7-6-2010

Low-Level Programming of a 9-Degree-of-Freedom Wheelchair-Mounted Robotic Arm with the Application of Different User Interfaces

Punya A. Basnayaka
University of South Florida

Low-Level Programming of a 9-Degree-of-Freedom Wheelchair-Mounted Robotic Arm

with the Application of Different User Interfaces

by

Punya A. Basnayaka

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Mechanical Engineering
Department of Mechanical Engineering
College of Engineering
University of South Florida

Major Professor: Rajiv Dubey, Ph.D.
Kyle B. Reed, Ph.D.
Redwan M. Alqasemi, Ph.D.
Shuh - Jing Ying, Ph.D.

Date of Approval:
July 6, 2010

Keywords: PID, GUI, rehabilitation robots, C++ functions, accuracy, repeatability

Dedication

This thesis is dedicated to my wonderful parents, who have raised me to be the person I am today and to my husband. You have been with me every step of the way, through good times and bad. Thank you for all the guidance and motivation that you have always given me.

Acknowledgements

It is my pleasure to acknowledge my supervisor, Dr. Rajiv Dubey for giving me the opportunity to carry out this research under his advice and supervision. I also take this opportunity to thank Dr. Redwan M. Alqasemi for his helpful guidance and encouragement throughout this research. I would also like to acknowledge Dr. Kyle B. Reed and Dr. Shuh–Jing Ying for being my committee members and for their valuable advice in different occasions.

I would like to thank all the members of the Assistive and Rehabilitation Robotics Technologies Center at USF including Dr. Kathryn De Laurentis and Dr. Stephanie Carey for their help. I am thankful to the USF WMRA research group including Garrett Pence, Ana Torres, John Capille, Mario Simoes and Fabian Farelo and all the other lab mates including Karan Khokar and Michelle Smith who helped me during different stages of this work.

.

Table of Contents

List of Tables iii

List of Figures iv

Abstract vi

Chapter 1: Introduction 1
1.1. Motivation 1
1.2. Thesis Objectives 2
1.3. Thesis Outline 3

Chapter 2: Background 4
2.1. Main Types of Rehabilitation Robots 4
2.1.1. Workstation Robots 5
2.1.2. Stand Alone Manipulators 7
2.1.3. Wheelchair-Mounted Robotic Systems 8
2.1.4. Mobile Robots 10
2.2. Robot Programming Languages 10
2.2.1. C++ Software Architecture of USF WMRA-I 12
2.2.2. Matlab Software Architecture of USF WMRA-I 12
2.3. Human Machine Interface 13
2.4. Safety 16
2.5. Overview of USF WMRA-I 16

Chapter 3: Closed Loop Motion Control System of USF WMRA-II 18
3.1. Introduction 18
3.2. Motion Controller 18
3.3. Motors 19
3.4. Position Sensor (Encoder) 19
3.5. Computer 20
3.6. PID Feedback Controller 21
3.6.1. Overview of Tuning Methods 22
3.6.2. PID Tuning Software 22

Chapter 4: Denavit-Hartenberg Parameters and Hardware of USF WMRA-II 24
4.1. Denavit-Hartenberg Rules 24
4.2. DH Parameter Assignment for USF WMRA-II 26
4.3. Hardware Architecture 27
4.3.1. Power Wheelchair 27
4.3.2. 7-DOF Robotic Arm 28
4.3.3. Power Supply 29
4.3.4. Controller Board 29

Chapter 5: Calibration and Programming of USF WMRA-II 31
5.1. Axis Calibration of WMRA-II 31

5.2. Calculation of Velocity and Acceleration 32
5.2.1. Position Control 33
5.3. PID Gains Setting of WMRA-II 34
5.3.1. Tuning Procedure 34
5.3.2. Scope 35
5.3.3. Auto Tuning 35
5.3.4. Manual Tuning 36
5.4. C++ Based Programming 37
5.5. High-Level Control Programming 37
5.6. C++ Low-Level Control Programming 38
5.6.1. Establishing the Communication of the Controller Board 39
5.6.2. Parameter Initialization 40
5.6.3. Motion Control Functions 40
5.6.4. Data Record 41
5.7. Matlab Low-Level Control Programming 42
5.8. User Interfaces 42
5.8.1. Touch-Screen 42
5.8.2. Spaceball 43
5.9. Test GUI for WMRA-II 44

Chapter 6: Testing and Evaluation of WMRA-II 46
6.1. Power Usage of the Robotic Arm 46
6.2. Repeatability and Accuracy of the Robotic Arm 47
6.2.1. Methodology 48
6.3. Use of Different Interfaces 51
6.3.1. Touch-Screen Evaluation 52
6.3.1.1. Testing Procedure and Results 52
6.3.2. Spaceball Evaluation 52
6.3.2.1. Testing Procedure and Results 53
6.4. Manipulability Measure of WMRA-II 53

Chapter 7: Conclusion 55

References 57

Appendices 60
Appendix A. C++ Programs 61
Appendix B. C++ Touch-Screen GUI Program 139

About the Author ....End Page

List of Tables

Table 1: Interface Devices Matching with Body Functions [20] 15

Table 2: WMRA-I Specifications [26].....17

Table 3: DH-Parameter Description 26

Table 4: DH-Parameter Comparison of Two Robotic Arms 27

Table 5: Specifications of the Wheelchair 28

Table 6: Data from Manufacturer's Datasheets 32

Table 7: Radians to Encoder Counts Conversion Factors 32

Table 8: Motion Profile Parameter Notation 33

Table 9: PID Gain Parameters of Motors 36

Table 10: Prototypes of WMRA-I and WMRA-II C++ Functions 38

Table 11: C++ Function Prototypes of ArmMotion Function 38

Table 12: C++ Controller Board Function for PID Gains Setting 40

Table 13: Controller Board Function for Position Control 41

Table 14: Function Example of Data Record 41

Table 15: System Specifications of WMRA-II 46

Table 16: Power Usage of WMRA-II 47

Table 17: Power Consumption Comparison of WMRA-II with WMRA-I.....47

Table 18: Selected Points for the Accuracy and Repeatability Test 49

Table 19: Repeatability Comparison of WMRA-I and WMRA-II.....50

Table 20: Average Percentage Relative Error of WMRA-I and WMRA-II 50

List of Figures

Figure 1: ProVAR (track-mounted upside-down PUMA-260) [4] 5

Figure 2: RAID Workstation Layout [6], [35].....6

Figure 3: My Spoon (left) and Handy-1 (right) Feeding Aids [8] 7

Figure 4: Handy-1's Washing, Shaving and Teeth Cleaning Tray (left)
and Make-Up Tray (right) [8].....7

Figure 5: Manus Robotic Arm [34], [35].....8

Figure 6: University of Bremen's FRIEND II Exhibited at ICORR'07 [35] 9

Figure 7: Care-O-Bot Serving Food (operated fully autonomously).....10

Figure 8: Basic Elements of a WMRA System 13

Figure 9: Head Tracker 15

Figure 10: WMRA-I [7].....17

Figure 11: Close Loop Motion Control System Block Diagram 18

Figure 12: Quadrature Detection of an Encoder and Elements of Optical Encoder [28] 20

Figure 13: PID Block Diagram of Galil Tuner [37] 22

Figure 14: Link Frames Assignment [22] 25

Figure 15: Frame Assignment of WMRA-II 26

Figure 16: Quickie 626 Wheelchair 28

Figure 17: CAD Drawing of WMRA-II Robotic Arm 29

Figure 18: WMRA-II Robotic Arm 29

Figure 19: Controller Board (Galil DMC-2183).....30

Figure 20: Labeling of Motors of the Robotic Arm.....31

Figure 21: Motion Profile Diagram.....33

Figure 22: Step Responses of Motor G.....35

Figure 23: Galil Tuner 36
Figure 24: Position and Velocity Profiles for Position Control 41
Figure 25: Screen Shot of Touch-Screen Control User Interface 43
Figure 26: Spaceball and Its Demo Program Window 44
Figure 27: Matlab-Spaceball Program Window 44
Figure 28: Test GUI for WMRA-II 45
Figure 29: Representation of Accuracy and Repeatability [33] 48
Figure 30: Experimental Setup for Repeatability and Accuracy Test 49
Figure 31: Repeatability and Accuracy of WMRA-I and WMRA-II 50
Figure 32: Evaluated User Interfaces: Touch-Screen and Spaceball 51
Figure 33: Manipulability Measure of WMRA-I and WMRA-II Robotic Arms 54

Low-Level Programming of a 9-Degree-of-Freedom Wheelchair-Mounted Robotic Arm with the Application of Different User Interfaces

Punya A. Basnayaka

Abstract

Implementation of C++ and Matlab based control of University of South Florida's latest wheelchair-mounted robotic arm (USF WMRA-II), which has 9 degrees-of-freedom, was carried out under this Master's thesis research. First, the rotational displacements about the 7 joints of the robotic arm were calibrated. It was followed by setting the control gains of the motors. Then existing high-level programs developed using C++ and Matlab for USF WMRA-I were modified for WMRA-II. The required low-level programs to provide complete kinematics of the joint movements to the controller board of WMRA-II (Galil DMC-2183) were developed using C++. A test GUI was developed using C++ to troubleshoot the control program and to evaluate the operation of the robotic arm. It was found that WMRA-II has higher repeatability, accuracy and manipulability as well as lower power consumption than WMRA-I. Touch-Screen and Spaceball user interfaces were successfully implemented to facilitate people with different disabilities.

Chapter 1:

Introduction

1.1. Motivation

Wheelchair-Mounted Robotic Arms (WMRAs) are used as assistive devices for people with manipulative and locomotive disabilities in performing their activities of daily living (ADL). Based on the combined mobility of the wheelchair and the manipulation of the robotic arm, these devices provide multiple degrees-of-freedom (DOF) for movement. Since most wheelchairs are non-holonomic and carry only 2 DOF, the use of robotic arms on the non-holonomic wheelchair platform can add extra DOF for better manipulation.

The Center for Assistive and Rehabilitation Robotics Technologies at the University of South Florida (USF) has been working on WMRAs with increased DOF. USF WMRA-I has 9 DOF (7-DOF robotic arm mounted on a 2-DOF power wheelchair), which give it the capability of performing sophisticated ADL tasks that the other commercial WMRAs cannot perform. USF WMRA-II has been fabricated with an improvised robotic arm, which employs more precise motors and a robust single compact controller board (Galil DMC-2183), with the intention of enhancing the accuracy of motion. Its workspace has also been expanded by changing the link lengths to reach objects placed further away.

Development of a WMRA has two major phases; (i) design and fabrication, and (ii) development of computer programs to control its operation. In programming WMRAs, communication between hardware (robotic arm and the wheelchair) and software (control program) is achieved through a controller board. The optimum kinematics for the required motion is calculated in the software and sent to the controller board, which eventually translates that information to voltages and currents to be supplied to the motors at each joint. WMRA-II had been newly fabricated and the control programs had to be developed ensuring that the intended

improvements are achieved. Furthermore, accessibility of differently disabled persons to WMRA-II had to be enabled by integrating suitable user interfaces.

1.2. Thesis Objectives

WMRA-I had originally been programmed using Matlab, with a C++ dynamic link library (dll) to communicate with the controller board. Matlab had been used as a convenient tool for accurate evaluation of mathematical equations and simulation. C++ had been used to communicate with the controller board, because it is widely used in embedded system applications as a low-level language that provides fast and efficient execution. However, the control operation of WMRA-I had been often obstructed by failure of the communication between the Matlab program and the C++ dll library. Therefore, the programming approach had to be changed to attain continuous controllability of the robotic arm. The Matlab codes developed to control WMRA-I were converted to C++ to achieve the above purpose.

Main objective of the research carried out under this thesis was to control the robotic arm of USF WMRA-II in autonomous mode and in teleoperation using different user interfaces to enhance the accessibility of people with different disabilities. Since the kinematic arrangement of WMRA-II is similar to WMRA-I except for the link lengths, WMRA-I's C++ and Matlab high-level control programs, which calculate the optimum trajectory for any required motion, could be used for WMRA-II, after changing the system parameters including the geometrical parameters. Since the motion controllers are different for WMRA-I and WMRA-II, low-level controller board functions, which provide complete kinematics of the required motion to WMRA-II controller board (Galil DMC-2183), had to be developed using C++ and Matlab. The major tasks carried out in achieving the above objectives are listed below.

1. Tuning the motors of WMRA-II for PID gains to minimize the overshoots and errors.
2. Calibrating rotational displacement at the 7 joints with respect to the encoder readings of the motors.
3. Developing C++ functions to communicate with the controller board.

4. Modifying existing high-level programs for WMRA-I written in C++ and Matlab to use them for WMRA-II.
5. Developing low-level C++ and Matlab programs to calculate the complete kinematics of all joints (position, velocity and acceleration in encoder readings) and to communicate the calculated kinematics to WMRA-II controller board.
6. Testing WMRA-II with the developed C++ and Matlab control programs for accuracy, repeatability and manipulability.
7. Developing C++ Graphical User Interface (GUI) for Touch-Screen control
8. Evaluating WMRA-II for Spaceball and Touch-Screen user interfaces by using C++ and Matlab programs.
9. Comparing performances of WMRA-I and WMRA-II
10. Implementing a test GUI for individual joint testing and for troubleshooting the robotic arm.

1.3. Thesis Outline

The rest of this thesis is organized under chapters as mentioned in this outline. Chapter 2 is a brief survey on previous work done on WMRAs, programming languages that have been used in robots, and other assistive devices used to facilitate disabled people. Chapter 3 describes the closed-loop control system of WMRA-II. Chapter 4 is on the hardware of WMRA-II, including the robotic arm, wheelchair and controller board. It also includes the comparison of DH-parameter of the two WMRAs. Chapter 5 shows how WMRA-II was programmed using C++ and Matlab languages including the development of controller board functions and the GUI for Touch-Screen user interface using C++. Chapter 5 also describes calibration of the joint axes of the robotic arm and PID gains setting of all the motors in the robotic arm. Chapter 6 presents the results from the testing of WMRA-II with the developed C++ functions and the evaluation of Spaceball and Touch-Screen user interfaces for controlling the robotic arm. The results obtained for WMRA-II were compared with those for WMRA-I at the end of Chapter 6. Chapter 7 concludes the thesis.

Chapter 2:

Background

The main purpose of developing rehabilitation robot systems is assisting people with disabilities in their Activities of Daily Living (ADL). The Rancho “Golden” arm, developed at Rancho Los Amigos Hospital in Downey, California in 1969 is the first successful rehabilitation robot manipulator found in the history [1]. Although these very first robotic arms were not extensively used by consumers, they established the foundation for further development in the field.

Rehabilitation robotic arms are developed to perform common tasks like carrying, grabbing, picking up and moving objects that a person will encounter in his/her ADL tasks. For example, robotic arm MANUS [14] is an assistive robot helping numerous ADL to be carried out in home, at work, and outdoors [12]. Robotic arm KARES [11] helps people with disabilities with 12 tasks of ADL [13].

2.1. Main Types of Rehabilitation Robots

Survey of the different rehabilitation robotic projects shows four main development concepts of robots in the rehabilitation field [2], [3].

1. Workstation Robots (static robots that operate in a structured environment)
2. Stand alone manipulators
3. Wheelchair-mounted robotic systems
4. Mobile manipulators

2.1.1. Workstation Robots

Workstations were the first robots designed for people with disabilities. The purpose was to give disabled people more autonomy in their daily work. Basically, a workstation is made of a desk and some shelves where a robotic manipulator is fixed. The robotic arm is programmed to get some objects, such as a telephone, book, etc. whose positions are perfectly known. DeVAR (Desktop Vocational Assistant Robot) [4], ProVAR (Professional Vocational Assistant Robot) [5], RAID (Robot for Assisting the Integration of the Disabled) [6] and Master are some examples of desks equipped with a robotic arm.

Stanford University built 4 generations of DeVAR systems. DeVAR III was a tabletop system laid out for daily living tasks, while DeVAR IV was used in a vocational environment. DeVAR III was successful for tasks like brushing teeth, preparing meals, and shaving. ProVAR was the successor of DeVAR [5]. It featured a Programmable Universal Machine for Assembly (PUMA-260) robotic arm and a human prosthetic end-effector mounted on a overhead track (Figure 1) that provided the open range of access to the object near the user.

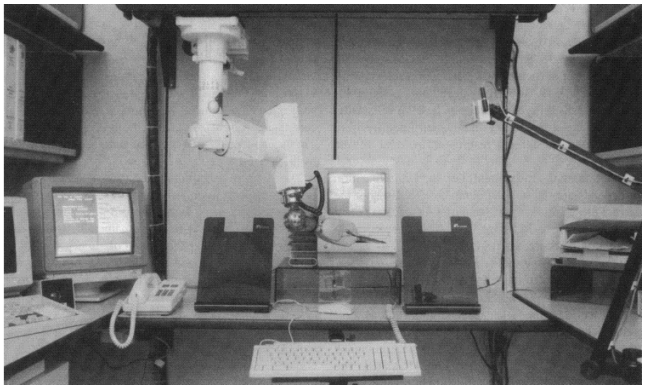

Figure 1: ProVAR (track-mounted upside-down PUMA-260) [4]

In an office environment, it is important to combine communication with manipulation, telephone control, and environmental control. The ProVAR interface included a single command structure for these three types of tasks. The voice recognition and digitizing card (VOTAN VPC-2100) allowed calls to be answered automatically in speakerphone mode. The robot could be commanded to pick up the phone receiver if a private conversation was desired. A computer-

interfaced environmental control unit (ECU) based on the X-10 standard allowed control of lights and small appliances by voice.

The majority of tasks within the RAID robot domain involved the transfer of physical media from one location to another. It used the RTX robot to manipulate computer, computer peripherals (scanner, printer, CDs) and telephone. Wheelchair user controlled RAID through a joystick which eliminated a mouse to access the user interface.

