
  orientation -0.1 0.9 0.25 2.4
  jump FALSE
},
Viewpoint {
  description "a_ft-lt-dn"
  position 1700 400 -1600
  orientation 0.031 1 -0.052 2.4
  jump FALSE
},
Viewpoint {
  description "a_ft-rt-up"
  position 1600 1900 1500
  orientation -0.4 0.5 0.14 0.85
  jump FALSE
},
Viewpoint {
  description "a_ft-rt-dn"
  position 1700 300 1900
  orientation 0.191 1 -0.075 0.615
  jump FALSE
},
Viewpoint {
  description "a_bk-rt-up"
  position -1700 1700 1700
  orientation -0.25 -0.5 -0.12 1
  jump FALSE
},
Viewpoint {
  description "a_bk-rt-dn"
  position -1800 500 1900
  orientation 0.116 -1 0.021 0.818
  jump FALSE
},
Viewpoint {
  description "a_birdeye"
  position -1100 4900 -1900
  orientation -0.56 -0.72 -0.4 2.2
  jump FALSE
},
Viewpoint {
  description "a_top"
  position 200 3100 0
  orientation -0.577 -0.577 -0.577 2.1
  jump FALSE
},
}]
Viewpoint {
  description "top"
  position 200 3100 0
  orientation -0.577 -0.577 -0.577 2.1
  jump FALSE
},
Viewpoint {
  description "birdeye"
  position -1100 4900 -1900
  orientation -0.56 -0.72 -0.4 2.2
  jump FALSE
},
Viewpoint {
  description "bk-rt-dn"
  position -1800 500 1900
  orientation 0.116 -1 0.021 0.818
```

Appendix B. (Continued)

```
    jump FALSE
  },
  Viewpoint {
    description "bk-rt-up"
    position -1700 1700 1700
    orientation -0.25 -0.5 -0.12 1
    jump FALSE
  },
  Viewpoint {
    description "ft-rt-dn"
    position 1700 300 1900
    orientation 0.191 1 -0.075 0.615
    jump FALSE
  },
  Viewpoint {
    description "ft-rt-up"
    position 1600 1900 1500
    orientation -0.4 0.5 0.14 0.85
    jump FALSE
  },
  Viewpoint {
    description "ft-lt-dn"
    position 1700 400 -1600
    orientation 0.031 1 -0.052 2.4
    jump FALSE
  },
  Viewpoint {
    description "ft-lt-up"
    position 1600 1800 -1400
    orientation -0.1 0.9 0.25 2.4
    jump FALSE
  },
  Viewpoint {
    description "bk-lt-dn"
    position -1400 400 -1800
    orientation 0.025 -1 0.037 2.4
    jump FALSE
  },
  Viewpoint {
    description "bk-lt-up"
    position -1300 1600 -1600
    orientation -0.1 -1 -0.25 2.4
    jump FALSE
  },
  Viewpoint {
    description "far"
    position 900 6000 -200
    orientation -0.601 -0.547 -0.582 2.172
    jump FALSE
  },
  Viewpoint {
    description "start"
    position 2500 500 1800
    orientation 0 1 0 0.8
    jump FALSE
  },
},

DEF GROUND Transform {
  rotation 1 0 0 0
  translation 0 0 0
  children [
    Shape {
      geometry Box { size 5000 1 5000 }
      appearance Appearance {
        texture ImageTexture { url "9_Z_Ground.jpg" repeatS TRUE repeatT TRUE }
      }
    }
  ]
}
```

Appendix B. (Continued)

```
        textureTransform TextureTransform {
            rotation 0
            center 0 0
            translation 0 0
            scale 3 3
        }},
    }},
}
}

# Transforming the wheelchair world coordinate system to the VR's world coordinate system:
DEF World Transform {
rotation 1 0 0 -1.5707963
translation 0 0 0
children [

DEF Chair Transform {
rotation 0 0 1 0
translation -440 -230 168
children [
#   DEF WCR SphereSensor {}
#   DEF WCT PlaneSensor { minPosition -400 0 maxPosition 400 0 }
Group {
children [Inline { url "0_Chair.wrl" }

DEF LWheel Transform {
rotation 0 1 0 0
translation 0 0 0
children [
#   DEF LW CylinderSensor { diskAngle 0 minAngle 1.5707963 maxAngle 1.5707963 }
Group {
children [Inline { url "0_LWheel.wrl" }]}]}

DEF RWheel Transform {
rotation 0 1 0 0
translation 0 0 0
children [
#   DEF RW CylinderSensor { diskAngle 0 minAngle 1.5707963 maxAngle 1.5707963 }
Group {
children [Inline { url "0_RWheel.wrl" }]}]}

DEF ARM1 Transform {
rotation 1 0 0 1.5707963
translation 440 220 139
children [
#   DEF JOINT1 CylinderSensor { diskAngle 0 minAngle 1.5707963 maxAngle 1.5707963 }
Group {
children [Inline { url "1.wrl" }

DEF ARM2 Transform {
rotation 0 0 -1 1.5707963
translation 0 42.69 -75.1
children [
#   DEF JOINT2 CylinderSensor { diskAngle 0 minAngle -1.5708 maxAngle 1.5708 }
Group {
children [Inline { url "2.wrl" }

DEF ARM3 Transform {
rotation 0 1 0 1.5707963
translation -1.73 75.08 -42.7
children [
#   DEF JOINT3 CylinderSensor { diskAngle 0 minAngle -3.1416 maxAngle 3.1416 }
Group {
children [Inline { url "3.wrl" }
```

Appendix B. (Continued)

```
DEF ARM4 Transform {
rotation 0 0 -1 0
translation -2.92 42.64 -75.08
children [
#   DEF JOINT4 CylinderSensor { diskAngle 0 minAngle -3.1416 maxAngle 3.1416 }
    Group {
    children [Inline { url "4.wrl" }

DEF ARM5 Transform {
rotation 0 1 0 1.5707963
translation -11.45 74.85 -423.58
children [
#   DEF JOINT5 CylinderSensor { diskAngle 0 minAngle -3.1416 maxAngle 3.1416 }
    Group {
    children [Inline { url "5.wrl" }

DEF ARM6 Transform {
rotation 0 0 -1 1.5707963
translation -2.17 45.99 -75.1
children [
#   DEF JOINT6 CylinderSensor { diskAngle 0 minAngle -3.1416 maxAngle 3.1416 }
    Group {
    children [Inline { url "6.wrl" }

DEF ARM7 Transform {
rotation 0 1 0 1.5707963
translation -2.92 -61.52 -161.49
children [
#   DEF JOINT7 CylinderSensor { diskAngle 0 minAngle -1.5708 maxAngle 1.5708 }
    Group {
    children [Inline { url "7.wrl" }

DEF ARM8 Transform {
rotation 0 0 -1 0
translation -1.78 61.39 -192.29
children [
#   DEF JOINT8 CylinderSensor { diskAngle 0 minAngle -3.1416 maxAngle 3.1614 }
    Group {
    children [Inline { url "8.wrl" }

}}}}}}}}}}}}}}}}}}}}

# ROUTE WCT.translation_changed TO Chair.set_translation
# ROUTE WCR.rotation_changed TO Chair.set_rotation
# ROUTE LW.rotation_changed TO LWheel.set_rotation
# ROUTE RW.rotation_changed TO RWheel.set_rotation
# ROUTE JOINT1.rotation_changed TO ARM1.set_rotation
# ROUTE JOINT2.rotation_changed TO ARM2.set_rotation
# ROUTE JOINT3.rotation_changed TO ARM3.set_rotation
# ROUTE JOINT4.rotation_changed TO ARM4.set_rotation
# ROUTE JOINT5.rotation_changed TO ARM5.set_rotation
# ROUTE JOINT6.rotation_changed TO ARM6.set_rotation
# ROUTE JOINT7.rotation_changed TO ARM7.set_rotation
# ROUTE JOINT8.rotation_changed TO ARM8.set_rotation

}}
```

Appendix B. (Continued)

B.2. Matlab Functions Listed Alphabetically

```
% This "new USF WMRA" function SIMULATES the arm going from any position to the ready
position with ANIMATION. All angles are in Radians.
% the ready position is assumed to be qd=[pi/2;pi/2;0;pi/2;pi/2;pi/2;0]] (Radians).
% ini=1 --> initialize animation figures, ini=2 or any --> just update the figures, ini=3
--> close the figures.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Function Declaration:
function WMRA_any2ready(ini, vr, ml, arm, Tiwc, qi)

% Closing the Arm library and Matlab Graphics Animation and Virtual Reality Animation and
Plots windows:
if ini==3
    if arm==1
        try
            WMRA_ARM_Motion(ini, 0, 0, 0);
        end
    end
    if vr==1
        try
            WMRA_VR_Animation(ini, 0, 0);
        end
    end
    if ml==1
        try
            WMRA_ML_Animation(ini, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
        end
    end
    return;
end

% Defining the used conditions:
qd=[pi/2;pi/2;0;pi/2;pi/2;pi/2;0]; % Final joint angles (Ready Position).
ts=10; % (5 or 10 or 20) Simulation time to move the arm from any position to the
ready position.
n=100; % Number of time steps.
dt=ts/n; % The time step to move the arm from any position to the ready position.

% Initializing the physical Arm:
if arm==1
    WMRA_ARM_Motion(ini, 2, [qi;0], dt);
    ddt=0;
end

% Initializing Virtual Reality Animation:
if vr==1
    WMRA_VR_Animation(ini, Tiwc, qi);
end

% Initializing Robot Animation in Matlab Graphics:
if ml==1
    % Inputting the D-H Parameters in a Matrix form:
    DH=WMRA_DH(qi);

    % Calculating the transformation matrices of each link:
```

Appendix B. (Continued)

```
T01=WMRA_rotx(DH(1,1))*WMRA_transl(DH(1,2),0,0)*WMRA_rotz(DH(1,4))*WMRA_transl(0,0,DH(1,3));
T12=WMRA_rotx(DH(2,1))*WMRA_transl(DH(2,2),0,0)*WMRA_rotz(DH(2,4))*WMRA_transl(0,0,DH(2,3));
T23=WMRA_rotx(DH(3,1))*WMRA_transl(DH(3,2),0,0)*WMRA_rotz(DH(3,4))*WMRA_transl(0,0,DH(3,3));
T34=WMRA_rotx(DH(4,1))*WMRA_transl(DH(4,2),0,0)*WMRA_rotz(DH(4,4))*WMRA_transl(0,0,DH(4,3));
T45=WMRA_rotx(DH(5,1))*WMRA_transl(DH(5,2),0,0)*WMRA_rotz(DH(5,4))*WMRA_transl(0,0,DH(5,3));
T56=WMRA_rotx(DH(6,1))*WMRA_transl(DH(6,2),0,0)*WMRA_rotz(DH(6,4))*WMRA_transl(0,0,DH(6,3));
T67=WMRA_rotx(DH(7,1))*WMRA_transl(DH(7,2),0,0)*WMRA_rotz(DH(7,4))*WMRA_transl(0,0,DH(7,3));
    % Calculating the Transformation Matrix of the initial and desired arm positions:
    Ti=Tiwc*T01*T12*T23*T34*T45*T56*T67;
    Td=Tiwc*WMRA_q2T(qd);
    WMRA_ML_Animation(ini, Ti, Td, Tiwc, T01, T12, T23, T34, T45, T56, T67);
end

% Check for the shortest route:
diff=qd-qi(1:7);
for i=1:7
    if diff(i) > pi
        diff(i)=diff(i)-2*pi;
    elseif diff(i) < (-pi)
        diff(i)=diff(i)+2*pi;
    end
end

% Joint angle change at every time step.
dq=[diff/n;0;0];

% Initialization:
qo=qi;
tt=0;

while tt <= (ts-dt)
    % Starting a timer:
    tic;

    % Calculating the new Joint Angles:
    qn=qo+dq;

    % Updating the physical Arm:
    if arm==1
        ddt=ddt+dt;
        if ddt>=0.5 || tt>=(ts-dt)
            WMRA_ARM_Motion(2, 1, [qn;0], ddt);
            ddt=0;
        end
    end

    % Updating Virtual Reality Animation:
    if vr==1
        WMRA_VR_Animation(2, Tiwc, qn);
    end

    % Updating Matlab Animation:
    if ml==1
```


Appendix B. (Continued)

```
    % Calculating the new Transformation Matrix:

T1a=WMRA_rotx(DH(1,1))*WMRA_transl(DH(1,2),0,0)*WMRA_rotz(qn(1))*WMRA_transl(0,0,DH(1,3))
;

T2a=WMRA_rotx(DH(2,1))*WMRA_transl(DH(2,2),0,0)*WMRA_rotz(qn(2))*WMRA_transl(0,0,DH(2,3))
;

T3a=WMRA_rotx(DH(3,1))*WMRA_transl(DH(3,2),0,0)*WMRA_rotz(qn(3))*WMRA_transl(0,0,DH(3,3))
;

T4a=WMRA_rotx(DH(4,1))*WMRA_transl(DH(4,2),0,0)*WMRA_rotz(qn(4))*WMRA_transl(0,0,DH(4,3))
;

T5a=WMRA_rotx(DH(5,1))*WMRA_transl(DH(5,2),0,0)*WMRA_rotz(qn(5))*WMRA_transl(0,0,DH(5,3))
;

T6a=WMRA_rotx(DH(6,1))*WMRA_transl(DH(6,2),0,0)*WMRA_rotz(qn(6))*WMRA_transl(0,0,DH(6,3))
;

T7a=WMRA_rotx(DH(7,1))*WMRA_transl(DH(7,2),0,0)*WMRA_rotz(qn(7))*WMRA_transl(0,0,DH(7,3))
;

    WMRA_ML_Animation(2, Ti, Td, Tiwc, T1a, T2a, T3a, T4a, T5a, T6a, T7a);
end

    % Updating the old values with the new values for the next iteration:
    qo=qn;
    tt=tt+dt;

    % Pausing for the speed sync:
    pause(dt-toc);

end
```

```
% This function communicates with the physical USF WMRA system with 9 DOF to get the
encoder readings and send the commands to be executed.
% The (.H) file and the (.DLL) file that contains the used functions should be in the
directory containing this program.
% config=0: Set the current encoder readings to zeros, config=1: Read the encoder
readings from the configuration txt file.
% config=2: Change the configuration file to the initial values provided by (qo), then
read the encoder readings from the configuration txt file.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Function Declaration:
```

```
function [qn] = WMRA_ARM_Motion(ind, config, qo, dt)
```

```
% Declaring the global variables:
```

```
global L ptr e2r1 e2r2 e2r3 e2r4 e2r5 e2r6 e2r7 e2r8 e2r9 e2d
```

```
% The initialization of the Arm library:
```

```
if ind==1
```

```
    % Reading the Wheelchair's constant dimentions, all dimentions are converted in
millimeters:
```

```
    L=WMRA_WCD;
```

```
    % Serial Communication properties:
```

```
    com=4;
```

```
    baud=19200;
```

Appendix B. (Continued)

```
% PID controller gains:
Kp=100;
Kd=1000;
Ki=0;

% Conversion of encoder readings to radians: Note that the encoder readings are
negative of the kinematic arrangements in the control code.
e2r1=-pi/900000;
e2r2=-pi/900000;
e2r3=-pi/950000;
e2r4=-pi/710000;
e2r5=-pi/580000;
%e2r6p=-pi/420000;
%e2r6n=-pi/500000;
e2r6=-pi/440000;
e2r7=-pi/630000;
e2r8=1; % Redwan: change this to forward motion when wheelchair controllers are
installed (Only when reading the encoders).
e2r9=1; % Redwan: change this to rotation motion when wheelchair controllers are
installed (Only when reading the encoders).
e2d =-1/100000;

% The case when changing the configuration file to qo is required:
if config==2
    % Converting the commanded angles to encoder readings:
    %qo=[qo(1)/e2r1; qo(2)/e2r2; qo(3)/e2r3; qo(4)/e2r4; qo(5)/e2r5; qo(6)/e2r6;
qo(7)/e2r7; qo(8)/e2r8; qo(9)/e2r9; qo(10)/e2d];
    qo=[qo(1)/e2r1; qo(2)/e2r2; qo(3)/e2r3; qo(4)/e2r4; qo(5)/e2r5; qo(6)/e2r6;
qo(7)/e2r7; qo(10)/e2d]; % Redwan: Replace this with the one above when wheelchair
controllers are installed.
    % Changing the configuration file to qo:
    fid = fopen('configuration.txt','w');
    fprintf(fid, ' %10.0f ',qo);
    fclose(fid);
    config=1;
end

try
    % Closing the library in case it was open:
    calllib ('controlMotor', 'close');
catch
end
try
    % Loading the DLL library of functions:
    loadlibrary('controlMotor.dll', 'controlMotor.h');
catch
end

% Establishing the connections, and setting the encoders to the current
configuration:
check=calllib('controlMotor', 'init', com, baud, config);
if check == 0
    fprintf('\nWMRA initialization has failed, Please check your communications.\n');
end

% Setting the PID controller gains (All motors the same gains in this case. Use
'setParamsPID' command to set each individual motor to different PID gains:
calllib ('controlMotor', 'setParamsPIDAll', Kp, Kd, Ki);

% Creating a pointer of for the 10 joints to be used to read or set the encoders:
dim = 1:8; % Redwan: Change 8 to 10 when wheelchair controllers are installed.
ptr = libpointer ('int32Ptr', dim);

% Reading the current positions and converting them to radians:
calllib ('controlMotor', 'getPosAll', ptr);
```

Appendix B. (Continued)

```
qc=double(ptr.Value);
qc=[qc(1:7), 0, 0, qc(8)]; % Redwan: Remove when wheelchair controllers are
installed.
qn=[qc(1)*e2r1; qc(2)*e2r2; qc(3)*e2r3; qc(4)*e2r4; qc(5)*e2r5; qc(6)*e2r6;
qc(7)*e2r7; qc(8)*e2r8; qc(9)*e2r9; qc(10)*e2d;];

% Closing the Arm library:
elseif ind==3
    % Reading the current positions to be saved in the configuration file:
    calllib ('controlMotor', 'getPosAll', ptr);
    % Closing the library and unloading:
    calllib ('controlMotor', 'close');
    unloadlibrary('controlMotor');
    % Reporting the function output to be zero (This value will not be used):
    qn=0;

% Updating the Arm:
else
    % Reading the current positions:
    calllib ('controlMotor', 'getPosAll', ptr);
    qc=double(ptr.Value)';

    % Converting the commanded angles to encoder readings:
    %qo=[qo(1)/e2r1; qo(2)/e2r2; qo(3)/e2r3; qo(4)/e2r4; qo(5)/e2r5; qo(6)/e2r6;
qo(7)/e2r7; qo(8)/e2r8; qo(9)/e2r9; qo(10)/e2d];
    qo=[qo(1)/e2r1; qo(2)/e2r2; qo(3)/e2r3; qo(4)/e2r4; qo(5)/e2r5; qo(6)/e2r6;
qo(7)/e2r7; qo(10)/e2d]; % Redwan: Replace this with the one above when wheelchair
controllers are installed.

    % finding the needed velocities for the arm, note that a factor of 33.8 is needed for
encoder velocities and position conversion:
    qdo(1:7)=33.8*abs(qo(1:7)-qc(1:7))/dt;
    qddo=500*[1; 1; 1; 1; 1; 1; 1; 10];

    % Calculating the gripper's commanded position and velocity:
    qo(8)=qo(8)+qc(8); % Redwan: Change 8 to 10 when wheelchair controllers are
installed.
    qdo(8)=33.8*abs(qo(8)-qc(8)); % Redwan: Change 8 to 10 when wheelchair controllers
are installed.

    % Splitting the negative sign to be used in the DLL functions:
    dir=[0;0;0;0;0;0;0;0]; % Redwan: Add two more zeros when wheelchair controllers are
installed.
    for i=1:8 % Redwan: Change 8 to 10 when wheelchair controllers are installed.
        if sign(qo(i)) == -1
            qo(i) = -qo(i);
            dir(i) = 1;
        end
    end

    % Sending the commanded angles to the controller boards:
    calllib ('controlMotor', 'posSelect', [1, 1, 1, 1, 1, 1, 1, 1, -1], qo', qdo', qddo',
dir);

    % Reading the current positions and converting them to radians:
    calllib ('controlMotor', 'getPosAll', ptr);
    qc=double(ptr.Value);
    qc=[qc(1:7), 0, 0, qc(8)]; % Redwan: Remove when wheelchair controllers are
installed.
    qn=[qc(1)*e2r1; qc(2)*e2r2; qc(3)*e2r3; qc(4)*e2r4; qc(5)*e2r5; qc(6)*e2r6;
qc(7)*e2r7; qc(8)*e2r8; qc(9)*e2r9; qc(10)*e2d;];
end
```

Appendix B. (Continued)

```

% This function uses a 3rd order Polynomial with a Blending factor to find a smooth
trajectory points of a variable "q" along a streight line, given the initial and final
variable values and the number of trajectory points.
% The output is the variable position.
% See Eq. 7.18 page 210 of Craig Book

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Function Declaration:
function [qt] = WMRA_BPolynomial(qi, qf, n)

% Blending Factor:
b=5;

% Initializing the time:
tt=0;
tf=abs((qf-qi));
dt=tf/(n-1);

if tf > 0.001
    % Blending procedure:
    % Time, position, velocity, and acceleration of the variable at the first blending
point:
    qddb=b*4*(qf-qi)/tf^2;
    tb=tf/2-sqrt(qddb^2*tf^2-4*qddb*(qf-qi))/abs(2*qddb);
    qdb=qddb*tb;
    qb=qi+qddb*tb^2/2;
    % Calculating the polynomial factors at the first blending point: From Eq.7.18 page
210 of Craig Book
    a01=qi;
    a11=0;
    a21=0.5*qddb;
    a31=(20*(qb-qi)-8*qdb*tb-2*qddb*tb^2)/(2*tb^3);
    %a41=(30*(qi-qb)+14*qdb*tb+qddb*tb^2)/(2*tb^4); % Uncomment for 5th order polynomial.
    %a51=(12*(qb-qi)-6*qdb*tb)/(2*tb^5); % Uncomment for 5th order polynomial.
    % Calculating the polynomial factors at the second blending point: From Eq.7.18 page
210 of Craig Book
    a02=qb+qdb*(tf-2*tb);
    a12=qdb;
    a22=-0.5*qddb;
    a32=(20*(qf-a02)-12*a12*tb+2*qddb*tb^2)/(2*tb^3);
    %a42=(30*(a02-qf)+16*a12*tb-qddb*tb^2)/(2*tb^4); % Uncomment for 5th order
polynomial.
    %a52=(12*(qf-a02)-6*a12*tb)/(2*tb^5); % Uncomment for 5th order polynomial.
end

% Calculating the intermediate joint angles along the trajectory from the initial to the
final position:
for i=1:n
    if tf <= 0.001
        qt(i)=qi;
    elseif tt<=tb
        qt(i)=a01+a11*tt+a21*tt^2+a31*tt^3; %+a41*tt^4+a51*tt^5; % Uncomment before
"+a41" for 5th order polynomial.
    elseif tt>=(tf-tb)
        qt(i)=a02+a12*(tt+tb-tf)+a22*(tt+tb-tf)^2+a32*(tt+tb-tf)^3; %+a42*(tt+tb-
tf)^4+a52*(tt+tb-tf)^5; % Uncomment before "+a42" for 5th order polynomial.
    else
        qt(i)=qb-qdb*(tb-tt);
    end
end

```

Appendix B. (Continued)

```
tt=tt+dt;
end
```

% This function is to stop the arm if it is moving towards a collision with itself, the wheelchair, or the human user.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

% Function Declaration:

```
function [dq]=WMRA_collide(dqi, T01, T12, T23, T34, T45, T56, T67)
```

% Reading the Wheelchair's constant dimentions, all dimentions are converted in millimeters:

```
L=WMRA_WCD;
```

% Collision Conditions:

```
gr=100-L(4)-L(5); % The ground buffer surface.
dq=dqi;
```

% 1- Collision of frame 3 using T03:

```
T03=T01*T12*T23;
```

% Collision with the ground:

```
if T03(3,4) <= gr
    dq=-0.01*dqi;
end
```

% Collision with the wheelchair's front left side:

```
if T03(1,4) >= 450 && T03(2,4) <= -150
    dq=-0.01*dqi;
end
```

% Collision with the wheelchair's rear left side:

```
if T03(1,4) <= 450 && T03(2,4) <= 100
    dq=-0.01*dqi;
end
```

% Collision with the wheelchair's rear left wheel:

```
if T03(1,4) <= 0 && T03(2,4) <= 100 && T03(3,4) <= 120
    dq=-0.01*dqi;
end
```

% 2- Collision of frame 4 using T04:

```
T04=T03*T34;
```

% Collision with the ground:

```
if T04(3,4) <= gr
    dq=-0.01*dqi;
end
```

% Collision with the wheelchair's front left side:

```
if T04(1,4) <= 450 && T04(1,4) >= -100 && T04(2,4) <= 0
    dq=-0.01*dqi;
end
```

% Collision with the wheelchair's rear left side:

```
if T04(1,4) <= -100 && T04(2,4) <= 100
    dq=-0.01*dqi;
end
```

% Collision with the wheelchair's rear left wheel:

```
if T04(1,4) <= -100 && T04(2,4) <= 100 && T04(3,4) <= 120
    dq=-0.01*dqi;
end
```

% 3- Collision of frame 5 using T05:

```
T05=T04*T45;
```

Appendix B. (Continued)

```
% Collision with the ground:
if T05(3,4) <= gr
    dq=-0.01*dqi;
end
% Collision with the wheelchair driver's left shoulder:
if T05(1,4) <= -100 && T05(1,4) >= -550 && T05(2,4) <= 150
    dq=-0.01*dqi;
end
% Collision with the wheelchair driver's lap:
if T05(1,4) <= 400 && T05(1,4) >= -100 && T05(2,4) <= 0 && T05(3,4) <= 470
    dq=-0.01*dqi;
end
% Collision with the wheelchair's battery pack:
if T05(1,4) <= -430 && T05(1,4) >= -630 && T05(2,4) <= 100 && T05(3,4) <= 50
    dq=-0.01*dqi;
end

% 4- Collision of frame 7 using T07:
T07=T05*T56*T67;
% Collision with the ground:
if T07(3,4) <= gr
    dq=-0.01*dqi;
end
% Collision with the wheelchair driver's left shoulder:
if T07(1,4) <= -50 && T07(1,4) >= -600 && T07(2,4) <= 200
    dq=-0.01*dqi;
end
% Collision with the wheelchair driver's lap:
if T07(1,4) <= 450 && T07(1,4) >= -50 && T07(2,4) <= 50 && T07(3,4) <= 520
    dq=-0.01*dqi;
end
% Collision with the wheelchair's battery pack:
if T07(1,4) <= -480 && T07(1,4) >= -680 && T07(2,4) <= 50 && T07(3,4) <= 100
    dq=-0.01*dqi;
end

% 5- Collision of the arm and itself using T37:
T37=T34*T45*T56*T67;
% Collision between the forearm and the upper arm:
if T37(1,4) <= 170 && T37(1,4) >= -170 && T37(2,4) >= -100 && T37(3,4) <= 0
    dq=-0.01*dqi;
end

% This function gives the WMRA's errors from the current position to the required
trajectory position.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Function Declaration:
function [delta]=WMRA_delta(Ti, Td)

ep=Td(1:3,4)-Ti(1:3,4);
eo=0.5*( cross(Ti(1:3,1),Td(1:3,1)) + cross(Ti(1:3,2),Td(1:3,2)) +
cross(Ti(1:3,3),Td(1:3,3)) ); % From equation 17 on page 189 of (Robot Motion Planning
and Control) Book by Micheal Brady et al. Taken from the paper (Resolved-Acceleration
Control of Mechanical Manipulators) By John Y. S. Luh et al.
delta=[ep; eo];
```

Appendix B. (Continued)

```
% This function gives the DH-Parameters matrix to be used in the program.
% Modifying the parameters on this file is sufficient to change these dimension in all
related programs.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Function Declaration:
function [DH]=WMRA_DH(q)

% Inputting the D-H Parameters in a Matrix form, dimensions are in millimeters and
radians:

% Dimensions based on the actual physical arm:
DH=[-pi/2 0 110 q(1) ; pi/2 0 146 q(2) ; -pi/2 0 549 q(3) ; pi/2 0 130 q(4) ;
    -pi/2 0 241 q(5) ; pi/2 0 0 q(6) ; -pi/2 0 179+131 q(7)];

% Dimensions based on the Virtual Reality arm model:
% DH=[-pi/2 0 109.72 q(1) ; pi/2 0 118.66 q(2) ; -pi/2 0 499.67 q(3) ; pi/2 0 121.78 q(4)
;
%     -pi/2 0 235.67 q(5) ; pi/2 0 0 q(6) ; -pi/2 0 276.68 q(7)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Thanks to Mayur Palankar %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = WMRA_error_gui(varargin)
% WMRA_ERROR_GUI M-file for WMRA_error_gui.fig
% WMRA_ERROR_GUI, by itself, creates a new WMRA_ERROR_GUI or raises the existing
% singleton*.
%
% H = WMRA_ERROR_GUI returns the handle to a new WMRA_ERROR_GUI or the handle to
% the existing singleton*.
%
% WMRA_ERROR_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
% function named CALLBACK in WMRA_ERROR_GUI.M with the given input arguments.
%
% WMRA_ERROR_GUI('Property','Value',...) creates a new WMRA_ERROR_GUI or raises the
% existing singleton*. Starting from the left, property value pairs are
% applied to the GUI before WMRA_error_gui_OpeningFunction gets called. An
% unrecognized property name or invalid value makes property application
% stop. All inputs are passed to WMRA_error_gui_OpeningFcn via varargin.
%
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
% instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help WMRA_error_gui

% Last Modified by GUIDE v2.5 03-Feb-2007 15:47:37

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',      mfilename, ...
```

Appendix B. (Continued)

```
        'gui_Singleton', gui_Singleton, ...
        'gui_OpeningFcn', @WMRA_error_gui_OpeningFcn, ...
        'gui_OutputFcn', @WMRA_error_gui_OutputFcn, ...
        'gui_LayoutFcn', [] , ...
        'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before WMRA_error_gui is made visible.
function WMRA_error_gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to WMRA_error_gui (see VARARGIN)

set(handles.edit1, 'String', varargin{1});

% Choose default command line output for WMRA_error_gui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes WMRA_error_gui wait for user response (see UIRESUME)
uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function WMRA_error_gui_OutputFcn(hObject, eventdata, handles)
% varargout  Cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
%varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close;

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```


Appendix B. (Continued)

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = WMRA_exit(varargin)
% WMRA_EXIT M-file for WMRA_exit.fig
%   WMRA_EXIT, by itself, creates a new WMRA_EXIT or raises the existing
%   singleton*.
%
%   H = WMRA_EXIT returns the handle to a new WMRA_EXIT or the handle to
%   the existing singleton*.
%
%   WMRA_EXIT('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in WMRA_EXIT.M with the given input arguments.
%
%   WMRA_EXIT('Property','Value',...) creates a new WMRA_EXIT or raises the
%   existing singleton*. Starting from the left, property value pairs are
%   applied to the GUI before WMRA_exit_OpeningFunction gets called. An
%   unrecognized property name or invalid value makes property application
%   stop. All inputs are passed to WMRA_exit_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help WMRA_exit