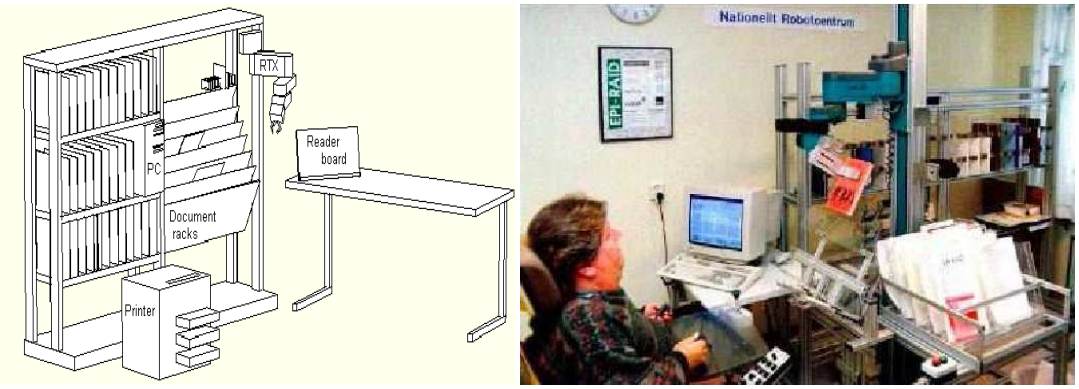

Figure 2: RAID Workstation Layout [6], [35]

Another type of workstation is low-cost workstations dedicated to self feeding tasks. They consist of a light weight robotic arm mounted on a special plate that is either put on a table or mounted on a stand. RAIL [7], Handy-1 [8] developed at, University of Kale, UK and MySpoon developed at SECOM's Intelligent Systems Laboratory, Japan [9] are some examples of this type of systems. Figure 3 shows Handy-I and My Spoon systems. Workstation robots are mechanically stable but are unable to react to changes in its environment. Main objective of designing Handy-I was to develop a low cost rehabilitation robotic system capable of being operated by people with severe disability, enabling them to gain independence in many daily living activities such as eating and drinking, shaving, cleaning teeth, applying make-up, washing, playing games and painting and drawing. In 1997 the implementation of sensors to the Handy-1 benefited the eating and drinking attachment and in 1998, the project showed a progress in drinking and eating.

Figure 3: My Spoon (left) and Handy-1 (right) Feeding Aids [8]

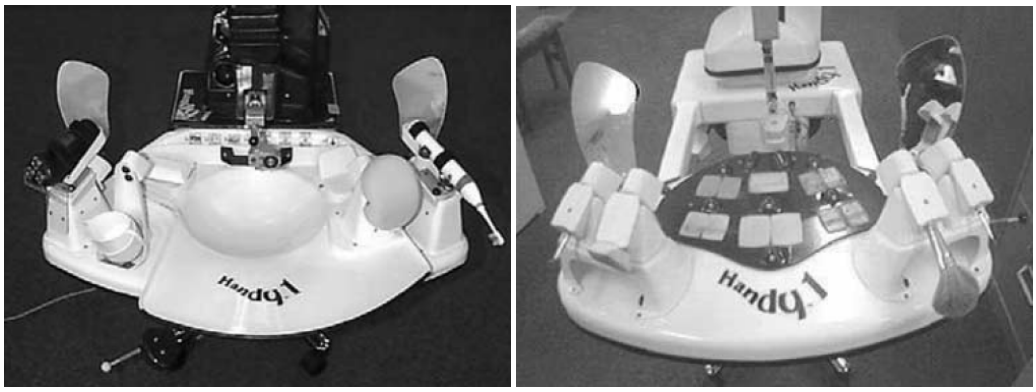

Figure 4: Handy-1's Washing, Shaving and Teeth Cleaning Tray (left) and Make-Up Tray (right) [8]

2.1.2. Stand Alone Manipulators

Unlike workstation-based systems, stand-alone systems typically use larger robotic arms to increase the workspace and they are not attached to any object. The work environment is not fixed as in the case of workstation robots. Sensors are used to get information about the robot's environment. The Tou robot [10] developed at the Polytechnic University of Catalunya in Spain and the ISAC (Intelligent Soft Arm Control) robot developed at the Intelligent Robotics Laboratory at Vanderbilt University [3] are two examples of stand-alone manipulators.

Tou is designed to provide tetraplegic people with some autonomy in their daily activities. It can perform simple tasks such as picking objects and rearranging the bed. The system consists of a robotic arm built from foam-rubber modules, a personal computer and the

communication interface. It uses three different communication interfaces: a speech recognition system, a joystick and a special keyboard. Tasks performed by ISAC include feeding soup, feeding fries and assisting in drinking. The user interacts with ISAC through a voice recognition system. ISAC uses pneumatic actuators called rubbertuators which behave like human muscles. This allows the robotic arm to have natural compliance.

2.1.3. Wheelchair-Mounted Robotic Systems

These systems allow people with disabilities to feed themselves and reach objects on the floor, on a table or above the head. Two types of wheelchair-mounted robotic arms are commercially available. The current market leader of this type is the MANUS (6-DOF robotic arm) [11] from Exact dynamics as shown in Figure 5. The second one is Rapture (4-DOF robotic arm), which is produced by the Rehabilitation Technologies Division of the Applied Resources Corporation. The Manus is a relatively high cost and sophisticated robotic arm whereas the Rapture robotic arm costs one third of the cost of the Manus, but caries very limited functionality.

Figure 5: Manus Robotic Arm [34], [35]

The Manus robotic arm has been controlled by the devices adapted to people with disabilities such as smart ball, breath control, movement base control, and joystick [20], [1]. The Rapture robotic arm [16] is controlled by a key pad and a secondary joystick. MANUS and Rapture robotic arms have been used by many research institutions to develop WMRA systems with enhanced performance by incorporating various sensing devices.

FRIEND [12] and VICTORIA [15] are another two examples for commercial WMRAs. Their purpose is to move the robotic arm by using information given by vision sensors. FRIEND I was developed by the Institute of Automation at the University of Bremen in Germany. It employs the Manus robotic arm. FRIEND II was the successor of FRIEND I. FRIEND I and FRIEND II have been controlled using voice commands. FRIEND II has also been controlled with a Brain Computer Interface (BCI) that reads the user's electroencephalographic (EEG) signals to control the robotic arm as shown in Figure 6. On the other hand, VICTORIA sets up a mimic recognition interface and a Touch-Screen to control the robotic arm.

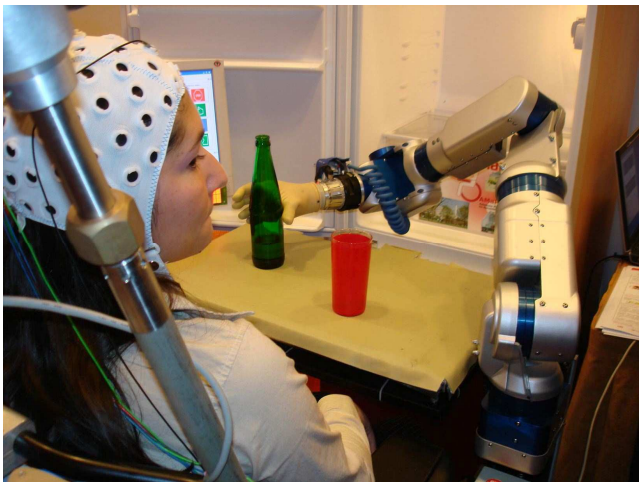

Figure 6: University of Bremen's FRIEND II Exhibited at ICORR'07 [35]

The Bath Institute of Mechanical Engineering in the United Kingdom implemented a custom WMRA called Weston. Weston was a successor of Wolfson, a workstation, and Wessex, a mobile manipulator [36]. It was designed to maximize the range of manipulation in horizontal plane and evaluated with four end-users: two spinal cord injury patients and two spinal muscular atrophy patients. Weston WMRA can be controlled in Cartesian space and in Polar coordinates and can be programmed with six tasks.

USF WMRA-I has a 7-DOF robotic arm mounted on a power wheelchair. This WMRA is unique because of its increased DOF. A complete description of USF WMRA-I is given in section 2.5.

2.1.4. Mobile Robots

Mobile robot systems navigate autonomously and perform tasks for the user. These systems are very useful for bed-bound users. WALKY [17], MoVAR, ARPH, HERMES, KARES 11 [13], and CARE-O-BOT are examples of such system. These robots have several characteristic advantages such as the ability to move independently from the wheelchair, moving from one room to another to fetch and carry objects, and serving more than one person.

MoVAR, developed at Stanford University, is one of the best known mobile robots. MoVAR uses DeVAR robot as the arm and omni-directional wheels on a mobile base. MoVAR was successfully tested by end users for different ADL. A localization system based on cameras and lasers has been employed on MoVAR to improve task execution in the home environment. KARES II was successfully tested for 12 tasks including wiping and scratching face, picking up objects, opening/closing doors, etc. A Visual Servoing module was adapted for vision based control of the robotic arm, and for a human-friendly interface for face recognition.

Figure 7: Care-O-Bot Serving Food (operated fully autonomously)

2.2. Robot Programming Languages

Robot programming languages have been developed to control robots in different environments and with different geometrical capabilities. Some of these programming languages were developed by modifying the general purpose computer languages, and others were specifically developed for Robots. Robot programming languages can be of three categories [29].

1. First generation languages.
2. Second generation languages.
3. World modeling and task-oriented object level languages.

First generation languages provide offline programming and the robot is usually controlled using a teach pendent. However, they are not capable of handling sensory data except ON/OFF binary signals and communication with other computers. VAL is one of the examples for the first generation programming languages. Second generation languages are structured programming languages with which complex tasks can be programmed. Force and torque sensors, which are connected to joints and grippers of robots, can be integrated to robot controller using a second generation language to achieve better motion control. Robots controlled with these languages behave more intelligently because of advanced sensory capabilities. They can recover in an event of malfunctioning. VAL-II and RAIL are two examples for second generation languages. World modeling and task-oriented object level languages are very popular because of their capability of task oriented controlling. These are general purpose computer languages. Many sensory devices (e.g. camera, laser, etc.) can be handled with these languages. They are used to control both industrial and rehabilitation robots using commands or automatically. Robots programmed with these languages are more intelligent than the others. C/C++, Matlab and LabVIEW are the most widely used general purpose languages [32].

C/C++ languages have been recently used in controlling many rehabilitation robots such as MIT Manus, UCF Manus and USF WMRA-I [30]. USF WMRA-I can also be controlled using Matlab. Robot library for ProVAR assistive robot has been developed using C++ and compiled with Watcom C/C++ compiler [31]. Some of the implemented functions include matrix operation, forward kinematics, inverse kinematics, trajectory generation, PID control, TCP/IP communication and various drivers for low-level hardware (e.g. encoders, force sensors, etc.) communication.

2.2.1. C++ Software Architecture of USF WMRA-I

The C++ control algorithm of WMRA-I has two parts; the main script and several functions supporting the main script. The main script is the stem of the control algorithm and all the libraries, which are used in supporting functions, are declared first in the main script. They are followed by the commands for displaying user options. When the options are input, the rest of the script passes those to supportive functions, where necessary calculations are performed and results are sent to a function called ArmMotion. This function then builds the communication with the controller board and sends necessary kinematics data to the controller board.

WMRA-I control algorithm uses both standard built-in C++ libraries (e.g. math, time, iostream, etc.) and external open source libraries (matrix and vector*)* [30]. It is a modular type algorithm that allows modifications. Every supporting function has two files called a header (.h) file and a source (.cpp) file. The header file consists of function prototypes and other supportive libraries while, the source file contains the function body including manipulation of data to calculate necessary outputs as a function return. The arguments and the returns of those functions are of matrix, vector or scalar data types.

2.2.2. Matlab Software Architecture of USF WMRA-I

Matlab control algorithm also consists of a main script assisted by other supportive functions. Logic of the program stays the same as that for the C++ control algorithm. However, the low-level program uses a C++ dll library to communicate with the controller board. The control algorithm utilizes built-in Matlab functions to perform all the calculations. Matlab control program is associated with a 3-DOF stick-figure simulation of the WMRA, which generates plots visualizing the real-time kinematics of the robotic arm and the wheelchair.

Both the control programs have their own pros and cons. It has been found that the processing time of the C++ algorithm is lower than that of the Matlab algorithm. Therefore, the C++ control program is more suitable for real-time control applications. However, Matrix manipulations in C++ are less convenient, and hence some of the C++ functions are more

complex and longer than their Matlab counterparts. Furthermore, Matlab control algorithm provides simulations and plots of the intended operations of the WMRA.

2.3. Human Machine Interface

In general, a complete WMRA system should consist of a main computer, a robotic arm with its low-level controller, sensors, a power wheelchair, and user interface devices that interact with the user (Figure 8). It should be able to perform certain predefined tasks autonomously, or be teleoperated by the user.

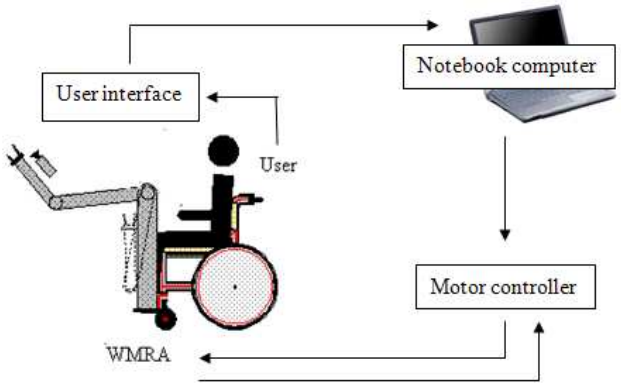

Figure 8: Basic Elements of a WMRA System

The ability of a person to understand and manipulate a device is a significant challenge for people with disabilities. People with disabilities can have a wide range of strengths and challenges. These differences can be in cognitive ability, sensory impairments, motor dexterity, behavioral skills, and social skills. Cognitive differences may be seen when a person is performing tasks that require problem solving or memory skills. Sensory skill issues are evident when a person has vision or hearing loss, tactile challenges, or difficulty to concentrate in distracting environments. Motor demands can be a barrier when a task involves isolating individual fingers, aiming at small targets, or other precise and graded movements. Many tasks require frustration tolerance, impulse control, turn taking, and other social skills. Devices designed to assist people with disabilities must take into account the above differences.

Generalizations over populations cannot be made for Human-Robot Interaction (HRI) in Assistive Technology (AT). The same task should be able to be performed in several ways. As the user adapts to the system, he/she can chose the most comfortable option to perform a certain task. In other words, each person is unique, and often has his/her own customized solution. For people with disabilities, the following HRI-AT design guidelines have been proposed by [25].

1. Interfaces should be easy to use without being complex.
2. Interfaces should have minimal length processes.
3. Ability to carry out multi-step processes.
4. Interfaces should adjust prompting levels. A person may require some form of prompting to perform a process.
5. Interfaces should leverage a person's sensory skills to augment their feedback.
6. Interfaces should accommodate multiple access devices.

According to the user's level of disability, functional capability and preference, a different set of human-robot interfaces can be chosen to drive the robotic system in a desirable manner. Devices, such as Touch-Screen, Spaceball, joystick, microphone and laser pointers are among user interface devices that can be used in WMRAs as interface devices. Different parts of the upper body such as fingers, wrist and head can be used to operate different interface devices. Table 1 illustrates the body parts that are capable of controlling different devices. The symbol √ is used to indicate that the device can be used appropriately with certain body function. For people with disabilities, depending on the level of the injury, dexterity may vary from finger motion to wrist motion to gross robotic arm motion. For example, for upper level Spinal Cord Injured (SCI) patients, dexterity is only available in the head and neck. So according to table 1, a head tracker, which is used to detect the movement of the head, can be used as interface devices [20] to control the robotic arm. Figure 9 shows one of the head trackers that are commercially available for people with disabilities with little or no control of their hands. This head tracker uses an infrared beam and according to the head movement, a mouse click events are created in the computer screen. It can be operated through a USB port of the computer.

Figure 9: Head Tracker

Table 1: Interface Devices Matching with Body Functions [20]

Device	Fingers	Wrist	Arm	Head/Neck
Touch-Screen	√	-	-	-
Spaceball	√	√	-	-
Microphone(Speech)	√	√	√	√
Joystick	√	√	√	-
Head Tracker	-	-	-	√

According to some surveys [3] [2], people with disabilities expect robots to give them a better autonomy in their everyday life. Furthermore, they are concerned about the quality and the cost of the tools provided by rehabilitation robotics. They demand for safe, reliable and uncomplicated devices that do not require too much training. Because of their low cost and ease of use, Handy-1 and MANUS robotic arms have been commercially successful.

All robots mentioned above are equipped with several control modes; manual, automatic and shared. In [18], it is shown that even though people find automatic and shared modes interesting, 80% judge the manual mode as a necessity for safety. However, in practice it is too slow and complicated.

2.4. Safety

Safety is one of the key factors when developing assistive systems for disabled people. Since the user is in close proximity to the assistive robot system the safety issues is very critical. The hardware of the system should be safe and the software should be very reliable to avoid unnecessary movements of joints. After designing them, such a system should be evaluated for safety. Some researchers consider safety as one of the evaluation criteria when they test their systems [1].

Potentially harmful situations can occur from unexpected movements of the body of the user or sneezing and coughing. The robotic systems should be able to detect these harmful situations and react accordingly. In DeVAR, the robot stops and shuts off if the user says 'stop' or shouts or if it encounters a resistance higher than a predefined threshold [23]. In USF WMRA-I two safety measures have been added to the hardware. The first is a panic stop button which is positioned under the right elbow of the user to stop the power supply to the robotic arm from the batteries without shutting off the logic power. The second safety measure is the timer. The power will cut off automatically if the system stays in idle for long time.

2.5. Overview of USF WMRA-I

USF WMRA-I [26] has a 7-DOF robotic arm, a 1- DOF gripper and a non-holonomic power wheelchair. Figure 10 shows WMRA-I and table 2 presents the specifications of it. WMRA-I provides a virtual robot interface and separates low-level software and hardware from the high-level robot control programming. It also provides a set of reusable and portable high-level control codes. Originally, it was programmed using Matlab and later the complete high-level program was converted to C++ due to some issues in reliably controlling the devices. Matlab and Virtual Reality have been used for simulation of WMRA-I.

The main control interface for WMRA-I uses a GUI which is used to control the robotic arm and the wheelchair in teleoperated mode and in autonomous mode. Other control interfaces include the Spaceball from which the WMRA-I can be controlled in 6-DOF, and the P300 Brain

Computer Interface (BCI) which is being developed in collaboration with the psychology department at USF. The BCI uses a fitted cap with electrodes mounted on it to sense EEG brain activities while the user is viewing a series of options on a computer. Other possible methods of controlling the robotic arm include joystick and the Phantom Omni, which is a six-degree-of-freedom device that provides a force feedback.

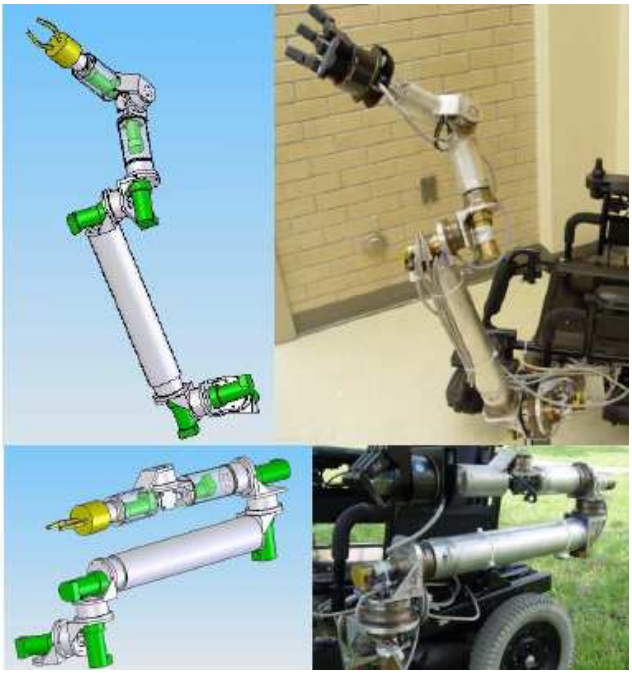

Figure 10: WMRA-I [7]

Table 2: WMRA-I Specifications [26]

Parameter	Value
Mass of the robotic arm/(kg)	12.5
Mass of the wheelchair (Including power supply)/(kg)	136
DOF	9
Maximum reachable height above floor/(m)	1.37
Designed payload/(kg)	6
Actuator Type	Brush DC servo motors
Controller board	Pic-Servo SC
Average Current Draw/(A)	2
Transmission	Harmonic drive
Chair width increase with side mount/(cm)	7.5

Chapter 3:

Closed Loop Motion Control System of USF WMRA-II

3.1. Introduction

To achieve a full WMRA control, high-level programming is necessary to calculate Cartesian trajectories, inverse kinematics and optimization based on criteria functions. Then low-level control is necessary to pass on the commands to the control board and drive the system. This thesis is continued with the low-level control of WMRA-II, and the high-level control is outside the scope of this thesis.

A typical closed loop motion control system (a servo control system) should consist of the main elements shown in Figure 11.

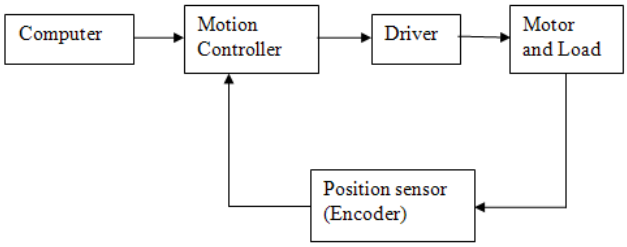

Figure 11: Close Loop Motion Control System Block Diagram

3.2. Motion Controller

The motion controller of WMRA-II (Galil DMC-2183) consists of a digital to analog converter (DAC) and a digital filter. In the Galil motion controller there are three types of elements added in series; a PID controller, a low pass filter and a notch filter. All the parameters in the filters are in terms of proportional gain (KP), integral gain (KI) and derivative gain (KD) of the PID controller. The signal passing through the DAC is converted to a signal in the range of +10 V to -10 V. This signal is then amplified (in the Driver shown in Figure 11) and finally sent to

the motor. Sending and receiving signals can be done using either both RS-232 or Ethernet TCP/IP ports. Functions of the motion controller can be listed as follows.

1. Generation of motion profiles in terms of voltage signals
2. Closing the position feedback loop
3. System compensation
4. Input/Output handling

3.3. Motors

Servo motors are used in industry in positioning applications. There are two types of servo motors used in WMRA-II, namely brushed and brushless DC motors. Brushed motors are lower in cost and simpler in design. However, the brushes and commutator can wear decreasing the life time of the motor. Brushless motors are generally capable of carrying more torque, and gearheads can be used to adjust the torque in these motors. These are more expensive than brushed motors. Heat dissipation and noise are generally lower in brushless motors. Brushless motors have been selected for the first 4 links of WMRA-II robotic arm and brushed motors were selected for the last three links. Each motor of the robotic arm consists of a gear head and a position sensor.

3.4. Position Sensor (Encoder)

Position sensors are used to get the position feedback of the joints to the control program. There are two main types of encoders, absolute encoders and incremental encoders or relative encoders. Absolute encoders provide a unique digital word for any rotational position of a shaft whereas incremental encoders provide digital pulses with the shaft rotation from which relative measurements can be obtained. Absolute encoders are used in applications where a device is inactive for long periods of time or moves at slow speeds. Incremental encoders are simple in application and also less expensive. The number of square wave cycles produced by an incremental encoder per complete rotation of the shaft is called the encoder resolution.

Servo motors used in WMRA-II (Maxon) include of incremental encoders. These encoders are usually supplied with two channels (A & B) that are offset from each other by 1/4 of a cycle (90 degrees). This signal pattern is referred to as quadrature and allows the user to determine not only the speed of rotation but its direction as well. By examining the phase relationship between A and B channels, one can determine if the encoder is turning clockwise (B leads A) or counterclockwise (A leads B). If an incremental encoder giving N number of pulses per revolution outputs two signals, channel A and B, which are in quadrature, the position resolution is increased to 4N quadrature counts/rev. The resolution of the encoder can be represented by a gain of $K_f = 4N/2\pi$ [count/rad]. Figure 12 illustrates details about an incremental encoder.

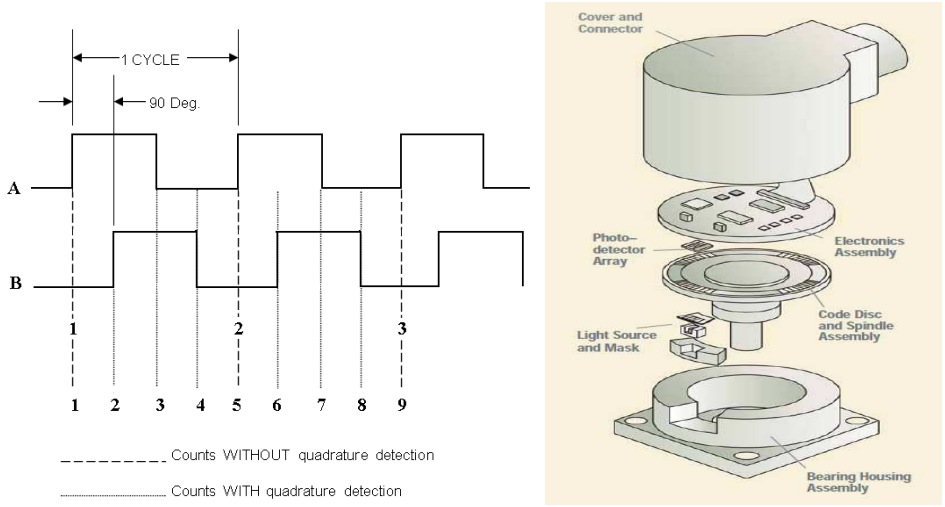

Figure 12: Quadrature Detection of an Encoder and Elements of Optical Encoder [28]

3.5. Computer

This is the place where the control program is stored and run. The users can use various interface devices such as joystick, Touch-Screen and Spaceball and BCI to input the information. The computer performs calculations and generates necessary outputs to be sent to the controller. Data transfer between the user interfaces, the control program and the controller board is accomplished either through Serial port or Ethernet TCP/IP port.

3.6. PID Feedback Controller

A proportional–integral–derivative controller (PID controller) is a generic control loop feedback mechanism (controller) widely used in industrial control systems. To achieve the best performance, the PID parameters used in the system must be tuned according to the nature of the system. The PID controller calculation (algorithm) involves three separate parameters: the proportional, the integral and derivative values, denoted KP, KI, and KD. The proportional value determines the reaction to the current error, the integral value determines the reaction based on the sum of recent errors, and the derivative value determines the reaction based on the rate at which the error has been changing [38].

The proportional term (KP gain) makes a change to the output that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by a constant *KP*, called the proportional gain. The proportional term is given by equation (1).

$$P_{out} = KP \times e(t) \quad (1)$$

The contribution from the integral term is proportional to both the magnitude of the error and the duration of the error. Summing the instantaneous error over time (integrating the error) gives the accumulated offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain and added to the controller output. The magnitude of the contribution of the integral term to the overall control action is determined by the equation (2). Equation (3) is the magnitude contribution of the derivative term.

$$I_{out} = KI \int e(t)dt \quad (2)$$

$$D_{out} = KD \frac{d}{dt} e(t) \quad (3)$$

Galil controllers provide a compensation filter, which includes a PID (Proportional-Integral-Derivative) filter followed by a notch filter as shown in Figure 13 [37].

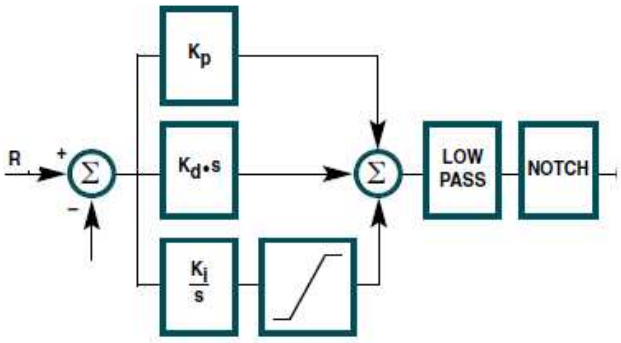

Figure 13: PID Block Diagram of Galil Tuner [37]

3.6.1. Overview of Tuning Methods

There are several methods for tuning a PID loop. The most effective methods generally involve the development of some form of process model, and then choosing KP, KI, and KD based on the dynamic model parameters. Manual tuning methods can be relatively inefficient, particularly if the loops have response times on the order of minutes or longer.

The choice of method will depend largely on whether or not the loop can be taken "offline" for tuning, and the response time of the system. If the system can be taken offline, the best tuning method often involves subjecting the system to a step change in input, measuring the output as a function of time, and using this response to determine the control parameters.

3.6.2. PID Tuning Software

Most modern industrial facilities no longer tune loops using the manual calculation methods shown above. Instead, PID tuning and loop optimization software are used to ensure consistent results. These software packages will gather the data, develop process models, and suggest optimal tuning. Some software packages can even develop tuning by gathering data from reference changes. [38]

Mathematical PID loop tuning induces an impulse in the system, and then uses the controlled system's frequency response to design the PID loop values. In loops with response

times of several minutes, mathematical loop tuning is recommended, because trial and error can literally take days just to find a stable set of loop values. Optimal values are harder to find. Some digital loop controllers offer a self-tuning feature in which very small setpoint changes are sent to the process, allowing the controller itself to calculate optimal tuning values.

Other formulas are available to tune the loop according to different performance criteria. Many patented formulas are now embedded within PID tuning software and hardware modules [38].

Chapter 4:

Denavit-Hartenberg Parameters and Hardware of USF WMRA-II

4.1. Denavit-Hartenberg Rules

The position and orientation of a robotic manipulator is described using kinematic equations. Two different approaches can be taken to express the position.

1. Joint space, which represents the angular position of each joint of the manipulator.
2. Cartesian space, which consists of position (*x*,*y*,*z*) and orientation (α, β, γ represented by a 3×3 matrix called the "rotation matrix") of the end-effector (gripper) in space.

Forward kinematics can be used to transform coordinates from joint space to Cartesian space. This transformation depends on the configuration of the robot (i.e. link lengths, joint positions, type of each joint, etc.). In order to describe the location of each link relative to its neighboring link, a frame assigned to each link is created, and a set of parameters are specified to characterize the frame. This representation is called the Denavit-Hartenberg (DH) notation [22]. Application of the DH rules for frame assignment of WMRA-II is briefly explained in this chapter.

There are two versions of DH rules.

1. Proximal
2. Distal

The main difference between the two is labeling of the indices. In this explanation, the proximal system will be used to find frame assignments. The steps in frame assignment using proximal labeling system are explained below with the notation shown in Figure 14.

1. Identify the joint axes and imagine lines of infinite length along them. For steps 2-5, consider two of these neighboring lines (at axes i and (i+1)).
2. Identify the common perpendicular between two axes, or if the axes intersect, the point of their intersection. Assign the link-frame origin at the point of intersection, or at the point where the common perpendicular meets the i^{th} axis.
3. Assign axis z_i pointing along the joint axis i.
4. Assign axis x_i pointing along the common perpendicular, or if the axes intersect, assign x_i to be normal to the plane containing the two axes.
5. Assign y_i to complete a right-hand coordinate system.
6. Assign {0} to match {1} when the first joint variable is zero. For {N}, choose an origin location and x_N free from the direction, keeping as many linkage parameters as zero.

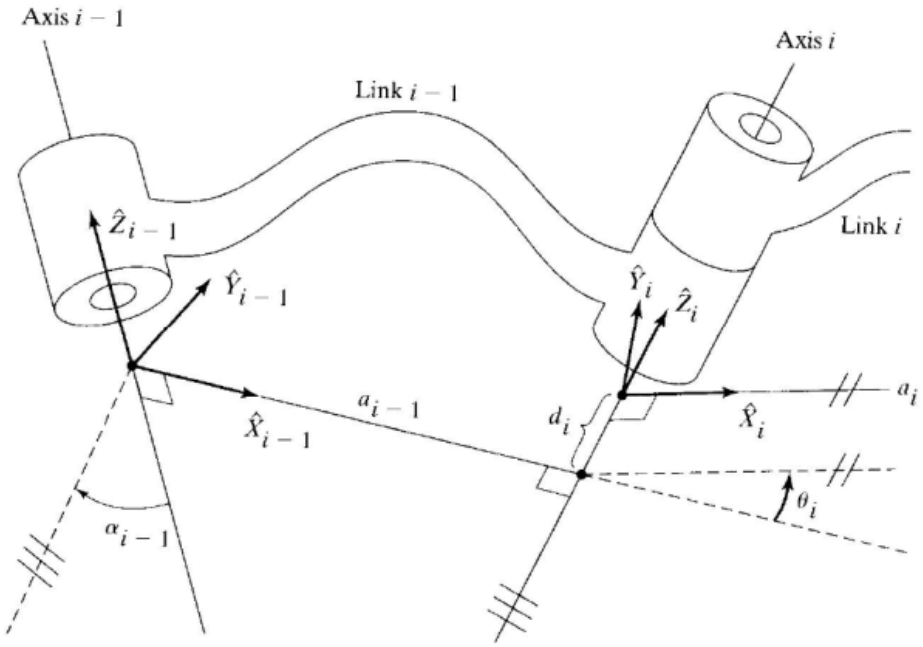

Figure 14: Link Frames Assignment [22]

If the DH frames are assigned properly, then the definition of the link parameters are as given in table 3.

Table 3: DH-Parameter Description

Parameter symbol	Description
a_i	the distance from z_i to z_{i+1} measured along x_i
α_i	the angle between z_i and z_{i+1} measured about x_i
d_i	the distance from x_{i-1} to $\hat{x}_i$ measured along z_i
θ_i	the angle between x_{i-1} and x_i measured about z_i

4.2. DH Parameter Assignment for USF WMRA-II

Figure 15 shows a Solid Works model of WMRA-II and link lengths are in millimeters as well as the frames assigned to each link. Table 5 shows the modified link lengths of WMRA-II from those of WMRA-I. Other parameters have been kept the same for the two WMRAs. Procedure for obtaining forward kinematics and inverse kinematics for WMRA-II robotic arm and combined mobility and manipulation theory used [26] are similar to those for WMRA-I.