% Last Modified by GUIDE v2.5 14-Mar-2007 23:20:09

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @WMRA_exit_OpeningFcn, ...
                  'gui_OutputFcn', @WMRA_exit_OutputFcn, ...
                  'gui_LayoutFcn', [] , ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before WMRA_exit is made visible.
function WMRA_exit_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
```

Appendix B. (Continued)

```
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% varargin command line arguments to WMRA_exit (see VARARGIN)

% Choose default command line output for WMRA_exit
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

global VAR_SCREENOPN
VAR_SCREENOPN = 1;

% UIWAIT makes WMRA_exit wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = WMRA_exit_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global VAR_SCREENOPN
VAR_SCREENOPN = 0;
close;

% This function gives the Jacobian Matrix and its determinant based on frame 0 of the new
USF WMRA, given the Transformation Matrices of each link.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Function Declaration:
function [J0,detJ0] = WMRA_J07(T1, T2, T3, T4, T5, T6, T7)

T=eye(4);

J0(1,7)=-T(1,1)*T(2,4)+T(2,1)*T(1,4);
J0(2,7)=-T(1,2)*T(2,4)+T(2,2)*T(1,4);
J0(3,7)=-T(1,3)*T(2,4)+T(2,3)*T(1,4);
J0(4,7)=T(3,1);
J0(5,7)=T(3,2);
J0(6,7)=T(3,3);

T=T7*T;

J0(1,6)=-T(1,1)*T(2,4)+T(2,1)*T(1,4);
J0(2,6)=-T(1,2)*T(2,4)+T(2,2)*T(1,4);
J0(3,6)=-T(1,3)*T(2,4)+T(2,3)*T(1,4);
```

Appendix B. (Continued)

```
J0(4,6)=T(3,1);
J0(5,6)=T(3,2);
J0(6,6)=T(3,3);
```

```
T=T6*T;
```

```
J0(1,5)=-T(1,1)*T(2,4)+T(2,1)*T(1,4);
J0(2,5)=-T(1,2)*T(2,4)+T(2,2)*T(1,4);
J0(3,5)=-T(1,3)*T(2,4)+T(2,3)*T(1,4);
J0(4,5)=T(3,1);
J0(5,5)=T(3,2);
J0(6,5)=T(3,3);
```

```
T=T5*T;
```

```
J0(1,4)=-T(1,1)*T(2,4)+T(2,1)*T(1,4);
J0(2,4)=-T(1,2)*T(2,4)+T(2,2)*T(1,4);
J0(3,4)=-T(1,3)*T(2,4)+T(2,3)*T(1,4);
J0(4,4)=T(3,1);
J0(5,4)=T(3,2);
J0(6,4)=T(3,3);
```

```
T=T4*T;
```

```
J0(1,3)=-T(1,1)*T(2,4)+T(2,1)*T(1,4);
J0(2,3)=-T(1,2)*T(2,4)+T(2,2)*T(1,4);
J0(3,3)=-T(1,3)*T(2,4)+T(2,3)*T(1,4);
J0(4,3)=T(3,1);
J0(5,3)=T(3,2);
J0(6,3)=T(3,3);
```

```
T=T3*T;
```

```
J0(1,2)=-T(1,1)*T(2,4)+T(2,1)*T(1,4);
J0(2,2)=-T(1,2)*T(2,4)+T(2,2)*T(1,4);
J0(3,2)=-T(1,3)*T(2,4)+T(2,3)*T(1,4);
J0(4,2)=T(3,1);
J0(5,2)=T(3,2);
J0(6,2)=T(3,3);
```

```
T=T2*T;
```

```
J0(1,1)=-T(1,1)*T(2,4)+T(2,1)*T(1,4);
J0(2,1)=-T(1,2)*T(2,4)+T(2,2)*T(1,4);
J0(3,1)=-T(1,3)*T(2,4)+T(2,3)*T(1,4);
J0(4,1)=T(3,1);
J0(5,1)=T(3,2);
J0(6,1)=T(3,3);
```

```
T=T1*T;
```

```
J0=[T(1:3,1:3),zeros(3,3);zeros(3,3),T(1:3,1:3)]*J0;
```

```
detJ0=sqrt(det(J0*J0'));
```

```
% This function gives the WMRA's base Jacobian Matrix based on the ground frame, given
the Wheelchair orientation angle about the z axis.
% Dimentions are as supplies, angles are in radians.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Appendix B. (Continued)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function Declaration:
function [J]=WMRA_Jga(ind, p, XY)

% Reading the Wheelchair's constant dimentions, all dimentions are converted in
millimeters:
L=WMRA_WCD;

% Deciding if the motion is in reference to the arm base (1) or the wheel axle center
(0):
if ind == 0, L(2:4)=[0;0;0]; end

% Calculating the Jacobian:
J = [eye(2) , zeros(2,3) , [-(XY(1)*sin(p)+XY(2)*cos(p)) ; XY(1)*cos(p)-XY(2)*sin(p)] ;
zeros(4,2) , eye(4)] * (L(5)/2)*[cos(p)+2*(L(2)*sin(p)+L(3)*cos(p))/L(1) , cos(p)-
2*(L(2)*sin(p)+L(3)*cos(p))/L(1) ; sin(p)-2*(L(2)*cos(p)-L(3)*sin(p))/L(1) ,
sin(p)+2*(L(2)*cos(p)-L(3)*sin(p))/L(1) ; 0 0;0 0;0 0; -2/L(1) , 2/L(1)] * [1 , -
L(1)/(2*L(5)) ; 1 , L(1)/(2*L(5))];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function gives the joint limit vector to be used in the program.
% Modifying the parameters on this file is sufficient to change these limits in all
related programs.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Function Declaration:
function [qmin,qmax]=WMRA_Jlimit()

% Inputting the joint limits in a vector form, dimensions are in radians:
% Dimentions based on the actual physical arm:
qmin=-[170;170;170;170;170;170;100;200]*pi/180;
qmax= [170;170;170;170;170;170;100;200]*pi/180;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This "new USF WMRA" script SIMULATES the Joint control of the WMRA system with
ANIMATION and plots for 9 DOF. All angles are in Radians.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Defining used parameters:
d2r=pi/180; % Conversions from Degrees to Radians.
r2d=180/pi; % Conversions from Radians to Degrees.

% Reading the Wheelchair's constant dimentions, all dimentions are converted in
millimeters:
L=WMRA_WCD;

% User input prompts:
choicel = input('\n Choose animation type or no animation: \n For Virtual Reality
Animation, press "1", \n For Matlab Graphics Animation, press "2", \n For BOTH
Animations, press "3", \n For NO Animation, press "4". \n','s');
if choicel=='2'
```

Appendix B. (Continued)

```
vr = 0; ml = 1;
elseif choice1=='3'
vr = 1; ml = 1;
elseif choice1=='4'
vr = 0; ml = 0;
else
vr = 1; ml = 0;
end
choice2 = input('\n Would you like to run the actual arm? \n For no, press "0", \n For
yes, press "1". \n', 's');
if choice2=='1'
arm=1;
else
arm=0;
end
choice3 = input('\n Press "1" if you want to start at the "ready" position, \n or press
"2" if you want to enter the initial joint angles. \n', 's');
if choice3=='2'
qi = input('\n Please enter the initial angles vector of the arm in radians (e.g.
[pi/2;pi/2;0;pi/2;pi/2;0]) \n');
Wci = input('\n Please enter the initial x,y position and z orientation of the WMRA
base in millimeters and radians (e.g. [200;500;0.3]) \n');
ini=0;
else
qi=[90;90;0;90;90;90;0]*d2r;
Wci=[0;0;0];
ini=0;
if vr==1 || ml==1 || arm==1
choice4 = input('\n Press "1" if you want to include "park" to "ready" motion, \n
or press "2" if not. \n', 's');
if choice4=='2'
ini=0;
else
ini=1;
end
end
end

% Calculating the Transformation Matrix of the initial position of the WMRA's base:
Tiwc=WMRA_p2T(Wci(1),Wci(2),Wci(3));

% Calculating the initial Wheelchair Variables:
qiwc=[sqrt(Wci(1)^2+Wci(2)^2);Wci(3)];

% Calculating the initial and desired joint positions:
qi=[qi;qiwc];
qd = input('\n Please enter the desired angles and distance vector in radians and mm
(e.g. [pi/3;-pi/3;pi/3;-pi/3;pi/3;-pi/3;500;pi/3]) \n');
ts = input('\n Please enter the desired execution time in seconds (e.g. 2) \n');

% Calculating the initial and final transformation matrices:
[Ti, Tia, Tiwc, T01, T12, T23, T34, T45, T56, T67]=WMRA_Tall(1, qi(1:7), qi(8:9), Tiwc);
[Td, Tda, Tdwc, T01d, T12d, T23d, T34d, T45d, T56d, T67d]=WMRA_Tall(2, qd(1:7), qd(8:9),
Tiwc);
% Calculating the number of iteration and the time increment (delta t):
dt=0.05; % Time increment in seconds.
total_time=ts; % Total time of animation.
n=round(total_time/dt); % Number of iterations rounded up.
dt=total_time/n; % Adjusted time increment in seconds.
dq=(qd-qi)/n;

% Initializing the joint angles, the Transformation Matrix, and time:
qo=qi;
To=Ti;
Toa=Tia;
Towc=Tiwc;
```

Appendix B. (Continued)

```
tt=0;
i=1;

% Initializing the WMRA:
if ini==0 % When no "park" to "ready" motion required.
    % Initializing Virtual Reality Animation:
    if vr==1
        WMRA_VR_Animation(1, Towc, qo);
    end
    % Initializing Robot Animation in Matlab Graphics:
    if ml==1
        WMRA_ML_Animation(1, To, Td, Towc, T01, T12, T23, T34, T45, T56, T67);
    end
    % Initializing the Physical Arm:
    if arm==1
        WMRA_ARM_Motion(1, 2, [qo;0], 0);
        ddt=0;
    end
elseif ini==1 && (vr==1 || ml==1 || arm==1) % When "park" to "ready" motion is required.
    WMRA_park2ready(1, vr, ml, arm, Towc, qo(8:9));
    if arm==1
        ddt=0;
    end
end

% Re-Drawing the Animation:
if vr==1 || ml==1
    drawnow;
end

% Starting a timer:
tic

% Starting the Iteration Loop:
while i<=n

    % Calculating the new Joint Angles:
    qn=qo+dq;

    % Calculating the new Transformation Matrices:
    [Tn, Tna, Tnwc, T01, T12, T23, T34, T45, T56, T67]=WMRA_Tall(2, qn, dq(8:9),
Towc);

    % Updating Physical Arm:
    if arm==1
        ddt=ddt+dt;
        if ddt>=0.5 || i>=(n+1)
            WMRA_ARM_Motion(2, 1, [qn;0], ddt);
            ddt=0;
        end
    end

    % Updating Virtual Reality Animation:
    if vr==1
        WMRA_VR_Animation(2, Tnwc, qn);
    end

    % Updating Matlab Graphics Animation:
    if ml==1
        WMRA_ML_Animation(2, Ti, Td, Tnwc, T01, T12, T23, T34, T45, T56, T67);
    end

    % Re-Drawing the Animation:
    if vr==1 || ml==1
        drawnow;
    end
end
```

Appendix B. (Continued)

```
    % Updating the old values with the new values for the next iteration:
    qo=qn;
    To=Tn;
    Toa=Tna;
    Towc=Tnwc;
    tt=tt+dt;
    i=i+1;

    % Delay to comply with the required speed:
    if toc < tt
        pause(tt-toc);
    end

end

% Reading the elapsed time and printing it with the simulation time:
toc

if vr==1 || ml==1 || arm==1

    % Going back to the ready position:
    choice6 = input('\n Do you want to go back to the "ready" position? \n Press "1" for
Yes, or press "2" for No. \n','s');
    if choice6=='1'
        WMRA_any2ready(2, vr, ml, arm, Tnwc, qn);
        % Going back to the parking position:
        choice7 = input('\n Do you want to go back to the "parking" position? \n Press
"1" for Yes, or press "2" for No. \n','s');
        if choice7=='1'
            WMRA_ready2park(2, vr, ml, arm, Tnwc, qn(8:9));
        end
    end

    % Closing the Arm library and Matlab Graphics Animation and Virtual Reality Animation
and Plots windows:
    choice8 = input('\n Do you want to close all simulation windows and arm controls? \n
Press "1" for Yes, or press "2" for No. \n','s');
    if choice8=='1'
        if arm==1
            WMRA_ARM_Motion(3, 0, 0, 0);
        end
        if vr==1
            WMRA_VR_Animation(3, 0, 0);
        end
        if ml==1
            WMRA_ML_Animation(3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
        end
    end

end

end

% This function uses a Linear function to find an equally-spaced trajectory points of a
variable "q" along a streight line, given the initial and final variable values and the
number of trajectory points.
% The output is the variable position.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Algasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function Declaration:
function [qt] = WMRA_Linear(qi, qf, n)
```

Appendix B. (Continued)

```
dq=(qf-qi)/(n-1);
```

```
for i=1:n
    qt(i)=qi+dq*(i-1);
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Thanks to Mayur Palankar %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function varargout = WMRA_matrix_entry(varargin)
% WMRA_MATRIX_ENTRY M-file for WMRA_matrix_entry.fig
% WMRA_MATRIX_ENTRY, by itself, creates a new WMRA_MATRIX_ENTRY or raises the
existing
% singleton*.
%
% H = WMRA_MATRIX_ENTRY returns the handle to a new WMRA_MATRIX_ENTRY or the handle
to
% the existing singleton*.
%
% WMRA_MATRIX_ENTRY('CALLBACK',hObject,eventData,handles,...) calls the local
% function named CALLBACK in WMRA_MATRIX_ENTRY.M with the given input arguments.
%
% WMRA_MATRIX_ENTRY('Property','Value',...) creates a new WMRA_MATRIX_ENTRY or
raises the
% existing singleton*. Starting from the left, property value pairs are
% applied to the GUI before WMRA_matrix_entry_OpeningFunction gets called. An
% unrecognized property name or invalid value makes property application
% stop. All inputs are passed to WMRA_matrix_entry_OpeningFcn via varargin.
%
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
% instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help WMRA_matrix_entry
% Last Modified by GUIDE v2.5 21-Feb-2007 13:19:38
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @WMRA_matrix_entry_OpeningFcn, ...
    'gui_OutputFcn', @WMRA_matrix_entry_OutputFcn, ...
    'gui_LayoutFcn', [], ...
    'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{2:nargout});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before WMRA_matrix_entry is made visible.
function WMRA_matrix_entry_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
```


Appendix B. (Continued)

```
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to WMRA_matrix_entry (see VARARGIN)

set(handles.edit3, 'String', varargin{1});

% Choose default command line output for WMRA_matrix_entry
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes WMRA_matrix_entry wait for user response (see UIRESUME)
uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function WMRA_matrix_entry_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
%varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%         str2double(get(hObject,'String')) returns contents of edit1 as a double

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VAR_MATRIX

Input=get(handles.edit1, 'String');
VAR_MATRIX = Input;
close;

function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%         str2double(get(hObject,'String')) returns contents of edit3 as a double
% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
```

Appendix B. (Continued)

```
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% This function does the animation of USF WMRA with 9 DOF using Matlab Graphics.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Function Declaration:
function WMRA_ML_Animation(i, Ti, Td, Towc, T01, T12, T23, T34, T45, T56, T67)

% Declaring the global variables:
global L arm_wheelchairl wheelchairu wheelchairc initial_desired_hand
global arm_base ground initialco desiredco handco arm_baseco

% The initialization of the animation plot:
if i==1

    % Reading the Wheelchair's constant dimentions, all dimentions are converted in
    millimeters:
    L=WMRA_WCD;

    % Arm:
    T1=Towc*T01;
    T2=T1*T12;
    T3=T2*T23;
    T4=T3*T34;
    T5=T4*T45;
    T6=T5*T56;
    T7=T6*T67;

    % Wheelchair:
    T8=Towc;
    T9=T8*WMRA_transl(-L(2),-L(3),-L(4));
    T10=T9*WMRA_transl(0,-L(1)/2,0);
    T11=T9*WMRA_transl(0,L(1)/2,0);
    % Lower Platform Corners:
    T12=T9*WMRA_transl(-200,-L(1)/2,0);
    T13=T9*WMRA_transl(-200,L(1)/2,0);
    T14=T9*WMRA_transl(L(2)+200,L(1)/2,0);
    T15=T9*WMRA_transl(L(2)+200,-L(1)/2,0);
    % Upper Platform Corners:
    T16=T9*WMRA_transl(-200,-L(1)/2,L(4));
    T17=T9*WMRA_transl(-200,L(1)/2,L(4));
    T18=T9*WMRA_transl(L(2)+200,L(1)/2,L(4));
    T19=T9*WMRA_transl(L(2)+200,-L(1)/2,L(4));

    % Arm Base Position.
    % Wheelbase Center.
    % Right Wheel Center.
    % Left Wheel Center.
    % Rear Right Wheelchair Corner.
    % Rear Left Wheelchair Corner.
    % Front Left Wheelchair Corner.
    % Front Right Wheelchair Corner.
    % Rear Right Wheelchair Corner.
    % Rear Left Wheelchair Corner.
    % Front Left Wheelchair Corner.
    % Front Right Wheelchair Corner.

    % Initial Animation Plot:
    figure(11);

    % Plots of the Arm and Wheelchair system:
    arm=plot3([T8(1,4), T1(1,4)], [T8(2,4), T1(2,4)], [T8(3,4), T1(3,4)], '-b', [T1(1,4),
T2(1,4)], [T1(2,4), T2(2,4)], [T1(3,4), T2(3,4)], '-g', [T2(1,4), T3(1,4)], [T2(2,4),
```

Appendix B. (Continued)

```

T3(2,4), [T2(3,4), T3(3,4)], '-b', [T3(1,4), T4(1,4)], [T3(2,4),
T4(2,4)], [T3(3,4), T4(3,4)], '-g', [T4(1,4), T5(1,4)], [T4(2,4),
T5(2,4)], [T4(3,4), T5(3,4)], '-b', [T5(1,4), T6(1,4)], [T5(2,4),
T6(2,4)], [T5(3,4), T6(3,4)], '-g', [T6(1,4), T7(1,4)], [T6(2,4),
T7(2,4)], [T6(3,4), T7(3,4)], '-g', 'LineWidth', 3);
hold on;
wheelchairl=plot3([T12(1,4), T13(1,4)], [T12(2,4), T13(2,4)], [T12(3,4), T13(3,4)], '-
g', [T13(1,4), T14(1,4)], [T13(2,4), T14(2,4)], [T13(3,4), T14(3,4)], '-g', [T14(1,4),
T15(1,4)], [T14(2,4), T15(2,4)], [T14(3,4), T15(3,4)], '-g', [T15(1,4), T12(1,4)], [T15(2,4),
T12(2,4)], [T15(3,4), T12(3,4)], '-g', [T10(1,4), T11(1,4)], [T10(2,4),
T11(2,4)], [T10(3,4), T11(3,4)], '-b', 'LineWidth', 3);
wheelchairc=plot3([T16(1,4), T17(1,4)], [T16(2,4), T17(2,4)], [T16(3,4), T17(3,4)], '-
g', [T17(1,4), T18(1,4)], [T17(2,4), T18(2,4)], [T17(3,4), T18(3,4)], '-g', [T18(1,4),
T19(1,4)], [T18(2,4), T19(2,4)], [T18(3,4), T19(3,4)], '-g', [T19(1,4), T16(1,4)], [T19(2,4),
T16(2,4)], [T19(3,4), T16(3,4)], '-g', 'LineWidth', 3);
wheelchair=plot3([T12(1,4), T16(1,4)], [T12(2,4), T16(2,4)], [T12(3,4), T16(3,4)], '-
g', [T13(1,4), T17(1,4)], [T13(2,4), T17(2,4)], [T13(3,4), T17(3,4)], '-g', [T14(1,4),
T18(1,4)], [T14(2,4), T18(2,4)], [T14(3,4), T18(3,4)], '-g', [T15(1,4), T19(1,4)], [T15(2,4),
T19(2,4)], [T15(3,4), T19(3,4)], '-g', 'LineWidth', 3);

% Plots of points of interest on the system:
initial=plot3(Ti(1,4), Ti(2,4), Ti(3,4), '-co', 'LineWidth', 5);
desired=plot3(Td(1,4), Td(2,4), Td(3,4), '-ro', 'LineWidth', 5);
hand=plot3(T7(1,4), T7(2,4), T7(3,4), '-yo', 'LineWidth', 5);
arm_base=plot3(Towc(1,4), Towc(2,4), Towc(3,4), '-mo', 'LineWidth', 5);
ground=plot3(0,0,0, '-ko', 'LineWidth', 5);

% Plots of the x-y-z local coordinate lines of the points of interest on the system:
initialco=plot3([Ti(1,4), Ti(1,4)+100*Ti(1,1)], [Ti(2,4),
Ti(2,4)+100*Ti(2,1)], [Ti(3,4), Ti(3,4)+100*Ti(3,1)], '-r', [Ti(1,4),
Ti(1,4)+100*Ti(1,2)], [Ti(2,4), Ti(2,4)+100*Ti(2,2)], [Ti(3,4), Ti(3,4)+100*Ti(3,2)], '-
g', [Ti(1,4), Ti(1,4)+100*Ti(1,3)], [Ti(2,4), Ti(2,4)+100*Ti(2,3)], [Ti(3,4),
Ti(3,4)+100*Ti(3,3)], '-b', 'LineWidth', 1);
desiredco=plot3([Td(1,4), Td(1,4)+100*Td(1,1)], [Td(2,4),
Td(2,4)+100*Td(2,1)], [Td(3,4), Td(3,4)+100*Td(3,1)], '-r', [Td(1,4),
Td(1,4)+100*Td(1,2)], [Td(2,4), Td(2,4)+100*Td(2,2)], [Td(3,4), Td(3,4)+100*Td(3,2)], '-
g', [Td(1,4), Td(1,4)+100*Td(1,3)], [Td(2,4), Td(2,4)+100*Td(2,3)], [Td(3,4),
Td(3,4)+100*Td(3,3)], '-b', 'LineWidth', 1);
handco=plot3([T7(1,4), T7(1,4)+100*T7(1,1)], [T7(2,4),
T7(2,4)+100*T7(2,1)], [T7(3,4), T7(3,4)+100*T7(3,1)], '-r', [T7(1,4),
T7(1,4)+100*T7(1,2)], [T7(2,4), T7(2,4)+100*T7(2,2)], [T7(3,4), T7(3,4)+100*T7(3,2)], '-
g', [T7(1,4), T7(1,4)+100*T7(1,3)], [T7(2,4), T7(2,4)+100*T7(2,3)], [T7(3,4),
T7(3,4)+100*T7(3,3)], '-b', 'LineWidth', 1);
arm_baseco=plot3([Towc(1,4), Towc(1,4)+100*Towc(1,1)], [Towc(2,4),
Towc(2,4)+100*Towc(2,1)], [Towc(3,4), Towc(3,4)+100*Towc(3,1)], '-r', [Towc(1,4),
Towc(1,4)+100*Towc(1,2)], [Towc(2,4),
Towc(2,4)+100*Towc(2,2)], [Towc(3,4), Towc(3,4)+100*Towc(3,2)], '-g', [Towc(1,4),
Towc(1,4)+100*Towc(1,3)], [Towc(2,4), Towc(2,4)+100*Towc(2,3)], [Towc(3,4),
Towc(3,4)+100*Towc(3,3)], '-b', 'LineWidth', 1);
groundco=plot3([100,0], [0,0], [0,0], '-r', [0,0], [100,0], [0,0], '-
g', [0,0], [0,0], [100,0], '-b', 'LineWidth', 1);

% Specifying plot properties:
view(40,15);
axis([-800 500 -500 800 0 1300]); grid on;
title('WMRA Animation'); xlabel('x, (mm)'); ylabel('y (mm)'); zlabel('z (mm)');
legend([arm(1), arm(2), wheelchairl(5), wheelchairl(1), initial(1), desired(1), hand(1),
arm_base(1), ground(1), initialco(1), initialco(2), initialco(3)], 'ROBOTIC -
', 'ARM', 'wheelaxle', 'wheelchair', 'initial position', 'desired position', 'current
position', 'arm base position', 'ground position', 'local x-axis', 'local y-axis', 'local z-
axis', -1);
hold off;

% Closing the animation plot:
elseif i==3
close (figure(11));

```

Appendix B. (Continued)

```

% Updating the animation plot:
else

    % Arm:
    T1=Towc*T01;
    T2=T1*T12;
    T3=T2*T23;
    T4=T3*T34;
    T5=T4*T45;
    T6=T5*T56;
    T7=T6*T67;

    % Wheelchair:
    T8=Towc;
    T9=T8*WMRA_transl(-L(2),-L(3),-L(4));
    T10=T9*WMRA_transl(0,-L(1)/2,0);
    T11=T9*WMRA_transl(0,L(1)/2,0);
    % Lower Platform Corners:
    T12=T9*WMRA_transl(-200,-L(1)/2,0);
    T13=T9*WMRA_transl(-200,L(1)/2,0);
    T14=T9*WMRA_transl(L(2)+200,L(1)/2,0);
    T15=T9*WMRA_transl(L(2)+200,-L(1)/2,0);
    % Upper Platform Corners:
    T16=T9*WMRA_transl(-200,-L(1)/2,L(4));
    T17=T9*WMRA_transl(-200,L(1)/2,L(4));
    T18=T9*WMRA_transl(L(2)+200,L(1)/2,L(4));
    T19=T9*WMRA_transl(L(2)+200,-L(1)/2,L(4));

    % Arm Base Position.
    % Wheelbase Center.
    % Right Wheel Center.
    % Left Wheel Center.
    % Rear Right Wheelchair Corner.
    % Rear Left Wheelchair Corner.
    % Front Left Wheelchair Corner.
    % Front Right Wheelchair Corner.
    % Rear Right Wheelchair Corner.
    % Rear Left Wheelchair Corner.
    % Front Left Wheelchair Corner.
    % Front Right Wheelchair Corner.

    % Updating Animation Plot:

    % Plots of the Arm and Wheelchair system:
    set(arm(1),'XData',[T8(1,4), T1(1,4)],'YData',[T8(2,4),
    T1(2,4)],'ZData',[T8(3,4),T1(3,4)]);
    set(arm(2),'XData',[T1(1,4), T2(1,4)],'YData',[T1(2,4),
    T2(2,4)],'ZData',[T1(3,4),T2(3,4)]);
    set(arm(3),'XData',[T2(1,4), T3(1,4)],'YData',[T2(2,4),
    T3(2,4)],'ZData',[T2(3,4),T3(3,4)]);
    set(arm(4),'XData',[T3(1,4), T4(1,4)],'YData',[T3(2,4),
    T4(2,4)],'ZData',[T3(3,4),T4(3,4)]);
    set(arm(5),'XData',[T4(1,4), T5(1,4)],'YData',[T4(2,4),
    T5(2,4)],'ZData',[T4(3,4),T5(3,4)]);
    set(arm(6),'XData',[T5(1,4), T6(1,4)],'YData',[T5(2,4),
    T6(2,4)],'ZData',[T5(3,4),T6(3,4)]);
    set(arm(7),'XData',[T6(1,4), T7(1,4)],'YData',[T6(2,4),
    T7(2,4)],'ZData',[T6(3,4),T7(3,4)]);
    set(wheelchairl(1),'XData',[T12(1,4), T13(1,4)],'YData',[T12(2,4),
    T13(2,4)],'ZData',[T12(3,4),T13(3,4)]);
    set(wheelchairl(2),'XData',[T13(1,4), T14(1,4)],'YData',[T13(2,4),
    T14(2,4)],'ZData',[T13(3,4),T14(3,4)]);
    set(wheelchairl(3),'XData',[T14(1,4), T15(1,4)],'YData',[T14(2,4),
    T15(2,4)],'ZData',[T14(3,4),T15(3,4)]);
    set(wheelchairl(4),'XData',[T15(1,4), T12(1,4)],'YData',[T15(2,4),
    T12(2,4)],'ZData',[T15(3,4),T12(3,4)]);
    set(wheelchairl(5),'XData',[T10(1,4), T11(1,4)],'YData',[T10(2,4),
    T11(2,4)],'ZData',[T10(3,4),T11(3,4)]);
    set(wheelchairu(1),'XData',[T16(1,4), T17(1,4)],'YData',[T16(2,4),
    T17(2,4)],'ZData',[T16(3,4),T17(3,4)]);
    set(wheelchairu(2),'XData',[T17(1,4), T18(1,4)],'YData',[T17(2,4),
    T18(2,4)],'ZData',[T17(3,4),T18(3,4)]);
    set(wheelchairu(3),'XData',[T18(1,4), T19(1,4)],'YData',[T18(2,4),
    T19(2,4)],'ZData',[T18(3,4),T19(3,4)]);
    set(wheelchairu(4),'XData',[T19(1,4), T16(1,4)],'YData',[T19(2,4),
    T16(2,4)],'ZData',[T19(3,4),T16(3,4)]);
    set(wheelchairc(1),'XData',[T12(1,4), T16(1,4)],'YData',[T12(2,4),
    T16(2,4)],'ZData',[T12(3,4),T16(3,4)]);

```

Appendix B. (Continued)

```
set(wheelchairc(2), 'XData', [T13(1,4), T17(1,4)], 'YData', [T13(2,4),
T17(2,4)], 'ZData', [T13(3,4), T17(3,4)]);
set(wheelchairc(3), 'XData', [T14(1,4), T18(1,4)], 'YData', [T14(2,4),
T18(2,4)], 'ZData', [T14(3,4), T18(3,4)]);
set(wheelchairc(4), 'XData', [T15(1,4), T19(1,4)], 'YData', [T15(2,4),
T19(2,4)], 'ZData', [T15(3,4), T19(3,4)]);

% Plots of points of interest on the system:
set(initial(1), 'XData', Ti(1,4), 'YData', Ti(2,4), 'ZData', Ti(3,4));
set(desired(1), 'XData', Td(1,4), 'YData', Td(2,4), 'ZData', Td(3,4));
set(hand(1), 'XData', T7(1,4), 'YData', T7(2,4), 'ZData', T7(3,4));
set(arm_base(1), 'XData', Towc(1,4), 'YData', Towc(2,4), 'ZData', Towc(3,4));