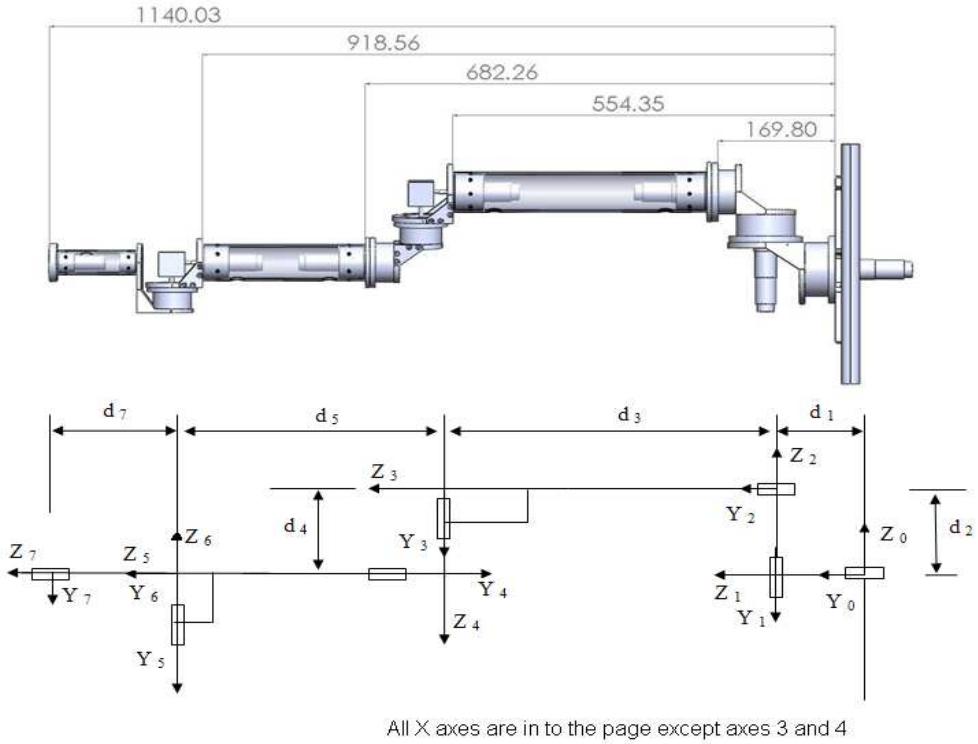

Figure 15: Frame Assignment of WMRA-II

Table 4: DH-Parameter Comparison of Two Robotic Arms

i	$\alpha_{(i-1)}$		$a_{(i-1)}$		d_i		θ_i	
	WMRA-I	WMRA-II	WMRA-I	WMRA-II	WMRA-I	WMRA-II	WMRA-I	WMRA-II
1	-90	-90	0	0	110	103	θ_1	θ_1
2	90	90	0	0	146	133	θ_2	θ_2
3	-90	-90	0	0	549	502	θ_3	θ_3
4	90	90	0	0	130	90	θ_4	θ_4
5	-90	-90	0	0	241	375	θ_5	θ_5
6	90	90	0	0	0	0	θ_6	θ_6
7	-90	-90	0	0	179	161	θ_7	θ_7

4.3. Hardware Architecture

WMRA-II system consists of a standard electric wheelchair, a 7 DOF robotic arm, an on-board controller, a power supply and a tablet PC.

4.3.1. Power Wheelchair

Powered wheelchairs were first developed in early 1970s. The performance of the wheelchairs is dependent on the weight of the entire system. Conventional wheelchairs are difficult to maneuver in constrained spaces because they only have two degrees of freedom to move the wheelchair in a 3-DOF plane.

A number of computer controlled wheelchairs have been developed in recent years, including CALL Smart Chair, NavChair, TinMan and WALKY [21]. Such chairs use a wide variety of sensors ranging from cameras, encoders, accelerometers, and gyroscopes with a desired input device such as communication aids, conventional joysticks, sip and puff switches, pressure pads, laser pointers, speech recognition systems and force reflecting joysticks.

Quickies 626 Wheelchair is the one used for WMRA-II (Figure 16). It is a conventional wheelchair with 2-DOF differential drive. A joystick is used as the input device to navigate the chair. Table 5 shows the specifications of this wheelchair.

Figure 16: Quickie 626 Wheelchair

Table 5: Specifications of the Wheelchair

Parameter	Value
Overall Length	29"
Overall Width	25"
Weight Capacity	250 (400lb optional)
Overall Weight	128 lb w/o batteries
Operating Voltage	24V
Performance	7 Mph

Although it is a conventional wheelchair, two incremental encoders are mounted on the wheels to make it a closed loop system. A computer input can also be used to control it.

4.3.2. 7-DOF Robotic Arm

The robotic arm of WMRA-II is similar to that of WMRA-I robotic arm except for the changes in link lengths and couplings. It was designed by the rehabilitation engineering research group of USF and fabricated in USF's machine shops. Figure 17 shows a CAD drawing of the robotic arm and Figure 18 shows the fabricated robotic arm. The robotic arm was initially

fabricated using carbon fiber tubes. Due to its incapability of withstanding considerable torques it was later fabricated using aluminum. More precise motors were mounted during this second fabrication to obtain more precise motion.

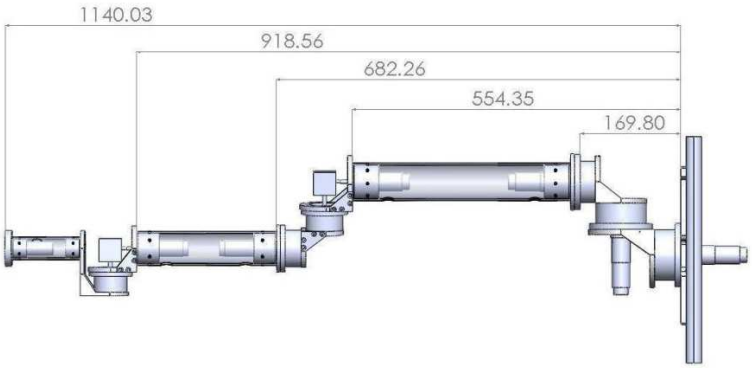

Figure 17: CAD Drawing of WMRA-II Robotic Arm

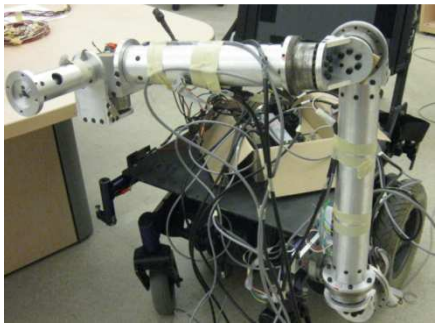

Figure 18: WMRA-II Robotic Arm

4.3.3. Power Supply

Both the controller board and the wheelchair use a 24V DC power supply which is a part of the wheelchair. The power unit is placed under the seat of the wheelchair. Battery life should be sufficiently long for the system to perform without any trouble.

4.3.4. Controller Board

Motion controller of WMRA-II is Galil DMC-2183. It is capable of controlling a maximum of 8 motors. Brushless and brushed motors can be controlled simultaneously using this controller. The driver system to which the controller transfers data in terms of a voltage signal consists of

brushless (AMP-20540) and brushed (AMP-20440) motors and two amplifiers. The two amplifiers are mounted on top of the controller board as shown in Figure 19. Both Serial port and Ethernet port can be used to communicate with the controller using a PC.

Figure 19: Controller Board (Galil DMC-2183)

Chapter 5:

Calibration and Programming of USF WMRA-II

5.1. Axis Calibration of WMRA-II

All the joint angles, which are in radians in the high-level program, should be converted to encoder counts in order to control the motors of the robotic arm. An experiment was carried out to find the radians-to-encoder-counts conversion factor for each motor. First, the theoretical number of encoder counts to run each motor by 180 degree was found from data sheets provided by the manufactures. Then all the motors were run by the amount obtained from data sheets. Amount of rotations were recorded for all the joints by using a laser tracker, protractor and a ruler. Then the theoretical values are adjusted to get the 180 degree movements of each joint.

Motors of the robotic arm were labeled as indicated in Figure 20. Table 6 shows the information provided by the data sheets and Table 6 contains the theoretical and experimental conversion factors. It can be seen from Table 7, that the theoretical conversion factors are slightly different from the experimental ones. The backlash present in the mechanical joints and the gears in the couplings and axis misalignment can be the reasons for these differences.

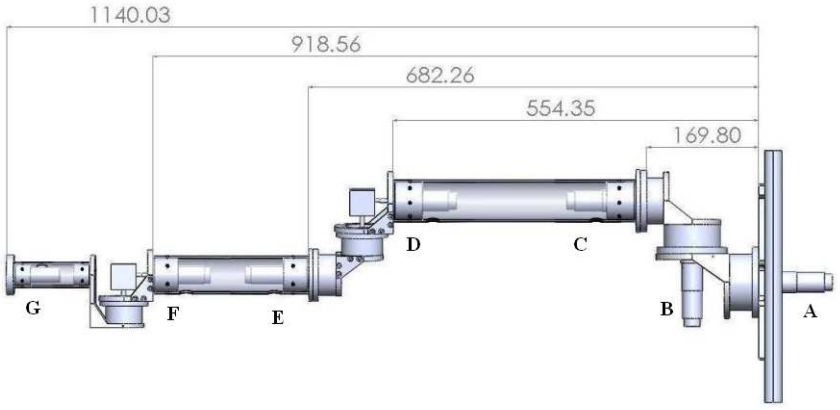

Figure 20: Labeling of Motors of the Robotic Arm

Table 6: Data from Manufacturer's Datasheets

Motor label	Encoder counts per revolution	Quadrature type	Gear head reduction	Harmonic drive reduction
A	500	yes	51:1	100:1
B	500	yes	51:1	100:1
C	500	yes	51:1	100:1
D	500	yes	51:1	100:1
E	500	yes	19:1	100:1
F	500	yes	19:1	100:1
G	512	yes	14:1	100:1

Table 7: Radians to Encoder Counts Conversion Factors

Axis Label	Theoretical factor	Experimental factor
A	$\pi/5100000$	$\pi/5250000$
B	$\pi/5100000$	$\pi/5250000$
C	$\pi/5100000$	$\pi/5250000$
D	$\pi/5100000$	$\pi/5250000$
E	$\pi/1900000$	$\pi/2000000$
F	$\pi/1900000$	$\pi/2000000$
G	$\pi/1433600$	$\pi/1500000$

5.2. Calculation of Velocity and Acceleration

Parameters for position controlling were calculated assuming that time and numbers of encoder counts were known.

$$v = a \times t \tag{4}$$

$$dq = \left(\frac{1}{2} \times v \times t\right) + [v \times (T - 2t)] + \left(\frac{1}{2} \times v \times t\right) \tag{5}$$

Solving equations (4) and (5) for velocity *v*, following expression can be obtained.

$$v^2 - a \times T \times v + dq \times a = 0$$

$$v = (a \times T)/2 \pm \left(\sqrt{(a \times T)^2 - 4 \times dq \times a}\right)/2 \tag{6}$$

5.2.1. Position Control

Acceleration and velocity can be calculated by considering the trapezoidal velocity profile as shown in Figure 21. It is assumed that the position (dq) and the total time (T) are given in the program.

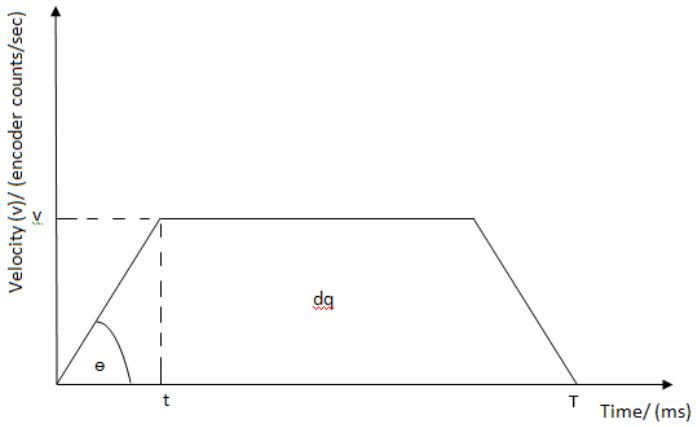

Figure 21: Motion Profile Diagram

Table 8: Motion Profile Parameter Notation

Parameter	Description
dq	Displacement in encoder counts
a	Acceleration of motor
v	Velocity
T	Total time
t	Acceleration time

By considering the real value of velocity, an expression for the acceleration can be calculated.

$(a \times T)^2 - 4 \times dq \times a \geq 0$

$$\mathrm{a} = (4 \times \mathrm{dq})/\mathrm{T}^2 \quad (7)$$

From equation (7) and (8), the applied velocity was found.

$$\mathrm{v} = (2 \times \mathrm{dq})/\mathrm{T} \quad (8)$$

5.3. PID Gains Setting of WMRA-II

The PID gains of the servo motors have to be adjusted (tuned) so that the motors follow the input signal as closely as possible. The gains of the system determine how well the servo system tries to reduce the error. Main purpose behind tuning motors of WMRA-II was to reduce the overshoots while improving the accuracy of the motion.

PID controllers consist of three gain parameters called proportional gain (KP), integral gain (KI) and derivative gain (KD). KP value determines the reaction to the current error, KI value determines the reaction based on the sum of errors, and KD value determines the reaction based on the rate at which the error is changing. Increasing of KI will increase the overshoot and settling time. Increasing of KD will decrease both overshoot and settling time. Increasing of KP will decrease rise time and steady state error, and will increase overshoot.

5.3.1. Tuning Procedure

GalilTools software was used to independently tune all the motors in the robotic arm. First PID gains were found by using Galil Tuner toolbox of the software shown in Figure 23. Then all the motors were tuned manually by keeping KI values for all the motors at zero. Data were recorded for all the motors during both manual and auto tuning. Finally, data were plotted to compare the step responses. For both tuning processes 100 counts were given within 100 ms. Ethernet connection was used to record the data from the controller board. The step responses for motor G are plotted in Figure 22.

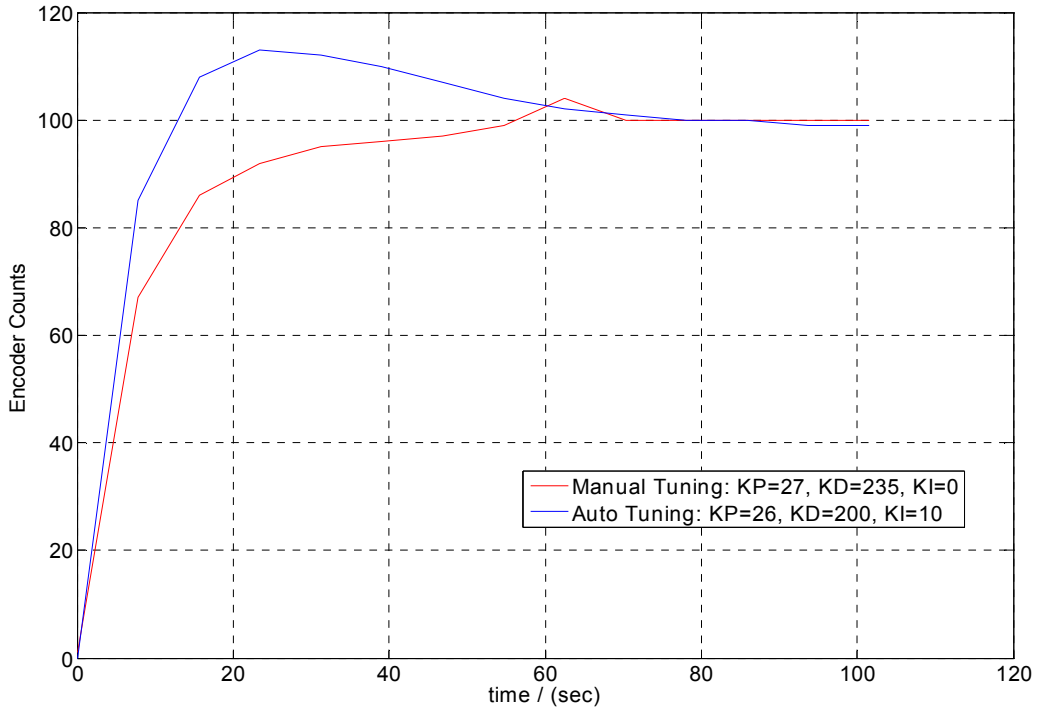

Figure 22: Step Responses of Motor G

5.3.2. Scope

Scope is another toolbox of the GalilTools software. Scope consists of 8-chanels and provides real-time visual feedback from the controller in the form of an oscilloscope signal. Eight different data types including position, position error, and velocity can be viewed from it.

5.3.3. Auto Tuning

The Autotune button, automatically adjusts optimal PID values by selecting axes through a series of predetermined movements, which may cause the motor to buzz, and finally shows a step response for the specified counts and time in the scope. Since Autotune provides accurate results for the default settings in the controller software (for minimum steady state error), manual tuning is required to obtain specific performances such as minimum overshoot, minimum settling time, etc.

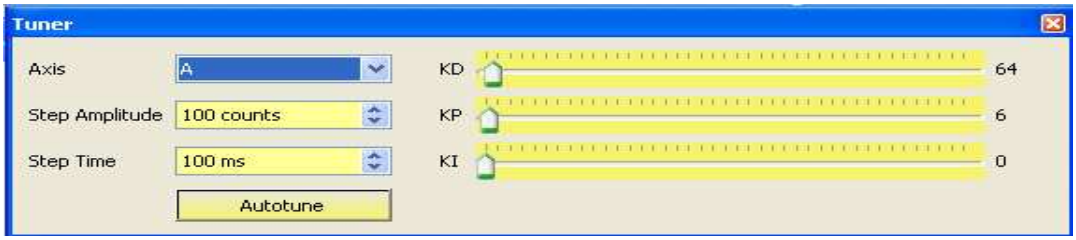

Figure 23: Galil Tuner

5.3.4. Manual Tuning

First the KI value of each motor was set to zero and then the KP and KI values were changed slightly to give a step input to the motors. Data were recorded and compared with values obtained from the Galil Tuner. It was noted, when the KI was set to zero, the overshoot was minimum. Therefore, values obtained from manual tuning were assigned in the low-level program. It can be seen from Figure 22, that overshoots and errors have been reduced by manual tuning when motor G was commanded to go to 100 encoder counts within 100 ms.

Since the payload is changing from one task to another, all the motors were tuned for three different payloads and the average values were found. Table 9 shows the average gain values obtained for three different payloads 1, 2 and 3.5 kg.

Table 9: PID Gain Parameters of Motors

Motor	KP (proportional gain)	KD (derivative gain)	KI (integral gain)
A	35	260	0
B	5	50	0
C	5	50	0
D	5	50	0
E	31	233	0
F	30	239	0
G	27	235	0
H	2	30	0

5.4. C++ Based Programming

C++ is one of the widely used programming languages at present. The language has been around for several decades and has won widespread acceptance because it gives programmers maximum control and efficiency. However, in controlling USF WMRA-I, C++ had only been used in the dll library functions that were used to physically communicate with the robotic arm. Matlab had been used as the programming language for major control tasks, which include optimization and calculation of kinematics, as well as for simulation. Various run time issues were encountered when the initial programs in Matlab were used to control the WMRA-I. Therefore, these Matlab programs were later converted to C++ in search of smooth and reliable control of WMRA-I. The concepts of kinematics and the control algorithms were not changed in this process. Visual Studio 2008 has been used to write all the C++ programs and its compiler was used to compile the program. The architecture of WMRA-II control software consists of three main layers:

1. User interface layer: Sends the required information to the high-level controller using an input device (Spaceball, Touch-Screen, joystick, etc.)
2. High-level control layer: Translates the user input into the input to the low-level controller of the output device (WMRA). In this layer, all the kinematic equations and optimization algorithms are included.
3. Low-level control layer: Deals with specific characteristics of the controller board of the output device and its communication protocol.

5.5. High-Level Control Programming

This includes the main script and several supporting functions that calculate kinematics, motion optimization algorithms and trajectory generation algorithms to control the robotic arm. The final output from the high-level program should be the accurate position of each and every motor for each time step within the trajectory.

C++ high-level program for WMRA-I was modified to suit WMRA-II. DH-parameters, wheelchair dimensions and joint limit constraints are among the modified parameters in the control program. The prototypes of the functions that include the ArmMotion function to communicate with the robotic arm were used in the newly developed controller board program after modifying them appropriately.

Table 10: Prototypes of WMRA-I and WMRA-II C++ Functions

WMRA	Function prototype
WMRA-I	void WMRA_park2ready(int ini, int vr, int ml, int arm, Matrix Tiwc, Matrix qiwc)
WMRA-II	void WMRA_park2ready(int ini, int vr, int ml, int arm, Matrix Tiwc, Matrix qiwc, Galil &g)

As shown in Table 10, the controller board function parameters for WMRA-II are passed by reference to the variable “g” whereas in WMRA-I all the parameters are passed by their values. The functions for the controller board were implemented using part of the Galil dll library.

5.6. C++ Low-Level Control Programming

ArmMotion function is the function that physically communicates with the robotic arm. It calculates necessary kinematic parameters (position, velocity, acceleration, and deceleration) based on encoder values and sends those parameters to the controller board for execution. The first part of the function consists of initialization of the controller board. During the initialization, PID gains are set and the controller board is set for the position tracking mode. The ArmMotion function prototype is different for WMRA-I and WMRA-II as indicated in Table 11.

Table 11: C++ Function Prototypes of ArmMotion Function

WMRAs	Function prototype
WMRA-I	Matrix WMRA_ARM_Motion(int ind, int config, Matrix qo, float dt)
WMRA-II	Matrix WMRA_ARM_Motion(int ind, int config, Matrix qo, float dt, Galil &g)

In the low-level layer, C++ functions were created to communicate with all the motors in WMRA-II using serial communication. In order to reach the required position at a given time,

motion parameters of each joint should be found for all the motors. Since the controller board used in WMRA-II is different from that of WMRA-I, all C++ functions for the motion controller were separately developed.

Galil motion controller board manufactures have developed a set of C++ communication functions. Parts of their codes were used to develop functions for WMRA-II controller board. The C++ controller board functions that were developed can be categorized under following major categories.

5.6.1. Establishing the Communication of the Controller Board

Communication can be achieved using both serial and Ethernet ports. For serial port communication, there are two parameters that should be selected to complete the function. These are,

- Com port: A port in the CPU where serial port cable is connected to the peripheral device (Galil motion controller). RS-232 is used to serially communicate between controller board and the PC.
- Baud rate: Baud rate is a measure of how fast the data is transferred between an instrument and a CPU using serial communication.

'Galil g (COM2 19200)' is an example function call for serial communication. The serial cable is connected to communication port 2 and the data transfer rate is selected as 19200 by this command.

Ethernet communication is very useful in data recording since its sampling period is higher than that for serial communication. If Ethernet TCP/IP is used to communicate with the controller boadr, the IP address, which is a 4-byte number, should be assigned to the Ethernet port before starting communication. The last two bytes of the IP address represent the two digit number of the controller board. 'Galil g (124.51.21.83)' is an example function call for Ethernet port communication.

5.6.2. Parameter Initialization

- PID gain settings

PID gain settings are done under parameter initialization. In this process, proportional, integral and derivative gains are set for each and every axis. All eight motors must be tuned for PID gains before using these functions. The procedure for finding the gains is explained in section 5.3 of this thesis.

Table 12: C++ Controller Board Function for PID Gains Setting

Function Prototype	Example
void PID(char PID_motor, float KP_value,float KD_value,float KI_value, Galil &g)	PID('A',6,64,0, g)

- Amplifier gain setting

This function is used to increase the bandwidth of the amplifier.

- Brushed and brushless axes settings

Both brushed and brushless motors are used in WMRA-II robotic arm. These functions are used to set each axis as brushless or brushed.

5.6.3. Motion Control Functions

Parameters given by the high-level program are the time steps, the total time to execute the task and the corresponding amounts of rotation for all the axes in radians. The low-level program should be able to create a motion profile based on the output of the high-level control program. Motion control functions developed for this purpose generate a motion profile that is sent to the motors in terms of voltages. Motion profiling can be done using both position controlling and velocity controlling.

1. Position control function

Figure 24 shows a sample velocity profiles for position control. The function developed for position control is shown in Table 13. Parameters used in the position control functions are, position, velocity, acceleration and deceleration.

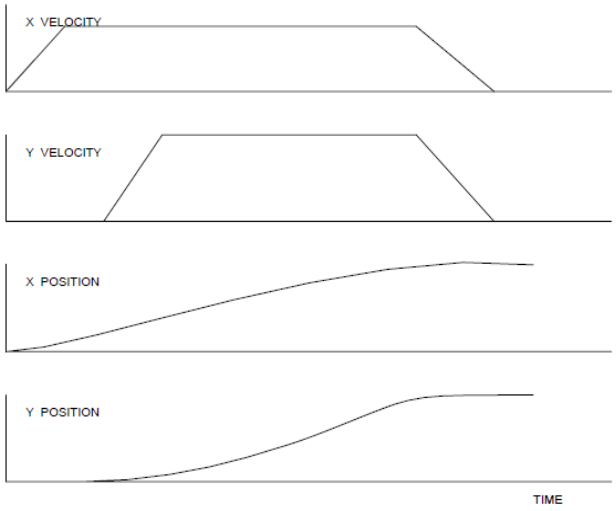

Figure 24: Position and Velocity Profiles for Position Control

Table 13: Controller Board Function for Position Control

Function prototype	Description
void Position_control(char motor,long int p_value,long int v_value, long int a_value, Galil &g)	Position_control('A',100000,20000,20000, g);

2. Velocity control function

Parameters used in the velocity control functions are, velocity, acceleration and deceleration.

5.6.4. Data Record

These functions are used for obtaining the position, velocity and torque values from the motion controller so that they can be used to minimize the position error and to keep track of the position, velocity or torque of each axis. Table 14 shows an example of the data record function.

Table 14: Function Example of Data Record

Function prototype	Example
float Tell_pos(char motor, Galil &g)	Tell_pos('A', g)

5.7. Matlab Low-Level Control Programming

Matlab low-level control program calculates all the kinematic parameters that the corresponding C++ program calculates. However, Matlab uses Component Object Model (COM) technology, which enables software components to communicate with each other, to link C++ functions to Matlab low-level program. All the parameters calculated must be converted to string type from numeric type before assigning them in C++ functions. This makes Matlab low-level program longer than the C++ program. Matlab built-in functions were used for converting parameter types and creating an instance of COM object.

5.8. User Interfaces

5.8.1. Touch-Screen

Users with different disabilities and individual needs, have to be considered in developing assistive systems. Touch-Screen is the GUI that can be used to control the robotic arm in teleoperated mode. Touch-Screen GUI implemented using C++ for USF WMRA-II is shown in Figure 25. In this interface a seven element vector written to a text file is used to pass the input data to the WMRA main program. When one or more buttons are selected, the corresponding elements of the vector are changed between 0 and 1, and the corresponding vector is written into the text file. The main program reads those values in real-time to update the motion.

There are two types of buttons used in the Touch-Screen GUI. Command buttons were used for Back, Exit and Stop operations. Check buttons were used to create all the other buttons. Mouse click events were created to pass the variable values to the main program. Properties of each button (size, location, fonts, background color, etc.) were selected from the available property editor which allows any required changes in the future. C++ codes related to the Touch-Screen GUI are given in Appendix B.

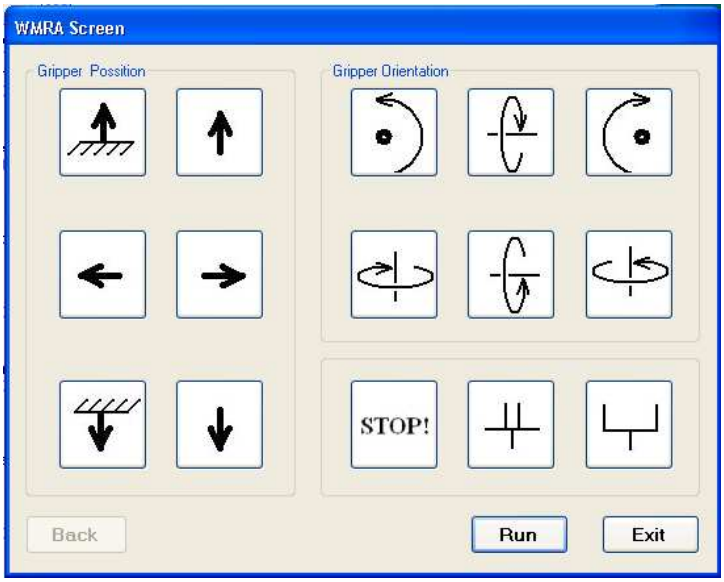

Figure 25: Screen Shot of Touch-Screen Control User Interface

5.8.2. Spaceball

Figure 26 shows a picture of a Spaceball. This device has three independent force sensors and three independent torque sensors to provide 6-DOF to represent an arbitrary position of a solid body in space. The Spaceball was originally developed to give 3D designers the ability to pan, spin, zoom, rotate and analyze their designs on the computer. Later it was used in controlling of robotic arms. By pushing and twisting the Spaceball, the robotic arm can be moved in the 3D space.

A demo program included in the Software Development Kit (SDK) called 3DxTest32 supplied by the manufacturer of the Spaceball was used to extract special data to be used in the WMRA-II control program. When the demo program is opened it creates a window on which the Spaceball movement data are written. Then these data are extracted to a text file as characters and the C++ WMRA-II program reads these data and converts the characters to numbers that are eventually used in the main program.

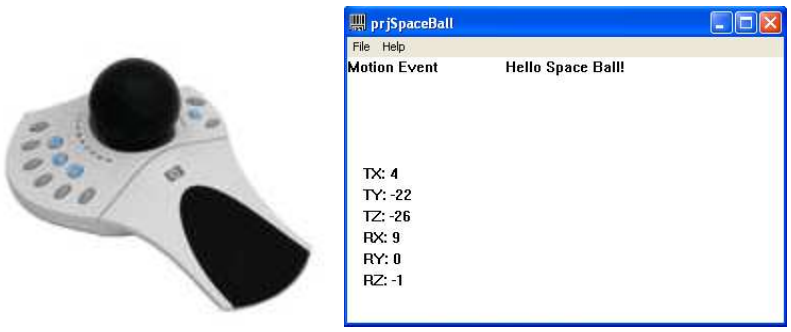

Figure 26: Spaceball and Its Demo Program Window

When the Spacball is used in Matlab, a separate Matlab demo program has to be used. A Matlab application is opened (Figure 27) automatically when the user opens the demo application program. The WMRA-II Matlab program should be run from the already opened Matlab command window so that both Matlab and Spaceball program share common global variables during the operation.

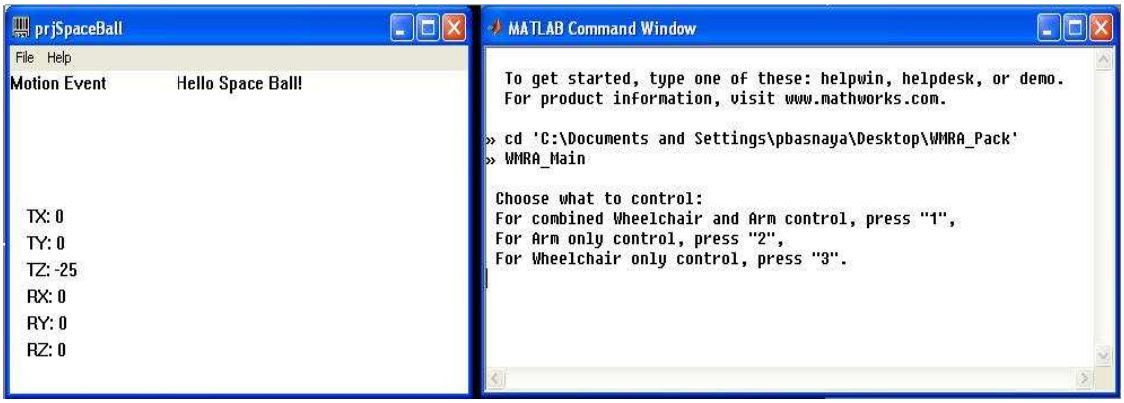

Figure 27: Matlab-Spaceball Program Window

5.9. Test GUI for WMRA-II

The joints of the robotic arm have to be operated individually for troubleshooting. When GalilTools software is used with Galil two-letter programming language for this purpose, the program has to be repeatedly changed for each joint. Therefore, a test GUI was separately developed to facilitate testing of the robotic arm operation. Figure 28 shows the developed GUI. This GUI avoids above mentioned difficulties in individual movement of joints during troubleshooting. In the test GUI, two separate buttons are implemented for clockwise and

counterclockwise operation of each motor. Separate group of buttons are implemented to open, close and stop the gripper.

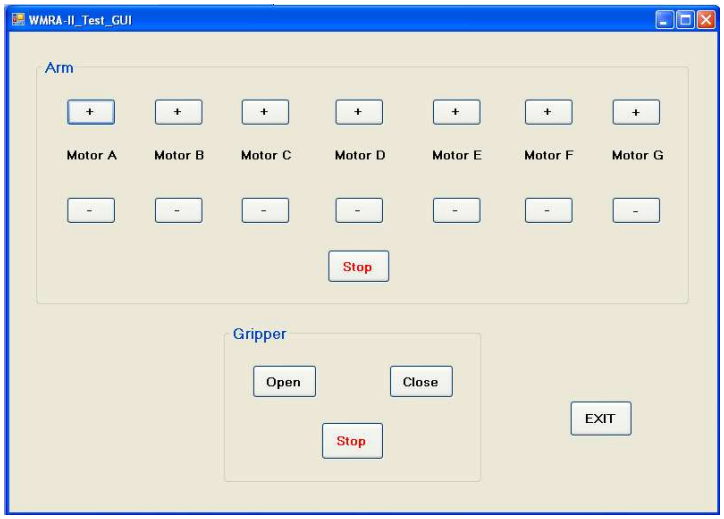

Figure 28: Test GUI for WMRA-II

Chapter 6:

Testing and Evaluation of WMRA-II

6.1. Power Usage of the Robotic Arm

The system specifications for WMRA-II obtained by testing are given in Table 15.

Table 15: System Specifications of WMRA-II

Mass of the robotic arm/(kg)	13.63
Mass of the wheelchair (not Including power supply)/(kg)	58
DOF	9
Maximum reachable height above floor/(m)	1.47
Designed payload/(kg)	3.85
Average current draw/(A)	1.75
Actuator type	Brushed and brushless DC servo motors
Controller board	Galil motion controller (DMC-2183) with two amplifiers for brushed and brushless motors

Power usage of the robotic arm and the whole system determines the battery life. The 24 V battery system of WMRA-II wheelchair, powers both the robotic arm and the wheelchair. As an indication of the power consumption, the electric current was recorded for the two cases in which the robotic arm was at rest and while it was running under the conditions given in Table 16. Power consumption of WMRA-II is compared with that of WMRA-I in Table 17.

Table 16: Power Usage of WMRA-II

Condition	Average Current / (A)
Idle - Controller only (all the motors off)	0.25
Holding self weight outstretched	1.25
Simultaneous movement of joints (without payload)	1.45
Simultaneous movement of joint with 2 kg payload	1.75
Lifting 6Kg with joint 1	1.75

It can be noted that WMRA-II consumes less power when it is in the idle state. However, batteries of the wheelchair can conserve power by implementing a timer that cuts off the power to the controller board when the robotic arm is in the idle state. During operation, WMRA-II consumes less power than WMRA-I.

Table 17: Power Consumption Comparison of WMRA-II with WMRA-I

Condition	WMRA-II	WMRA-I
Idling	0.25	0.36
Holding self-weight outstretched	1.25	0.58
Lifting 6Kg with joint 1	1.75	3.30

6.2. Repeatability and Accuracy of the Robotic Arm

The accuracy refers to the robot's ability to accurately position its wrist end at a desired target point within its work volume. Repeatability is a statistical term associated with accuracy and it describes how a point is repeated. By keeping equal conditions, the robot can be moved to a certain point a number of times to calculate the repeatability error. It does not describe the error with respect to absolute coordinates. It is a positional deviation from the average of displacement [33]. Figure 29 shows a representation of accuracy and repeatability.