% Plots of the x-y-z local coordinate lines of the points of interest on the system:
set(initialco(1), 'XData', [Ti(1,4), Ti(1,4)+100*Ti(1,1)], 'YData', [Ti(2,4),
Ti(2,4)+100*Ti(2,1)], 'ZData', [Ti(3,4), Ti(3,4)+100*Ti(3,1)]);
set(initialco(2), 'XData', [Ti(1,4), Ti(1,4)+100*Ti(1,2)], 'YData', [Ti(2,4),
Ti(2,4)+100*Ti(2,2)], 'ZData', [Ti(3,4), Ti(3,4)+100*Ti(3,2)]);
set(initialco(3), 'XData', [Ti(1,4), Ti(1,4)+100*Ti(1,3)], 'YData', [Ti(2,4),
Ti(2,4)+100*Ti(2,3)], 'ZData', [Ti(3,4), Ti(3,4)+100*Ti(3,3)]);
set(desiredco(1), 'XData', [Td(1,4), Td(1,4)+100*Td(1,1)], 'YData', [Td(2,4),
Td(2,4)+100*Td(2,1)], 'ZData', [Td(3,4), Td(3,4)+100*Td(3,1)]);
set(desiredco(2), 'XData', [Td(1,4), Td(1,4)+100*Td(1,2)], 'YData', [Td(2,4),
Td(2,4)+100*Td(2,2)], 'ZData', [Td(3,4), Td(3,4)+100*Td(3,2)]);
set(desiredco(3), 'XData', [Td(1,4), Td(1,4)+100*Td(1,3)], 'YData', [Td(2,4),
Td(2,4)+100*Td(2,3)], 'ZData', [Td(3,4), Td(3,4)+100*Td(3,3)]);
set(handco(1), 'XData', [T7(1,4), T7(1,4)+100*T7(1,1)], 'YData', [T7(2,4),
T7(2,4)+100*T7(2,1)], 'ZData', [T7(3,4), T7(3,4)+100*T7(3,1)]);
set(handco(2), 'XData', [T7(1,4), T7(1,4)+100*T7(1,2)], 'YData', [T7(2,4),
T7(2,4)+100*T7(2,2)], 'ZData', [T7(3,4), T7(3,4)+100*T7(3,2)]);
set(handco(3), 'XData', [T7(1,4), T7(1,4)+100*T7(1,3)], 'YData', [T7(2,4),
T7(2,4)+100*T7(2,3)], 'ZData', [T7(3,4), T7(3,4)+100*T7(3,3)]);
set(arm_baseco(1), 'XData', [Towc(1,4), Towc(1,4)+100*Towc(1,1)], 'YData', [Towc(2,4),
Towc(2,4)+100*Towc(2,1)], 'ZData', [Towc(3,4), Towc(3,4)+100*Towc(3,1)]);
set(arm_baseco(2), 'XData', [Towc(1,4), Towc(1,4)+100*Towc(1,2)], 'YData', [Towc(2,4),
Towc(2,4)+100*Towc(2,2)], 'ZData', [Towc(3,4), Towc(3,4)+100*Towc(3,2)]);
set(arm_baseco(3), 'XData', [Towc(1,4), Towc(1,4)+100*Towc(1,3)], 'YData', [Towc(2,4),
Towc(2,4)+100*Towc(2,3)], 'ZData', [Towc(3,4), Towc(3,4)+100*Towc(3,3)]);

end
```

```
% This function is for the resolved rate and optimization solution of the USF WMRA with 9
DOF.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Function Declaration:
```

```
function [dq]=WMRA_Opt(i, JLA, JLO, Jo, detJo, dq, dx, dt, q)
```

```
% Declaring a global variable:
```

```
global dHo
```

```
% Reading the Wheelchair's constant dimentions, all dimentions are converted in
millimeters:
```

```
L=WMRA_WCD;
```

```
% The case when wheelchair-only control is required with no arm motion:
```

```
if i==0
    WCA=3;
```

Appendix B. (Continued)

```
% calculating the Inverse of the Jacobian, which is always non-singular:
pinvJo=inv(Jo(1:2,1:2));
% calculating the joint angle change:
% Here, dq of the wheels are translated from radians to distances travelled after
using the Jacobian.
dq=pinvJo*dx;
dq(1)=dq(1)*L(5);
else
% Reading the physical joint limits of the arm:
[qmin,qmax]=WMRA_Jlimit;
% Creating the gradient of the optimization function to avoid joint limits:
dH=[0;0;0;0;0;0;0];
if JLA==1
for j=1:7
dH(j)=-0.25*(qmax(j)-qmin(j))^2*(2*q(j)-qmax(j)-qmin(j))/((qmax(j)-
q(j))^2*(q(j)-qmin(j))^2);
% Re-defining the weight in case the joint is moving away from it's limit or
the joint limit was exceeded:
if abs(dH(j)) < abs(dHo(j)) && q(j) < qmax(j) && q(j) > qmin(j)
dH(j)=0;
elseif abs(dH(j)) < abs(dHo(j)) && (q(j) >= qmax(j) || q(j) <= qmin(j))
dH(j)=inf;
elseif abs(dH(j)) > abs(dHo(j)) && (q(j) >= qmax(j) || q(j) <= qmin(j))
dH(j)=0;
end
end
end
dHo=dH;
% The case when arm-only control is required with no wheelchair motion:
if max(size(dq))==7
WCA=2;
wo=200000000;
ko=350000;
% The weight matrix to be used for the Weighted Least Norm Solution with Joint
Limit Avoidance:
W=diag(1*[1;1;1;1;1;1;1]+1*abs(dH));
% The inverse of the diagonal weight matrix:
dia=diag(W);
Winv=diag([1/dia(1); 1/dia(2); 1/dia(3); 1/dia(4); 1/dia(5); 1/dia(6);
1/dia(7)]);
% The case when wheelchair-and-arm control is required:
else
WCA=1;
wo=340000000;
ko=13;
% The weight matrix to be used for the Weighted Least Norm Solution:
W=diag([1*[1;1;1;1;1;1;1]+1*abs(dH);10*[1;1]]);
% The inverse of the diagonal weight matrix:
dia=diag(W);
Winv=diag([1/dia(1); 1/dia(2); 1/dia(3); 1/dia(4); 1/dia(5); 1/dia(6); 1/dia(7);
1/dia(8); 1/dia(9)]);
end
% Redefining the determinant based on the weight:
if i==1 || i==2
detJo=sqrt(det(Jo*Winv*Jo'));
end
dof=max(size(dx));
end

% SR-Inverse and Weighted Least Norm Optimization:
if i==1
% Calculating the variable scale factor, sf:
if detJo<wo
sf=ko*(1-detJo/wo)^2; % from eq. 9.79 page 268 of Nakamura's book.
else
sf=0;
end
end
```

Appendix B. (Continued)

```
end
% claculating the SR-Inverse of the Jacobian:
pinvJo=Winv*Jo'*inv(Jo*Winv*Jo'+sf*eye(dof));
% calculating the joint angle change optimized based on the Weighted Least Norm
Solution:
% Here, dq of the wheels are translated from radians to distances travelled after
using the Jacobian.
if WCA==2
    dq=pinvJo*dx;
else
    dq=pinvJo*dx;
    dq(8)=dq(8)*L(5);
end

% Pseudo Inverse and Weighted Least Norm Optimization:
elseif i==2
    % claculating the Pseudo Inverse of the Jacobian:
    pinvJo=Winv*Jo'*inv(Jo*Winv*Jo');
    % calculating the joint angle change optimized based on the Weighted Least Norm
Solution:
    % Here, dq of the wheels are translated from radians to distances travelled after
using the Jacobian.
    if WCA==2
        dq=pinvJo*dx;
    else
        dq=pinvJo*dx;
        dq(8)=dq(8)*L(5);
    end

% SR-Inverse and Projection Gradient Optimization based on Euclidean norm of errors:
elseif i==3
    % Calculating the variable scale factor, sf:
    if detJo<wo
        sf=ko*(1-detJo/wo)^2;          % from eq. 9.79 page 268 of Nakamura's book.
    else
        sf=0;
    end
    % claculating the SR-Inverse of the Jacobian:
    pinvJo=Jo'*inv(Jo*Jo'+sf*eye(dof));
    % calculating the joint angle change optimized based on minimizing the Euclidean norm
of errors:
    % Here, dq of the wheels are translated from distances travelled to radians, and back
after using the Jacobian.
    if WCA==2
        %dq=pinvJo*dx+(eye(7)-pinvJo*Jo)*dq;
        dq=pinvJo*dx+0.001*(eye(7)-pinvJo*Jo)*dH;
    else
        %dq(8)=dq(8)/L(5);
        %dq=pinvJo*dx+(eye(9)-pinvJo*Jo)*dq;
        dq=pinvJo*dx+0.001*(eye(9)-pinvJo*Jo)*[dH;0;0];
        dq(8)=dq(8)*L(5);
    end

% Pseudo Inverse and Projection Gradient Optimization based on Euclidean norm of errors:
elseif i==4
    % claculating the Pseudo Inverse of the Jacobian:
    pinvJo=Jo'*inv(Jo*Jo');
    % calculating the joint angle change optimized based on minimizing the Euclidean norm
of errors:
    % Here, dq of the wheels are translated from distances travelled to radians, and back
after using the Jacobian.
    if WCA==2
        %dq=pinvJo*dx+(eye(7)-pinvJo*Jo)*dq;
        dq=pinvJo*dx+0.001*(eye(7)-pinvJo*Jo)*dH;
    else
        %dq(8)=dq(8)/L(5);
```

Appendix B. (Continued)

```

    %dq=pinvJo*dx+(eye(9)-pinvJo*Jo)*dq;
    dq=pinvJo*dx+0.001*(eye(9)-pinvJo*Jo)*[dH;0;0];
    dq(8)=dq(8)*L(5);
end
end

if JLO==1
    % A safety condition to stop the joint that reaches the joint limits in the arm:
    if WCA~=3
        for k=1:7
            if q(k) >= qmax(k) || q(k) <= qmin(k)
                dq(k)=0;
            end
        end
    end
    % A safety condition to slow the joint that exceeds the velocity limits in the WMRA:
    if WCA==3
        dqmax=dt*[100;0.15]; % Joiny velocity limits when the time increment is dt
        second.
    else
        dqmax=dt*[0.5;0.5;0.5;0.5;0.5;0.5;0.5;100;0.15]; % Joiny velocity limits when the
        time increment is dt second.
    end
    for k=1:max(size(dq))
        if abs(dq(k)) >= dqmax(k)
            dq(k)=sign(dq(k))*dqmax(k);
        end
    end
end
end
end

```

```

% This function gives the Transformation Matrix of the WMRA's base on the wheelchair with
2 DOF, given the desired x,y position and z rotation angle.
% Dimentions are as supplies, angles are in radians.

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

% Function Declaration:
function [T]=WMRA_p2T(x, y, a)

```

```

% Reading the Wheelchair's constant dimentions, all dimentions are converted in
millimeters:
L=WMRA_WCD;

```

```

% Defining the Transformation Matrix:
T=[cos(a) -sin(a) 0 x ; sin(a) cos(a) 0 y ; 0 0 1 L(4)+L(5) ; 0 0 0 1];

```

```

% This "new USF WMRA" function SIMULATES the arm going from the parking position to the
ready position with ANIMATION. All angles are in Radians.
% The parking position is assumed to be qi=[0;pi/2;0;pi;0;0;0] (Radians),
% the ready position is assumed to be qd=[pi/2;pi/2;0;pi/2;pi/2;0] (Radians).
% ini=1 --> initialize animation figures, ini=2 or any --> just update the figures, ini=3
--> close the figures.

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```


Appendix B. (Continued)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Function Declaration:
function WMRA_park2ready(ini, vr, ml, arm, Tiwc, qiwc)

% Closing the Arm library and Matlab Graphics Animation and Virtual Reality Animation and
Plots windows:
if ini==3
    if arm==1
        try
            WMRA_ARM_Motion(ini, 0, 0, 0);
        end
    end
    if vr==1
        try
            WMRA_VR_Animation(ini, 0, 0);
        end
    end
    if ml==1
        try
            WMRA_ML_Animation(ini, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
        end
    end
    return;
end

% Defining the used conditions:
qi=[0;pi/2;0;pi;0;0;0]; % Initial joint angles (Parking Position).
qd=[pi/2;pi/2;0;pi/2;pi/2;pi/2;0]; % Final joint angles (Ready Position).
ts=10; % (5 or 10 or 20) Simulation time to move the arm from the parking position
to the ready position.
n=100; % Number of time steps.
dt=ts/n; % The time step to move the arm from the parking position to the ready
position.
dq=(qd-qi)/(0.5*n+5); % Joint angle change at every time step.

% Initializing the physical Arm:
if arm==1
    WMRA_ARM_Motion(ini, 2, [qi;qiwc;0], dt);
    ddt=0;
end

% Initializing Virtual Reality Animation:
if vr==1
    WMRA_VR_Animation(ini, Tiwc, [qi;qiwc]);
end

% Initializing Robot Animation in Matlab Graphics:
if ml==1
    % Inputting the D-H Parameters in a Matrix form:
    DH=WMRA_DH(qi);

    % Calculating the transformation matrices of each link:

    T01=WMRA_rotx(DH(1,1))*WMRA_transl(DH(1,2),0,0)*WMRA_rotz(DH(1,4))*WMRA_transl(0,0,DH(1,3));
    T12=WMRA_rotx(DH(2,1))*WMRA_transl(DH(2,2),0,0)*WMRA_rotz(DH(2,4))*WMRA_transl(0,0,DH(2,3));
    T23=WMRA_rotx(DH(3,1))*WMRA_transl(DH(3,2),0,0)*WMRA_rotz(DH(3,4))*WMRA_transl(0,0,DH(3,3));
    T34=WMRA_rotx(DH(4,1))*WMRA_transl(DH(4,2),0,0)*WMRA_rotz(DH(4,4))*WMRA_transl(0,0,DH(4,3));
end
```

Appendix B. (Continued)

```
T45=WMRA_rotx(DH(5,1))*WMRA_transl(DH(5,2),0,0)*WMRA_rotz(DH(5,4))*WMRA_transl(0,0,DH(5,3));
T56=WMRA_rotx(DH(6,1))*WMRA_transl(DH(6,2),0,0)*WMRA_rotz(DH(6,4))*WMRA_transl(0,0,DH(6,3));
T67=WMRA_rotx(DH(7,1))*WMRA_transl(DH(7,2),0,0)*WMRA_rotz(DH(7,4))*WMRA_transl(0,0,DH(7,3));
% Calculating the Transformation Matrix of the initial and desired arm positions:
Ti=Tiwc*T01*T12*T23*T34*T45*T56*T67;
Td=Tiwc*WMRA_q2T(qd);
WMRA_ML_Animation(ini, Ti, Td, Tiwc, T01, T12, T23, T34, T45, T56, T67);
end

% Initialization:
qo=qi;
tt=0;

while tt <= (ts)
    % Starting a timer:
    tic;

    % Calculating the new Joint Angles:
    if tt==0
        qn=qo;
    elseif tt < (dt*(0.5*n-5))
        qn(1)=qo(1)+dq(1);
    elseif tt < (dt*(0.5*n+5))
        qn=qo+dq;
    elseif tt < (dt*(n-1))
        qn(2:7)=qo(2:7)+dq(2:7);
    end

    % Updating the physical Arm:
    if arm==1
        ddt=ddt+dt;
        if ddt>=0.5 || tt>=(ts)
            WMRA_ARM_Motion(2, 1, [qn;qiwc;0], ddt);
            ddt=0;
        end
    end

    % Updating Virtual Reality Animation:
    if vr==1
        WMRA_VR_Animation(2, Tiwc, [qn;qiwc]);
    end

    % Updating Matlab Animation:
    if ml==1
        % Calculating the new Transformation Matrix:
        T1a=WMRA_rotx(DH(1,1))*WMRA_transl(DH(1,2),0,0)*WMRA_rotz(qn(1))*WMRA_transl(0,0,DH(1,3));
        T2a=WMRA_rotx(DH(2,1))*WMRA_transl(DH(2,2),0,0)*WMRA_rotz(qn(2))*WMRA_transl(0,0,DH(2,3));
        T3a=WMRA_rotx(DH(3,1))*WMRA_transl(DH(3,2),0,0)*WMRA_rotz(qn(3))*WMRA_transl(0,0,DH(3,3));
        T4a=WMRA_rotx(DH(4,1))*WMRA_transl(DH(4,2),0,0)*WMRA_rotz(qn(4))*WMRA_transl(0,0,DH(4,3));
        T5a=WMRA_rotx(DH(5,1))*WMRA_transl(DH(5,2),0,0)*WMRA_rotz(qn(5))*WMRA_transl(0,0,DH(5,3));
```

Appendix B. (Continued)

```
T6a=WMRA_rotx(DH(6,1))*WMRA_transl(DH(6,2),0,0)*WMRA_rotz(qn(6))*WMRA_transl(0,0,DH(6,3))
;

T7a=WMRA_rotx(DH(7,1))*WMRA_transl(DH(7,2),0,0)*WMRA_rotz(qn(7))*WMRA_transl(0,0,DH(7,3))
;

    WMRA_ML_Animation(2, Ti, Td, Tiwc, T1a, T2a, T3a, T4a, T5a, T6a, T7a);
end

% Updating the old values with the new values for the next iteration:
qo=qn;
tt=tt+dt;

% Pausing for the speed sync:
pause(dt-toc);

end
```

```
% This function plots different animation variables for USF WMRA with 9 DOF.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Function Declaration:
```

```
function WMRA_Plots(st, L, r2d, dt, i, tt, qn, dq, Tn, Tnwc, detjoa, detjo)
```

```
% Declaring the global variables:
```

```
global time q1 q2 q3 q4 q5 q6 q7 q11 qrr
global qd1 qd2 qd3 qd4 qd5 qd6 qd7 qd1 qdr
global x y z roll pitch yaw xc yc zc rollc pitchc yawc detJoa detJo
```

```
% Data collection at every iteration:
```

```
if st==1
```

```
    % Generating a time vector for plotting:
```

```
    time(i)=tt;
```

```
    % Joint Angles:
```

```
    q1(i)=qn(1)*r2d;
```

```
    q2(i)=qn(2)*r2d;
```

```
    q3(i)=qn(3)*r2d;
```

```
    q4(i)=qn(4)*r2d;
```

```
    q5(i)=qn(5)*r2d;
```

```
    q6(i)=qn(6)*r2d;
```

```
    q7(i)=qn(7)*r2d;
```

```
    q11(i)=qn(8)-L(1)*qn(9)/2;
```

```
    qrr(i)=qn(8)+L(1)*qn(9)/2;
```

```
    % Joint Velocities:
```

```
    qd1(i)=r2d*dq(1)/dt;
```

```
    qd2(i)=r2d*dq(2)/dt;
```

```
    qd3(i)=r2d*dq(3)/dt;
```

```
    qd4(i)=r2d*dq(4)/dt;
```

```
    qd5(i)=r2d*dq(5)/dt;
```

```
    qd6(i)=r2d*dq(6)/dt;
```

```
    qd7(i)=r2d*dq(7)/dt;
```

```
    qd1(i)=(dq(8)-L(1)*dq(9)/2)/dt;
```

```
    qdr(i)=(dq(8)+L(1)*dq(9)/2)/dt;
```

```
    % Hand Position and Orientation:
```

```
    x(i)=Tn(1,4);
```

```
    y(i)=Tn(2,4);
```

```
    z(i)=Tn(3,4);
```

```
    or=WMRA_T2rpy(Tn);
```

Appendix B. (Continued)

```
roll(i)=or(1)*r2d;
pitch(i)=or(2)*r2d;
yaw(i)=or(3)*r2d;
% Arm Base Position and Orientation on the Wheelchair:
xc(i)=Tnwc(1,4);
yc(i)=Tnwc(2,4);
zc(i)=Tnwc(3,4);
orc=WMRA_T2rpy(Tnwc);
rollc(i)=orc(1)*r2d;
pitchc(i)=orc(2)*r2d;
yawc(i)=orc(3)*r2d;
% Manipulability Measure:
detJoa(i)=detjao;
detJo(i)=detjo;

% Plotting the data in graphas:
else

figure(2);
plot(time,qd1,'-y',time,qd2,'-m',time,qd3,'-c',time,qd4,'-r',time,qd5,'-
g',time,qd6,'-b',time,qd7,'-k');
grid on; title('Joint Angular Velocities vs Time');xlabel('time,
(sec)');ylabel('joint velocities,
(deg/s)');legend('\theta_1d','\theta_2d','\theta_3d','\theta_4d','\theta_5d','\theta_6d',
'\theta_7d',-1);

figure(3);
plot(time,qdl,'-b',time,qdr,'-g');
grid on; title('Wheels Track Velocities vs Time');xlabel('time,
(sec)');ylabel('wheels track velocities, (mm/s)');legend('\theta_Ld','\theta_Rd',-1);

figure(4);
plot(time,q1,'-y',time,q2,'-m',time,q3,'-c',time,q4,'-r',time,q5,'-g',time,q6,'-
b',time,q7,'-k');
grid on; title('Joint Angular Displacements vs Time');xlabel('time,
(sec)');ylabel('joint angles,
(deg)');legend('\theta_1','\theta_2','\theta_3','\theta_4','\theta_5','\theta_6','\theta_
7',-1);

figure(5);
plot(time,qll,'-b',time,qrr,'-g');
grid on; title('Wheels Track distances vs Time');xlabel('time, (sec)');ylabel('wheels
track distances, (mm)');legend('\theta_L','\theta_R',-1);

figure(6);
plot(time,x,'-y',time,y,'-m',time,z,'-c');
grid on; title('Hand Position vs Time');xlabel('time, (sec)');ylabel('position,
(mm)');legend('x','y','z',-1);

figure(7);
plot(time,roll,'-y',time,pitch,'-m',time,yaw,'-c');
grid on; title('Hand Orientation vs Time');xlabel('time, (sec)');ylabel('orientation,
(deg)');legend('roll','pitch','yaw',-1);

figure(8);
plot(time,xc,'-y',time,yc,'-m',time,zc,'-c');
grid on; title('Arm Base Position vs Time');xlabel('time, (sec)');ylabel('position,
(mm)');legend('x','y','z',-1);

figure(9);
plot(time,rollc,'-y',time,pitchc,'-m',time,yawc,'-c');
grid on; title('Arm Base Orientation vs Time');xlabel('time,
(sec)');ylabel('orientation, (deg)');legend('roll','pitch','yaw',-1);

figure(10);
plot(time,detJoa,'-y',time,detJo,'-m');
```

Appendix B. (Continued)

```
grid on; title('Manipulability Measure vs Time');xlabel('time,  
(sec)');ylabel('Manipulability Measure'); legend('W_A_R_M', 'W_W_M_R_A',-1);  
end
```

```
% This function uses a 3rd order Polynomial with no Blending factor to find a smooth  
trajectory points of a variable "q" along a streight line, given the initial and final  
variable values and the number of trajectory points.  
% The output is the variable position.  
% See Eqs. 7.3 and 7.6 page 204,205 of Craig Book
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Function Declaration:  
function [qt] = WMRA_Polynomial(qi, qf, n)
```

```
tt=0;  
tf=abs((qf-qi));  
dt=tf/(n-1);
```

```
for i=1:n  
    if tf <= 0.001  
        qt(i)=qi;  
    else  
        qt(i)=qi+(qf-qi)*3*tt^2/tf^2-(qf-qi)*2*tt^3/tf^3; % From Eq.7.3 and 7.6 page  
204,205 of Craig Book  
    end  
    tt=tt+dt;  
end
```

```
% This function reads the BCI 2000 device output through TCP/IP port,  
% extracts the selected row and column of the screen interface out of the sent data from  
the BCI 2000,  
% converts these row and column data to a commanded Cartesian velocities from the BCI  
device  
% and sends it as an output.  
% The optional input to this function is the port number.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Thanks to Eduardo Veras %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Function declaration:  
function dx=WMRA_psy(port1)
```

```
% Assigning the port number in case the user did not input it:  
if nargin<1  
    port1=19711;  
end
```

```
% Openning the port:  
try  
    udp=pnet('udpsocket',port1);
```

Appendix B. (Continued)

```
catch
    pnet(udp, 'close');
    clear udp;
end

% Initializing the loop variable:
i=1;

% Starting the loop:
while(i<2)

    % Trying to read the data packet from the port:
    try
        len=pnet(udp, 'readpacket');

        % Condition to make sure that the data is read:
        if len>0

            % Reading a data block:
            data=pnet(udp, 'read', 36, 'uint64', 'ieee-be', 'block');

            % Condition to make sure that the read data is not blank:
            if (isempty(data)~=1)

                % Condition to make sure that the read data is not a blank line:
                if length(data) > 0

                    % Finding the string 'SelectedRow' out of that data line:
                    k1 = findstr(data, 'SelectedRow');

                    % Condition in case the required string is found:
                    if (isempty(k1)~=1)
                        % The length of the string:
                        num1 = length(k1);
                        % Condition to ensure that the string length > 0:
                        if num1 > 0
                            % Reading the string that comes right after the selected
string and converting it to a number:
                            mrow = str2double(data(13));
                        end
                    end

                    % Finding the string 'SelectedColumn' out of that data line:
                    k2 = findstr(data, 'SelectedColumn');

                    % Condition in case the required string is found:
                    if (isempty(k2)~=1)
                        % The length of the string:
                        num2 = length(k2);
                        % Condition to ensure that the string length > 0:
                        if num2 > 0
                            % Reading the string that comes right after the selected
string and converting it to a number:
                            mcol = str2double(data(16));
                        end
                    end

                    %Modifying the output to the proper format:
                    rc=[mrow mcol];

                    % Assigning the directional velocity vector based on the selected row
and column from the interface screen:
                    dx=[0;0;0;0;0;0;0];
                    if rc == [1 1]
                        dx=[0;0;1;0;0;0;0];
                    elseif rc == [1 2]
```

Appendix B. (Continued)

```
dx=[1;0;0;0;0;0;0];
elseif rc == [1 3]
dx=[0;0;0;0;0;0.003;0];
elseif rc == [1 4]
dx=[0;0;0;0;0.003;0;0];
elseif rc == [1 5]
dx=[0;0;0;0;0;-0.003;0];
elseif rc == [2 1]
dx=[0;1;0;0;0;0;0];
elseif rc == [2 2]
dx=[0;-1;0;0;0;0;0];
elseif rc == [2 3]
dx=[0;0;0;-0.003;0;0;0];
elseif rc == [2 4]
dx=[0;0;0;0;-0.003;0;0];
elseif rc == [2 5]
dx=[0;0;0;0.003;0;0;0];
elseif rc == [3 1]
dx=[0;0;-1;0;0;0;0];
elseif rc == [3 2]
dx=[-1;0;0;0;0;0;0];
elseif rc == [3 3]
dx=[0;0;0;0;0;0;0];
elseif rc == [3 4]
dx=[0;0;0;0;0;0;1];
elseif rc == [3 5]
dx=[0;0;0;0;0;0;-1];
else
fprintf('ERROR');
end
% dx=dx(1:6);
% Once we get the reading, we can get out of the loop:
i=2;

end
end
catch
end
end
% Be a good citizen and cleanup your mess:
pnet(udp, 'close');
clear udp;

% This function gives the Transformation Matrix of the new USF WMRA with 7 DOF, given the
joint angles in Radians.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Function Declaration:
function [T]=WMRA_q2T(q)

% Inputting the D-H Parameters in a Matrix form:
DH=WMRA_DH(q);

% Calculating the transformation matrices of each link:
T1=WMRA_rotx(DH(1,1))*WMRA_transl(DH(1,2),0,0)*WMRA_rotz(DH(1,4))*WMRA_transl(0,0,DH(1,3));
```

Appendix B. (Continued)

```

T2=WMRA_rotx(DH(2,1))*WMRA_transl(DH(2,2),0,0)*WMRA_rotz(DH(2,4))*WMRA_transl(0,0,DH(2,3)
);
T3=WMRA_rotx(DH(3,1))*WMRA_transl(DH(3,2),0,0)*WMRA_rotz(DH(3,4))*WMRA_transl(0,0,DH(3,3)
);
T4=WMRA_rotx(DH(4,1))*WMRA_transl(DH(4,2),0,0)*WMRA_rotz(DH(4,4))*WMRA_transl(0,0,DH(4,3)
);
T5=WMRA_rotx(DH(5,1))*WMRA_transl(DH(5,2),0,0)*WMRA_rotz(DH(5,4))*WMRA_transl(0,0,DH(5,3)
);
T6=WMRA_rotx(DH(6,1))*WMRA_transl(DH(6,2),0,0)*WMRA_rotz(DH(6,4))*WMRA_transl(0,0,DH(6,3)
);
T7=WMRA_rotx(DH(7,1))*WMRA_transl(DH(7,2),0,0)*WMRA_rotz(DH(7,4))*WMRA_transl(0,0,DH(7,3)
);

% Calculating the total Transformation Matrix of the given arm position:
T=T1*T2*T3*T4*T5*T6*T7;

```

```

% This "new USF WMRA" function SIMULATES the arm going from the ready position to any
position with ANIMATION. All angles are in Radians.
% the ready position is assumed to be qd=[pi/2;pi/2;0;pi/2;pi/2;0] (Radians).
% ini=1 --> initialize animation figures, ini=2 or any --> just update the figures, ini=3
--> close the figures.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Function Declaration:
function WMRA_ready2any(ini, vr, ml, arm, Tiwc, qd)

% Closing the Arm library and Matlab Graphics Animation and Virtual Reality Animation and
Plots windows:
if ini==3
    if arm==1
        try
            WMRA_ARM_Motion(ini, 0, 0, 0);
        end
    end
    if vr==1
        try
            WMRA_VR_Animation(ini, 0, 0);
        end
    end
    if ml==1
        try
            WMRA_ML_Animation(ini, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
        end
    end
    return;
end

% Defining the used conditions:
qi=[pi/2;pi/2;0;pi/2;pi/2;0]; % Initial joint angles (Ready Position).
ts=10; % (5 or 10 or 20) Simulation time to move the arm from the ready position to
any position.
n=100; % Number of time steps.
dt=ts/n; % The time step to move the arm from the ready position to any position.

% Initializing the physical Arm:
if arm==1
    WMRA_ARM_Motion(ini, 2, [qi;0;0;0], dt);
    ddt=0;

```


Appendix B. (Continued)

```
end

% Initializing Virtual Reality Animation:
if vr==1
    WMRA_VR_Animation(ini, Tiwc, [qi;0;0]);
end

% Initializing Robot Animation in Matlab Graphics:
if ml==1
    % Inputting the D-H Parameters in a Matrix form:
    DH=WMRA_DH(qi);

    % Calculating the transformation matrices of each link:

T01=WMRA_rotx(DH(1,1))*WMRA_transl(DH(1,2),0,0)*WMRA_rotz(DH(1,4))*WMRA_transl(0,0,DH(1,3));
T12=WMRA_rotx(DH(2,1))*WMRA_transl(DH(2,2),0,0)*WMRA_rotz(DH(2,4))*WMRA_transl(0,0,DH(2,3));
T23=WMRA_rotx(DH(3,1))*WMRA_transl(DH(3,2),0,0)*WMRA_rotz(DH(3,4))*WMRA_transl(0,0,DH(3,3));
T34=WMRA_rotx(DH(4,1))*WMRA_transl(DH(4,2),0,0)*WMRA_rotz(DH(4,4))*WMRA_transl(0,0,DH(4,3));
T45=WMRA_rotx(DH(5,1))*WMRA_transl(DH(5,2),0,0)*WMRA_rotz(DH(5,4))*WMRA_transl(0,0,DH(5,3));
T56=WMRA_rotx(DH(6,1))*WMRA_transl(DH(6,2),0,0)*WMRA_rotz(DH(6,4))*WMRA_transl(0,0,DH(6,3));
T67=WMRA_rotx(DH(7,1))*WMRA_transl(DH(7,2),0,0)*WMRA_rotz(DH(7,4))*WMRA_transl(0,0,DH(7,3));
    % Calculating the Transformation Matrix of the initial and desired arm positions:
    Ti=Tiwc*T01*T12*T23*T34*T45*T56*T67;
    Td=Tiwc*WMRA_q2T(qd);
    WMRA_ML_Animation(ini, Ti, Td, Tiwc, T01, T12, T23, T34, T45, T56, T67);
end

% Check for the shortest route:
diff=qd-qi;
for i=1:7
    if diff(i) > pi
        diff(i)=diff(i)-2*pi;
    elseif diff(i) < (-pi)
        diff(i)=diff(i)+2*pi;
    end
end

% Joint angle change at every time step.
dq=diff/n;

% Initialization:
qo=qi;
tt=0;

while tt <= (ts-dt)
    % Starting a timer:
    tic;

    % Calculating the new Joint Angles:
    qn=qo+dq;

    % Updating the physical Arm:
    if arm==1
```

Appendix B. (Continued)

```

        ddt=ddt+dt;
        if ddt>=0.5 || tt>=(ts-dt)
            WMRA_ARM_Motion(2, 1, [qn;0;0;0], ddt);
            ddt=0;
        end
    end

    % Updating Virtual Reality Animation:
    if vr==1
        WMRA_VR_Animation(2, Tiwc, [qn;0;0]);
    end

    % Updating Matlab Animation:
    if ml==1
        % Calculating the new Transformation Matrix:

T1a=WMRA_rotx(DH(1,1))*WMRA_transl(DH(1,2),0,0)*WMRA_rotz(qn(1))*WMRA_transl(0,0,DH(1,3))
;

T2a=WMRA_rotx(DH(2,1))*WMRA_transl(DH(2,2),0,0)*WMRA_rotz(qn(2))*WMRA_transl(0,0,DH(2,3))
;

T3a=WMRA_rotx(DH(3,1))*WMRA_transl(DH(3,2),0,0)*WMRA_rotz(qn(3))*WMRA_transl(0,0,DH(3,3))
;

T4a=WMRA_rotx(DH(4,1))*WMRA_transl(DH(4,2),0,0)*WMRA_rotz(qn(4))*WMRA_transl(0,0,DH(4,3))
;

T5a=WMRA_rotx(DH(5,1))*WMRA_transl(DH(5,2),0,0)*WMRA_rotz(qn(5))*WMRA_transl(0,0,DH(5,3))
;

T6a=WMRA_rotx(DH(6,1))*WMRA_transl(DH(6,2),0,0)*WMRA_rotz(qn(6))*WMRA_transl(0,0,DH(6,3))
;

T7a=WMRA_rotx(DH(7,1))*WMRA_transl(DH(7,2),0,0)*WMRA_rotz(qn(7))*WMRA_transl(0,0,DH(7,3))
;
        WMRA_ML_Animation(2, Ti, Td, Tiwc, T1a, T2a, T3a, T4a, T5a, T6a, T7a);
    end

    % Updating the old values with the new values for the next iteration:
    qo=qn;
    tt=tt+dt;

    % Pausing for the speed sync:
    pause(dt-toc);

end

```

```

% This "new USF WMRA" function SIMULATES the arm going from the ready position to the
parking position with ANIMATION. All angles are in Radians.
% The parking position is assumed to be qi=[0;pi/2;0;pi;0;0;0] (Radians),
% the ready position is assumed to be qd=[pi/2;pi/2;0;pi/2;pi/2;pi/2;0] (Radians).
% ini=1 --> initialize animation figures, ini=2 or any --> just update the figures, ini=3
--> close the figures.

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

% Function Declaration:
function WMRA_ready2park(ini, vr, ml, arm, Tiwc, qiwc)

```

Appendix B. (Continued)

```
% Closing the Arm library and Matlab Graphics Animation and Virtual Reality Animation and
Plots windows:
if ini==3
    if arm==1
        try
            WMRA_ARM_Motion(ini, 0, 0, 0);
        end
    end
    if vr==1
        try
            WMRA_VR_Animation(ini, 0, 0);
        end
    end
    if ml==1
        try
            WMRA_ML_Animation(ini, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
        end
    end
    return;
end

% Defining the used conditions:
qi=[pi/2;pi/2;0;pi/2;pi/2;pi/2;0]; % Initial joint angles (Ready Position).
qd=[0;pi/2;0;pi/2;0;0]; % Final joint angles (Parking Position).
ts=10; % (5 or 10 or 20) Simulation time to move the arm from the ready position to
the parking position.
n=100; % Number of time steps.
dt=ts/n; % The time step to move the arm from the parking position to the ready
position.
dq=(qd-qi)/(0.5*n+5); % Joint angle change at every time step.

% Initializing the physical Arm:
if arm==1
    WMRA_ARM_Motion(ini, 2, [qi;qiw;0], dt);
    ddt=0;
end

% Initializing Virtual Reality Animation:
if vr==1
    WMRA_VR_Animation(ini, Tiwc, [qi;qiw]);
end

% Initializing Robot Animation in Matlab Graphics:
if ml==1
    % Inputting the D-H Parameters in a Matrix form:
    DH=WMRA_DH(qi);

    % Calculating the transformation matrices of each link:

T01=WMRA_rotx(DH(1,1))*WMRA_transl(DH(1,2),0,0)*WMRA_rotz(DH(1,4))*WMRA_transl(0,0,DH(1,3));
T12=WMRA_rotx(DH(2,1))*WMRA_transl(DH(2,2),0,0)*WMRA_rotz(DH(2,4))*WMRA_transl(0,0,DH(2,3));
T23=WMRA_rotx(DH(3,1))*WMRA_transl(DH(3,2),0,0)*WMRA_rotz(DH(3,4))*WMRA_transl(0,0,DH(3,3));
T34=WMRA_rotx(DH(4,1))*WMRA_transl(DH(4,2),0,0)*WMRA_rotz(DH(4,4))*WMRA_transl(0,0,DH(4,3));
T45=WMRA_rotx(DH(5,1))*WMRA_transl(DH(5,2),0,0)*WMRA_rotz(DH(5,4))*WMRA_transl(0,0,DH(5,3));
T56=WMRA_rotx(DH(6,1))*WMRA_transl(DH(6,2),0,0)*WMRA_rotz(DH(6,4))*WMRA_transl(0,0,DH(6,3));
```

Appendix B. (Continued)

```
T67=WMRA_rotx(DH(7,1))*WMRA_transl(DH(7,2),0,0)*WMRA_rotz(DH(7,4))*WMRA_transl(0,0,DH(7,3));
);
% Calculating the Transformation Matrix of the initial and desired arm positions:
Ti=Tiwc*T01*T12*T23*T34*T45*T56*T67;
Td=Tiwc*WMRA_q2T(qd);
WMRA_ML_Animation(ini, Ti, Td, Tiwc, T01, T12, T23, T34, T45, T56, T67);
end

% Initialization:
qo=qi;
tt=0;

while tt <= (ts)
% Starting a timer:
tic;

% Calculating the new Joint Angles:
if tt==0
qn=qo;
elseif tt < (dt*(0.5*n-5))
qn(2:7)=qo(2:7)+dq(2:7);
elseif tt < (dt*(0.5*n+5))
qn=qo+dq;
elseif tt < (dt*(n-1))
qn(1)=qo(1)+dq(1);
end

% Updating the physical Arm:
if arm==1
ddt=ddt+dt;
if ddt>=0.5 || tt>=(ts)
WMRA_ARM_Motion(2, 1, [qn;qiwc;0], ddt);
ddt=0;
end
end

% Updating Virtual Reality Animation:
if vr==1
WMRA_VR_Animation(2, Tiwc, [qn;qiwc]);
end

% Updating Matlab Animation:
if ml==1
% Calculating the new Transformation Matrix:

T1a=WMRA_rotx(DH(1,1))*WMRA_transl(DH(1,2),0,0)*WMRA_rotz(qn(1))*WMRA_transl(0,0,DH(1,3));
;

T2a=WMRA_rotx(DH(2,1))*WMRA_transl(DH(2,2),0,0)*WMRA_rotz(qn(2))*WMRA_transl(0,0,DH(2,3));
;

T3a=WMRA_rotx(DH(3,1))*WMRA_transl(DH(3,2),0,0)*WMRA_rotz(qn(3))*WMRA_transl(0,0,DH(3,3));
;

T4a=WMRA_rotx(DH(4,1))*WMRA_transl(DH(4,2),0,0)*WMRA_rotz(qn(4))*WMRA_transl(0,0,DH(4,3));
;

T5a=WMRA_rotx(DH(5,1))*WMRA_transl(DH(5,2),0,0)*WMRA_rotz(qn(5))*WMRA_transl(0,0,DH(5,3));
;

T6a=WMRA_rotx(DH(6,1))*WMRA_transl(DH(6,2),0,0)*WMRA_rotz(qn(6))*WMRA_transl(0,0,DH(6,3));
;

T7a=WMRA_rotx(DH(7,1))*WMRA_transl(DH(7,2),0,0)*WMRA_rotz(qn(7))*WMRA_transl(0,0,DH(7,3));
;
WMRA_ML_Animation(2, Ti, Td, Tiwc, T1a, T2a, T3a, T4a, T5a, T6a, T7a);
```

Appendix B. (Continued)

```
end

% Updating the old values with the new values for the next iteration:
qo=qn;
tt=tt+dt;

% Pausing for the speed sync:
pause(dt-toc);

end
```

```
% This function gives the homogeneous transformation matrix, given the rotation angle
about the X axis.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [T]=WMRA_rotx(t)

c=cos(t);
s=sin(t);
T=[1 0 0 0 ; 0 c -s 0 ; 0 s c 0 ; 0 0 0 1];
```

```
% This function gives the homogeneous transformation matrix, given the rotation angle
about the Z axis.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [T]=WMRA_roty(t)

c=cos(t);
s=sin(t);
T=[c 0 s 0 ; 0 1 0 0 ; -s 0 c 0 ; 0 0 0 1];
```

```
% This function gives the homogeneous transformation matrix, given the rotation angle
about the Z axis.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [T]=WMRA_rotz(t)

c=cos(t);
s=sin(t);
T=[c -s 0 0 ; s c 0 0 ; 0 0 1 0 ; 0 0 0 1];
```

Appendix B. (Continued)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Thanks to Mayur Palankar %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = WMRA_screen(varargin)
% WMRA_SCREEN M-file for WMRA_screen.fig
%   WMRA_SCREEN, by itself, creates a new WMRA_SCREEN or raises the existing
%   singleton*.
%
%   H = WMRA_SCREEN returns the handle to a new WMRA_SCREEN or the
%   handle to
%   the existing singleton*.
%
%   WMRA_SCREEN('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in WMRA_SCREEN.M with the given input arguments.
%
%   WMRA_SCREEN('Property','Value',...) creates a new WMRA_SCREEN or raises the
%   existing singleton*. Starting from the left, property value pairs are
%   applied to the GUI before WMRA_screen_OpeningFunction gets called. An
%   unrecognized property name or invalid value makes property application
%   stop. All inputs are passed to WMRA_screen_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help WMRA_screen

% Last Modified by GUIDE v2.5 04-Mar-2007 20:56:51

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @WMRA_screen_OpeningFcn, ...
                  'gui_OutputFcn',  @WMRA_screen_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before WMRA_screen is made visible.
function WMRA_screen_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to WMRA_screen (see VARARGIN)

% Choose default command line output for WMRA_screen
handles.output = hObject;
```

Appendix B. (Continued)

```
% Update handles structure
guidata(hObject, handles);

global VAR_DX
global VAR_SCREENOPN

VAR_DX=[0;0;0;0;0;0;0];
VAR_SCREENOPN = 1;

if (varargin{1} == '1')
    set(handles.pushbutton1,'Enable','on');
else
    set(handles.pushbutton1,'Enable','off');
end

set(handles.togglebutton2,'cdata',button11);
set(handles.togglebutton3,'cdata',button12);
set(handles.togglebutton8,'cdata',button13);
set(handles.togglebutton9,'cdata',button14);
set(handles.togglebutton15,'cdata',button15);
set(handles.togglebutton4,'cdata',button21);
set(handles.togglebutton5,'cdata',button22);
set(handles.togglebutton10,'cdata',button23);
set(handles.togglebutton11,'cdata',button24);
set(handles.togglebutton16,'cdata',button25);
set(handles.togglebutton6,'cdata',button31);
set(handles.togglebutton7,'cdata',button32);
set(handles.pushbutton37,'cdata',button33);
set(handles.togglebutton13,'cdata',button34);
set(handles.togglebutton14,'cdata',button35);

% UIWAIT makes WMRA_screen wait for user response (see UIRESUME)
% uiwait(handles.figure1);

function play = button11
play = iconize(imread('11.jpg'));
function play = button12
play = iconize(imread('12.jpg'));
function play = button13
play = iconize(imread('13.jpg'));
function play = button14
play = iconize(imread('14.jpg'));
function play = button15
play = iconize(imread('15.jpg'));
function play = button21
play = iconize(imread('21.jpg'));
function play = button22
play = iconize(imread('22.jpg'));
function play = button23
play = iconize(imread('23.jpg'));
function play = button24
play = iconize(imread('24.jpg'));
function play = button25
play = iconize(imread('25.jpg'));
function play = button31
play = iconize(imread('31.jpg'));
function play = button32
play = iconize(imread('32.jpg'));
function play = button33
play = iconize(imread('33.jpg'));
function play = button34
play = iconize(imread('34.jpg'));
function play = button35
play = iconize(imread('35.jpg'));

function out = iconize(a)
```

Appendix B. (Continued)

```
[r,c,d] = size(a);
r_skip = ceil(r/70);
c_skip = ceil(c/70);
out = a(1:r_skip:end,1:c_skip:end,:);

% --- Outputs from this function are returned to the command line.
function varargout = WMRA_screen_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
WMRA_Main_GUI;

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global VAR_SCREENOPN
VAR_SCREENOPN = 0;
close;

% --- Executes on button press in togglebutton2.
function togglebutton2_Callback(hObject, eventdata, handles)
% hObject handle to togglebutton2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global VAR_DX
% 1 1
if (get(hObject,'Value') == get(hObject,'Max'))
    set(hObject, 'BackgroundColor', 'red');
    set(handles.togglebutton6,'Enable','off');
    VAR_DX(3) = 1;
elseif (get(hObject,'Value') == get(hObject,'Min'))
    set(hObject, 'BackgroundColor', 'white');
    set(handles.togglebutton6,'Enable','on');
    VAR_DX(3) = 0;
end
% Hint: get(hObject,'Value') returns toggle state of togglebutton2

% --- Executes on button press in togglebutton3.
function togglebutton3_Callback(hObject, eventdata, handles)
% hObject handle to togglebutton3 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global VAR_DX
% 1 2
if (get(hObject,'Value') == get(hObject,'Max'))
    set(hObject, 'BackgroundColor', 'red');
    set(handles.togglebutton7,'Enable','off');
    VAR_DX(1) = 1;
elseif (get(hObject,'Value') == get(hObject,'Min'))
    set(hObject, 'BackgroundColor', 'white');
    set(handles.togglebutton7,'Enable','on');
    VAR_DX(1) = 0;
end
% Hint: get(hObject,'Value') returns toggle state of togglebutton3
```


Appendix B. (Continued)

```
% --- Executes on button press in togglebutton8.
function togglebutton8_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VAR_DX
% 1 3
if (get(hObject,'Value') == get(hObject,'Max'))
    set(hObject, 'BackgroundColor', 'red');
    set(handles.togglebutton15,'Enable','off');
    VAR_DX(6) = 0.003;
elseif (get(hObject,'Value') == get(hObject,'Min'))
    set(hObject, 'BackgroundColor', 'white');
    set(handles.togglebutton15,'Enable','on');
    VAR_DX(6) = 0;
end
% Hint: get(hObject,'Value') returns toggle state of togglebutton8

% --- Executes on button press in togglebutton9.
function togglebutton9_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VAR_DX
% 1 4
if (get(hObject,'Value') == get(hObject,'Max'))
    set(hObject, 'BackgroundColor', 'red');
    set(handles.togglebutton11,'Enable','off');
    VAR_DX(5) = 0.003;
elseif (get(hObject,'Value') == get(hObject,'Min'))
    set(hObject, 'BackgroundColor', 'white');
    set(handles.togglebutton11,'Enable','on');
    VAR_DX(5) = 0;
end
% Hint: get(hObject,'Value') returns toggle state of togglebutton9

% --- Executes on button press in togglebutton15.
function togglebutton15_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VAR_DX
% 1 5
if (get(hObject,'Value') == get(hObject,'Max'))
    set(hObject, 'BackgroundColor', 'red');
    set(handles.togglebutton8,'Enable','off');
    VAR_DX(6) = -0.003;
elseif (get(hObject,'Value') == get(hObject,'Min'))
    set(hObject, 'BackgroundColor', 'white');
    set(handles.togglebutton8,'Enable','on');
    VAR_DX(6) = 0;
end
% Hint: get(hObject,'Value') returns toggle state of togglebutton15

% --- Executes on button press in togglebutton4.
function togglebutton4_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VAR_DX
% 2 1
if (get(hObject,'Value') == get(hObject,'Max'))
    set(hObject, 'BackgroundColor', 'red');
    set(handles.togglebutton5,'Enable','off');
    VAR_DX(2) = 1;
elseif (get(hObject,'Value') == get(hObject,'Min'))
```

Appendix B. (Continued)

```
    set(hObject, 'BackgroundColor', 'white');
    set(handles.togglebutton5, 'Enable', 'on');
    VAR_DX(2) = 0;
end
% Hint: get(hObject, 'Value') returns toggle state of togglebutton4

% --- Executes on button press in togglebutton5.
function togglebutton5_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VAR_DX
% 2 2
if (get(hObject, 'Value') == get(hObject, 'Max'))
    set(hObject, 'BackgroundColor', 'red');
    set(handles.togglebutton4, 'Enable', 'off');
    VAR_DX(2) = -1;
elseif (get(hObject, 'Value') == get(hObject, 'Min'))
    set(hObject, 'BackgroundColor', 'white');
    set(handles.togglebutton4, 'Enable', 'on');
    VAR_DX(2) = 0;
end
% Hint: get(hObject, 'Value') returns toggle state of togglebutton5

% --- Executes on button press in togglebutton10.
function togglebutton10_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VAR_DX
% 2 3
if (get(hObject, 'Value') == get(hObject, 'Max'))
    set(hObject, 'BackgroundColor', 'red');
    set(handles.togglebutton16, 'Enable', 'off');
    VAR_DX(4) = -0.003;
elseif (get(hObject, 'Value') == get(hObject, 'Min'))
    set(hObject, 'BackgroundColor', 'white');
    set(handles.togglebutton16, 'Enable', 'on');
    VAR_DX(4) = 0;
end
% Hint: get(hObject, 'Value') returns toggle state of togglebutton10

% --- Executes on button press in togglebutton11.
function togglebutton11_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VAR_DX
% 2 4
if (get(hObject, 'Value') == get(hObject, 'Max'))
    set(hObject, 'BackgroundColor', 'red');
    set(handles.togglebutton9, 'Enable', 'off');
    VAR_DX(5) = -0.003;
elseif (get(hObject, 'Value') == get(hObject, 'Min'))
    set(hObject, 'BackgroundColor', 'white');
    set(handles.togglebutton9, 'Enable', 'on');
    VAR_DX(5) = 0;
end
% Hint: get(hObject, 'Value') returns toggle state of togglebutton11

% --- Executes on button press in togglebutton16.
function togglebutton16_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VAR_DX
```

Appendix B. (Continued)

```
% 2 5
if (get(hObject,'Value') == get(hObject,'Max'))
    set(hObject, 'BackgroundColor', 'red');
    set(handles.togglebutton10,'Enable','off');
    VAR_DX(4) = 0.003;
elseif (get(hObject,'Value') == get(hObject,'Min'))
    set(hObject, 'BackgroundColor', 'white');
    set(handles.togglebutton10,'Enable','on');
    VAR_DX(4) = 0;
end
% Hint: get(hObject,'Value') returns toggle state of togglebutton16

% --- Executes on button press in togglebutton6.
function togglebutton6_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VAR_DX
% 3 1
if (get(hObject,'Value') == get(hObject,'Max'))
    set(hObject, 'BackgroundColor', 'red');
    set(handles.togglebutton2,'Enable','off');
    VAR_DX(3) = -1;
elseif (get(hObject,'Value') == get(hObject,'Min'))
    set(hObject, 'BackgroundColor', 'white');
    set(handles.togglebutton2,'Enable','on');
    VAR_DX(3) = 0;
end
% Hint: get(hObject,'Value') returns toggle state of togglebutton6

% --- Executes on button press in togglebutton7.
function togglebutton7_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VAR_DX
% 3 2
if (get(hObject,'Value') == get(hObject,'Max'))
    set(hObject, 'BackgroundColor', 'red');
    set(handles.togglebutton3,'Enable','off');
    VAR_DX(1) = -1;
elseif (get(hObject,'Value') == get(hObject,'Min'))
    set(hObject, 'BackgroundColor', 'white');
    set(handles.togglebutton3,'Enable','on');
    VAR_DX(1) = 0;
end
% Hint: get(hObject,'Value') returns toggle state of togglebutton7

% --- Executes on button press in pushbutton37.
function pushbutton37_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton37 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VAR_DX
% 3 3
dx = [0;0;0;0;0;0;0];
% dx=dx(1:6);Redwan.
VAR_DX = dx;

set(handles.togglebutton2, 'BackgroundColor', 'white');
set(handles.togglebutton3, 'BackgroundColor', 'white');
set(handles.togglebutton8, 'BackgroundColor', 'white');
set(handles.togglebutton9, 'BackgroundColor', 'white');
set(handles.togglebutton15, 'BackgroundColor', 'white');

set(handles.togglebutton4, 'BackgroundColor', 'white');
```

Appendix B. (Continued)

```
set(handles.togglebutton5, 'BackgroundColor', 'white');
set(handles.togglebutton10, 'BackgroundColor', 'white');
set(handles.togglebutton11, 'BackgroundColor', 'white');
set(handles.togglebutton16, 'BackgroundColor', 'white');

set(handles.togglebutton6, 'BackgroundColor', 'white');
set(handles.togglebutton7, 'BackgroundColor', 'white');
set(handles.togglebutton13, 'BackgroundColor', 'white');
set(handles.togglebutton14, 'BackgroundColor', 'white');

set(handles.togglebutton2, 'Value', 0);
set(handles.togglebutton3, 'Value', 0);
set(handles.togglebutton8, 'Value', 0);
set(handles.togglebutton9, 'Value', 0);
set(handles.togglebutton15, 'Value', 0);

set(handles.togglebutton4, 'Value', 0);
set(handles.togglebutton5, 'Value', 0);
set(handles.togglebutton10, 'Value', 0);
set(handles.togglebutton11, 'Value', 0);
set(handles.togglebutton16, 'Value', 0);

set(handles.togglebutton6, 'Value', 0);
set(handles.togglebutton7, 'Value', 0);
set(handles.togglebutton13, 'Value', 0);
set(handles.togglebutton14, 'Value', 0);

set(handles.togglebutton2, 'Enable', 'on');
set(handles.togglebutton3, 'Enable', 'on');
set(handles.togglebutton8, 'Enable', 'on');
set(handles.togglebutton9, 'Enable', 'on');
set(handles.togglebutton15, 'Enable', 'on');

set(handles.togglebutton4, 'Enable', 'on');
set(handles.togglebutton5, 'Enable', 'on');
set(handles.togglebutton10, 'Enable', 'on');
set(handles.togglebutton11, 'Enable', 'on');
set(handles.togglebutton16, 'Enable', 'on');

set(handles.togglebutton6, 'Enable', 'on');
set(handles.togglebutton7, 'Enable', 'on');
set(handles.togglebutton13, 'Enable', 'on');
set(handles.togglebutton14, 'Enable', 'on');

% --- Executes on button press in togglebutton13.
function togglebutton13_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VAR_DX
% 3 4
if (get(hObject, 'Value') == get(hObject, 'Max'))
    set(hObject, 'BackgroundColor', 'red');
    set(handles.togglebutton14, 'Enable', 'off');
    VAR_DX(7) = 1;
elseif (get(hObject, 'Value') == get(hObject, 'Min'))
    set(hObject, 'BackgroundColor', 'white');
    set(handles.togglebutton14, 'Enable', 'on');
    VAR_DX(7) = 0;
end
% Hint: get(hObject, 'Value') returns toggle state of togglebutton13

% --- Executes on button press in togglebutton14.
function togglebutton14_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

Appendix B. (Continued)

```
% handles structure with handles and user data (see GUIDATA)
global VAR_DX
% 3 5
if (get(hObject,'Value') == get(hObject,'Max'))
    set(hObject, 'BackgroundColor', 'red');
    set(handles.togglebutton13, 'Enable', 'off');
    VAR_DX(7) = -1;
elseif (get(hObject,'Value') == get(hObject,'Min'))
    set(hObject, 'BackgroundColor', 'white');
    set(handles.togglebutton13, 'Enable', 'on');
    VAR_DX(7) = 0;
end
% Hint: get(hObject,'Value') returns toggle state of togglebutton14



---



% This function gives the Roll, Pitch, Taw angles, given the transformation matrix.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [rpy]=WMRA_T2rpy(T)

rpy = zeros(1,3);

% Making sure there is no singularity:
if abs(T(1,1))<eps && abs(T(2,1))<eps
    rpy(1)=0;
    rpy(2)=atan2(-T(3,1), T(1,1));
    rpy(3)=atan2(-T(2,3), T(2,2));
else
    rpy(1)=atan2(T(2,1), T(1,1));
    s=sin(rpy(1));
    c=cos(rpy(1));
    rpy(2)=atan2(-T(3,1), c*T(1,1)+s*T(2,1));
    rpy(3)=atan2(s*T(1,3)-c*T(2,3), c*T(2,2)-s*T(1,2));
end



---



% This function is for getting the transformations of the USF WMRA with 9 DOF.
% q is for the 7 joints in radians, and dq is for the wheelchair only in millimeters and
radians.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Function Declaration:
function [T, Ta, Twc, T1, T2, T3, T4, T5, T6, T7]=WMRA_Tall(i, q, dq, Twc)

% Declaring the global variables:
global DH

if i==1

    % Inputting the D-H Parameters in a Matrix form:
    DH=WMRA_DH(q);
```

Appendix B. (Continued)

```

    % Calculating the transformation matrices of each link:
T1=WMRA_rotx(DH(1,1))*WMRA_transl(DH(1,2),0,0)*WMRA_rotz(DH(1,4))*WMRA_transl(0,0,DH(1,3));
);
T2=WMRA_rotx(DH(2,1))*WMRA_transl(DH(2,2),0,0)*WMRA_rotz(DH(2,4))*WMRA_transl(0,0,DH(2,3));
);
T3=WMRA_rotx(DH(3,1))*WMRA_transl(DH(3,2),0,0)*WMRA_rotz(DH(3,4))*WMRA_transl(0,0,DH(3,3));
);
T4=WMRA_rotx(DH(4,1))*WMRA_transl(DH(4,2),0,0)*WMRA_rotz(DH(4,4))*WMRA_transl(0,0,DH(4,3));
);
T5=WMRA_rotx(DH(5,1))*WMRA_transl(DH(5,2),0,0)*WMRA_rotz(DH(5,4))*WMRA_transl(0,0,DH(5,3));
);
T6=WMRA_rotx(DH(6,1))*WMRA_transl(DH(6,2),0,0)*WMRA_rotz(DH(6,4))*WMRA_transl(0,0,DH(6,3));
);
T7=WMRA_rotx(DH(7,1))*WMRA_transl(DH(7,2),0,0)*WMRA_rotz(DH(7,4))*WMRA_transl(0,0,DH(7,3));
);

    % Calculating the Transformation Matrix of the initial arm position:
Ta=T1*T2*T3*T4*T5*T6*T7;

    % Calculating the Transformation Matrix of the initial WMRA system position:
T=Twc*Ta;
else
T1=WMRA_rotx(DH(1,1))*WMRA_transl(DH(1,2),0,0)*WMRA_rotz(q(1))*WMRA_transl(0,0,DH(1,3));
T2=WMRA_rotx(DH(2,1))*WMRA_transl(DH(2,2),0,0)*WMRA_rotz(q(2))*WMRA_transl(0,0,DH(2,3));
T3=WMRA_rotx(DH(3,1))*WMRA_transl(DH(3,2),0,0)*WMRA_rotz(q(3))*WMRA_transl(0,0,DH(3,3));
T4=WMRA_rotx(DH(4,1))*WMRA_transl(DH(4,2),0,0)*WMRA_rotz(q(4))*WMRA_transl(0,0,DH(4,3));
T5=WMRA_rotx(DH(5,1))*WMRA_transl(DH(5,2),0,0)*WMRA_rotz(q(5))*WMRA_transl(0,0,DH(5,3));
T6=WMRA_rotx(DH(6,1))*WMRA_transl(DH(6,2),0,0)*WMRA_rotz(q(6))*WMRA_transl(0,0,DH(6,3));
T7=WMRA_rotx(DH(7,1))*WMRA_transl(DH(7,2),0,0)*WMRA_rotz(q(7))*WMRA_transl(0,0,DH(7,3));
Ta=T1*T2*T3*T4*T5*T6*T7;
Twc=WMRA_w2T(1, Twc, dq);
T=Twc*Ta;
end

% This function finds the trajectory points along a streight line, given the initial and
final transformations. Single-angle rotation about a single axis is used
% See Eqs. 1.73-1.103 pages 30-32 of Richard Paul's book " Robot Manipulators"

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Function Declaration:
function [Tt] = WMRA_traj(ind, Ti, Td, n)

% Finding the rotation of the desired point based on the initial point:
R=Ti(1:3,1:3) '*Td(1:3,1:3);

```

Appendix B. (Continued)

```
% Initial single-angle representation of the rotation:
a=atan2(sqrt((R(3,2)-R(2,3))^2+(R(1,3)-R(3,1))^2+(R(2,1)-R(1,2))^2),
(R(1,1)+R(2,2)+R(3,3)-1));
s=sin(a);
c=cos(a);
v=1-c;
% Finding the single-vector components for the rotation:
if a<0.001
    kx=1;
    ky=0;
    kz=0;
elseif a<pi/2+0.001
    kx=(R(3,2)-R(2,3))/(2*s);
    ky=(R(1,3)-R(3,1))/(2*s);
    kz=(R(2,1)-R(1,2))/(2*s);
else
    kx=sign(R(3,2)-R(2,3))*sqrt((R(1,1)-c)/v);
    ky=sign(R(1,3)-R(3,1))*sqrt((R(2,2)-c)/v);
    kz=sign(R(2,1)-R(1,2))*sqrt((R(3,3)-c)/v);
    if kx>ky && kx>kz
        ky=(R(2,1)+R(1,2))/(2*kx*v);
        kz=(R(1,3)+R(3,1))/(2*kx*v);
    elseif ky>kx && ky>kz
        kx=(R(2,1)+R(1,2))/(2*ky*v);
        kz=(R(3,2)+R(2,3))/(2*ky*v);
    else
        kx=(R(1,3)+R(3,1))/(2*kz*v);
        ky=(R(3,2)+R(2,3))/(2*kz*v);
    end
end

% Running the desired trajectory method:
% 1 == Polynomial with Blending function,
% 2 == Polynomial without Blending function,
% 3 == Linear function.
if ind == 2
    at=WMRA_Polynomial(0,a,n);
    xt=WMRA_Polynomial(Ti(1,4), Td(1,4), n);
    yt=WMRA_Polynomial(Ti(2,4), Td(2,4), n);
    zt=WMRA_Polynomial(Ti(3,4), Td(3,4), n);
elseif ind == 3
    at=WMRA_Linear(0,a,n);
    xt=WMRA_Linear(Ti(1,4), Td(1,4), n);
    yt=WMRA_Linear(Ti(2,4), Td(2,4), n);
    zt=WMRA_Linear(Ti(3,4), Td(3,4), n);
else
    at=WMRA_BPolynomial(0,a,n);
    xt=WMRA_BPolynomial(Ti(1,4), Td(1,4), n);
    yt=WMRA_BPolynomial(Ti(2,4), Td(2,4), n);
    zt=WMRA_BPolynomial(Ti(3,4), Td(3,4), n);
end