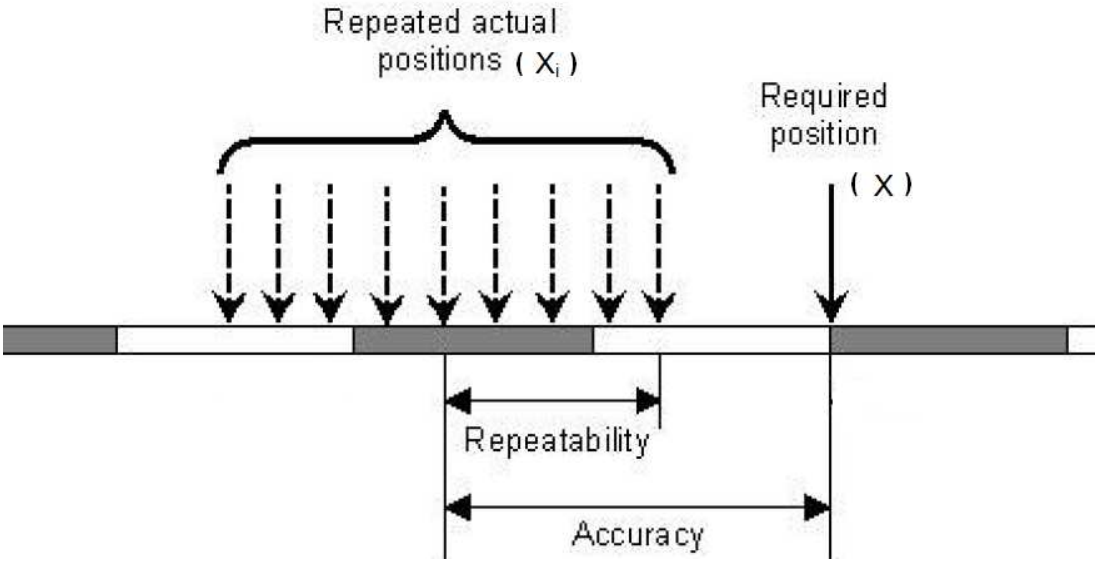

Figure 29: Representation of Accuracy and Repeatability [33]

The formulas that calculate accuracy and repeatability of the robotic arm are shown in equation (5), (6) and (7).

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} \sqrt{(x_i - x)^2}}{n} \quad (5)$$

$$\text{Repeatability} = \frac{\sum_{i=1}^{n} \sqrt{(x_i - \bar{x})^2}}{n} \quad (6)$$

Finally, relative true error was calculated using equation (7).

$$\text{Relative true error} = \frac{\text{Accuracy}}{\text{True value}} \quad (7)$$

6.2.1. Methodology

WMRA's are designed to perform daily living activities and therefore they should be reliable, accurate and repeatable. As a test of repeatability and accuracy, five positions as shown in the Table 18 were selected (without changing the orientation of the end-effector) and the two robotic arms were commanded to travel from the same ready position to each target position five

times. All the user options such as operation velocity, method of optimization, etc were kept as constants for all the trials. All the measurements were made based on the robotic arm base coordinate systems (Figure 30) and the data were recorded. Laser pointer was attached to a gripper of the arm to measure the vertical distance travelled (Which is z axis w.r.t arm base coordinate system) by the dripper. X and y coordinates were drawn on the floor and WMRAs were positioned properly on the drawn coordinate. X and Y distances travel by the gripper were measured.

Table 18: Selected Points for the Accuracy and Repeatability Test

No of points	x-y-z Coordinate
1	(700,200,700)
2	(200,700,700)
3	(1000,500,300)
4	(550,300,900)
5	(850,950,500)

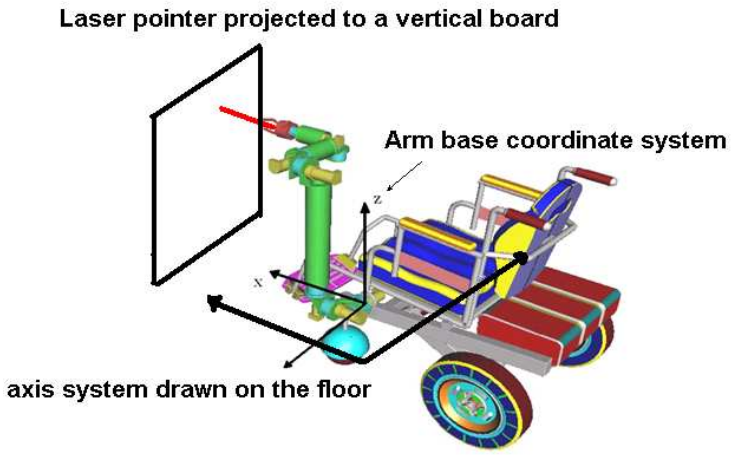

Figure 30: Experimental Setup for Repeatability and Accuracy Test

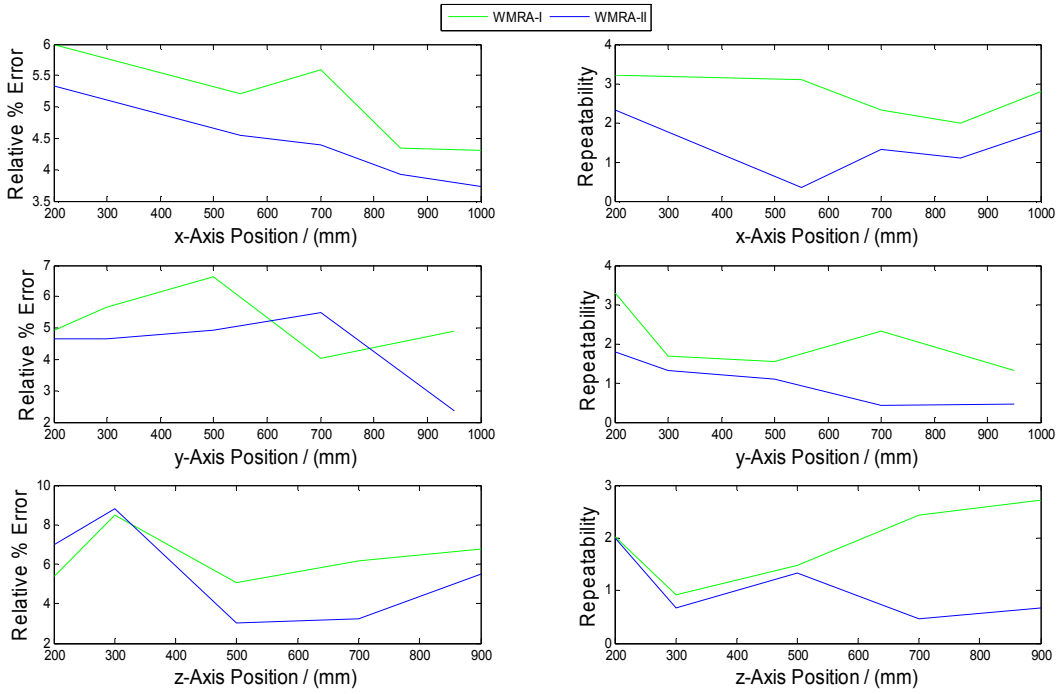

Figure 31: Repeatability and Accuracy of WMRA-I and WMRA-II

According to Figure 31, both WMRA-II and WMRA-I show reasonable repeatability. However, WMRA-II always recorded considerably less repeatability and percentage relative error than WMRA-I as shown in Table 19 and 20 respectively.

Table 19: Repeatability Comparison of WMRA-I and WMRA-II

Coordinate	WMRA-I(Average Repeatability) / mm	WMRA-II(Average repeatability) / mm
X	2.69	1.38
Y	2.03	1.02
Z	1.92	1.03

Table 20: Average Percentage Relative Error of WMRA-I and WMRA-II

Coordinate	WMRA-I(Average % Relative error) / %	WMRA-II(Average % Relative error) / %
X	5.10	4.39
Y	5.22	4.42
Z	6.38	5.50

According to above results both accuracy and repeatability of WMRA-II are higher than those for WMRA-I. Both of the systems were tested using the same C++ control algorithm and the same communication protocol (RS-232) was used to communicate with the controller board. Main differences are in the controller boards and the motors used in the two robotic arms. WMRA-II consists of the Galil motion controller (24 V DC) with two separate driver units for brushed and brushless motors and WMRA-I consists of the JRKERR PIC servo SC controller (12V DC) with separate driver unit for each motor. The motors used in WMRA-II are more precise than those used in WMRA-I.

The final accuracy in a robotic arm depends on the control program resolution and the measuring system. The mechanical inaccuracies are mainly caused by the backlash in the robotic arm joints and backlash in the gear mechanisms of the robotic arm. The measuring system with a laser tracker, a level and a ruler with a white board may have caused some errors during the measurements.

6.3. Use of Different Interfaces

Since this is preliminary testing stage of the WMRA-II, due to safety concerns, the system was tested without a user being seated on the wheelchair. Spaceball and Touch-Screen were the different interfaces used to run the robotic arm.

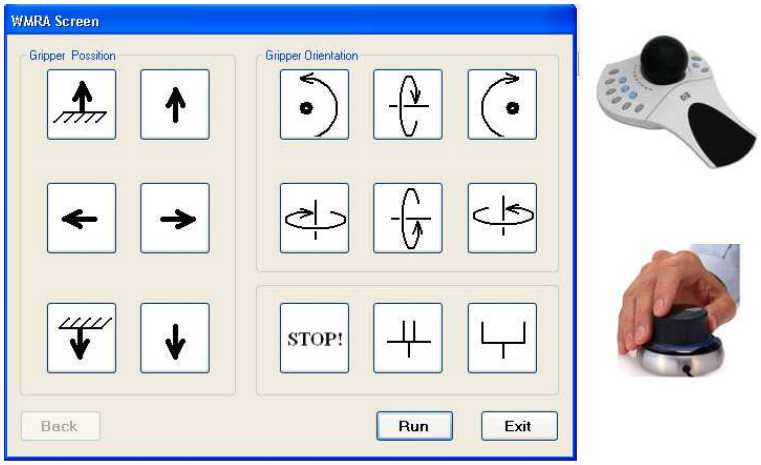

Figure 32: Evaluated User Interfaces: Touch-Screen and Spaceball

6.3.1. Touch-Screen Evaluation

Touch-Screen is one of the popular and convenient user interface used in assistive devices. Its graphical user interface should be simple enough to provide directions to control the robotic arm as desired. As mentioned in Chapter 5, it provides buttons indicating directions of movement of the end-effector so that the users can clearly visualize and identify which buttons to be used to achieve the desired motion. In order to begin the motion, the user should click on the proper buttons that provide the required direction of motion and select *Run* button to move the robotic arm. *Stop* button of the Touch-Screen should be clicked if the user wants to stop the motion. An Exit button is provided to exit from the Touch-Screen user interface.

6.3.1.1. Testing Procedure and Results

First the Touch-Screen was separately evaluated for three translations and rotations of the end-effector using both C++ and Matlab control programs. Then it was tested for combined motion of the end-effector such as three translations, two translations and one rotation, etc. It was tested for different cases of combined motion with different speeds of the end-effector. The robotic arm could be moved successfully to the intended directions.

6.3.2. Spaceball Evaluation

For some users who do not have enough dexterity in their finger movement, Touch-Screen is not a convenient interface to control the robotic arm. Spaceball, which is a device that consist of a ball and sensors to detect its 6-DOF motion, can be helpful in such situations (see section 5.8.2). By using the Spaceball, the user can separately control three translations and three rotation of the gripper. However, the user should be trained properly before using the Spaceball to move the robotic arm. Gripper open and close operations cannot be performed using the Spaceball. The gripper operation can be programmed to the buttons available on the spaceball.

6.3.2.1. Testing Procedure and Results

First, the coordinate system of the gripper was mapped to match the coordinate system at the base of the robotic arm. Then the Spaceball could be easily tested for forward, backward, left and right motion of the robotic arm. These directions were selected according to conventional description of movement of the robotic arm for a user sitting on the wheelchair.

Sensitivity of the Spaceball motion can be set from the Spaceball software or from the twelve-way buttons. Accuracy of the displacement reading of the Spaceball, maximum value for each translation and rotation, and the rate of change of the Spaceball movement are important in deciding the movement of the robotic arm. Sensitivity of the Spaceball was adjusted using a trial and error procedure. Finally, the Spaceball was tested for the robotic arm motion for independent translation and rotation.

It was noticed that the response delay is high when using Matlab program than C++ program. This is because both Spacball and Matlab programs share common global variables and it will take some time to pass the Spaceball variable to Matlab programs. Also Matlab take longer time to interpret the codes.

6.4. Manipulability Measure of WMRA-II

Manipulability measure can be used as a criterion to be maximized to place the robotic arm in a configuration far from its singular configurations and to ensure dexterous manipulation. The method proposed by Yoshikawa [26] was used in this work to calculate the manipulability measure. When the manipulator is redundant, infinite solutions exist for the inverse kinematics. In this case, criteria are needed in order to extract the best solution to run the robotic arm.

Manipulability of WMRA-I and WMRA-II were evaluated and compared for several positions in the workspace using an already developed Matlab simulation. When both robotic arms were in ready position [90, 90, 90, 0,-90, 0], it was found that the manipulability measure for WMRA-II is higher than that for WMRA-I. Furthermore, for the selected end-effector positions, WMRA-II showed higher manipulability measures during the motion. Figure 33 shows the details

of how manipulability measure changed when the end-effector moved from ready position to selected positions with respect to wheelchair base frame coordinate system.

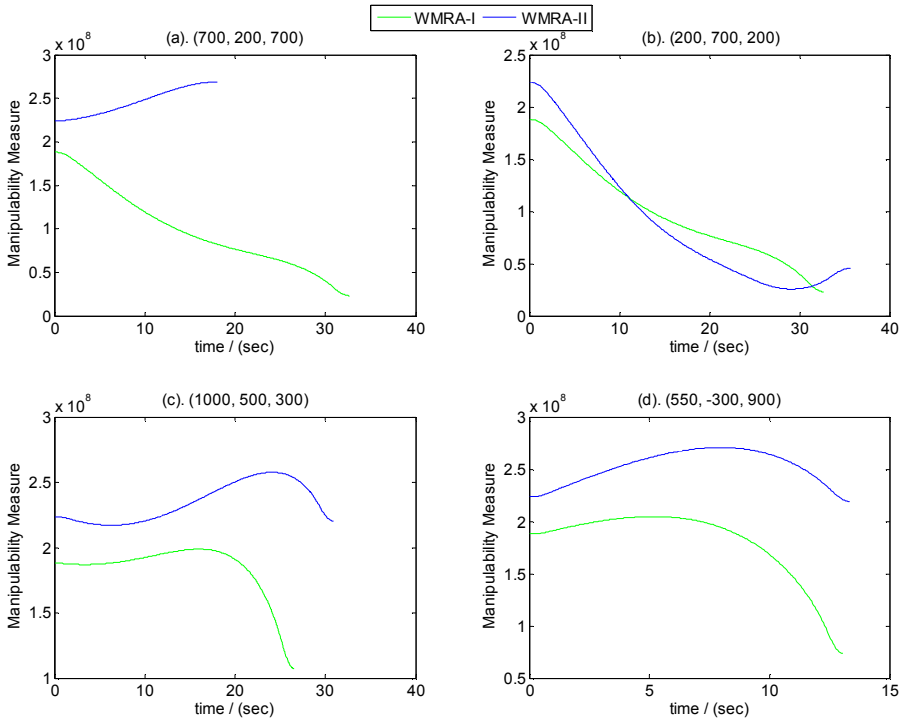

Figure 33: Manipulability Measure of WMRA-I and WMRA-II Robotic Arms

Chapter 7:

Conclusion

Research carried out in this thesis yielded satisfactory results in controlling and testing an assistive robotic wheelchair system (USF WMRA-II) that can be integrated into the everyday life of people with disabilities. The outcome of this research can be summarized as follows.

1. The robotic arm of USF WMRA-II was successfully controlled using not only C++ but also Matlab programming languages.
2. USF WMRA-II was tested in autonomous mode as well as using Touch-Screen and Spaceball as convenient user interfaces. It was identified that the Touch-Screen control is more satisfactory than the Spaceball control, because, Spaceball requires more training and it should be calibrated depending on the disability level of the user. Touch-Screen requires minimal or no training and provides clear and simple insight to the user. However, motion of the WMRA-I for both Touch-Screen and Spaceball was found to be less smooth and less reliable.
3. It was found that WMRA-II is more repeatable and accurate in performance than WMRA-I.
4. Manipulability measure of WMRA-II is higher when it is in its ready position. It also showed higher manipulability measures for most of the predefined positions.
5. WMRA-I consumes less power when it is in idle and during the motion. The power usage of the system mainly depends on payload on the gripper, type of motors used in the arm and the type of the controller board used. When WMRA-II is holding its own weight it consumes more power than WMRA-I. This is because weight of the WMRA-II is higher than WMRA-I and torque applies on motors are high in WMRA-II.

Many improvements can be done to the system to increase the capacity and quality of doing ADL tasks. Wheelchair controllers have to be integrated to the control algorithm. So that the capability of performing combined motion tasks such as opening a door can be achieved. Further force/torque sensors should be added to control the compliance of the system when performing tasks like opening a door. User evaluation can be done for ADL tasks. Sensors for collision avoidance and autonomous navigation can also integrate to make the system more intelligent.

References

[1] J R Allen, A Karchak, and E.L. Bontrager, “Design and fabrication of a pair of Rancho anthropomorphic arm”, Technical report, Rancho Loas Amigos Hospital, Inc, 1972

[2] Z.Zenn Bien, Dimitar Stefanov “Advances in Rehabilitation Robotics”, Vol. 306, pp. 25, 2004

[3] K. Kawamura and M. Iskarous, “Trends in service robots for the disabled and the elderly”, Special session for the disabled and elderly people, 1994.