Tt(:, :, 1)=Ti;

for i=2:n
    % Single-angle Change:
    da=at(i)-at(1);

    s=sin(da);
    c=cos(da);
    v=1-c;

    % Rotation and Position Change:
    dR=[kx^2*v+c kx*ky*v-kz*s kx*kz*v+ky*s;
        kx*ky*v+kz*s ky^2*v+c ky*kz*v-kx*s;
        kx*kz*v-ky*s ky*kz*v+kx*s kz^2*v+c];
```

Appendix B. (Continued)

```
% Finding the trajectory points along the trajectory line:
    Tt(:, :, i)=[Ti(1:3,1:3)*dR [xt(i);yt(i);zt(i)] ; 0 0 0 1];
end

%
% % Rotational Trajectory:
% % Single-angle Change:
% da=2*pi/(n-1);
% kx=1; ky=0; kz=0;
% s=sin(da);
% c=cos(da);
% v=1-c;
%
% % Rotation and Position Change:
% dR=[kx^2*v+c kx*ky*v-kz*s kx*kz*v+ky*s;
%     kx*ky*v+kz*s ky^2*v+c ky*kz*v-kx*s;
%     kx*kz*v-ky*s ky*kz*v+kx*s kz^2*v+c];
%
% % Finding the trajectory points along the trajectory line:
% Tt(:, :, 1)=Ti;
% for i=2:n
%     x=Ti(1,4)+2000*cos((i-1)*da);
%     y=Ti(2,4)+2000*sin((i-1)*da);
%     Tt(:, :, i)=[Ti(1:3,1:3)*(dR)^(i-1) [x;y;Ti(3,4)] ; 0 0 0 1];
% end
```

% This function gives the homogeneous transformation matrix, given the X, Y, Z cartesian translation values.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [T]=WMRA_transl(x, y, z)
```

```
T=[eye(3) [x;y;z]; 0 0 0 1];
```

% This function does the animation of USF WMRA with 9 DOF using Virtual Reality Toolbox.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

% Function Declaration:

```
function WMRA_VR_Animation(i, Twc, q)
```

% Declaring the global variables:

```
global L WMRA
```

% The initialization of the animation plot:

```
if i==1
```

```
    % Reading the Wheelchair's constant dimentions, all dimentions are converted in millimeters:
```

```
    L=WMRA_WCD;
```

```
    % Opening the WMRA file:
```


Appendix B. (Continued)

```
WMRA = vrworld('\9_wmra.wrl');
open(WMRA);
% Changing the View Point of the simulation:
WMRA.DynamicView.translation=[Twc(1,4) 0 -Twc(2,4)];
% Calculating the wheelaxle transform instead of the arm base transform:
Twc=Twc*[eye(3) -L(2:4) ; 0 0 0 1];
% The orientation about Z of the wheelchair:
phi=q(9);
% Calculating wheelchair's wheels' angles:
ql=q(8)/L(5)-L(1)*q(9)/(2*L(5));
qr=q(8)/L(5)+L(1)*q(9)/(2*L(5));
% Updating the VRML file for the new angles and distances:
WMRA.Chair.translation=[Twc(1,4) Twc(2,4) L(5)];
WMRA.Chair.rotation=[0 0 1 phi];
WMRA.LWheel.rotation=[0 1 0 ql];
WMRA.RWheel.rotation=[0 1 0 qr];
WMRA.ARM2.rotation=[0 0 -1 q(1)];
WMRA.ARM3.rotation=[0 1 0 q(2)];
WMRA.ARM4.rotation=[0 0 -1 q(3)];
WMRA.ARM5.rotation=[0 1 0 q(4)];
WMRA.ARM6.rotation=[0 0 -1 q(5)];
WMRA.ARM7.rotation=[0 1 0 q(6)];
WMRA.ARM8.rotation=[0 0 -1 q(7)];
% Viewing the simulation:
view(WMRA);

% Closing the animation plot:
elseif i==3
    close(WMRA);
    delete(WMRA);

% Updating the animation plot:
else
    WMRA.DynamicView.translation=[Twc(1,4) 0 -Twc(2,4)];
    Twc=Twc*[eye(3) -L(2:4) ; 0 0 0 1];
    phi=q(9);
    ql=q(8)/L(5)-L(1)*q(9)/(2*L(5));
    qr=q(8)/L(5)+L(1)*q(9)/(2*L(5));
    WMRA.Chair.translation=[Twc(1,4) Twc(2,4) L(5)];
    WMRA.Chair.rotation=[0 0 1 phi];
    WMRA.LWheel.rotation=[0 1 0 ql];
    WMRA.RWheel.rotation=[0 1 0 qr];
    WMRA.ARM2.rotation=[0 0 -1 q(1)];
    WMRA.ARM3.rotation=[0 1 0 q(2)];
    WMRA.ARM4.rotation=[0 0 -1 q(3)];
    WMRA.ARM5.rotation=[0 1 0 q(4)];
    WMRA.ARM6.rotation=[0 0 -1 q(5)];
    WMRA.ARM7.rotation=[0 1 0 q(6)];
    WMRA.ARM8.rotation=[0 0 -1 q(7)];
end
```

% This function gives the Transformation Matrix of the wheelchair with 2 DOF (Ground to WMRA base), given the previous transformation matrix and the required wheelchair's travel distance and angle.

% Dimentionations are as supplies, angles are in radians.

%%
%% COPY RIGHTS RESERVED %%%

%% Developed By: Redwan M. Alqasemi %%%

%% April 2007 %%%
%%

Appendix B. (Continued)

```

% Function Declaration:
function [T]=WMRA_w2T(ind, Tp, q)

% Reading the Wheelchair's constant dimentions, all dimentions are converted in
millimeters:
L=WMRA_WCD;

% Deciding if the motion is in reference to the arm base (1) or the wheel axle center
(0):
if ind == 0, L(2:4)=[0;0;0]; end

% Defining the inverse of Transformation Matrix between the wheelchair center and the
WMRA's base:
Twa=[eye(3) L(2:4) ; 0 0 0 1];

% The previous transformation matrix from the ground to the wheelchair center:
Tp=Tp*inv(Twa);

% Defining the Transformation Matrix between the ground and the wheelchair center and
WMRA's base:
if abs(q(2))<=eps % Streight line motion.
    Tp(1:2,4)=Tp(1:2,4)+q(1)*Tp(1:2,1);
    T=Tp*Twa;
else
    po=atan2(Tp(2,1),Tp(1,1));
    p=q(2);
    r=q(1)/p-L(1)/2;
    Tgw=[cos(po+p) -sin(po+p) 0 Tp(1,4)+sin(pi/2+po+p/2)*(r+L(1)/2)*sin(p)/cos(p/2) ;
sin(po+p) cos(po+p) 0 Tp(2,4)-cos(pi/2+po+p/2)*(r+L(1)/2)*sin(p)/cos(p/2) ; 0 0 1 Tp(3,4)
; 0 0 0 1];
    T=Tgw*Twa;
end

% This function gives the wheelchair dimentions matrix to be used in the program.
% Modifying the dimentons on this file is sufficient to change these dimention in all
related programs.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Function Declaration:
function L=WMRA_WCD()

L=[0;0;0;0;0];

% All dimentions are in millimeters.
L(1)=560; % Distance between the two driving wheels.
L(2)=440; % Horizontal distance between the wheels axix of rotation and the arm mounting
position (along x).
L(3)=230; % Horizontal distance between the middle point between the two driving wheels
and the arm mounting position (along y).
L(4)=182; % Vertical distance between the wheels axix of rotation and the arm mounting
position (along z).
L(5)=168; % Radius of the driving wheels.

```

Appendix B. (Continued)

B.3. Matlab Main Script and GUI Main File

```
% This "new USF WMRA" script SIMULATES the WMRA system with ANIMATION and plots for 9
DOF. All angles are in Radians.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Declaring the global variables to be used for the touch screen control:
global VAR_DX
global VAR_SCREENOPN
global dHo

% Defining used parameters:
d2r=pi/180; % Conversions from Degrees to Radians.
r2d=180/pi; % Conversions from Radians to Degrees.

% Reading the Wheelchair's constant dimentions, all dimentions are converted in
millimeters:
L=WMRA_WCD;

% User input prompts:

choice000 = input('\n Choose what to control: \n For combined Wheelchair and Arm control,
press "1", \n For Arm only control, press "2", \n For Wheelchair only control, press "3".
\n','s');
if choice000=='2'
    WCA=2;
    choice00000 = input('\n Choose whose frame to base the control on: \n For Ground
Frame, press "1", \n For Wheelchair Frame, press "2", \n For Gripper Frame, press "3".
\n','s');
    if choice00000=='2'
        coord=2;
    elseif choice00000=='3'
        coord=3;
    else
        coord=1;
    end
    choice0000 = input('\n Choose the cartesian coordinates to be controlled: \n For
Position and Orientation, press "1", \n For Position only, press "2". \n','s');
    if choice0000=='2'
        cart=2;
    else
        cart=1;
    end
    choice5 = input('\n Please enter the desired optimization method: (1= SR-I & WLN, 2=
P-I & WLN, 3= SR-I & ENE, 4= P-I & ENE) \n','s');
    if choice5=='2'
        optim=2;
    elseif choice5=='3'
        optim=3;
    elseif choice5=='4'
        optim=4;
    else
        optim=1;
    end
    choice50 = input('\n Do you want to include Joint Limit Avoidance? (1= Yes, 2= No)
\n','s');
    if choice50=='2'
        JLA=0;
```

Appendix B. (Continued)

```
    else
        JLA=1;
    end
    choice500 = input('\n Do you want to include Joint Limit/Velocity and Obstacle Safety
Stop? (1= Yes, 2= No) \n','s');
    if choice500=='2'
        JLO=0;
    else
        JLO=1;
    end
elseif choice000=='3'
    WCA=3;
    choice00000 = input('\n Choose whose frame to base the control on: \n For Ground
Frame, press "1", \n For Wheelchair Frame, press "2". \n','s');
    if choice00000=='2'
        coord=2;
    else
        coord=1;
    end
    choice500 = input('\n Do you want to include Joint Velocity Safety Stop? (1= Yes, 2=
No) \n','s');
    if choice500=='2'
        JLO=0;
    else
        JLO=1;
    end
    cart=2;
    optim=0;
    JLA=0;
else
    WCA=1;
    choice00000 = input('\n Choose whose frame to base the control on: \n For Ground
Frame, press "1", \n For Wheelchair Frame, press "2", \n For Gripper Frame, press "3".
\n','s');
    if choice00000=='2'
        coord=2;
    elseif choice00000=='3'
        coord=3;
    else
        coord=1;
    end
    choice0000 = input('\n Choose the cartesian coordinates to be controlled: \n For
Position and Orientation, press "1", \n For Position only, press "2". \n','s');
    if choice0000=='2'
        cart=2;
    else
        cart=1;
    end
    choice5 = input('\n Please enter the desired optimization method: (1= SR-I & WLN, 2=
P-I & WLN, 3= SR-I & ENE, 4= P-I & ENE) \n','s');
    if choice5=='2'
        optim=2;
    elseif choice5=='3'
        optim=3;
    elseif choice5=='4'
        optim=4;
    else
        optim=1;
    end
    choice50 = input('\n Do you want to include Joint Limit Avoidance? (1= Yes, 2= No)
\n','s');
    if choice50=='2'
        JLA=0;
    else
        JLA=1;
    end
end
```

Appendix B. (Continued)

```
choice500 = input('\n Do you want to include Joint Limit/Velocity and Obstacle Safety
Stop? (1= Yes, 2= No) \n','s');
if choice500=='2'
    JLO=0;
else
    JLO=1;
end
end

choice0 = input('\n Choose the control mode: \n For position control, press "1", \n For
velocity control, press "2", \n For SpaceBall control, press "3", \n For Psychology Mask
control, press "4", \n For Touch Screen control, press "5". \n','s');
if choice0=='1'
    cont=1;
    Td = input('\n Please enter the transformation matrix of the desired position and
orientation from the control-based frame \n (e.g. [0 0 1 1455;-1 0 0 369;0 -1 0 999; 0 0
0 1]) \n');
    v = input('\n Please enter the desired linear velocity of the gripper in mm/s (e.g.
50) \n');
    choice00 = input('\n Chose the Trajectory generation function: \n Press "1" for a
Polynomial function with Blending, or \n press "2" for a Polynomial function without
Blending, or \n press "3" for a Linear function. \n','s');
    if choice00=='2'
        trajf=2;
    elseif choice00=='3'
        trajf=3;
    else
        trajf=1;
    end
elseif choice0=='2'
    cont=2;
    ts = input('\n Please enter the desired simulation time in seconds (e.g. 2) \n');
    if cart==2
        Vd = input('\n Please enter the desired 3x1 cartesian velocity vector of the
grripper (in mm/s) (e.g. [70;70;-70]) \n');
    else
        Vd = input('\n Please enter the desired 6x1 cartesian velocity vector of the
grripper (in mm/s, radians/s) (e.g. [70;70;-70;0.001;0.001;0.001]) \n');
    end
elseif choice0=='3'
    cont=3;
    % Space Ball will be used for control.
    v = input('\n Please enter the desired linear velocity of the gripper in mm/s (e.g.
50) \n');
elseif choice0=='4'
    cont=4;
    % BCI 2000 Psychology Mask will be used for control.
    v = input('\n Please enter the desired linear velocity of the gripper in mm/s (e.g.
50) \n');
    port1 = input('\n Please enter the desired port number (e.g. 19711) \n');
else
    cont=5;
    % Touch Screen will be used for control.
    v = input('\n Please enter the desired linear velocity of the gripper in mm/s (e.g.
50) \n');
end

choice1 = input('\n Choose animation type or no animation: \n For Virtual Reality
Animation, press "1", \n For Matlab Graphics Animation, press "2", \n For BOTH
Animations, press "3", \n For NO Animation, press "4". \n','s');
if choice1=='2'
    vr = 0; ml = 1;
elseif choice1=='3'
    vr = 1; ml = 1;
elseif choice1=='4'
    vr = 0; ml = 0;
```

Appendix B. (Continued)

```
else
    vr = 1; ml = 0;
end

choice10 = input('\n Would you like to run the actual WMRA? \n For yes, press "1", \n For
no, press "2". \n', 's');
if choice10=='1'
    arm=1;
else
    arm=0;
end
choice2 = input('\n Press "1" if you want to start at the "ready" position, \n or press
"2" if you want to enter the initial joint angles. \n', 's');
if choice2=='2'
    qi = input('\n Please enter the arms initial angles vector in radians (e.g.
[pi/2;pi/2;0;pi/2;pi/2;pi/2;0]) \n');
    WCi = input('\n Please enter the initial x,y position and z orientation of the WMRA
base from the ground base in millimeters and radians (e.g. [200;500;0.3]) \n');
    ini=0;
else
    qi=[90;90;0;90;90;90;0]*d2r;
    WCi=[0;0;0];
    ini=0;
    if vr==1 || ml==1 || arm==1
        choice3 = input('\n Press "1" if you want to include "park" to "ready" motion, \n
or press "2" if not. \n', 's');
        if choice3=='2'
            ini=0;
        else
            ini=1;
        end
    end
end
choice4 = input('\n Press "1" if you do NOT want to plot the simulation results, \n or
press "2" if do. \n', 's');
if choice4=='2'
    plt=2;
else
    plt=1;
end

% Calculating the Transformation Matrix of the initial position of the WMRA's base:
Tiwc=WMRA_p2T(WCi(1),WCi(2),WCi(3));

% Calculating the initial Wheelchair Variables:
qiwc=[sqrt(WCi(1)^2+WCi(2)^2);WCi(3)];

% Calculating the initial transformation matrices:
[Ti, Tia, Tiwc, T01, T12, T23, T34, T45, T56, T67]=WMRA_Tall(1, qi, [0;0], Tiwc);

if cont==1
    % Calculating the linear distance from the initial position to the desired position
and the linear velocity:
    if coord==2
        D=sqrt( (Td(1,4)-Tia(1,4))^2 + (Td(2,4)-Tia(2,4))^2 + (Td(3,4)-Tia(3,4))^2);
    elseif coord==3
        D=sqrt( (Td(1,4))^2 + (Td(2,4))^2 + (Td(3,4))^2);
    else
        D=sqrt( (Td(1,4)-Ti(1,4))^2 + (Td(2,4)-Ti(2,4))^2 + (Td(3,4)-Ti(3,4))^2);
    end
    % Calculating the number of iteration and the time increment (delta t) if the linear
step increment of the tip is 1 mm:
    dt=0.05; % Time increment in seconds.
    total_time=D/v; % Total time of animation.
    n=round(total_time/dt); % Number of iterations rounded up.
    dt=total_time/n; % Adjusted time increment in seconds.
end
```

Appendix B. (Continued)

```
% Calculating the Trajectory of the end effector, and once the trajectory is
calculated, we should redefine "Td" based on the ground frame:
if coord==2
    Tt=WMRA_traj(trajf, Tia, Td, n+1);
    Td=Tiwc*Td;
elseif coord==3
    Tt=WMRA_traj(trajf, eye(4), Td, n+1);
    Td=Ti*Td;
else
    Tt=WMRA_traj(trajf, Ti, Td, n+1);
end
elseif cont==2
    % Calculating the number of iterations and the time increment (delta t) if the linear
step increment of the gripper is 1 mm:
    dt=0.05; % Time increment in seconds.
    total_time=ts; % Total time of animation.
    n=round(total_time/dt); % Number of iterations rounded up.
    dt=total_time/n; % Adjusted time increment in seconds.
    dx=Vd*dt;
    Td=Ti;
elseif cont==3
    WMRA_exit(); % This is to stop the simulation in SpaceBall control when the user
presses the exit key.
    dt=0.05;
    dx=v*dt*[spdata1(3)/20 ; -spdata1(1)/40 ; spdata1(2)/30 ; spdata1(6)/1500 ; -
spdata1(4)/900 ; spdata1(5)/1300];
    dg=spdata1(7);
    Td=Ti;
    n=1;
elseif cont==4
    WMRA_exit(); % This is to stop the simulation in Psychology Mask control when the
user presses the exit key.
    dt=0.05;
    dx=v*dt*WMRA_psy(port1);
    dg=dx(7);
    dx=dx(1:6);
    Td=Ti;
    n=1;
else
    WMRA_screen('0'); % This is to start the screen controls. Argument: '0'=BACK button
disabled, '1'=BACK button enabled.
    dt=0.05;
    dx=v*dt*VAR_DX(1:6);
    dg=VAR_DX(7);
    Td=Ti;
    n=1;
end

% Initializing the joint angles, the Transformation Matrix, and time:
dq=zeros(9,1);
dg=0;
qo=[qi;qiwc];
To=Ti;
Toa=Tia;
Towc=Tiwc;
tt=0;
i=1;
dHo=[0;0;0;0;0;0;0];

% Initializing the WMRA:
if ini==0 % When no "park" to "ready" motion required.
    % Initializing Virtual Reality Animation:
    if vr==1
        WMRA_VR_Animation(1, Towc, qo);
    end
    % Initializing Robot Animation in Matlab Graphics:
```

Appendix B. (Continued)

```
    if ml==1
        WMRA_ML_Animation(1, To, Td, Towc, T01, T12, T23, T34, T45, T56, T67);
    end
    % Initializing the Physical Arm:
    if arm==1
        WMRA_ARM_Motion(1, 2, [qo;dg], 0);
        ddt=0;
    end
elseif ini==1 && (vr==1 || ml==1 || arm==1) % When "park" to "ready" motion is required.
    WMRA_park2ready(1, vr, ml, arm, Towc, qo(8:9));
    if arm==1
        ddt=0;
    end
end

% Re-Drawing the Animation:
if vr==1 || ml==1
    drawnow;
end

% Starting a timer:
tic

% Starting the Iteration Loop:
while i<=(n+1)
    % Calculating the 6X7 Jacobian of the arm in frame 0:
    [Joa,detJoa]=WMRA_J07(T01, T12, T23, T34, T45, T56, T67);

    % Calculating the 6X2 Jacobian based on the WMRA's base in the ground frame:
    phi=atan2(Towc(2,1),Towc(1,1));
    Jowc=WMRA_Jga(1, phi, Toa(1:2,4));

    % Changing the Jacobian reference frame based on the choice of which coordinates
    frame are referenced in the Cartesian control:
    % coord=1 for Ground Coordinates Control.
    % coord=2 for Wheelchair Coordinates Control.
    % coord=3 for Gripper Coordinates Control.
    if coord==2
        Joa=Joa;
        Jowc=[Towc(1:3,1:3)' zeros(3); zeros(3) Towc(1:3,1:3)']*Jowc;
    elseif coord==3
        Joa=[Toa(1:3,1:3)' zeros(3); zeros(3) Toa(1:3,1:3)']*Joa;
        Jowc=[To(1:3,1:3)' zeros(3); zeros(3) To(1:3,1:3)']*Jowc;
    elseif coord==1
        Joa=[Towc(1:3,1:3) zeros(3); zeros(3) Towc(1:3,1:3)]*Joa;
        Jowc=Jowc;
    end

    % Calculating the 3X9 or 6X9 augmented Jacobian of the WMRA system based on the
    ground frame:
    if cart==2
        Joa=Joa(1:3,1:7);
        detJoa=sqrt(det(Joa*Joa'));
        Jowc=Jowc(1:3,1:2);
        Jo=[Joa Jowc];
        detJo=sqrt(det(Jo*Jo'));
    else
        Jo=[Joa Jowc];
        detJo=sqrt(det(Jo*Jo'));
    end

    % Finding the Cartesian errors of the end effector:
    if cont==1
        % Calculating the Position and Orientation errors of the end effector, and the
        rates of motion of the end effector:
        if coord==2
```


Appendix B. (Continued)

```

        invTowc=[Towc(1:3,1:3)' , -Towc(1:3,1:3)*Towc(1:3,4);0 0 0 1];
        Ttnew=invTowc*Tiwc*Tt(:, :, i);
        dx=WMRA_delta(Toa, Ttnew);
    elseif coord==3
        invTo=[To(1:3,1:3)' , -To(1:3,1:3)*To(1:3,4);0 0 0 1];
        Ttnew=invTo*Ti*Tt(:, :, i);
        dx=WMRA_delta(eye(4), Ttnew);
    else
        dx=WMRA_delta(To, Tt(:, :, i));
    end
elseif cont==2

elseif cont==3
    dx=v*dt*[spdata1(3)/20 ; -spdata1(1)/40 ; spdata1(2)/30 ; spdata1(6)/1500 ; -
spdata1(4)/900 ; spdata1(5)/1300];
    dg=spdata1(7);
elseif cont==4
    dx=v*dt*WMRA_psy(port1);
    dg=dx(7);
    dx=dx(1:6);
else
    dx=v*dt*VAR_DX(1:6);
    dg=VAR_DX(7);
end

% Changing the order of Cartesian motion in the case when gripper reference frame is
selected for control with the screen or psy or SpaceBall interfaces:
if coord==3 && cont>=3
    dx=[-dx(2);-dx(3);dx(1);-dx(5);-dx(6);dx(4)];
end

if cart==2
    dx=dx(1:3);
end

% Calculating the resolved rate with optimization:
% Index input values for "optim": 1= SR-I & WLN, 2= P-I & WLN, 3= SR-I & ENE, 4= P-I
& ENE:
if WCA==2
    dq=WMRA_Opt(optim, JLA, JLO, Joa, detJoa, dq(1:7), dx, dt, qo);
    dq=[dq;0;0];
elseif WCA==3
    dq=WMRA_Opt(optim, JLA, JLO, Jowc, 1, dq(8:9), dx(1:2), dt, 1);
    dq=[0;0;0;0;0;0;0;dq];
else
    dq=WMRA_Opt(optim, JLA, JLO, Jo, detJo, dq, dx, dt, qo);
end

% Calculating the new Joint Angles:
qn=qo+dq;

% Calculating the new Transformation Matrices:
[Tn, Tna, Tnwc, T01, T12, T23, T34, T45, T56, T67]=WMRA_Tall(2, qn, dq(8:9), Towc);

% A safety condition function to stop the joints that may cause colision of the arm
with itself, the wheelchair, or the human user:
if JLO==1 && WCA~=3
    dq(1:7)=WMRA_collide(dq(1:7), T01, T12, T23, T34, T45, T56, T67);
    % Re-calculating the new Joint Angles:
    qn=qo+dq;
    % Re-calculating the new Transformation Matrices:
    [Tn, Tna, Tnwc, T01, T12, T23, T34, T45, T56, T67]=WMRA_Tall(2, qn, dq(8:9),
Towc);
end

% Saving the plot data in case plots are required:

```

Appendix B. (Continued)

```
if plt==2
    WMRA_Plots(1, L, r2d, dt, i, tt, qn, dq, Tn, Tnwc, detJoa, detJo);
end

% Updating Physical Arm:
if arm==1
    ddt=ddt+dt;
    if ddt>=0.5 || i>=(n+1)
        WMRA_ARM_Motion(2, 1, [qn;dq], ddt);
        ddt=0;
    end
end

% Updating Virtual Reality Animation:
if vr==1
    WMRA_VR_Animation(2, Tnwc, qn);
end

% Updating Matlab Graphics Animation:
if ml==1
    WMRA_ML_Animation(2, Ti, Td, Tnwc, T01, T12, T23, T34, T45, T56, T67);
end

% Re-Drawing the Animation:
if vr==1 || ml==1
    drawnow;
end

% Updating the old values with the new values for the next iteration:
qo=qn;
To=Tn;
Toa=Tna;
Towc=Tnwc;
tt=tt+dt;
i=i+1;

% Stopping the simulation when the exit button is pressed:
if cont==3 || cont==4 || cont==5
    if (VAR_SCREENOPN == -1)
        n=n+1;
    else
        break
    end
end

% Delay to comply with the required speed:
if toc < tt
    pause(tt-toc);
end

end

% Reading the elapsed time and printing it with the simulation time:
if cont==1 || cont==2, fprintf('\nSimula. time is %6.6f seconds.\n' , total_time); end
toc

% Plotting:
if plt==2
    WMRA_Plots(2, L, r2d, dt, i, tt, qn, dq, Tn, Tnwc, detJoa, detJo);
end

if vr==1 || ml==1 || arm==1

    % Going back to the ready position:
    choice6 = input('\n Do you want to go back to the "ready" position? \n Press "1" for
Yes, or press "2" for No. \n', 's');
    if choice6=='1'
```

Appendix B. (Continued)

```
WMRA_any2ready(2, vr, ml, arm, Tnwc, qn);
% Going back to the parking position:
choice7 = input('\n Do you want to go back to the "parking" position? \n Press
"1" for Yes, or press "2" for No. \n', 's');
if choice7=='1'
    WMRA_ready2park(2, vr, ml, arm, Tnwc, qn(8:9));
end
end

% Closing the Arm library and Matlab Graphics Animation and Virtual Reality Animation
and Plots windows:
choice8 = input('\n Do you want to close all simulation windows and arm controls? \n
Press "1" for Yes, or press "2" for No. \n', 's');
if choice8=='1'
    if arm==1
        WMRA_ARM_Motion(3, 0, 0, 0);
    end
    if vr==1
        WMRA_VR_Animation(3, 0, 0);
    end
    if ml==1
        WMRA_ML_Animation(3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    end
    if plt==2
        close
    end
    (figure(2), figure(3), figure(4), figure(5), figure(6), figure(7), figure(8), figure(9), figure(1
0));
end
end

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Thanks to Mayur Palankar %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function varargout = WMRA_Main_GUI(varargin)
% WMRA_Main_GUI M-file for WMRA_Main_GUI.fig
% WMRA_Main_GUI, by itself, creates a new WMRA_Main_GUI or raises the existing
% singleton*.
%
% H = WMRA_Main_GUI returns the handle to a new WMRA_Main_GUI or the handle to
% the existing singleton*.
%
% WMRA_Main_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
% function named CALLBACK in WMRA_Main_GUI.M with the given input arguments.
%
% WMRA_Main_GUI('Property','Value',...) creates a new WMRA_Main_GUI or raises the
% existing singleton*. Starting from the left, property value pairs
% are
% applied to the GUI before WMRA_Main_GUI_OpeningFunction gets called. An
% unrecognized property name or invalid value makes property application
% stop. All inputs are passed to WMRA_Main_GUI_OpeningFcn via varargin.
%
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
% instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help WMRA_Main_GUI
```

Appendix B. (Continued)

```
% Last Modified by GUIDE v2.5 31-Mar-2007 16:02:05

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @WMRA_Main_GUI_OpeningFcn, ...
                  'gui_OutputFcn',  @WMRA_Main_GUI_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before WMRA_Main_GUI is made visible.
function WMRA_Main_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to WMRA_Main_GUI (see VARARGIN)

% Choose default command line output for WMRA_Main_GUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

global VAR_SCREENOPN
global VAR_LOOP
global VAR_WCI
global VAR_QI

VAR_WCI = [0; 0; 0];
VAR_QI = [1.5708; 1.5708; 0; 1.5708; 1.5708; 1.5708; 0];
VAR_SCREENOPN = 0;
VAR_LOOP = 0;

% UIWAIT makes WMRA_Main_GUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = WMRA_Main_GUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% -----
function file_menu_Callback(hObject, eventdata, handles)
% hObject    handle to file menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function open_menu_Callback(hObject, eventdata, handles)
```

Appendix B. (Continued)

```
% hObject    handle to open_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function save_menu_Callback(hObject, eventdata, handles)
% hObject    handle to save_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function saveas_menu_Callback(hObject, eventdata, handles)
% hObject    handle to saveas_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% -----
function exit_menu_Callback(hObject, eventdata, handles)
% hObject    handle to exit_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close;

% -----
function help_menu_Callback(hObject, eventdata, handles)
% hObject    handle to help_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

val = get(hObject, 'Value');
switch val
case 1
    if (get(handles.popupmenu17, 'Value') == 2)
        set(handles.edit65, 'Enable', 'on');
        set(handles.edit66, 'Enable', 'on');
        set(handles.edit67, 'Enable', 'on');
        set(handles.text22, 'Enable', 'on');
    end
    if (get(handles.popupmenu17, 'Value') == 1)
        if (strcmp(get(handles.edit36, 'Enable'), 'off'))
            set(handles.edit36, 'String', 0);
            set(handles.edit37, 'String', 0);
            set(handles.edit38, 'String', 1);
            set(handles.edit39, 'String', -1);
            set(handles.edit40, 'String', 0);
            set(handles.edit41, 'String', 0);
            set(handles.edit42, 'String', 0);
            set(handles.edit43, 'String', -1);
            set(handles.edit44, 'String', 0);

            set(handles.edit36, 'Enable', 'on');
            set(handles.edit37, 'Enable', 'on');
            set(handles.edit38, 'Enable', 'on');
            set(handles.edit39, 'Enable', 'on');
            set(handles.edit40, 'Enable', 'on');
            set(handles.edit41, 'Enable', 'on');
            set(handles.edit42, 'Enable', 'on');
            set(handles.edit43, 'Enable', 'on');
            set(handles.edit44, 'Enable', 'on');
        end
end
```

Appendix B. (Continued)

```
end
case 2
if (get(handles.popupmenu17, 'Value') == 2)
set(handles.edit65, 'Enable', 'off');
set(handles.edit66, 'Enable', 'off');
set(handles.edit67, 'Enable', 'off');
set(handles.text22, 'Enable', 'off');
end
if (get(handles.popupmenu17, 'Value') == 1)
set(handles.edit36, 'String', 1);
set(handles.edit37, 'String', 0);
set(handles.edit38, 'String', 0);
set(handles.edit39, 'String', 0);
set(handles.edit40, 'String', 1);
set(handles.edit41, 'String', 0);
set(handles.edit42, 'String', 0);
set(handles.edit43, 'String', 0);
set(handles.edit44, 'String', 1);

set(handles.edit36, 'Enable', 'off');
set(handles.edit37, 'Enable', 'off');
set(handles.edit38, 'Enable', 'off');
set(handles.edit39, 'Enable', 'off');
set(handles.edit40, 'Enable', 'off');
set(handles.edit41, 'Enable', 'off');
set(handles.edit42, 'Enable', 'off');
set(handles.edit43, 'Enable', 'off');
set(handles.edit44, 'Enable', 'off');
end
end

% Hints: contents = get(hObject, 'String') returns popupmenu1 contents as cell array
%         contents{get(hObject, 'Value')} returns selected item from popupmenu1

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject, 'BackgroundColor'),
get(0, 'defaultUicontrolBackgroundColor'))
set(hObject, 'BackgroundColor', 'white');
end

% --- Executes on selection change in popupmenu3.
function popupmenu3_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
val = get(hObject, 'Value');
switch val
case 1
if (strcmp(get(handles.popupmenu1, 'Enable'), 'off'))
set(handles.popupmenu8, 'Enable', 'on');
set(handles.checkbox1, 'Value', 1);
set(handles.checkbox1, 'Enable', 'on');

set(handles.popupmenu1, 'Enable', 'on');
string_list = {'Ground'; 'Wheelchair'; 'Gripper'};
set(handles.popupmenu21, 'String', string_list);
end
case 2
if (strcmp(get(handles.popupmenu1, 'Enable'), 'off'))
```

Appendix B. (Continued)

```
set(handles.popupmenu8,'Enable','on');
set(handles.checkbox1,'Value',1);
set(handles.checkbox1,'Enable','on');

set(handles.popupmenu1,'Enable','on');
string_list = {'Ground'; 'Wheelchair'; 'Gripper'};
set(handles.popupmenu21,'String',string_list);
end
case 3
if (get(handles.popupmenu21,'Value') == 3)
set(handles.popupmenu21,'Value',2);
end

string_list = get(handles.popupmenu21,'String');
set(handles.popupmenu21,'String',string_list(1:2));

set(handles.popupmenu8,'Enable','off');
set(handles.checkbox1,'Value',0);
set(handles.checkbox1,'Enable','off');

set(handles.popupmenu1,'Value',2);
set(handles.popupmenu1,'Enable','off');
if (get(handles.popupmenu17,'Value') == 2)
set(handles.edit65,'Enable','off');
set(handles.edit66,'Enable','off');
set(handles.edit67,'Enable','off');
set(handles.text22,'Enable','off');
end
if (get(handles.popupmenu17,'Value') == 1)
set(handles.edit36,'String',1);
set(handles.edit37,'String',0);
set(handles.edit38,'String',0);
set(handles.edit39,'String',0);
set(handles.edit40,'String',1);
set(handles.edit41,'String',0);
set(handles.edit42,'String',0);
set(handles.edit43,'String',0);
set(handles.edit44,'String',1);

set(handles.edit36,'Enable','off');
set(handles.edit37,'Enable','off');
set(handles.edit38,'Enable','off');
set(handles.edit39,'Enable','off');
set(handles.edit40,'Enable','off');
set(handles.edit41,'Enable','off');
set(handles.edit42,'Enable','off');
set(handles.edit43,'Enable','off');
set(handles.edit44,'Enable','off');
end
end
% Hints: contents = get(hObject,'String') returns popupmenu3 contents as cell array
%         contents{get(hObject,'Value')} returns selected item from popupmenu3

% --- Executes during object creation, after setting all properties.
function popupmenu3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
set(hObject,'BackgroundColor','white');
end
end
```

Appendix B. (Continued)

```
% --- Executes on selection change in popupmenu4.
function popupmenu4_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
val = get(hObject, 'Value');
switch val
case 1
    if (get(handles.popupmenu15, 'Value') == 1)
        set(handles.popupmenu16, 'Enable', 'on');
    end

    set(handles.popupmenu11, 'Enable', 'on');
    if (get(handles.popupmenu11, 'Value') == 1)
        set(handles.popupmenu14, 'Enable', 'on');
    end

    set(handles.popupmenu10, 'Enable', 'on');
case 2
    if (get(handles.popupmenu15, 'Value') == 1)
        set(handles.popupmenu16, 'Enable', 'on');
    end

    set(handles.popupmenu11, 'Enable', 'on');
    if (get(handles.popupmenu11, 'Value') == 1)
        set(handles.popupmenu14, 'Enable', 'on');
    end

    set(handles.popupmenu10, 'Enable', 'on');
case 3
    if (get(handles.popupmenu15, 'Value') == 1)
        set(handles.popupmenu16, 'Enable', 'on');
    end

    set(handles.popupmenu11, 'Enable', 'on');
    if (get(handles.popupmenu11, 'Value') == 1)
        set(handles.popupmenu14, 'Enable', 'on');
    end

    set(handles.popupmenu10, 'Enable', 'on');
case 4
    if (get(handles.popupmenu6, 'Value') == 1)
        if (get(handles.popupmenu15, 'Value') == 1)
            set(handles.popupmenu16, 'Enable', 'off');
        end

        set(handles.popupmenu11, 'Enable', 'off');
        if (get(handles.popupmenu11, 'Value') == 1)
            set(handles.popupmenu14, 'Enable', 'off');
        end

        if (get(handles.popupmenu7, 'Value') == 1)
            set(handles.popupmenu10, 'Enable', 'off');
        end
    end
end

% Hints: contents = get(hObject, 'String') returns popupmenu4 contents as cell array
%         contents{get(hObject, 'Value')} returns selected item from popupmenu4

% --- Executes during object creation, after setting all properties.
function popupmenu4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```


Appendix B. (Continued)

```
% Hint: popupmenu controls usually have a white background on Windows.
%     See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu6.
function popupmenu6_Callback(hObject, eventdata, handles)
% hObject     handle to popupmenu6 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
val = get(hObject,'Value');
switch val
case 1
    if (get(handles.popupmenu4, 'Value') == 4)
        if (get(handles.popupmenu15, 'Value') == 1)
            set(handles.popupmenu16,'Enable','off');
        end

        set(handles.popupmenu11,'Enable','off');
        if (get(handles.popupmenu11, 'Value') == 1)
            set(handles.popupmenu14,'Enable','off');
        end

        if (get(handles.popupmenu7, 'Value') == 1)
            set(handles.popupmenu10,'Enable','off');
        end
    end
case 2
    if (get(handles.popupmenu15, 'Value') == 1)
        set(handles.popupmenu16,'Enable','on');
    end

    set(handles.popupmenu11,'Enable','on');
    if (get(handles.popupmenu11, 'Value') == 1)
        set(handles.popupmenu14,'Enable','on');
    end

    set(handles.popupmenu10,'Enable','on');
end

% Hints: contents = get(hObject,'String') returns popupmenu6 contents as cell array
%     contents{get(hObject,'Value')} returns selected item from popupmenu6

% --- Executes during object creation, after setting all properties.
function popupmenu6_CreateFcn(hObject, eventdata, handles)
% hObject     handle to popupmenu6 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%     See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu7.
function popupmenu7_Callback(hObject, eventdata, handles)
% hObject     handle to popupmenu7 (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu7 contents as cell array
%     contents{get(hObject,'Value')} returns selected item from popupmenu7
```

Appendix B. (Continued)

```
val = get(hObject,'Value');
switch val
case 1
    if ((get(handles.popupmenu4, 'Value') == 4) && (get(handles.popupmenu6, 'Value') ==
1))
        set(handles.popupmenu10,'Enable','off');
    end
case 2
    set(handles.popupmenu10,'Enable','on');
end

% --- Executes during object creation, after setting all properties.
function popupmenu7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu8.
function popupmenu8_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu8 contents as cell array
%         contents{get(hObject,'Value')} returns selected item from popupmenu8

% --- Executes during object creation, after setting all properties.
function popupmenu8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu10.
function popupmenu10_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu10 contents as cell array
%         contents{get(hObject,'Value')} returns selected item from popupmenu10

% --- Executes during object creation, after setting all properties.
function popupmenu10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

Appendix B. (Continued)

```
end

% --- Executes on selection change in popupmenu14.
function popupmenu14_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu14 contents as cell array
%         contents{get(hObject,'Value')} returns selected item from popupmenu14

% --- Executes during object creation, after setting all properties.
function popupmenu14_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu11.
function popupmenu11_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

val = get(hObject,'Value');
switch val
case 1
    set(handles.popupmenu14,'Enable','on');
case 2
    set(handles.popupmenu14,'Enable','off');
end

% Hints: contents = get(hObject,'String') returns popupmenu11 contents as cell array
%         contents{get(hObject,'Value')} returns selected item from popupmenu11

% --- Executes during object creation, after setting all properties.
function popupmenu11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu16.
function popupmenu16_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu16 contents as cell array
%         contents{get(hObject,'Value')} returns selected item from popupmenu16

% --- Executes during object creation, after setting all properties.
function popupmenu16_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu16 (see GCBO)
```

Appendix B. (Continued)

```
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu15.
function popupmenu15_Callback(hObject, eventdata, handles)
% hObject handle to popupmenu15 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global VAR_WCI
global VAR_QI

val = get(hObject,'Value');
switch val
case 1
    if ((get(handles.popupmenu4, 'Value') ~= 4) || (get(handles.popupmenu6, 'Value') ==
2))
        set(handles.popupmenu16,'Enable','on');
    end
    set(handles.edit49, 'String', 1.5708);
    set(handles.edit50, 'String', 1.5708);
    set(handles.edit51, 'String', 0);
    set(handles.edit52, 'String', 1.5708);
    set(handles.edit53, 'String', 1.5708);
    set(handles.edit54, 'String', 1.5708);
    set(handles.edit55, 'String', 0);

    set(handles.edit49,'Enable','off');
    set(handles.edit50,'Enable','off');
    set(handles.edit51,'Enable','off');
    set(handles.edit52,'Enable','off');
    set(handles.edit53,'Enable','off');
    set(handles.edit54,'Enable','off');
    set(handles.edit55,'Enable','off');

    set(handles.pushbutton5,'Enable','off');
    set(handles.pushbutton6,'Enable','off');

    set(handles.edit56, 'String', 0);
    set(handles.edit57, 'String', 0);
    set(handles.edit58, 'String', 0);

    set(handles.edit56,'Enable','off');
    set(handles.edit57,'Enable','off');
    set(handles.edit58,'Enable','off');
case 2
    set(handles.popupmenu16,'Enable','off');

    set(handles.edit49, 'String', VAR_QI(1,1));
    set(handles.edit50, 'String', VAR_QI(2,1));
    set(handles.edit51, 'String', VAR_QI(3,1));
    set(handles.edit52, 'String', VAR_QI(4,1));
    set(handles.edit53, 'String', VAR_QI(5,1));
    set(handles.edit54, 'String', VAR_QI(6,1));
    set(handles.edit55, 'String', VAR_QI(7,1));

    set(handles.edit49,'Enable','off');
    set(handles.edit50,'Enable','off');
    set(handles.edit51,'Enable','off');
    set(handles.edit52,'Enable','off');
```

Appendix B. (Continued)

```
set(handles.edit53,'Enable','off');
set(handles.edit54,'Enable','off');
set(handles.edit55,'Enable','off');

set(handles.pushbutton5,'Enable','off');
set(handles.pushbutton6,'Enable','off');

set(handles.edit56,'String',VAR_WCI(1,1));
set(handles.edit57,'String',VAR_WCI(2,1));
set(handles.edit58,'String',VAR_WCI(3,1));

set(handles.edit56,'Enable','off');
set(handles.edit57,'Enable','off');
set(handles.edit58,'Enable','off');
case 3
set(handles.popupmenu16,'Enable','off');

set(handles.edit49,'Enable','on');
set(handles.edit50,'Enable','on');
set(handles.edit51,'Enable','on');
set(handles.edit52,'Enable','on');
set(handles.edit53,'Enable','on');
set(handles.edit54,'Enable','on');
set(handles.edit55,'Enable','on');

set(handles.pushbutton5,'Enable','on');
set(handles.pushbutton6,'Enable','on');

set(handles.edit56,'Enable','on');
set(handles.edit57,'Enable','on');
set(handles.edit58,'Enable','on');
end
% Hints: contents = get(hObject,'String') returns popupmenu15 contents as cell array
%         contents{get(hObject,'Value')} returns selected item from popupmenu15

% --- Executes during object creation, after setting all properties.
function popupmenu15_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu19.
function popupmenu19_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu19 contents as cell array
%         contents{get(hObject,'Value')} returns selected item from popupmenu19

% --- Executes during object creation, after setting all properties.
function popupmenu19_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
```

Appendix B. (Continued)

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu17.
function popupmenu17_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VAR_SCREENOPN

val = get(hObject,'Value');
switch val
case 1
    if (get(handles.popupmenu1, 'Value') == 1)
        set(handles.edit36,'Enable','on');
        set(handles.edit37,'Enable','on');
        set(handles.edit38,'Enable','on');
        set(handles.edit39,'Enable','on');
        set(handles.edit40,'Enable','on');
        set(handles.edit41,'Enable','on');
        set(handles.edit42,'Enable','on');
        set(handles.edit43,'Enable','on');
        set(handles.edit44,'Enable','on');
    else
        set(handles.edit36, 'String', 1);
        set(handles.edit37, 'String', 0);
        set(handles.edit38, 'String', 0);
        set(handles.edit39, 'String', 0);
        set(handles.edit40, 'String', 1);
        set(handles.edit41, 'String', 0);
        set(handles.edit42, 'String', 0);
        set(handles.edit43, 'String', 0);
        set(handles.edit44, 'String', 1);

        set(handles.edit36,'Enable','off');
        set(handles.edit37,'Enable','off');
        set(handles.edit38,'Enable','off');
        set(handles.edit39,'Enable','off');
        set(handles.edit40,'Enable','off');
        set(handles.edit41,'Enable','off');
        set(handles.edit42,'Enable','off');
        set(handles.edit43,'Enable','off');
        set(handles.edit44,'Enable','off');
    end

    set(handles.edit15,'Enable','on');
    set(handles.edit16,'Enable','on');
    set(handles.edit17,'Enable','on');

    set(handles.text15,'Enable','on');
    set(handles.text16,'Enable','on');
    set(handles.text17,'Enable','on');
    set(handles.text18,'Enable','on');

    set(handles.edit45,'Enable','on');
    set(handles.text14,'Enable','on');
    set(handles.text13,'Enable','on');

    set(handles.pushbutton3,'Enable','on');
    set(handles.popupmenu20,'Enable','on');
    %%%%%%%%%%%%%%%
    set(handles.edit62,'Enable','off');
    set(handles.edit63,'Enable','off');
    set(handles.edit64,'Enable','off');
```

Appendix B. (Continued)

```
set(handles.text21,'Enable','off');

set(handles.edit65,'Enable','off');
set(handles.edit66,'Enable','off');
set(handles.edit67,'Enable','off');
set(handles.text22,'Enable','off');

set(handles.edit4,'Enable','off');
set(handles.text6,'Enable','off');
set(handles.text5,'Enable','off');

set(handles.pushbutton4,'Enable','off');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
set(handles.edit1,'Enable','off');
set(handles.text1,'Enable','off');
set(handles.text2,'Enable','off');

set(handles.edit69,'Enable','off');
set(handles.text23,'Enable','off');
case 2
set(handles.edit36,'Enable','off');
set(handles.edit37,'Enable','off');
set(handles.edit38,'Enable','off');
set(handles.edit39,'Enable','off');
set(handles.edit40,'Enable','off');
set(handles.edit41,'Enable','off');
set(handles.edit42,'Enable','off');
set(handles.edit43,'Enable','off');
set(handles.edit44,'Enable','off');

set(handles.edit15,'Enable','off');
set(handles.edit16,'Enable','off');
set(handles.edit17,'Enable','off');

set(handles.text15,'Enable','off');
set(handles.text16,'Enable','off');
set(handles.text17,'Enable','off');
set(handles.text18,'Enable','off');

set(handles.edit45,'Enable','off');
set(handles.text14,'Enable','off');
set(handles.text13,'Enable','off');

set(handles.pushbutton3,'Enable','off');
set(handles.popupmenu20,'Enable','off');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
set(handles.edit62,'Enable','on');
set(handles.edit63,'Enable','on');
set(handles.edit64,'Enable','on');
set(handles.text21,'Enable','on');

if (get(handles.popupmenu1, 'Value') == 1)
    set(handles.edit65,'Enable','on');
    set(handles.edit66,'Enable','on');
    set(handles.edit67,'Enable','on');
    set(handles.text22,'Enable','on');
end

set(handles.edit4,'Enable','on');
set(handles.text6,'Enable','on');
set(handles.text5,'Enable','on');

set(handles.pushbutton4,'Enable','on');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
set(handles.edit1,'Enable','off');
set(handles.text1,'Enable','off');
```

Appendix B. (Continued)

```
set(handles.text2,'Enable','off');

set(handles.edit69,'Enable','off');
set(handles.text23,'Enable','off');
case 3
set(handles.edit36,'Enable','off');
set(handles.edit37,'Enable','off');
set(handles.edit38,'Enable','off');
set(handles.edit39,'Enable','off');
set(handles.edit40,'Enable','off');
set(handles.edit41,'Enable','off');
set(handles.edit42,'Enable','off');
set(handles.edit43,'Enable','off');
set(handles.edit44,'Enable','off');

set(handles.edit15,'Enable','off');
set(handles.edit16,'Enable','off');
set(handles.edit17,'Enable','off');

set(handles.text15,'Enable','off');
set(handles.text16,'Enable','off');
set(handles.text17,'Enable','off');
set(handles.text18,'Enable','off');

set(handles.edit45,'Enable','off');
set(handles.text14,'Enable','off');
set(handles.text13,'Enable','off');

set(handles.pushbutton3,'Enable','off');
set(handles.popupmenu20,'Enable','off');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
set(handles.edit62,'Enable','off');
set(handles.edit63,'Enable','off');
set(handles.edit64,'Enable','off');
set(handles.text21,'Enable','off');

set(handles.edit65,'Enable','off');
set(handles.edit66,'Enable','off');
set(handles.edit67,'Enable','off');
set(handles.text22,'Enable','off');

set(handles.edit4,'Enable','off');
set(handles.text6,'Enable','off');
set(handles.text5,'Enable','off');

set(handles.pushbutton4,'Enable','off');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
set(handles.edit1,'Enable','on');
set(handles.text1,'Enable','on');
set(handles.text2,'Enable','on');

set(handles.edit69,'Enable','off');
set(handles.text23,'Enable','off');
case 4
set(handles.edit36,'Enable','off');
set(handles.edit37,'Enable','off');
set(handles.edit38,'Enable','off');
set(handles.edit39,'Enable','off');
set(handles.edit40,'Enable','off');
set(handles.edit41,'Enable','off');
set(handles.edit42,'Enable','off');
set(handles.edit43,'Enable','off');
set(handles.edit44,'Enable','off');

set(handles.edit15,'Enable','off');
set(handles.edit16,'Enable','off');
```


Appendix B. (Continued)

```
set(handles.edit17,'Enable','off');

set(handles.text15,'Enable','off');
set(handles.text16,'Enable','off');
set(handles.text17,'Enable','off');
set(handles.text18,'Enable','off');

set(handles.edit45,'Enable','off');
set(handles.text14,'Enable','off');
set(handles.text13,'Enable','off');

set(handles.pushbutton3,'Enable','off');
set(handles.popupmenu20,'Enable','off');
#####
set(handles.edit62,'Enable','off');
set(handles.edit63,'Enable','off');
set(handles.edit64,'Enable','off');
set(handles.text21,'Enable','off');

set(handles.edit65,'Enable','off');
set(handles.edit66,'Enable','off');
set(handles.edit67,'Enable','off');
set(handles.text22,'Enable','off');

set(handles.edit4,'Enable','off');
set(handles.text6,'Enable','off');
set(handles.text5,'Enable','off');

set(handles.pushbutton4,'Enable','off');
#####
set(handles.edit1,'Enable','on');
set(handles.text1,'Enable','on');
set(handles.text2,'Enable','on');

set(handles.edit69,'Enable','on');
set(handles.text23,'Enable','on');
case 5
set(handles.edit36,'Enable','off');
set(handles.edit37,'Enable','off');
set(handles.edit38,'Enable','off');
set(handles.edit39,'Enable','off');
set(handles.edit40,'Enable','off');
set(handles.edit41,'Enable','off');
set(handles.edit42,'Enable','off');
set(handles.edit43,'Enable','off');
set(handles.edit44,'Enable','off');

set(handles.edit15,'Enable','off');
set(handles.edit16,'Enable','off');
set(handles.edit17,'Enable','off');

set(handles.text15,'Enable','off');
set(handles.text16,'Enable','off');
set(handles.text17,'Enable','off');
set(handles.text18,'Enable','off');

set(handles.edit45,'Enable','off');
set(handles.text14,'Enable','off');
set(handles.text13,'Enable','off');

set(handles.pushbutton3,'Enable','off');
set(handles.popupmenu20,'Enable','off');
#####
set(handles.edit62,'Enable','off');
set(handles.edit63,'Enable','off');
set(handles.edit64,'Enable','off');
```

Appendix B. (Continued)

```
set(handles.text21,'Enable','off');

set(handles.edit65,'Enable','off');
set(handles.edit66,'Enable','off');
set(handles.edit67,'Enable','off');
set(handles.text22,'Enable','off');

set(handles.edit4,'Enable','off');
set(handles.text6,'Enable','off');
set(handles.text5,'Enable','off');

set(handles.pushbutton4,'Enable','off');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
set(handles.edit1,'Enable','on');
set(handles.text1,'Enable','on');
set(handles.text2,'Enable','on');

set(handles.edit69,'Enable','off');
set(handles.text23,'Enable','off');

if (VAR_SCREENOPN ~= 1)
    WMRA_screen ('1');
    drawnow;
end
end

% Hints: contents = get(hObject,'String') returns popupmenu17 contents as cell array
%         contents{get(hObject,'Value')} returns selected item from popupmenu17

% --- Executes during object creation, after setting all properties.
function popupmenu17_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%         str2double(get(hObject,'String')) returns contents of edit1 as a double

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

Appendix B. (Continued)

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%         str2double(get(hObject,'String')) returns contents of edit4 as a double

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit36_Callback(hObject, eventdata, handles)
% hObject    handle to edit36 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit36 as text
%         str2double(get(hObject,'String')) returns contents of edit36 as a double

% --- Executes during object creation, after setting all properties.
function edit36_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit36 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit37_Callback(hObject, eventdata, handles)
% hObject    handle to edit37 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit37 as text
%         str2double(get(hObject,'String')) returns contents of edit37 as a double

% --- Executes during object creation, after setting all properties.
function edit37_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit37 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit38_Callback(hObject, eventdata, handles)
% hObject    handle to edit38 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit38 as text
```

Appendix B. (Continued)

```
%          str2double(get(hObject,'String')) returns contents of edit38 as a double

% --- Executes during object creation, after setting all properties.
function edit38_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit38 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%          See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit39_Callback(hObject, eventdata, handles)
% hObject    handle to edit39 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit39 as text
%          str2double(get(hObject,'String')) returns contents of edit39 as a double

% --- Executes during object creation, after setting all properties.
function edit39_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit39 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%          See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit40_Callback(hObject, eventdata, handles)
% hObject    handle to edit40 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit40 as text
%          str2double(get(hObject,'String')) returns contents of edit40 as a double

% --- Executes during object creation, after setting all properties.
function edit40_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit40 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%          See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit41_Callback(hObject, eventdata, handles)
% hObject    handle to edit41 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit41 as text
%          str2double(get(hObject,'String')) returns contents of edit41 as a double

% --- Executes during object creation, after setting all properties.
```

Appendix B. (Continued)

```
function edit41_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit41 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit42_Callback(hObject, eventdata, handles)
% hObject    handle to edit42 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit42 as text
%         str2double(get(hObject,'String')) returns contents of edit42 as a double

% --- Executes during object creation, after setting all properties.
function edit42_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit42 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit43_Callback(hObject, eventdata, handles)
% hObject    handle to edit43 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit43 as text
%         str2double(get(hObject,'String')) returns contents of edit43 as a double

% --- Executes during object creation, after setting all properties.
function edit43_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit43 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit44_Callback(hObject, eventdata, handles)
% hObject    handle to edit44 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit44 as text
%         str2double(get(hObject,'String')) returns contents of edit44 as a double

% --- Executes during object creation, after setting all properties.
function edit44_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit44 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

Appendix B. (Continued)

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit16_Callback(hObject, eventdata, handles)
% hObject    handle to edit16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit16 as text
%         str2double(get(hObject,'String')) returns contents of edit16 as a double

% --- Executes during object creation, after setting all properties.
function edit16_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit15_Callback(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit15 as text
%         str2double(get(hObject,'String')) returns contents of edit15 as a double

% --- Executes during object creation, after setting all properties.
function edit15_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit17_Callback(hObject, eventdata, handles)
% hObject    handle to edit17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit17 as text
%         str2double(get(hObject,'String')) returns contents of edit17 as a double

% --- Executes during object creation, after setting all properties.
function edit17_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
```

Appendix B. (Continued)

```
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit49_Callback(hObject, eventdata, handles)
% hObject    handle to edit49 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit49 as text
%         str2double(get(hObject,'String')) returns contents of edit49 as a double

% --- Executes during object creation, after setting all properties.
function edit49_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit49 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit50_Callback(hObject, eventdata, handles)
% hObject    handle to edit50 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit50 as text
%         str2double(get(hObject,'String')) returns contents of edit50 as a double

% --- Executes during object creation, after setting all properties.
function edit50_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit50 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit51_Callback(hObject, eventdata, handles)
% hObject    handle to edit51 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit51 as text
%         str2double(get(hObject,'String')) returns contents of edit51 as a double

% --- Executes during object creation, after setting all properties.
function edit51_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit51 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

Appendix B. (Continued)

```
    set(hObject,'BackgroundColor','white');
end

function edit52_Callback(hObject, eventdata, handles)
% hObject    handle to edit52 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit52 as text
%         str2double(get(hObject,'String')) returns contents of edit52 as a double

% --- Executes during object creation, after setting all properties.
function edit52_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit52 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit53_Callback(hObject, eventdata, handles)
% hObject    handle to edit53 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit53 as text
%         str2double(get(hObject,'String')) returns contents of edit53 as a double

% --- Executes during object creation, after setting all properties.
function edit53_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit53 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit54_Callback(hObject, eventdata, handles)
% hObject    handle to edit54 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit54 as text
%         str2double(get(hObject,'String')) returns contents of edit54 as a double

% --- Executes during object creation, after setting all properties.
function edit54_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit54 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```


Appendix B. (Continued)

```
function edit55_Callback(hObject, eventdata, handles)
% hObject    handle to edit55 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit55 as text
%        str2double(get(hObject,'String')) returns contents of edit55 as a double

% --- Executes during object creation, after setting all properties.
function edit55_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit55 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit56_Callback(hObject, eventdata, handles)
% hObject    handle to edit56 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit56 as text
%        str2double(get(hObject,'String')) returns contents of edit56 as a double

% --- Executes during object creation, after setting all properties.
function edit56_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit56 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit57_Callback(hObject, eventdata, handles)
% hObject    handle to edit57 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit57 as text
%        str2double(get(hObject,'String')) returns contents of edit57 as a double

% --- Executes during object creation, after setting all properties.
function edit57_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit57 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit58_Callback(hObject, eventdata, handles)
% hObject    handle to edit58 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

Appendix B. (Continued)

```
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit58 as text
%         str2double(get(hObject,'String')) returns contents of edit58 as a double

% --- Executes during object creation, after setting all properties.
function edit58_CreateFcn(hObject, eventdata, handles)
% hObject      handle to edit58 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit45_Callback(hObject, eventdata, handles)
% hObject      handle to edit45 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit45 as text
%         str2double(get(hObject,'String')) returns contents of edit45 as a double

% --- Executes during object creation, after setting all properties.
function edit45_CreateFcn(hObject, eventdata, handles)
% hObject      handle to edit45 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu20.
function popupmenu20_Callback(hObject, eventdata, handles)
% hObject      handle to popupmenu20 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu20 contents as cell array
%         contents{get(hObject,'Value')} returns selected item from popupmenu20

% --- Executes during object creation, after setting all properties.
function popupmenu20_CreateFcn(hObject, eventdata, handles)
% hObject      handle to popupmenu20 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit62_Callback(hObject, eventdata, handles)
% hObject      handle to edit62 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
```

Appendix B. (Continued)

```
% Hints: get(hObject,'String') returns contents of edit62 as text
%         str2double(get(hObject,'String')) returns contents of edit62 as a double

% --- Executes during object creation, after setting all properties.
function edit62_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit62 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit63_Callback(hObject, eventdata, handles)
% hObject    handle to edit63 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit63 as text
%         str2double(get(hObject,'String')) returns contents of edit63 as a double

% --- Executes during object creation, after setting all properties.
function edit63_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit63 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit64_Callback(hObject, eventdata, handles)
% hObject    handle to edit64 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit64 as text
%         str2double(get(hObject,'String')) returns contents of edit64 as a double
% --- Executes during object creation, after setting all properties.
function edit64_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit64 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit65_Callback(hObject, eventdata, handles)
% hObject    handle to edit65 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit65 as text
%         str2double(get(hObject,'String')) returns contents of edit65 as a double

% --- Executes during object creation, after setting all properties.
```

Appendix B. (Continued)

```
function edit65_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit65 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit66_Callback(hObject, eventdata, handles)
% hObject    handle to edit66 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit66 as text
%         str2double(get(hObject,'String')) returns contents of edit66 as a double

% --- Executes during object creation, after setting all properties.
function edit66_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit66 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit67_Callback(hObject, eventdata, handles)
% hObject    handle to edit67 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit67 as text
%         str2double(get(hObject,'String')) returns contents of edit67 as a double

% --- Executes during object creation, after setting all properties.
function edit67_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit67 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenu21.
function popupmenu21_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu21 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu21 contents as cell array
%         contents{get(hObject,'Value')} returns selected item from popupmenu21

% --- Executes during object creation, after setting all properties.
function popupmenu21_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu21 (see GCBO)
```