[4] H.F. Machiel Van der Loos “Lesson learned in the application of robotics technology to the field of rehabilitation”, IEEE Transaction on Rehabiliatation Engineering, Vol. 3, No. 1, pp 46-55, 1995.

[5] H.F.M Van der Loos, J.J Wanger, N. Smaby, K. Chang, O. Madrigal, L.J. Leifer and O.Khotib, “ProVAR assistive robot system architecture”, IEEE International Conference on Robotics and Automation, 1999.

[6] T. Jones, “Raid: Toward greater independence in the office and home environment.” IEEE International Conference on Rehabilitation Robotics”, pp 201-206, 1999.

[7] Kevin Edwards, Redwan Alqasemi, Rajiv Dubey, “Design, Construction and Testing of a Wheelchair-Mounted Robotic Arm", Proceedings of the 2006 IEEE International Conference on Robotics and Automation,2006.

[8] M. Topping, "An Overview of the Development of Handy 1, a Rehabilitation Robot to Assist the Severely Disabled”, Journal of Intelligent and Robotic Systems, pp. 253, 2002.

[9] S.Ishii, S. Tanaka, and F. Hiramatsu, “Meal assistance robot for severely handicapped people”, IEEE Conference on Rehabilitation and Automation, pp.1308 1995.

[10] A. Casals, R. Villa, and D. Casals, “A soft assistance arm for telegraphic”, 1st TIDE cong., pp. 103, 1993.

[11] H. Eftring and K. Boschiann, “Technical results from Manus user trials”, IEEE Conference on Rehabilitation robotics, pp. 136, 1999.

[12] N. Suzuki, K. Masamune, I. Sakuma, " System assisting walking and carrying daily necessities with an overhead robot arm for in- home elderlies" IEEE Int. Conf. on Engineering and Medicine and Biology. Vol. 3. pp. 3371, 2000.

[13] Z.N. Bien, M. J. Chung, P.H. Chang, D.S. kwon, “ Integration of a Rehabilitation Robotic System (KARES 11) with Human-Friendly Man-Machine Interaction Units”, Autonomous Robots, Vol.16, No.2. pp. 165, 2004.

[14] H.H. Duimel Kwee, Tuinhofde Moed and J.A. Van Woerden, “The manus wheelchair-borne manipulator: System Review and first results”, In IARP, 2nd Workshop Medical and Healthcare Robotics, pp. 385, 1989.

[15] F. Bley, M. Rous, U. Canzler, and K. Karl-Frieddrich, “Supervised navigation and manipulation for impaired wheelchair users”, IEEE Transaction on Machine Man and Cybernetics, pp. 152, 2004.

[16] R. Mahoney, “The raptor wheelchair robot system”, IEEE International Conference on Rehabilitation Robotics”, pp 135, 2001.

[17] H. Neveryd, G. Bolmsj. Walky, “An ultrasonic navigating mobile robot for the disabiled”, TIDE, pp. 366, 1995.

[18] M.Busnel, R Cammoun, F. Coulon-Lauture, j-M Detriche, G Le Claire, and B.Lasigne, “The robotized workstation master for users with tetraplegia: Description and Evaluation" Journal of Rehabilitation Research and Development” Vol. 36, No. 3, 1999.

[19] S.D. Prior , “A Review of World Rehabilitation Robotics”, Middlesex Polytechnic internal report, 1989.

[20] Dae-Jin Kim Aman Behal, “Human-in-the-Loop Control of an Assistive Robotic Arm in Unstructured Environments for Spinal Cord Injured Users”, 4th ACM/IEEE International Conference on Human Robot Interaction, pp. 285, 2009.

[21] Vijay Kumar, “Assistive Devices For People With Motor Disabilities”, Wiley Encyclopedia of Electrical and Electronics Engineering, 1997.

[22] John J. Craid, "Introduction to Robotics”, Third Edition, 1989.

[23] H.L Iiassler, “Robotics for health care: A review of the literature,” Robotica, vol. 11, pp. 495, 1993.

[24] Noriyuki Tejima, “Evaluation of Rehabilitation Robots for Eating”, IEEE International Workshop on Robot and Human Communication , pp. 118, 1996.

[25] Katherine Tsui and Holly Yanco, David Kontak and Linda Beliveau, "Development and Evaluation of a Flexible Interface for a Wheelchair Mounted Robotic Arm", HRI, 2008.

[26] Redwan M. Alqasemi, "Maximizing Manupulation Capabilites of Persons with DIsabilities Using a Smatrt 9-Degree-of-Freedom Wheelchair-Mounted Robotic Arm System ", University of South Florida, 2007.

[27] Galil motion controller user manual

[28] http://www.motion-control-info.com/optical-encoder_design_guide.html

[29] S.R. Deb, "Robotics technology and flexible automation", 1994.

[30] Ana Catalina Torres Rocco, “Development and Testing of a New C-Based Algorithm to Control a 9-Degree-of-Freedom Wheelchair-Mounted-Robotic-Arm System", University of South Florida, 2010.

[31] Van der Loos, H. F. M., Wagne, J. J., Smaby, N., 1999, "ProVAR Assistive Robot System Architecture", International Conference on Robotics and Automation, pp. 741,1999.

[32] Machiel, H. F., and Van, d. L., 2000, "Immersive user environments in rehabilitation robotics and mechatronics", Artificial Life and Robotics, Vol. 4, pp. 176, 2006.

[33] Ahmad Rasdan Ismail Azmi, Hassan Syamimi Syamsuddin, Mohd Zaki Nuawi, Shahrum Abdullah, Hairunnisa Mohamad Ibrahim, "The performance analysis of industrial robot under loaded conditions and various distance", 8th WSEAS International Conference on Robotics, Control and Manufacturing Technology, pp.75,2008.

[34] Axel Graser, “Technological Solutions to Autonomous Robot Control”, TIDE, 1998.

[35] Katherine M. Tsui, "Design and Evaluation of a Visual Control Interface of a Wheelchair Mounted Robotic Arm for Users with Cognitive Impairments", University of Massachusetts Lowell, 2008.

[36] Michael Hillman, Karen Hagan, Sean Hagan, Jill Jepson, Roger Orpwood, "The Weston Wheelchair Mounted Assistive Robot - The Design Story", Robotica, Vol. 20, pp. 125, 2002.

[37] http://www.galilmc.com/catalog/techref.pdf

[38] kiam heong ang, gregory chongpid,"control system analysis, design, and technology" IEEE Transactions on ontrol systems technology, Vol. 13, No. 4, 2005.

Appendices

Appendix A. C++ Programs

```
/* 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 %%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%Modified By: Ana Catalina Torres & Punya A. Basnayaka %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% June 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "any2ready.h"
#include "ArmMotion.h"
#include "DH.h"
#include "q2T.h"
#include <time.h>
using namespace std;
using namespace math;

void WMRA_any2ready(int ini, int vr, int ml, int arm, Matrix Tiwc, Matrix qi, Galil &g){
Matrix zero(1,1);
zero.Null(1,1);
        if (ini==3){
                if (arm==1){
                        try {
             WMRA_ARM_Motion(ini, 0, zero, 0, g);
                        cout<< "any ini "<<endl;
                        }
                        catch (...) {
                                cout << "Exception 1 occurred";
                        }
                }
                if (vr==1){

                }
                if (ml==1){
                        /*try {
                                WMRA_ML_Animation(ini, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
                        }
                        catch (...) {
                                cout << "Exception 3 occurred";
                        }*/
                }
                return;
        }

        // Defining the used conditions:
        Matrix qd(7,1);
        float ts, dt, ddt;
        int n, j;
        float qd2 [7]= {PI/2, PI/2, 0, PI/2, PI/2, PI/2, 0}; // Final joint angles (Ready
Position).
        for ( j = 0 ; j < 7 ; j++ ) {
                qd(j,0)=qd2[j];
        }
```

Appendix A. (Continued)

```
        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.

        FILE * fel;
        fel = fopen("Any2ready.txt","w");
        fprintf(fel," qd is: \n\n");
        int k;
        for (j=0;j<qd.RowNo();j++){
                for (k=0;k<qd.ColNo();k++){
                        fprintf(fel," %4.2f ",  qd(j,k));
                }
                fprintf(fel," \n");
        }
        fprintf(fel," \n\n\n");
        fprintf(fel," ts is %4.2f ",  ts);
        fprintf(fel," \n\n\n");
        fprintf(fel," n is %d ",  n);
        fprintf(fel," \n\n\n");
        fprintf(fel," dt is %4.2f ",  dt);
        fprintf(fel," \n\n\n");

        // Initializing the physical Arm:
        if (arm==1){
                zero.Null(10,1);
                for ( j = 0 ; j < 9 ; j++ ) {
                        zero(j,0)=qi(j,0);
                }
                cout<<"0";
                WMRA_ARM_Motion(ini, 2, zero, dt, g);
                ddt=0;
                cout<<"1"<<endl;
        }
        fprintf(fel," qi is: \n\n");
        for (j=0;j<qi.RowNo();j++){
                for (k=0;k<qi.ColNo();k++){
                        fprintf(fel," %4.2f ",  qi(j,k));
                }
                fprintf(fel," \n");
        }
        fprintf(fel," \n\n\n");

        if (vr==1){
        //      WMRA_VR_Animation(ini, Tiwc, qi);
        }

        // Initializing Robot Animation in Matlab Graphics:
        Matrix DH(1,1);
        Matrix T01(4,4), T12(4,4), T23(4,4), T34(4,4), T45(4,4), T56(4,4), T67(4,4);
        Matrix Ti(4,4), Td(4,4);

        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(0,0))*WMRA_transl(DH(0,1),0,0)*WMRA_rotz(DH(0,3))*WMRA_transl(0,0,DH(0,2));
                                T12 =
WMRA_rotx(DH(1,0))*WMRA_transl(DH(1,1),0,0)*WMRA_rotz(DH(1,3))*WMRA_transl(0,0,DH(1,2));
                //T3
                T23 =
WMRA_rotx(DH(2,0))*WMRA_transl(DH(2,1),0,0)*WMRA_rotz(DH(2,3))*WMRA_transl(0,0,DH(2,2));
```

Appendix A. (Continued)

```
                //T4
                T34 =
WMRA_rotx(DH(3,0))*WMRA_transl(DH(3,1),0,0)*WMRA_rotz(DH(3,3))*WMRA_transl(0,0,DH(3,2));
                //T5
                T45 =
WMRA_rotx(DH(4,0))*WMRA_transl(DH(4,1),0,0)*WMRA_rotz(DH(4,3))*WMRA_transl(0,0,DH(4,2));
                //T6
                T56 =
WMRA_rotx(DH(5,0))*WMRA_transl(DH(5,1),0,0)*WMRA_rotz(DH(5,3))*WMRA_transl(0,0,DH(5,2));
                //T7
                T67 =
WMRA_rotx(DH(6,0))*WMRA_transl(DH(6,1),0,0)*WMRA_rotz(DH(6,3))*WMRA_transl(0,0,DH(6,2));
                // 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);
        }

        // Check for the shortest route:
        Matrix diff(7,1);
        for (j=0;j<7;j++){
                diff(j,0)=qd(j,0)-qi(j,0);
        }
        for (j=0;j<7;j++){
                if (diff(j,0) > PI) {
                        diff(j,0)=diff(j,0)-2*PI;
                }
                else if (diff(j,0) < (-PI)) {
                        diff(j,0)=diff(j,0)+2*PI;
                }
        }

        fprintf(fel," diff is: \n\n");
        for (j=0;j<diff.RowNo();j++){
                for (k=0;k<diff.ColNo();k++){
                        fprintf(fel," %4.2f ",  diff(j,k));
                }
                fprintf(fel," \n");
        }
        fprintf(fel," \n\n\n");

                diff/=(n);
        Matrix dq(1,1);
        dq.Null(9,1);
        for ( j = 0 ; j < 7 ; j++ ) {
                dq(j,0)=diff(j,0);
        }

        fprintf(fel," dq is: \n\n");
        for (j=0;j<dq.RowNo();j++){
                for (k=0;k<dq.ColNo();k++){
                        fprintf(fel," %4.2f ",  dq(j,k));
                }
                fprintf(fel," \n");
        }
        fprintf(fel," \n\n\n");

        // Initialization:
        Matrix qo(1,1);
        float tt;
        qo=qi;
        tt=0;
        fprintf(fel," qo is: \n\n");
        for (j=0;j<qo.RowNo();j++){
                for (k=0;k<qo.ColNo();k++){
```

Appendix A. (Continued)

```
                fprintf(fel," %4.2f ",  qo(j,k));
                }
                fprintf(fel," \n");
        }
        fprintf(fel," \n\n\n");
        fprintf(fel," tt is %4.2f ",  tt);
        fprintf(fel," \n\n\n");

        Matrix qn(1,1);
        clock_t startt, endt, endf;
        double timedif;
        Matrix T1a(1,1), T2a(1,1), T3a(1,1), T4a(1,1), T5a(1,1), T6a(1,1), T7a(1,1);

        while (tt <= (ts-dt)) {
                // Starting a timer:
                startt=0;
                startt=clock()/ CLOCKS_PER_SEC;

                // Calculating the new Joint Angles:
                qn.Null(9,1);
                qn=qo+dq;

                fprintf(fel," qn is: \n\n");
                for (j=0;j<qn.RowNo();j++){
                        for (k=0;k<qn.ColNo();k++){
                                fprintf(fel," %4.2f ",  qn(j,k));
                        }
                        fprintf(fel," \n");
                }
                fprintf(fel," \n\n\n");
                fprintf(fel," ddt is %4.2f ",  ddt);
                fprintf(fel," \n\n\n");

                // Updating the physical Arm:
                if (arm==1) {
                        float ddt2=0;
                        ddt2=ddt+dt;
                        if (ddt2>=0.5 || tt>=(ts-dt)){
                                zero.Null(10,1);
                                for ( j = 0 ; j < 9 ; j++ ) {
                                        zero(j,0)=qn(j,0);
                                }
                                cout<<"2";
                                WMRA_ARM_Motion(2, 1, zero, ddt2, g);
                                cout<<"3";
                                ddt2=0;
                        }
                        ddt=0;
                        ddt=ddt2;
                }

                // Updating Virtual Reality Animation:
                if (vr==1){
                        //WMRA_VR_Animation(2, Tiwc, qn);
                }

                // Updating Matlab Animation:
                if (ml==1){
                        // Calculating the new Transformation Matrix:
                        T1a =
WMRA_rotx(DH(0,0))*WMRA_transl(DH(0,1),0,0)*WMRA_rotz(qn(0,0))*WMRA_transl(0,0,DH(0,2));
                //T2
                T2a =
WMRA_rotx(DH(1,0))*WMRA_transl(DH(1,1),0,0)*WMRA_rotz(qn(1,0))*WMRA_transl(0,0,DH(1,2));
                //T3
```

Appendix A. (Continued)

```
                T3a=
WMRA_rotx(DH(2,0))*WMRA_transl(DH(2,1),0,0)*WMRA_rotz(qn(2,0))*WMRA_transl(0,0,DH(2,2));
                //T4
                T4a =
WMRA_rotx(DH(3,0))*WMRA_transl(DH(3,1),0,0)*WMRA_rotz(qn(3,0))*WMRA_transl(0,0,DH(3,2));
                //T5
                T5a =
WMRA_rotx(DH(4,0))*WMRA_transl(DH(4,1),0,0)*WMRA_rotz(qn(4,0))*WMRA_transl(0,0,DH(4,2));
                //T6
                T6a =
WMRA_rotx(DH(5,0))*WMRA_transl(DH(5,1),0,0)*WMRA_rotz(qn(5,0))*WMRA_transl(0,0,DH(5,2));
                //T7
                T7a =
WMRA_rotx(DH(6,0))*WMRA_transl(DH(6,1),0,0)*WMRA_rotz(qn(6,0))*WMRA_transl(0,0,DH(6,2));

                //WMRA_ML_Animation(2, Ti, Td, Tiwc, T1a, T2a, T3a, T4a, T5a, T6a, T7a);
                }

                // Updating the old values with the new values for the next iteration:

                qo.Null(9,1);
                qo=qn;
                tt=tt+dt;

                fprintf(fel," qo is: \n\n");
                for (j=0;j<qo.RowNo();j++){
                        for (k=0;k<qo.ColNo();k++){
                                fprintf(fel," %4.2f ",  qo(j,k));
                        }
                        fprintf(fel," \n");
                }
                fprintf(fel," \n\n\n");
                fprintf(fel," tt is %4.2f ",  tt);
                fprintf(fel," \n\n\n");

                // Pausing for the speed sync:
                endt=0;
                endt=clock()/ CLOCKS_PER_SEC;
                timedif=0;
                timedif=endt-startt;
                wait((dt-timedif));
        }
        fclose(fel);
}
```

Appendix A. (Continued)

```
/* 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: Punya Basnayaka %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Feb 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "WCD.h"
#include "ArmMotion.h"
#include <math.h>
#include <string>
#include <iostream>
#include <sstream>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <time.h> //if #include<windows.h> is included, there will be error messages
#include "control.h"
//#include "C:\Documents and Settings\pbasnaya\Desktop\My
Research\WMRA2all\functiontest\functiontest\controlt2.h"
#include "Galil.h"
using namespace std;
using namespace math;
using namespace System;

void sleep(long d) //sleep function takes in ms value
{ //sleep(1000) sleeps for 1 second
        clock_t start=clock();
        while(clock() - start < d);
}

Matrix WMRA_ARM_Motion(int ind, int config, Matrix qo, float dt, Galil &g){

        //extern long * ptr;
        extern float e2r1, e2r2, e2r3, e2r4, e2r5, e2r6, e2r7, e2r8, e2r9, e2d;
        long int aa, bb, cc, dd, ee, ff, gg, hh;
        //long int aa1, bb2, cc3, dd4, ee5, ff6, gg7, hh8;
        string a1, b1, c1,d1,e1,f1,g1;
        float  Kp, Kd, Ki;
        Matrix qn(10,1), qntemp(10,1);
        Matrix L(1,1);

// Reading the Wheelchair's constant dimentions, all dimentions are converted in
millimeters:
        L=WMRA_WCD();

//The initialization of the Arm library:
        if (ind==1){

// PID controller gains:

                Brushed_Motor ('A', 1,g);
```