Appendix B. (Continued)

```
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit69_Callback(hObject, eventdata, handles)
% hObject handle to edit69 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit69 as text
% str2double(get(hObject,'String')) returns contents of edit69 as a double

% --- Executes during object creation, after setting all properties.
function edit69_CreateFcn(hObject, eventdata, handles)
% hObject handle to edit69 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton3 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global VAR_MATRIX

if (get(handles.popupmenu1, 'Value') == 1)
    WMRA_matrix_entry(' (3 x 4) ');
    TS = str2num(VAR_MATRIX);
    [row col] = size(TS);

    if ((row == 3) && (col == 4))
        set(handles.edit36, 'String', TS(1,1));
        set(handles.edit37, 'String', TS(1,2));
        set(handles.edit38, 'String', TS(1,3));
        set(handles.edit16, 'String', TS(1,4));
        set(handles.edit39, 'String', TS(2,1));
        set(handles.edit40, 'String', TS(2,2));
        set(handles.edit41, 'String', TS(2,3));
        set(handles.edit15, 'String', TS(2,4));
        set(handles.edit42, 'String', TS(3,1));
        set(handles.edit43, 'String', TS(3,2));
        set(handles.edit44, 'String', TS(3,3));
        set(handles.edit17, 'String', TS(3,4));
    else
        WMRA_error_gui ('Matrix size wrong. Size (3 x 4) expected');
    end
else
    WMRA_matrix_entry(' (3 x 1) ');
    TS = str2num(VAR_MATRIX);
    [row col] = size(TS);

    if ((row == 3) && (col == 1))
        set(handles.edit16, 'String', TS(1,1));
```

Appendix B. (Continued)

```
        set(handles.edit15, 'String', TS(2,1));
        set(handles.edit17, 'String', TS(3,1));
    else
        WMRA_error_gui ('Matrix size wrong. Size (3 x 1) expected');
    end
end

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VAR_MATRIX

if (get(handles.popupmenu1, 'Value') == 1)
    WMRA_matrix_entry('2 x 3');
    TS = str2num(VAR_MATRIX);
    [row col] = size(TS);

    if ((row == 2) && (col == 3))
        set(handles.edit62, 'String', TS(1,1));
        set(handles.edit63, 'String', TS(1,2));
        set(handles.edit64, 'String', TS(1,3));
        set(handles.edit65, 'String', TS(2,1));
        set(handles.edit66, 'String', TS(2,2));
        set(handles.edit67, 'String', TS(2,3));
    else
        WMRA_error_gui ('Matrix size wrong. Size (2 x 3) expected');
    end
else
    WMRA_matrix_entry('1 x 3');
    TS = str2num(VAR_MATRIX);
    [row col] = size(TS);

    if ((row == 1) && (col == 3))
        set(handles.edit62, 'String', TS(1,1));
        set(handles.edit63, 'String', TS(1,2));
        set(handles.edit64, 'String', TS(1,3));
    else
        WMRA_error_gui ('Matrix size wrong. Size (1 x 3) expected');
    end
end

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles, varargin)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VAR_MATRIX

WMRA_matrix_entry('3 x 1');
TS = str2num(VAR_MATRIX);
[row col] = size(TS);

if ((row == 3) && (col == 1))
    set(handles.edit56, 'String', TS(1,1));
    set(handles.edit57, 'String', TS(2,1));
    set(handles.edit58, 'String', TS(3,1));
else
    WMRA_error_gui ('Matrix size wrong. Size (3 x 1) expected');
end

% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

Appendix B. (Continued)

```
% handles structure with handles and user data (see GUIDATA)
global VAR_MATRIX

WMRA_matrix_entry('(7 x 1)');
TS = str2num(VAR_MATRIX);
[row col] = size(TS);

if ((row == 7) && (col == 1))
    set(handles.edit49, 'String', TS(1,1));
    set(handles.edit50, 'String', TS(2,1));
    set(handles.edit51, 'String', TS(3,1));
    set(handles.edit52, 'String', TS(4,1));
    set(handles.edit53, 'String', TS(5,1));
    set(handles.edit54, 'String', TS(6,1));
    set(handles.edit55, 'String', TS(7,1));
else
    WMRA_error_gui('Matrix size wrong. Size (7 x 1) expected');
end

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global VAR_STOP

VAR_STOP = 1;
drawnow;

set(handles.pushbutton1, 'Enable', 'off');
set(handles.pushbutton7, 'Enable', 'off');
set(handles.pushbutton8, 'Enable', 'on');
set(handles.pushbutton2, 'Enable', 'on');

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global VAR_STOP
global VAR_DX
global VAR_SCREENOPN
global VAR_WCI
global VAR_QI
global spdata1
global VAR_LOOP

VAR_DX = [0;0;0;0;0;0;0];
VAR_STOP = 0;

error = 0;
notfilled = 0;
td_err = 0;
ts_err = 0;

set(handles.pushbutton1, 'Enable', 'on');
set(handles.pushbutton2, 'Enable', 'off');
set(handles.pushbutton8, 'Enable', 'off');

if (get(handles.popupmenu17, 'Value') == 3) || (get(handles.popupmenu17, 'Value') == 4)
|| (get(handles.popupmenu17, 'Value') == 5)
    set(handles.pushbutton7, 'Enable', 'on');
    VAR_LOOP = 1;
end

cart = get(handles.popupmenu1, 'Value');
```

Appendix B. (Continued)

```
WCA = get(handles.popupmenu3, 'Value');
if WCA == 3
    optim = 0;
    JLA = 0;
    JLO = get(handles.checkbox2, 'Value');
else
    optim = get(handles.popupmenu8, 'Value');
    JLA = get(handles.checkbox1, 'Value');
    JLO = get(handles.checkbox2, 'Value');
end

coord = get(handles.popupmenu21, 'Value');

t1 = get(handles.popupmenu4, 'Value');
if t1 == 1
    vr = 1;
    ml = 0;
elseif t1 == 2
    vr = 0;
    ml = 1;
elseif t1 == 3
    vr = 1;
    ml = 1;
else
    vr = 0;
    ml = 0;
end

arm = get(handles.popupmenu6, 'Value');
arm = arm - 1;
plt = get(handles.popupmenu7, 'Value');
choice8 = get(handles.popupmenu10, 'Value');

drawnow;
if VAR_STOP == 1
    stop = 1;
else
    stop = 0;
end

if stop == 0
    cont = get(handles.popupmenu17, 'Value');
    if cont == 1
        if cart == 1
            v_str = get(handles.edit36, 'String');
            v_1 = str2double(v_str);
            if isnan(v_1)
                error = 1;
                notfilled = 1;
            end
            if (v_1 > 1) || (v_1 < -1)
                error = 1;
                td_err = 1;
            end
            v_str = get(handles.edit37, 'String');
            v_2 = str2double(v_str);
            if isnan(v_2)
                error = 1;
                notfilled = 1;
            end
            if (v_2 > 1) || (v_2 < -1)
                error = 1;
                td_err = 1;
            end
            v_str = get(handles.edit38, 'String');
            v_3 = str2double(v_str);
```


Appendix B. (Continued)

```
if isnan (v_3)
    error = 1;
    notfilled = 1;
end
if (v_3 > 1) || (v_3 < -1)
    error = 1;
    td_err = 1;
end
v_str = get(handles.edit39, 'String');
v_5 = str2double(v_str);
if isnan (v_5)
    error = 1;
    notfilled = 1;
end
if (v_5 > 1) || (v_5 < -1)
    error = 1;
    td_err = 1;
end
v_str = get(handles.edit40, 'String');
v_6 = str2double(v_str);
if isnan (v_6)
    error = 1;
    notfilled = 1;
end
if (v_6 > 1) || (v_6 < -1)
    error = 1;
    td_err = 1;
end
v_str = get(handles.edit41, 'String');
v_7 = str2double(v_str);
if isnan (v_7)
    error = 1;
    notfilled = 1;
end
if (v_7 > 1) || (v_7 < -1)
    error = 1;
    td_err = 1;
end
v_str = get(handles.edit42, 'String');
v_9 = str2double(v_str);
if isnan (v_9)
    error = 1;
    notfilled = 1;
end
if (v_9 > 1) || (v_9 < -1)
    error = 1;
    td_err = 1;
end
v_str = get(handles.edit43, 'String');
v_10 = str2double(v_str);
if isnan (v_10)
    error = 1;
    notfilled = 1;
end
if (v_10 > 1) || (v_10 < -1)
    error = 1;
    td_err = 1;
end
v_str = get(handles.edit44, 'String');
v_11 = str2double(v_str);
if isnan (v_11)
    error = 1;
    notfilled = 1;
end
if (v_11 > 1) || (v_11 < -1)
    error = 1;
```

Appendix B. (Continued)

```
                td_err = 1;
            end
        else
            v_1 = 1;
            v_2 = 0;
            v_3 = 0;
            v_5 = 0;
            v_6 = 1;
            v_7 = 0;
            v_9 = 0;
            v_10 = 0;
            v_11 = 1;
        end
        v_str = get(handles.edit16, 'String');
        v_4 = str2double(v_str);
        if isnan (v_4)
            error = 1;
            notfilled = 1;
        end
        v_str = get(handles.edit15, 'String');
        v_8 = str2double(v_str);
        if isnan (v_8)
            error = 1;
            notfilled = 1;
        end
        v_str = get(handles.edit17, 'String');
        v_12 = str2double(v_str);
        if isnan (v_12)
            error = 1;
            notfilled = 1;
        end
        Td = [v_1 v_2 v_3 v_4; v_5 v_6 v_7 v_8; v_9 v_10 v_11 v_12; 0 0 0 1];

        temp_str = get(handles.edit45, 'String');
        v = str2double(temp_str);
        if isnan (v)
            error = 1;
            notfilled = 1;
        end
        trajf = get(handles.popupmenu20, 'Value');
    elseif cont == 2
        v_str = get(handles.edit62, 'String');
        v_1 = str2double(v_str);
        if isnan (v_1)
            error = 1;
            notfilled = 1;
        end
        v_str = get(handles.edit63, 'String');
        v_2 = str2double(v_str);
        if isnan (v_2)
            error = 1;
            notfilled = 1;
        end
        v_str = get(handles.edit64, 'String');
        v_3 = str2double(v_str);
        if isnan (v_3)
            error = 1;
            notfilled = 1;
        end
        if (cart == 1)
            v_str = get(handles.edit65, 'String');
            v_4 = str2double(v_str);
            if isnan (v_4)
                error = 1;
                notfilled = 1;
            end
        end
    end
end
```

Appendix B. (Continued)

```
v_str = get(handles.edit66, 'String');
v_5 = str2double(v_str);
if isnan (v_5)
    error = 1;
    notfilled = 1;
end
v_str = get(handles.edit67, 'String');
v_6 = str2double(v_str);
if isnan (v_6)
    error = 1;
    notfilled = 1;
end
end
if (cart == 1)
    Vd = [v_1; v_2; v_3; v_4; v_5; v_6];
else
    Vd = [v_1; v_2; v_3];
end

temp_str = get(handles.edit4, 'String');
ts = str2double(temp_str);
if isnan (ts)
    error = 1;
    notfilled = 1;
end
if ts < 0
    error = 1;
    ts_err = 1;
end
else
temp_str = get(handles.edit1, 'String');
v = str2double(temp_str);
if isnan (v)
    error = 1;
    notfilled = 1;
end
if cont == 4
temp_str = get(handles.edit69, 'String');
port1 = str2double(temp_str);
if isnan (port1)
    error = 1;
    notfilled = 1;
end
end
end
end

if stop == 0
drawnow;
if VAR_STOP == 1
    stop = 1;
else
    stop = 0;
end

if stop == 0
start = get(handles.popupmenu15, 'Value');
if start == 1
    if vr == 1 || ml == 1 || arm == 1
        if (get(handles.popupmenu16, 'Value') == 1)
            ini = 1;
        else
            ini = 0;
        end
    else
        ini = 0;
    end
end
```

Appendix B. (Continued)

```
end
qi = [90;90;0;90;90;90;0] * pi/180;
VAR_QI = qi;
WCI = [0;0;0];
VAR_WCI = WCI;
else
v_str = get(handles.edit49, 'String');
v_1 = str2double(v_str);
if isnan (v_1)
    error = 1;
    notfilled = 1;
end
v_str = get(handles.edit50, 'String');
v_2 = str2double(v_str);
if isnan (v_2)
    error = 1;
    notfilled = 1;
end
v_str = get(handles.edit51, 'String');
v_3 = str2double(v_str);
if isnan (v_3)
    error = 1;
    notfilled = 1;
end
v_str = get(handles.edit52, 'String');
v_4 = str2double(v_str);
if isnan (v_4)
    error = 1;
    notfilled = 1;
end
v_str = get(handles.edit53, 'String');
v_5 = str2double(v_str);
if isnan (v_5)
    error = 1;
    notfilled = 1;
end
v_str = get(handles.edit54, 'String');
v_6 = str2double(v_str);
if isnan (v_6)
    error = 1;
    notfilled = 1;
end
v_str = get(handles.edit55, 'String');
v_7 = str2double(v_str);
if isnan (v_7)
    error = 1;
    notfilled = 1;
end
end
qi = [v_1; v_2; v_3; v_4; v_5; v_6; v_7];
VAR_QI = qi;

v_str = get(handles.edit56, 'String');
v_1 = str2double(v_str);
if isnan (v_1)
    error = 1;
    notfilled = 1;
end
v_str = get(handles.edit57, 'String');
v_2 = str2double(v_str);
if isnan (v_2)
    error = 1;
    notfilled = 1;
end
v_str = get(handles.edit58, 'String');
v_3 = str2double(v_str);
if isnan (v_3)
```

Appendix B. (Continued)

```
        error = 1;
        notfilled = 1;
    end
    WCi = [v_1; v_2; v_3];
    VAR_WCI = WCi;

    ini = 0;
end

choice6 = get(handles.popupmenu11, 'Value');
if (choice6 == 1)
    choice7 = get(handles.popupmenu14, 'Value');
end
end
end

if (error == 1)
    if notfilled == 1
        WMRA_error_gui ('One or more required inputs are not filled or filled wrongly');
    end
    if td_err == 1
        WMRA_error_gui ('Elements of Rd (Rotation Matrix) should be in between -1 to
+1');
    end
    if ts_err == 1
        WMRA_error_gui ('Ts should be greater than zero');
    end
end

if cont == 5
    if (VAR_SCREENOPN ~= 1)
        WMRA_screen ('1');
        drawnow;
    end
end

% Declaring a global variable for Optimization in WMRA_Opt():
global dHo

if (stop == 0) && (error == 0)    %Redwan's Code Entry
    drawnow;
    if VAR_STOP == 1
        stop = 1;
    else
        stop = 0;
    end

    if stop == 0    % 1st point
        % This "new USF WMRA" script SIMULATES the WMRA system with ANIMATION and plots
        for 9 DOF. All angles are in Radians.
        % Defining used parameters:
        d2r=pi/180; % Conversions from Degrees to Radians.
        r2d=180/pi; % Conversions from Radians to Degrees.

        % Reading the Wheelchair's constant dimentions, all dimentions are converted in
        millimeters:
        L=WMRA_WCD;

        % Calculating the Transformation Matrix of the initial position of the WMRA's
        base:
        Tiwc=WMRA_p2T(WCi(1),WCI(2),WCI(3));

        % Calculating the initial Wheelchair Variables:
        qiwc=[sqrt(WCi(1)^2+WCI(2)^2);WCI(3)];

        % Calculating the initial transformation matrices:
```

Appendix B. (Continued)

```

[Ti, Tia, Tiwc, T01, T12, T23, T34, T45, T56, T67]=WMRA_Tall(1, qi, [0;0], Tiwc);

if cont==1
    % Calculating the linear distance from the initial position to the desired
    position and the linear velocity:
    if coord==2
        D=sqrt( (Td(1,4)-Tia(1,4))^2 + (Td(2,4)-Tia(2,4))^2 + (Td(3,4)-
Tia(3,4))^2);
    elseif coord==3
        D=sqrt( (Td(1,4))^2 + (Td(2,4))^2 + (Td(3,4))^2);
    else
        D=sqrt( (Td(1,4)-Ti(1,4))^2 + (Td(2,4)-Ti(2,4))^2 + (Td(3,4)-Ti(3,4))^2);
    end
    % Calculating the number of iteration and the time increment (delta t) if the
linear step increment of the tip is 1 mm:
    dt=0.05; % Time increment in seconds.
    total_time=D/v; % Total time of animation.
    n=round(total_time/dt); % Number of iterations rounded up.
    dt=total_time/n; % Adjusted time increment in seconds.
    % Calculating the Trajectory of the end effector, and once the trajectory is
calculated, we should redefine "Td" based on the ground frame:
    if coord==2
        Tt=WMRA_traj(trajf, Tia, Td, n+1);
        Td=Tiwc*Td;
    elseif coord==3
        Tt=WMRA_traj(trajf, eye(4), Td, n+1);
        Td=Ti*Td;
    else
        Tt=WMRA_traj(trajf, Ti, Td, n+1);
    end
elseif cont==2
    % Calculating the number of iteration and the time increment (delta t) if the
linear step increment of the gripper is 1 mm:
    dt=0.05; % Time increment in seconds.
    total_time=ts; % Total time of animation.
    n=round(total_time/dt); % Number of iterations rounded up.
    dt=total_time/n; % Adjusted time increment in seconds.
    dx=Vd*dt;
    Td=Ti;
elseif cont==3
    dt=0.05;
    dx=v*dt*[spdata1(3)/20 ; -spdata1(1)/40 ; spdata1(2)/30 ; spdata1(6)/1500 ; -
spdata1(4)/900 ; spdata1(5)/1300];
    dg=spdata1(7);
    Td=Ti;
    n=1;
elseif cont==4
    dt=0.05;
    dx=v*dt*WMRA_psy(port1);
    dg=dx(7);
    dx=dx(1:6);
    Td=Ti;
    n=1;
else
    dt=0.05;
    dx=v*dt*VAR_DX(1:6);
    dg=VAR_DX(7);
    Td=Ti;
    n=1;
end

drawnow;
if VAR_STOP == 1
    stop = 1;
else
    stop = 0;

```

Appendix B. (Continued)

```
end

if stop == 0 % 2nd point

    % Initializing the joint angles, the Transformation Matrix, and time:
    dq=zeros(9,1);
    dg=0;
    qo=[qi;qiwc];
    To=Ti;
    Toa=Tia;
    Towc=Tiwc;
    tt=0;
    i=1;
    dHo=[0;0;0;0;0;0;0];

    % Initializing the WMRA:
    if ini==0 % When no "park" to "ready" motion required.
        % Initializing Virtual Reality Animation:
        if vr==1
            WMRA_VR_Animation(1, Towc, qo);
        end
        % Initializing Robot Animation in Matlab Graphics:
        if ml==1
            WMRA_ML_Animation(1, To, Td, Towc, T01, T12, T23, T34, T45, T56,
T67);

            end
            % Initializing the Physical Arm:
            if arm==1
                WMRA_ARM_Motion(1, 2, [qo;dg], 0);
                ddt=0;
            end
        elseif ini==1 && (vr==1 || ml==1 || arm==1) % When "park" to "ready" motion
is required.
            WMRA_park2ready(1, vr, ml, arm, Towc, qo(8:9));
            if arm==1
                ddt=0;
            end
        end

        % Re-Drawing the Animation:
        if vr==1 || ml==1
            drawnow;
        end

        % Starting a timer:
        tic

        drawnow;
        if VAR_STOP == 1
            stop = 1;
        else
            stop = 0;
        end

        % Starting the Iteration Loop:
        while (i<=(n+1)) && (stop == 0) % 3rd point
            % Calculating the 6X7 Jacobian of the arm in frame 0:
            [Joa,detJoa]=WMRA_J07(T01, T12, T23, T34, T45, T56, T67);

            % Calculating the 6X2 Jacobian based on the WMRA's base in the ground
frame:

            phi=atan2(Towc(2,1),Towc(1,1));
            Jowc=WMRA_Jga(1, phi, Toa(1:2,4));

            % Changing the Jacobian reference frame based on the choice of which
coordinates frame are referenced in the Cartesian control:
```

Appendix B. (Continued)

```

% coord=1 for Ground Coordinates Control.
% coord=2 for Wheelchair Coordinates Control.
% coord=3 for Gripper Coordinates Control.
if coord==2
    Joa=Joa;
    Jowc=[Towc(1:3,1:3)' zeros(3); zeros(3) Towc(1:3,1:3)']*Jowc;
elseif coord==3
    Joa=[Toa(1:3,1:3)' zeros(3); zeros(3) Toa(1:3,1:3)']*Joa;
    Jowc=[To(1:3,1:3)' zeros(3); zeros(3) To(1:3,1:3)']*Jowc;
elseif coord==1
    Joa=[Towc(1:3,1:3) zeros(3); zeros(3) Towc(1:3,1:3)]*Joa;
    Jowc=Jowc;
end

% Calculating the 3X9 or 6X9 augmented Jacobian of the WMRA system based
on the ground frame:
if cart==2
    Joa=Joa(1:3,1:7);
    detJoa=sqrt(det(Joa*Joa'));
    Jowc=Jowc(1:3,1:2);
    Jo=[Joa Jowc];
    detJo=sqrt(det(Jo*Jo'));
else
    Jo=[Joa Jowc];
    detJo=sqrt(det(Jo*Jo'));
end

% Finding the Cartesian errors of the end effector:
if cont==1
    % Calculating the Position and Orientation errors of the end
effector, and the rates of motion of the end effector:
    if coord==2
        invTowc=[Towc(1:3,1:3)' , -Towc(1:3,1:3)']*Towc(1:3,4);0 0 0 1];
        Ttnew=invTowc*Tiwc*Tt(:, :, i);
        dx=WMRA_delta(Toa, Ttnew);
    elseif coord==3
        invTo=[To(1:3,1:3)' , -To(1:3,1:3)']*To(1:3,4);0 0 0 1];
        Ttnew=invTo*Ti*Tt(:, :, i);
        dx=WMRA_delta(eye(4), Ttnew);
    else
        dx=WMRA_delta(To, Tt(:, :, i));
    end
elseif cont==2
elseif cont==3
    dx=v*dt*[spdata1(3)/20 ; -spdata1(1)/40 ; spdata1(2)/30 ;
spdata1(6)/1500 ; -spdata1(4)/900 ; spdata1(5)/1300];
    dg=spdata1(7);
elseif cont==4
    dx=v*dt*WMRA_psy(port1);
    dg=dx(7);
    dx=dx(1:6);
else
    dx=v*dt*VAR_DX(1:6);
    dg=VAR_DX(7);
end

% Changing the order of Cartesian motion in the case when gripper
reference frame is selected for control with the screen or psy or SpaceBall interfaces:
if coord==3 && cont>=3
    dx=[-dx(2) ; -dx(3) ; dx(1) ; -dx(5) ; -dx(6) ; dx(4) ];
end

if cart==2
    dx=dx(1:3);
end

```


Appendix B. (Continued)

```

        % Calculating the resolved rate with optimization:
        % Index input values for "optim": 1= SR-I & WLN, 2= P-I & WLN, 3= SR-I &
ENE, 4= P-I & ENE:
        if WCA==2
            dq=WMRA_Opt(optim, JLA, JLO, Joa, detJoa, dq(1:7), dx, dt, qo);
            dq=[dq;0;0];
        elseif WCA==3
            dq=WMRA_Opt(optim, JLA, JLO, Jowc, 1, dq(8:9), dx(1:2), dt, 1);
            dq=[0;0;0;0;0;0;0;dq];
        else
            dq=WMRA_Opt(optim, JLA, JLO, Jo, detJo, dq, dx, dt, qo);
        end

        drawnow;
        if VAR_STOP == 1
            stop = 1;
        else
            stop = 0;
        end

        if stop == 0    % 4nd point

            % Calculating the new Joint Angles:
            qn=qo+dq;

            % Calculating the new Transformation Matrices:
            [Tn, Tna, Tnwc, T01, T12, T23, T34, T45, T56, T67]=WMRA_Tall(2, qn,
dq(8:9), Towc);

            % A safety condition function to stop the joints that may cause
collision of the arm with itself, the wheelchair, or the human user:
            if JLO==1 && WCA~=3
                dq(1:7)=WMRA_collide(dq(1:7), T01, T12, T23, T34, T45, T56, T67);
                % Re-calculating the new Joint Angles:
                qn=qo+dq;
                % Re-calculating the new Transformation Matrices:
                [Tn, Tna, Tnwc, T01, T12, T23, T34, T45, T56, T67]=WMRA_Tall(2,
qn, dq(8:9), Towc);
            end

            % Saving the plot data in case plots are required:
            if plt==2
                WMRA_Plots(1, L, r2d, dt, i, tt, qn, dq, Tn, Tnwc, detJoa,
detJo);
            end

            % Updating Physical Arm:
            if arm==1
                ddt=ddt+dt;
                if ddt>=0.5 || i>=(n+1)
                    WMRA_ARM_Motion(2, 1, [qn;dq], ddt);
                    ddt=0;
                end
            end

            % Updating Virtual Reality Animation:
            if vr==1
                WMRA_VR_Animation(2, Tnwc, qn);
            end

            % Updating Matlab Graphics Animation:
            if ml==1
                WMRA_ML_Animation(2, Ti, Td, Tnwc, T01, T12, T23, T34, T45, T56,
T67);
            end

```

Appendix B. (Continued)

```
% Re-Drawing the Animation:
if vr==1 || ml==1
    drawnow;
end

% Updating the old values with the new values for the next iteration:
qo=qn;
To=Tn;
Toa=Tna;
Towc=Tnwc;
tt=tt+dt;
i=i+1;

% Stopping the simulation when the exit button is pressed:
if cont==3 || cont==4
    if (VAR_LOOP == 1)
        n=n+1;
    else
        break
    end
end
if cont==5
    if (VAR_SCREENOPN == 1) && (VAR_LOOP == 1)
        n=n+1;
    else
        break
    end
end

% Delay to comply with the required speed:
if toc < tt
    pause(tt-toc);
end
end

drawnow; % 5th point
if VAR_STOP == 1
    stop = 1;
else
    stop = 0;
end
end

drawnow; % 6th point
if VAR_STOP == 1
    stop = 1;
else
    stop = 0;
end

if stop == 0 % 7th point
    % Reading the elapsed time and printing it with the simulation time:
    if cont==1 || cont==2, fprintf('\nSimulation time is %6.6f seconds.\n' ,
total_time); end
    toc

% Plotting:
if plt==2
    WMRA_Plots(2, L, r2d, dt, i, tt, qn, dq, Tn, Tnwc, detJoa, detJo);
end

if vr==1 || ml==1 || arm==1

% Going back to the ready position:
if choice6==1
    WMRA_any2ready(2, vr, ml, arm, Tnwc, qn);
```

Appendix B. (Continued)

```
% Going back to the parking position:
if choice7==1
    WMRA_ready2park(2, vr, ml, arm, Tnwc, qn(8:9));
end
end

% Closing the Arm library and Matlab Graphics Animation and Virtual
Reality Animation and Plots windows:or press "2" for No. \n','s');
if choice8==1
    if arm==1
        WMRA_ARM_Motion(3, 0, 0, 0);
    end
    if vr==1
        WMRA_VR_Animation(3, 0, 0);
    end
    if ml==1
        WMRA_ML_Animation(3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    end
    if plt==2
        close
        (figure(2),figure(3),figure(4),figure(5),figure(6),figure(7),figure(8),figure(9),figure(1
0));
    end
end
end
end
end
end
end
end
end

VAR_QI = qn(1:7);
VAR_WCI = [Tnwc(1,4); Tnwc(2,4); phi];

if (get(handles.popupmenu15, 'Value') == 2)
    set(handles.edit49, 'String', VAR_QI(1,1));
    set(handles.edit50, 'String', VAR_QI(2,1));
    set(handles.edit51, 'String', VAR_QI(3,1));
    set(handles.edit52, 'String', VAR_QI(4,1));
    set(handles.edit53, 'String', VAR_QI(5,1));
    set(handles.edit54, 'String', VAR_QI(6,1));
    set(handles.edit55, 'String', VAR_QI(7,1));

    set(handles.edit56, 'String', VAR_WCI(1,1));
    set(handles.edit57, 'String', VAR_WCI(2,1));
    set(handles.edit58, 'String', VAR_WCI(3,1));
end

set(handles.pushbutton1,'Enable','off');
set(handles.pushbutton7,'Enable','off');
set(handles.pushbutton2,'Enable','on');
set(handles.pushbutton8,'Enable','on');

% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global VAR_LOOP

VAR_LOOP = 0;
drawnow;

set(handles.pushbutton1,'Enable','off');
set(handles.pushbutton7,'Enable','off');
```

Appendix B. (Continued)

```
set(handles.pushbutton2,'Enable','on');
set(handles.pushbutton8,'Enable','on');

% --- Executes on button press in checkbox1.
function checkbox1_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox1

% --- Executes on button press in checkbox2.
function checkbox2_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox2

% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%1st coll

set(handles.popupmenu1,'Enable','on');
set(handles.popupmenu1,'Value',1);

set(handles.popupmenu3,'Enable','on');
set(handles.popupmenu3,'Value',1);

set(handles.popupmenu21,'Enable','on');
set(handles.popupmenu21,'Value',1);

set(handles.popupmenu4,'Enable','on');
set(handles.popupmenu4,'Value',1);

set(handles.popupmenu6,'Enable','on');
set(handles.popupmenu6,'Value',1);

set(handles.popupmenu7,'Enable','on');
set(handles.popupmenu7,'Value',1);

set(handles.popupmenu8,'Enable','on');
set(handles.popupmenu8,'Value',1);

set(handles.checkbox1,'Enable','on');
set(handles.checkbox1,'Value',1);
set(handles.checkbox2,'Enable','on');
set(handles.checkbox2,'Value',1);

set(handles.popupmenu10,'Enable','on');
set(handles.popupmenu10,'Value',1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%2nd coll

set(handles.popupmenu17,'Enable','on');
set(handles.popupmenu17,'Value',1);

set(handles.edit36,'String',0);
set(handles.edit37,'String',0);
set(handles.edit38,'String',1);
set(handles.edit39,'String',-1);
set(handles.edit40,'String',0);
```

Appendix B. (Continued)

```
set (handles.edit41, 'String', 0);
set (handles.edit42, 'String', 0);
set (handles.edit43, 'String', -1);
set (handles.edit44, 'String', 0);

set (handles.edit36, 'Enable', 'on');
set (handles.edit37, 'Enable', 'on');
set (handles.edit38, 'Enable', 'on');
set (handles.edit39, 'Enable', 'on');
set (handles.edit40, 'Enable', 'on');
set (handles.edit41, 'Enable', 'on');
set (handles.edit42, 'Enable', 'on');
set (handles.edit43, 'Enable', 'on');
set (handles.edit44, 'Enable', 'on');

set (handles.edit15, 'String', 369);
set (handles.edit16, 'String', 1455);
set (handles.edit17, 'String', 999);

set (handles.edit15, 'Enable', 'on');
set (handles.edit16, 'Enable', 'on');
set (handles.edit17, 'Enable', 'on');

set (handles.text15, 'Enable', 'on');
set (handles.text16, 'Enable', 'on');
set (handles.text17, 'Enable', 'on');
set (handles.text18, 'Enable', 'on');

set (handles.edit45, 'String', 100);
set (handles.edit45, 'Enable', 'on');
set (handles.text14, 'Enable', 'on');
set (handles.text13, 'Enable', 'on');

set (handles.pushbutton3, 'Enable', 'on');
set (handles.popupmenu20, 'Enable', 'on');
set (handles.popupmenu20, 'Value', 1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

set (handles.edit62, 'String', 70);
set (handles.edit63, 'String', 70);
set (handles.edit64, 'String', -70);
set (handles.edit62, 'Enable', 'off');
set (handles.edit63, 'Enable', 'off');
set (handles.edit64, 'Enable', 'off');
set (handles.text21, 'Enable', 'off');

set (handles.edit65, 'String', 0.001);
set (handles.edit66, 'String', 0.001);
set (handles.edit67, 'String', 0.001);
set (handles.edit65, 'Enable', 'off');
set (handles.edit66, 'Enable', 'off');
set (handles.edit67, 'Enable', 'off');
set (handles.text22, 'Enable', 'off');

set (handles.edit4, 'String', 2);
set (handles.edit4, 'Enable', 'off');
set (handles.text6, 'Enable', 'off');
set (handles.text5, 'Enable', 'off');

set (handles.pushbutton4, 'Enable', 'off');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

set (handles.edit1, 'String', 50);
set (handles.edit1, 'Enable', 'off');
```

Appendix B. (Continued)

```
set(handles.text1,'Enable','off');
set(handles.text2,'Enable','off');

set(handles.edit69,'String',19711);
set(handles.edit69,'Enable','off');
set(handles.text23,'Enable','off');

set(handles.pushbutton7,'Enable','off');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%3rd coll

set(handles.popupmenu15,'Enable','on');
set(handles.popupmenu15,'Value',1);

set(handles.popupmenu16,'Enable','on');
set(handles.popupmenu16,'Value',1);

set(handles.edit49,'String',1.5708);
set(handles.edit50,'String',1.5708);
set(handles.edit51,'String',0);
set(handles.edit52,'String',1.5708);
set(handles.edit53,'String',1.5708);
set(handles.edit54,'String',1.5708);
set(handles.edit55,'String',0);

set(handles.edit49,'Enable','off');
set(handles.edit50,'Enable','off');
set(handles.edit51,'Enable','off');
set(handles.edit52,'Enable','off');
set(handles.edit53,'Enable','off');
set(handles.edit54,'Enable','off');
set(handles.edit55,'Enable','off');

set(handles.pushbutton5,'Enable','off');
set(handles.pushbutton6,'Enable','off');

set(handles.edit56,'String',0);
set(handles.edit57,'String',0);
set(handles.edit58,'String',0);

set(handles.edit56,'Enable','off');
set(handles.edit57,'Enable','off');
set(handles.edit58,'Enable','off');

set(handles.popupmenu11,'Enable','on');
set(handles.popupmenu11,'Value',1);

set(handles.popupmenu14,'Enable','on');
set(handles.popupmenu14,'Value',1);

set(handles.pushbutton2,'Enable','on');
set(handles.pushbutton1,'Enable','off');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Appendix C. C++ Programs and DLL Library

C.1. DLL Library Functions

Return Type	Function Name	Arguments
int32	brk	(int32)
int32	brkAll	
[int32, int32Ptr]	brkSelect	(int32Ptr)
int32	changeBaudRate	(int32)
int32	changeCommPort	(int32)
int32	clear	(int32)
int32	clearAll	
[int32, int32Ptr]	clearSelect	(int32Ptr)
int32	close	
int32	enable	(int32)
int32	enableAll	
[int32, int32Ptr]	enableSelect	(int32Ptr)
uint8	getAd	(int32)
[int32, int32Ptr]	getAdAll	(int32Ptr)
int32	getHm	(int32)
[int32, int32Ptr]	getHmAll	(int32Ptr)
int32	getPos	(int32)
[int32, int32Ptr]	getPosAll	(int32Ptr)
int32	getVel	(int32)
[int32, int32Ptr]	getVelAll	(int32Ptr)
int32	init	(int32, int32, int32)
int32	lastErrorCode	
cstring	lastErrorString	
int32	off	(int32)
int32	offAll	
[int32, int32Ptr]	offSelect	(int32Ptr)

Appendix C. (Continued)

Return Type	Function Name	Arguments
int32	pos	(int32, uint32, uint32, uint32, int32)
int32	posAll	(uint32, uint32, uint32, int32)
[int32, int32Ptr, uint32Ptr, uint32Ptr, uint32Ptr, int32Ptr]	posSelect	(int32Ptr, uint32Ptr, uint32Ptr, uint32Ptr, int32Ptr)
[int32, int32Ptr]	posSelectAll	(int32Ptr, uint32, uint32, uint32, int32)
int32	pwm	(int32, uint8, int32)
int32	pwmAll	(uint8, int32)
[int32, int32Ptr, uint8Ptr, int32Ptr]	pwmSelect	(int32Ptr, uint8Ptr, int32Ptr)
[int32, int32Ptr]	pwmSelectAll	(int32Ptr, uint8, int32)
int32	reset	(int32)
int32	resetAll	
[int32, int32Ptr]	resetSelect	(int32Ptr)
int32	setParams	(int32, int16, int16, int16, int16, int16, uint8, uint8, uint8, uint8)
int32	setParamsAll	(int16, int16, int16, int16, int16, uint8, uint8, uint8, uint8)
int32	setParamsLEL	(int32, int16, int16)
int32	setParamsLELAll	(int16, int16)
int32	setParamsLIMIT	(int32, uint8, uint8)
int32	setParamsLIMITAll	(uint8, uint8)
int32	setParamsPID	(int32, int16, int16, int16)
int32	setParamsPIDAll	(int16, int16, int16)
int32	setParamsRCM	(int32, uint8, uint8, uint8)
int32	setParamsRCMAll	(uint8, uint8, uint8)
int32	setPos	(int32, uint32, int32)
int32	setPosAll	(uint32, int32)
[int32, int32Ptr, uint32Ptr, int32Ptr]	setPosSelect	(int32Ptr, uint32Ptr, int32Ptr)
[int32, int32Ptr]	setPosSelectAll	(int32Ptr, uint32, int32)
[int32, int32Ptr]	status	(int32, int32Ptr)
[int32, int32Ptr]	statusAll	(int32Ptr)
int32	stop	(int32)
int32	stopAll	
[int32, int32Ptr]	stopSelect	(int32Ptr)
int32	vel	(int32, uint32, uint32, int32)
int32	velAll	(uint32, uint32, int32)
[int32, int32Ptr, uint32Ptr, uint32Ptr, int32Ptr]	velSelect	(int32Ptr, uint32Ptr, uint32Ptr, int32Ptr)
[int32, int32Ptr]	velSelectAll	(int32Ptr, uint32, uint32, int32)

Appendix C. (Continued)

C.2. DLL Library Documentation

controlMotor.dll is a Windows DLL that includes functions for communicating with PIC-SERVO (v.4, v.5 and v.10) modules. It can be used with almost all Windows programming languages. This DLL was created using Microsoft Visual Studio 2003, and the source code is included with the DLL. This portion is being developed by Mayur Palankar.

Initialization/Closing

1) Initialize PIC-SERVO Modules

Opens the COM port and initializes the PIC-SERVO modules.

Command:

init

Syntax:

int init (int CommPort, long BaudRate, int fileOpen);

Return Value:

Negative number or zero: Error or Failure (Corresponds to a unique error number) or zero if no modules found

Positive number: Success. Number corresponds to the number of modules found in the system.

Arguments:

CommPort: COM port number (1 to 8)

BaudRate: Communication rate (19200, 57600, 115200 or 230400)

fileOpen: Set PIC-SERVO modules based on the last stored configuration.(0 or 1)

Description:

Opens the COM port specified by CommPort at the specified BaudRate, and initializes the network of motor controllers. Controller addresses are dynamically assigned, starting with the furthest controller with address 1. All group addresses are set to 0xFF. Returns the number of controllers found on in the network. The PIC-SERVO modules are initialized to the last saved configuration if the fileOpen argument is passed a positive number else they are initialized to zero. Please note that this doesn't mean the arm will move; this means the modules is assumed to be at the initial configuration that was saved previously.

CommPort and BaudRate: These arguments tell which communication port and the rate at which it should communicate to it.

Appendix C. (Continued)

fileOpen: If a positive number is passed, the old configuration last stored (when close() is executed) will be restored. This is done using a file 'configuration.txt' which is local to this dll. If the file is not found, initial position are used. Any changes manually made to the file will also be reflected if the option is chosen.

Examples:

```
init(4, 19200, 0)
```

This command will try to initialize the PIC-SERVO modules at COMM port 4 with a baud rate 19200 and they will be initialized at their start position (fileOpen = 0).

2) Close communication

Saves the current configuration, resets all the buffers and gracefully closes the COMM port.

Command:

```
close
```

Syntax:

```
int close ();
```

Return Value:

Negative number or zero: Error or Failure (Corresponds to a unique error number)

Positive number: Success.

Description:

This command saves the current configuration in the 'configuration.txt' file which is local to the dll. This file is used to open the PIC-SERVO modules in the previous configuration. It resets all the PIC-SERVO modules and shuts it down.

NOTE: Any functions of this dll can't be executed unless the init function is executed. After that, all functions (except init) can be executed. A new init function can only be executed unless the old connection is closed first.

CAUTION: The init command creates threads for internal usage for each of the PIC-SERVO modules found. For graceful shutdown, the close() should be executed. If the parent process using the dll doesn't execute the close() before exiting, the lock on the PIC-SERVO modules still exist causing the system to behave unpredictably. This responsibility rests solely on the programmer using this dll.

Appendix C. (Continued)

Reset/Clear Motor

1) Reset

Resets a particular PIC-SERVO module or group of modules.

Commands included:

reset
resetAll
resetSelect

Syntax:

```
int reset (int module);  
int resetAll ();  
int resetSelect (int module[]);
```

Return Value:

Negative number or zero: Error or Failure (Corresponds to a unique error number)
Positive number: Success

Arguments:

module/module[]: module address (1 to 32)

Description:

This command resets a PIC-SERVO motor module to its start up status.

module: This argument describes the PIC-SERVO module address which has to be reseted. The number sent should be less than or equal to the number of modules present else it will result in an error (error code: -1). When an array of module number is passed, the index is used to address the module and its value decides if the motor module will be reseted or not. For ex. [1 0 1 1] when passed, will reset the motor module numbers 1, 3 and 4; while the motor module number 2 wont be affected. The length of the array (in this case 4) should be less than or equal to the modules present in the circuit.

Types:

reset: This command is used to reset one PIC-SERVO module.

resetAll: This command resets all the PIC-SERVO modules.

resetSelect: Using this command, individual PIC-SERVO modules can be reseted. This command has the highest flexibility and the others are a special case of this command.

Examples:

```
reset(1)
```

Resets the PIC-SERVO module 1 to its start up state.

Appendix C. (Continued)

2) Clears

Clears a particular PIC-SERVO module or group of modules.

Commands included:

clear
clearAll
clearSelect

Syntax:

```
int clear (int module);  
int clearAll ();  
int clearSelect (int module[]);
```

Return Value:

Negative number or zero: Error or Failure (Corresponds to a unique error number)

Positive number: Success

Arguments:

module/module[]: module address (1 to 32)

Description:

This command clears a PIC-SERVO motor module's status bits.

module: This argument describes the PIC-SERVO module address whose status bits have to be cleared. The number sent should be less than or equal to the number of modules present else it will result in an error (error code: -1). When an array of module number is passed, the index is used to address the module and its value decides if the motor module will be cleared or not. For ex. [1 0 1 1] when passed, will clear the motor module numbers 1, 3 and 4; while the motor module number 2 wont be affected. The length of the array (in this case 4) should be less than or equal to the modules present in the circuit.

Types:

clear: This command is used to clear one PIC-SERVO module.

clearAll: This command clears all the PIC-SERVO modules.

clearSelect: Using this command, individual PIC-SERVO modules can be cleared. This command has the highest flexibility and the others are a special case of this command.

Examples:

```
clear(1)
```

Clears the PIC-SERVO module 1's status bits.

Appendix C. (Continued)

Enable/Disable Motors

1) Enable

Enables a particular PIC-SERVO module or group of modules.

Commands included:

enable
enableAll
enableSelect

Syntax:

```
int enable (int module);  
int enableAll ();  
int enableSelect (int module[]);
```

Return Value:

Negative number or zero: Error or Failure (Corresponds to a unique error number)

Positive number: Success

Arguments:

module/module[]: module address (1 to 32)

Description:

This command enables a PIC-SERVO motor module. Any move command can't be executed if the motor modules are disabled.

module: This argument describes the PIC-SERVO module address which has to be enabled. The number sent should be less than or equal to the number of modules present else it will result in an error (error code: -1). When an array of module numbers is passed, the index is used to address the module and its value decides if the motor module will be enabled or not. For ex. [1 0 1 1] when passed, will enable the motor module numbers 1, 3 and 4; while the motor module number 2 wont be affected. The length of the array (in this case 4) should be less than or equal to the modules present in the circuit.

Types:

enable: This command is used to enable one PIC-SERVO module.

enableAll: This command enables all the PIC-SERVO modules.

enableSelect: Using this command, individual PIC-SERVO modules can be enabled. This command has the highest flexibility and the others are a special case of this command.

Examples:

```
enable(1)
```

Enables the PIC-SERVO module 1.

Appendix C. (Continued)

2) Off

Disables a particular PIC-SERVO module or group of modules.

Commands included:

off

offAll

offSelect

Syntax:

int off (int module);

int offAll ();

int offSelect (int module[]);

Return Value:

Negative number or zero: Error or Failure (Corresponds to a unique error number)

Positive number: Success

Arguments:

module/module[]: module address (1 to 32)

Description:

This command disables a PIC-SERVO motor module. Any move command can now be executed.

module: This argument describes the PIC-SERVO module address which has to be disabled. The number sent should be less than or equal to the number of modules present else it will result in an error (error code: -1). When an array of module number is passed, the index is used to address the module and its value decides if the motor module will be disabled or not. For ex. [1 0 1 1] when passed, will disable the motor module numbers 1, 3 and 4; while the motor module number 2 wont be affected. The length of the array (in this case 4) should be less than or equal to the modules present in the circuit.

Types:

off: This command is used to disable one PIC-SERVO module.

offAll: This command disables all the PIC-SERVO modules.

offSelect: Using this command, individual PIC-SERVO modules can be disabled. This command has the highest flexibility and the others are a special case of this command.

Examples:

off(1)

Disables the PIC-SERVO module 1.

Appendix C. (Continued)

Move Commands

1)Position Control

Loads a motion trajectory to move to a certain position.

Commands included:

pos
posAll
posSelect
posSelectAll

Syntax:

int pos (int module, unsigned long pos, unsigned long vel, unsigned long acc, int rev);
int posAll (unsigned long pos, unsigned long vel, unsigned long acc, int rev);
int posSelect (int module[], unsigned long pos[], unsigned long vel[], unsigned long acc[], int rev[]);
int posSelectAll (int module[], unsigned long pos, unsigned long vel, unsigned long acc, int rev);

Return Value:

Negative number or zero: Error or Failure (Corresponds to a unique error number)

Positive number: Success

Arguments:

module/module[]: module address (1 to 32)
pos / pos[] : positive 32 bit position data
(0 to +2,147,483,647)
vel / vel[] : positive 32 bit velocity data
(0 to +83,886,080)
acc / acc[] : positive 32 bit acceleration data
(0 to +2,147,483,647)
rev / rev[] : reverse motion (0 or 1)

Description:

This command sends the position, velocity and acceleration data needed for a particular motion to the appropriate PIC-SERVO motor module.

module: This argument describes the PIC-SERVO module address where the corresponding trajectory information has to be sent. The number sent should be less than or equal to the number of modules present else it will result in an error (error code: -1). When an array of module number is passed, the index is used to address the module and its value decides if the motor module will be enabled or disabled. For ex. [1 0 1 1] when passed, will load the corresponding trajectory information to the module numbers 1, 3

Appendix C. (Continued)

and 4; while the module number 2 wont be affected. The length of the array (in this case 4) should be less than or equal to the modules present in the circuit. Array elements greater then the number of modules present will be discarded. Care should be taken so that the corresponding entries of module array matches with the pos, vel and acc arguments and is left solely on the user.

pos, vel and acc: Data for the motion trajectory.

rev: This argument determines the direction in which the motor moves. When its value is 1 or a positive number, the direction will taken as reverse and the sign of the position field will be reversed. When 0 the direction will be taken as forward.

Types:

pos: This command is used to control one PIC-SERVO module.

posAll: This command controls all the PIC-SERVO modules and moves all of them to the same position with the same velocity and acceleration in the same direction.

posSelect: Using this command, individual PIC-SERVO modules can be controlled to move to their corresponding positions with their own corresponding velocity, acceleration and direction. This command has the highest flexibility and the others are a special case of this command.

posSelectAll: Similar to the above one but the trajectory for all selected modules will the same.

Examples:

pos(1, 100000, 10000, 100, 1)

Moves the PIC-SERVO module 1 to the position 100000 with velocity 10000 and acceleration 100. The direction is reverse.

posSelect([1 0 0 1], [200000 100 100000 100000], [10000 0 0 300000], [100 0 100 300], [0 1 1 1])

Moves the PIC-SERVO module 1 to the position 200000 in the forward direction (velocity 100000 and acceleration 100) and moves the PIC-SERVO module 4 to the position 100000 in the reverse direction (velocity 300000 and acceleration 300). The PIC-SERVO module 2 and 3 won't have any effect and its entries will be discarded.

2)Velocity Control

Loads a motion trajectory to move with a certain velocity.

Commands included:

vel

velAll

velSelect

velSelectAll

Appendix C. (Continued)

Syntax:

```
int vel (int module, unsigned long vel, unsigned long acc, int rev);
int velAll (unsigned long vel, unsigned long acc, int rev);
int velSelect (int module[], unsigned long vel[], unsigned long acc[], int rev[]);
int velSelectAll (int module[], unsigned long vel, unsigned long acc, int rev);
```

Return Value:

Negative number or zero: Error or Failure (Corresponds to a unique error number)
Positive number: Success

Arguments:

module/module[]: module address (1 to 32)
vel / vel[] : positive 32 bit velocity data
(0 to +83,886,080)
acc / acc[] : positive 32 bit acceleration data
(0 to +2,147,483,647)
rev / rev[] : reverse motion (0 or 1)

Description:

This command sends the velocity and acceleration data needed for a particular motion to the appropriate PIC-SERVO motor module.

module: This argument describes the PIC-SERVO module address where the corresponding trajectory information has to be sent. The number sent should be less than or equal to the number of modules present else it will result in an error (error code: -1). When an array of module number is passed, the index is used to address the module and its value decides if the motor module will be enabled or disabled. For ex. [1 0 1 1] when passed, will load the corresponding trajectory information to the module numbers 1, 3 and 4; while the module number 2 won't be affected. The length of the array (in this case 4) should be less than or equal to the modules present in the circuit. Array elements greater than the number of modules present will be discarded. Care should be taken so that the corresponding entries of module array matches with the vel and acc arguments and is left solely on the user.

vel and acc: Data for the motion trajectory.

rev: This argument determines the direction in which the motor moves. When its value is 1 or a positive number, the direction will be taken as reverse. When 0 the direction will be taken as forward.

Types:

vel: This command is used to control one PIC-SERVO module.

Appendix C. (Continued)

velAll: This command controls all the PIC-SERVO modules and moves all of them with the same velocity and acceleration in the same direction.

velSelect: Using this command, individual PIC-SERVO modules can be controlled to move with their own corresponding velocity, acceleration and direction. This command has the highest flexibility and the others are a special case of this command.

velSelectAll: Similar to the above one but the trajectory for all selected modules will be the same.

Examples:

```
vel(1, 10000, 100, 1)
```

Moves the PIC-SERVO module 1 with velocity 10000 and acceleration 100. The direction is reverse.

```
velSelect([1 0 0 1], [10000 1000 30 300000], [100 1 100 300], [0 1 1 1])
```

Moves the PIC-SERVO module 1 in the forward direction with velocity 100000 and acceleration 100 and moves the PIC-SERVO module 4 in the reverse direction with velocity 300000 and acceleration 300. The PIC-SERVO module 2 and 3 won't have any effect and its corresponding entries will be discarded.

3)PWM Control

Loads a motion trajectory to move with certain PWM information.

Commands included:

pwm

pwmAll

pwmSelect

pwmSelectAll

Syntax:

```
int pwm (int module, unsigned char pwm, int rev);
```

```
int pwmAll (unsigned char pwm, int rev);
```

```
int pwmSelect (int module[], unsigned char pwm[], int rev[]);
```

```
int pwmSelectAll (int module[], unsigned char pwm, int rev);
```

Return Value:

Negative number or zero: Error or Failure (Corresponds to a unique error number)

Positive number: Success

Arguments:

module/module[]: module address (1 to 32)

pwm / pwm[] : positive 8 bit PWM data (0 to +255)

rev / rev[] : reverse motion (0 or 1)

Appendix C. (Continued)

Description:

This command sends the PWM data needed for a particular motion to the appropriate PIC-SERVO motor module.

module: This argument describes the PIC-SERVO module address where the corresponding PWM information has to be sent. The number sent should be less than or equal to the number of modules present else it will result in an error (error code: -1). When an array of module number is passed, the index is used to address the module and its value decides if the motor module will be enabled or disabled. For ex. [1 0 1 1] when passed, will load the corresponding trajectory information to the module numbers 1, 3 and 4; while the module number 2 won't be affected. The length of the array (in this case 4) should be less than or equal to the modules present in the circuit. Array elements greater than the number of modules present will be discarded. Care should be taken so that the corresponding entries of module array matches with the vel and acc arguments and is left solely on the user.

pwm: Data for the motion trajectory.

rev: This argument determines the direction in which the motor moves. When its value is 1 or a positive number, the direction will be taken as reverse. When 0 the direction will be taken as forward.

Types:

pwm: This command is used to control one PIC-SERVO module.

pwmAll: This command controls all the PIC-SERVO modules and moves all of them with the same PWM information.

pwmSelect: Using this command, individual PIC-SERVO modules can be controlled to move with their own PWM information. This command has the highest flexibility and the others are a special case of this command.

pwmSelectAll: Similar to the above one but the trajectory for all selected modules will be the same.

Examples:

```
pwm(1, 100, 1)
```

Moves the PIC-SERVO module 1 with PWM 100. The direction is reverse.

```
pwmSelect([1 0 0 1], [100 1 100 200], [0 1 1 1])
```

Moves the PIC-SERVO module 1 in the forward direction with PWM 100 and moves the PIC-SERVO module 4 in the reverse direction with PWM 200. The PIC-SERVO module 2 and 3 won't have any effect and its corresponding entries will be discarded.

Appendix C. (Continued)

Stop Commands

1)Stop

De-accelerates a PIC-SERVO module to a complete stop.

Commands included:

stop
stopAll
stopSelect

Syntax:

```
int stop (int module);  
int stopAll ();  
int stopSelect (int module[]);
```

Return Value:

Negative number or zero: Error or Failure (Corresponds to a unique error number)

Positive number: Success

Arguments:

module/module[]: module address (1 to 32)

Description:

This command de-accelerates a moving PIC-SERVO motor module to a complete stop. The de-acceleration will be the same amount with which the motor was moving at the time of execution. If the motor is already stopped, the command will have no effect.

module: This argument describes the PIC-SERVO module address which has to be stopped. The number sent should be less than or equal to the number of modules present else it will result in an error (error code: -1). When an array of module number is passed, the index is used to address the module and its value decides if the motor module will be stopped or left to run. For ex. [1 0 1 1] when passed, will stop the motor module numbers 1, 3 and 4; while the motor module number 2 wont be affected. The length of the array (in this case 4) should be less than or equal to the modules present in the circuit.

Types:

stop: This command is used to stop one PIC-SERVO module.

stopAll: This command stops all the PIC-SERVO modules.

stopSelect: Using this command, individual PIC-SERVO modules can be stopped. This command has the highest flexibility and the others are a special case of this command.

Examples:

stop(1)

De-accelerates the PIC-SERVO module 1 to a complete stop.

Appendix C. (Continued)

stopAll()

De-accelerates all the PIC-SERVO modules to a complete stop.

stopSelect([1 0 1 0])

De-accelerates the PIC-SERVO modules 1 and 3 to a complete stop. PIC-SERVO modules 2 and 4 won't be affected.

2)Break

Immediately stops a PIC-SERVO module.

Commands included:

brk

brkAll

brkSelect

Syntax:

int brk (int module);

int brkAll ();

int brkSelect (int module[]);

Return Value:

Negative number or zero: Error or Failure (Corresponds to a unique error number)

Positive number: Success

Arguments:

module/module[]: module address (1 to 32)

Description:

This command Immediately stops a moving PIC-SERVO motor module. If the motor is already stopped, the command will have no effect.

module: This argument describes the PIC-SERVO module address which has to be stopped. The number sent should be less than or equal to the number of modules present else it will result in an error (error code: -1). When an array of module number is passed, the index is used to address the module and its value decides if the motor module will be stopped or left to run. For ex. [1 0 1 1] when passed, will stop the motor module numbers 1, 3 and 4; while the motor module number 2 wont be affected. The length of the array (in this case 4) should be less than or equal to the modules present in the circuit.

Types:

brk: This command is used to stop one PIC-SERVO module.

brkAll: This command stops all the PIC-SERVO modules.

brkSelect: Using this command, individual PIC-SERVO modules can be stopped. This command has the highest flexibility and the others are a special case of this command.

Appendix C. (Continued)

Examples:

`brk(1)`

Immediately stops the PIC-SERVO module 1.

`brkAll()`

Immediately stops all the PIC-SERVO modules.

`brkSelect([1 0 1 0])`

Immediately stops the PIC-SERVO modules 1 and 3. PIC-SERVO modules 2 and 4 won't be affected.

Appendix C. (Continued)

Status Commands

1) Individual Parameters per PIC-SERVO

Returns the status of the individual parameter for a particular PIC-SERVO module.

Commands included:

getPos
getVel
getAd
getHm

Syntax:

```
long getPos (int module);  
long getVel (int module);  
unsigned char getAd (int module);  
long getHm (int module);
```

Return Value:

Negative number: Error or Failure (Corresponds to a unique error number)
Positive number or zero: Corresponding status.

Arguments:

module/module[]: module address (1 to 32)

Description:

This command returns the specific status parameter that was asked for the particular PIC-SERVO module.

module: This argument describes the PIC-SERVO module address whose status has to be read. The number sent should be less than or equal to the number of modules present else it will result in an error (error code: -1).

Types:

getPos: This command returns the current position of the PIC-SERVO module.
getVel: This command returns the current velocity of the PIC-SERVO module.
getAd: This command returns the current A/D value of the PIC-SERVO module.
getHm: This command returns the current motor home position of the PIC-SERVO module.

Examples:

```
getPos(1)  
Returns the current position of PIC-SERVO module 1.
```

Appendix C. (Continued)

2) Individual Parameters

Returns the status of the individual parameter for all PIC-SERVO modules.

Commands included:

getPosAll
getVelAll
getAdAll
getHmAll

Syntax:

```
int getPosAll (long * x);  
int getVelAll (long * x);  
int getAdAll (long * x);  
int getHmAll (long * x);
```

Return Value:

Negative number: Error or Failure (Corresponds to a unique error number)

Positive number or zero: Success.

Arguments:

(*) x: A pointer to the array which has the status values.

Description:

This command returns the specific status parameter for all PIC-SERVO modules.

x: A pointer to the array which has the status values for all the PIC-SERVO modules. The size of the array will be equal to the modules present in the system.

Types:

getPosAll: This command returns the current position of all the PIC-SERVO modules.

getVelAll: This command returns the current velocity of all the PIC-SERVO modules.

getAdAll: This command returns the current A/D value of all the PIC-SERVO modules.

getHmAll: This command returns the current motor home position of all the PIC-SERVO modules.

Examples:

```
getPosAll(1)
```

Returns the current position of all PIC-SERVO modules.

```
getVelAll(1)
```

Returns the current velocities of all PIC-SERVO modules.

3) Complete Status

Returns the status of all parameters for all or one PIC-SERVO modules.

Appendix C. (Continued)

Commands included:

status
statusAll

Syntax:

int status (int module, long * x);
int statusAll (long * x);

Return Value:

Negative number: Error or Failure (Corresponds to a unique error number)
Positive number or zero: Success.

Arguments:

module: module address (1 to 32)
(* x): A pointer to the array which has the status values.

Description:

This command returns the all status parameters for all or one PIC-SERVO modules.

x: A pointer to the array which has the status values for all the PIC-SERVO modules. The size of the array will be either 4 or 4 * (modules present in the system).

Types:

status: This command returns all the parameters for a particular PIC-SERVO module.
statusAll: This command returns all the parameters for all PIC-SERVO modules.

Appendix C. (Continued)

Set Motor

Changes the position value of the PIC-SERVO modules.

Commands included:

setPos
setPosAll
setPosSelect
setPosSelectAll

Syntax:

```
int setPos (int module, unsigned long pos, int rev);  
int setPosAll (unsigned long pos, int rev);  
int setPosSelect (int module[], unsigned long pos[], int rev[]);  
int setPosSelectAll (int module[], unsigned long pos, int rev);
```

Return Value:

Negative number: Error or Failure (Corresponds to a unique error number)
Positive number or zero: Success.

Arguments:

module/module[]: module address (1 to 32)
pos / pos[] : positive 32 bit position data
(0 to +2,147,483,647)
rev / rev[] : reverse motion (0 or 1)

Description:

This command sets the position variable for a specific PIC-SERVO module or a set of modules.

module: This argument describes the PIC-SERVO module address whose position variable has to be changed. The number sent should be less than or equal to the number of modules present else it will result in an error (error code: -1). When an array of module number is passed, the index is used to address the module and its value decides if the motor module's position will be changed or not. For ex. [1 0 1 1] when passed, the position variable for the module numbers 1, 3 and 4 will be changed; while of the module number 2 wont be affected. The length of the array (in this case 4) should be less than or equal to the modules present in the circuit. Array elements greater then the number of modules present will be discarded. Care should be taken so that the corresponding entries of module array matches with the pos argument and is left solely on the user.

Appendix C. (Continued)

pos: New data.

rev: This argument determines the direction in which the motor is present. When its value is 1 or a positive number, the direction will taken as reverse and the sign of the position field will be reversed. When 0 the direction will be taken as forward.

Types:

setPos: This command sets the position variable of one PIC-SERVO module.

setPosAll: This command sets the position variable of all PIC-SERVO modules with the same position and direction.

setPosSelect: Using this command, individual PIC-SERVO modules can be controlled to change to their corresponding positions and direction. This command has the highest flexibility and the others are a special case of this command.

setPosSelectAll: Similar to the above one but the position variable changed for the selected modules is the same.

Examples:

setPos(1, 100000, 1)

Sets the position of the PIC-SERVO module 1 to 100000.

setPosSelect([1 0 0 1], [200000 100 100000 100000], [0 1 1 1])

Sets the position variable of PIC-SERVO module 1 to position 200000 in the forward direction and sets the position of PIC-SERVO module 4 to position 100000 in the reverse direction. The PIC-SERVO module 2 and 3 won't have any effect and its corresponding entries will be discarded.

About the Author

Redwan M. Alqasemi graduated with honor from King Abdulaziz University in Jeddah, Saudi Arabia with his Bachelor degree in Mechanical Engineering in 1995. He was awarded the Best University Student award for achievements during his undergraduate studies. He worked at the Pepsi, and then moved to Wichita, Kansas in 1996 and got his Master's degree from Wichita State University in 2001. He taught several labs, and was awarded the best project award by Boeing and Raytheon aircraft companies for building a wall-climbing robot for aircraft maintenance.

Redwan moved to Tampa, Florida in 2001 to pursue his Ph.D. degree in the field of Rehabilitation Robotics. He taught several courses and labs, and worked in the Center for Rehabilitation Engineering and Technology as the leader of their research group. He completed his Ph.D. degree in the field of combining mobility and manipulation of wheelchair-mounted robotic arms in 2007.

Redwan is a member of Tau-Beta-Pi and Phi-Kappa-Phi honor societies. He is also a member of ASME and IEEE societies. He has been actively involved in research and has published many papers in many prestigious journals and conferences.