Appendix A. (Continued)

```
                e2r1=PI/6595000;//a
                e2r2=PI/5100000;//b
                e2r3=PI/1800000;//c
                e2r4=PI/2300000;//d
                e2r5=PI/2000000;//e
                e2r6=-PI/2000000;//f
                e2r7=PI/1500000; //g
                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;//h

// The case when changing the configuration file to qo is required:

                if (config==2){

// Converting the commanded angles to encoder readings:

                        qo(0,0)=qo(0,0)/e2r1;
                        qo(1,0)=qo(1,0)/e2r2;
                        qo(2,0)=qo(2,0)/e2r3;
                        qo(3,0)=qo(3,0)/e2r4;
                        qo(4,0)=qo(4,0)/e2r5;
                        qo(5,0)=qo(5,0)/e2r6;
                        qo(6,0)=qo(6,0)/e2r7;
                        qo(7,0)=qo(7,0)/e2r8;
                        qo(8,0)=qo(8,0)/e2r9;
                        qo(9,0)=0;

// Changing the configuration file to qo:
                        FILE * fid;
                        fid = fopen("configuration.txt","w");
                        int j;
                        for (j=0;j<7;j++){
                                fprintf(fid," %10.0f ",qo(j,0));
                        }
                        fclose (fid);
                        config=1;
                }
                g.command("KPH=2");
                g.command("KDH=30");
                g.command("KIH=0");
                g.command("KPA=35");
                g.command("KDA=260");
                g.command("KIA=0");
// This command sets the controller board in postion tracking mode

                Pos_track (g);
                Decceleration_all (90000, g);
                Time_constant (0.6, g);

                long int qp[10];

                qp[0]=qo(0,0);
                //cout<<"qp[0] = "<<qp[0]<<endl;
                qp[1]=qo(1,0);
                //cout<<"qp[1] = "<<qp[1]<<endl;
                qp[2]=qo(2,0);
                //cout<<"qp[2] = "<<qp[2]<<endl;
                qp[3]=qo(3,0);
                //cout<<"qp[3] = "<<qp[3]<<endl;
                qp[4]=qo(4,0);
                //cout<<"qp[4] = "<<qp[4]<<endl;
                qp[5]=qo(5,0);
```

Appendix A. (Continued)

```
                qp[6]=qo(6,0);
                //cout<<"qp[6] = "<<qp[6]<<endl;
                qp[7]=qo(7,0);
                //cout<<"qp[7] = "<<qp[7]<<endl;
                qp[8]=qo(8,0);
                //cout<<"qp[8] = "<<qp[8]<<endl;
                qp[9]=qo(9,0);
                cout<<"qp[9] = "<<qp[9]<<endl;

                Define_Position ('A', qo(0,0), g);
                Define_Position ('B', qo(1,0), g);
                Define_Position ('C', qo(2,0), g);
                Define_Position ('D', qo(3,0), g);
                Define_Position ('E', qo(4,0), g);
                Define_Position ('F', qo(5,0), g);
                Define_Position ('G', qo(6,0), g);
                Define_Position ('H', qo(9,0), g);

        //PID Gains setting

                PID_all( 6,64,0,g);

        aa=Tell_pos('A', g);
        bb=Tell_pos('B', g);
        cc=Tell_pos('C', g);
        dd=Tell_pos('D', g);
        ee=Tell_pos('E', g);
        ff=Tell_pos('F', g);
        gg=Tell_pos('G', g);
        hh=Tell_pos('H', g);

                long int qc[10];

                qc[0]=aa;
                qc[1]=bb;
                qc[2]=cc;
                qc[3]=dd;
                qc[4]=ee;
                qc[5]=ff;
                qc[6]=gg;
                qc[7]=0;
                qc[8]=0;
                qc[9]=hh;

                qn(0,0)=qc[0]*e2r1;
                qn(1,0)=qc[1]*e2r2;
                qn(2,0)=qc[2]*e2r3;
                qn(3,0)=qc[3]*e2r4;
                qn(4,0)=qc[4]*e2r5;
                qn(5,0)=qc[5]*e2r6;
                qn(6,0)=qc[6]*e2r7;
                qn(7,0)=qc[7]*e2r8;
                qn(8,0)=qc[8]*e2r9;
                qn(9,0)=qc[9]*e2d;// gripper
                cout<<"qn(9,0)"<<qn(9,0)<<endl;

                }

// Closing the Arm library:

        else{
                if (ind==3){
                qn.Null(10,1);
        cout<<"qn" <<qn<<endl;
                }
```

Appendix A. (Continued)

```
// Updating the Arm:
                else {

                if (ind==2){

                        try
        {
                long int qin[10];

        aa=Tell_pos('A', g);
        bb=Tell_pos('B', g);
        cc=Tell_pos('C', g);
        dd=Tell_pos('D', g);
        ee=Tell_pos('E', g);
        ff=Tell_pos('F', g);
        gg=Tell_pos('G', g);
        hh=Tell_pos('H', g);

        long int qc[10];

                qc[0]=aa;
                qc[1]=bb;
                qc[2]=cc;
                qc[3]=dd;
                qc[4]=ee;
                qc[5]=ff;
                qc[6]=gg;
                qc[7]=0;
                qc[8]=0;
                qc[9]=hh;
                cout<<"qc[9] = "<<qc[9]<<endl;


                float qt[10];
                float qg[1];
                 qo(0,0)=qo(0,0)/e2r1;
                 qo(1,0)=qo(1,0)/e2r2;
                 qo(2,0)=qo(2,0)/e2r3;
                 qo(3,0)=qo(3,0)/e2r4;
                 qo(4,0)=qo(4,0)/e2r5;
                 qo(5,0)=qo(5,0)/e2r6;
                 qo(6,0)=qo(6,0)/e2r7;
                 qo(7,0)=qo(7,0)/e2r8 ;
                 qo(8,0)=qo(8,0)/e2r9;

                 if(qo(9,0)==0)
                 {
                        qo(9,0)=0;
                        qo(9,0)=qo(9,0)+qc[9];

                 }
                 else
                 {
                        if(qo(9,0)==1){
                 qo(9,0)=-qo(9,0)*100000 ;
                 qo(9,0)=qo(9,0)+qc[9];
                        }
                        if (qo(9,0)==-1){
                        qo(9,0)=-qo(9,0)*100000 ;
                 qo(9,0)=qo(9,0)+qc[9];
                        }

                 }
```

Appendix A. (Continued)

```
            cout<<"qg[1] =  "<<qo(9,0)<<endl;

            float dq[8];

             dq[0]=qo(0,0)-qc[0];
             dq[1]=qo(1,0)-qc[1];
             dq[2]=qo(2,0)-qc[2];
             dq[3]=qo(3,0)-qc[3];
             dq[4]=qo(4,0)-qc[4];
             dq[5]=qo(5,0)-qc[5];
             dq[6]=qo(6,0)-qc[6];
             dq[7]=qo(9,0)-qc[9];


      long int dq1[8];


             dq1[0]=abs(dq[0]);
             dq1[1]=abs(dq[1]);
             dq1[2]=abs(dq[2]);
             dq1[3]=abs(dq[3]);
             dq1[4]=abs(dq[4]);
             dq1[5]=abs(dq[5]);
             dq1[6]=abs(dq[6]);
             dq1[7]=abs(dq[7]);

             cout<<"dq[7] =  "<<dq[7]<<endl;


                    long int dq01[8];
                    long int dq01t[8];
                    long int qddo1[8];

                    dq01t[7]=dq1[7]/2;


                    qddo1[7]=5000;

            if (abs(dq[0])>25000)
            {
                    dq01[0]=170000;
                    qddo1[0]=75000;
            }
            else{
                    dq01[0]=2*abs(dq[0])/0.3;;
                    qddo1[0]=2*abs(dq[0])/0.7;

            }
            cout<<"dq01[0] =  "<<dq01[0]<<endl;
            cout<<"qddo1[0]= "<<qddo1[0]<<endl;
            ///////////////

            if (abs(dq[1])>25000)
            {
                    dq01[1]=170000;
                    qddo1[1]=75000;
            }
            else{
                    dq01[1]=2*abs(dq[1])/0.3;;
                    qddo1[1]=2*abs(dq[1])/0.7;

            }
            cout<<"dq01[1] =  "<<dq01[1]<<endl;
            cout<<"qddo1[1]= "<<qddo1[1]<<endl;
            ////////////////////////
```

Appendix A. (Continued)

```
if (abs(dq[2])>25000)
{
        dq01[2]=170000;
        qddo1[2]=75000;
}
else{
        dq01[2]=2*abs(dq[2])/0.3;;
        qddo1[2]=2*abs(dq[2])/0.7;

}
cout<<"dq01[2] =  "<<dq01[2]<<endl;
cout<<"qddo1[2]= "<<qddo1[2]<<endl;
////////////////////
if (abs(dq[3])>25000)
{
        dq01[3]=170000;
        qddo1[3]=75000;
}
else{
        dq01[3]=2*abs(dq[3])/0.3;;
        qddo1[3]=2*abs(dq[3])/0.7;

}
cout<<"dq01[3] =  "<<dq01[3]<<endl;
cout<<"qddo1[3]= "<<qddo1[3]<<endl;
////////////////////////

if (abs(dq[4])>25000)
{
        dq01[4]=170000;
        qddo1[4]=75000;
}
else{
        dq01[4]=2*abs(dq[4])/0.3;;
        qddo1[4]=2*abs(dq[4])/0.7;

}
cout<<"dq01[4] =  "<<dq01[4]<<endl;
cout<<"qddo1[4]= "<<qddo1[4]<<endl;
////////////////////
if (abs(dq[5])>25000)
{
        dq01[5]=170000;
        qddo1[5]=75000;
}
else{
        dq01[5]=2*abs(dq[5])/0.3;;
        qddo1[5]=2*abs(dq[5])/0.7;

}
cout<<"dq01[5] =  "<<dq01[5]<<endl;
cout<<"qddo1[5]= "<<qddo1[5]<<endl;
////////////////////
if (abs(dq[6])>25000)
{
        dq01[6]=170000;
        qddo1[6]=75000;
}
else{
        dq01[6]=2*abs(dq[6])/0.3;;
        qddo1[6]=2*abs(dq[6])/0.7;

}
cout<<"dq01[6] =  "<<dq01[6]<<endl;
```

Appendix A. (Continued)

```
int j;
FILE * fid1;
fid1 = fopen("arm1.txt","a");
fprintf(fid1," \n\n\n");
                fprintf(fid1," qc[3] is %4.2f ",  qc[3]);
                fprintf(fid1," \n\n\n");



                Position_control('a',qo(0,0),dq01[0],qddo1[0], g);

                Position_control('b',qo(1,0),dq01[1],qddo1[1], g);

                Position_control('c',qo(2,0),dq01[2],qddo1[2], g);

                Position_control('d',qo(3,0),dq01[3],qddo1[3], g);

                Position_control('e',qo(4,0),dq01[4],qddo1[4], g);

                Position_control('f',qo(5,0),dq01[5],qddo1[5], g);
                //Position_control('g',qo(6,0),dq01[6],qddo1[6], g);
                Position_control('h',qo(9,0),dq01t[7],5000, g);


        aa=Tell_pos('A', g);
        bb=Tell_pos('B', g);
        cc=Tell_pos('C', g);
        dd=Tell_pos('D', g);
        ee=Tell_pos('E', g);
        ff=Tell_pos('F', g);
        gg=Tell_pos('G', g);
        hh=Tell_pos('H', g);


                qc[0]=aa;
                qc[1]=bb;
                qc[2]=cc;
                qc[3]=dd;
                qc[4]=ee;
                qc[5]=ff;
                qc[6]=gg;
                qc[7]=0;
                qc[8]=0;
                qc[9]=hh;

                qn(0,0)=qc[0]*e2r1;
                qn(1,0)=qc[1]*e2r2;
                qn(2,0)=qc[2]*e2r3;
                qn(3,0)=qc[3]*e2r4;
                //cout<<"qn(3,0) = "<<qn(3,0)<<endl;
                qn(4,0)=qc[4]*e2r5;
                qn(5,0)=qc[5]*e2r6;
                qn(6,0)=qc[6]*e2r7;
                qn(7,0)=qc[7]*e2r8;
                qn(8,0)=qc[8]*e2r9;
                qn(9,0)=qc[9]*e2d;
                cout <<"qn"<<qn<<endl;
                fprintf(fid1," \n\n\n");
                fprintf(fid1," qn(3,0) is %4.2f ", qn(3,0));
                fprintf(fid1," \n\n\n");
                fclose(fid1);
                }
        catch(string e)
        {
                cout << e;
```

Appendix A. (Continued)

```
}

                }
                }
                }


        return qn;
}

                }

/* 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 %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%Modified by: Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/
#include "matrix.h"
#include "BPolynomial.h"
using namespace std;
using namespace math;

Matrix WMRA_BPolynomial(float qi, float qf, float n){
        Matrix qtb(2,1);
        // Blending Factor:
        int b;
        b=5;
        // Initializing the time:
        float tt, tf, dt, qddb, tb, qdb, qb;
        float a01, a11, a21, a31, a41, a51, a02, a12, a22, a32, a42, a52;
        int i;
        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)/pow(tf,2);
                tb=tf/2-sqrt(pow(qddb,2)*pow(tf,2)-4*qddb*(qf-qi))/abs(2*qddb);
                qdb=qddb*tb;
                qb=qi+qddb*pow(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*pow(tb,2))/(2*pow(tb,3));
                // a41=(30*(qi-qb)+14*qdb*tb+qddb*pow(tb,2))/(2*pow(tb,4)); % Uncomment
for 5th order polynomial.
                // a51=(12*(qb-qi)-6*qdb*tb)/(2*pow(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);
```

Appendix A. (Continued)

```
                a12=qdb;
                a22=-0.5*qddb;
                a32=(20*(qf-a02)-12*a12*tb+2*qddb*pow(tb,2))/(2*pow(tb,3));
                // a42=(30*(a02-qf)+16*a12*tb-qddb*pow(tb,2))/(2*pow(tb,4)); % Uncomment
for 5th order polynomial
                // a52=(12*(qf-a02)-6*a12*tb)/(2*pow(tb,5)); % Uncomment for 5th order
polynomial.
        }

        // Calculating the intermediate joint angles along the trajectory from the initial
to the final position:
        float *qttemp;
        qttemp = new float[n];
        for (i=0; i<n; i++){
                if (tf<=0.001){
                        qttemp[i]=qi;
                }
                else if (tt<=tb){
                        qttemp[i]=a01+a11*tt+a21*pow(tt,2)+a31*pow(tt,3);
//+a41*pow(tt,4)+a51*pow(tt,5); // Uncomment before "+a41" for 5th order polynomial.
                }
                else if (tt>=(tf-tb)){
                        qttemp[i]=a02+a12*(tt+tb-tf)+a22*pow((tt+tb-tf),2)+a32*pow((tt+tb-
tf),3); //+a42*pow((tt+tb-tf),4)+a52*pow((tt+tb-tf),5); // Uncomment before "+42" for 5th
order polynomial.
                }
                else {
                        qttemp[i]=qb-qdb*(tb-tt);
                }
                tt = tt + dt;
        }
        qtb.SetSize(n,1);
        for (i=0; i < n; i++){
                qtb(i,0) = qttemp[i];
        }
        delete [] qttemp;
        return qtb;
        }
```

```
/* 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 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%Modified By:Ana Catalina Torres%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "Collide.h"
#include "WCD.h"
using namespace std;
using namespace math;

Matrix WMRA_collide(Matrix dqi, Matrix T01, Matrix T12, Matrix T23, Matrix T34, Matrix
T45, Matrix T56, Matrix T67){
```

Appendix A. (Continued)

```
Matrix L(1,1);
L=WMRA_WCD();

// Collision Conditions:
int gr;
Matrix dq(1,1);
gr=100-L(0,3)-L(0,4); // The ground buffer surface.
dq=dqi;

// 1- Collision of frame 3 using T03:
Matrix T03(1,1);
T03=T01*T12*T23;
// Collision with the ground:
if (T03(2,3) <= gr){
        dq=dqi;
        dq*=(-0.01);
}
// Collision with the wheelchair's front left side:
if (T03(0,3) >= 450 && T03(1,3) <= -150){
        dq=dqi;
        dq*=(-0.01);
}
// Collision with the wheelchair's rear left side:
if (T03(0,3) <= 450 && T03(1,3) <= 100){
        dq=dqi;
        dq*=(-0.01);
}
// Collision with the wheelchair's rear left wheel:
if (T03(0,3) <= 0 && T03(1,3) <= 100 && T03(2,3) <= 120){
        dq=dqi;
        dq*=(-0.01);
}

// 2- Collision of frame 4 using T04:
Matrix T04(1,1);
T04=T03*T34;
// Collision with the ground:
if (T04(2,3) <= gr){
        dq=dqi;
        dq*=(-0.01);
}
// Collision with the wheelchair's front left side:
if (T04(0,3) <= 450 && T04(0,3) >= -100 && T04(1,3) <= 0){
        dq=dqi;
        dq*=(-0.01);
}
// Collision with the wheelchair's rear left side:
if (T04(0,3) <= -100 && T04(1,3) <= 100){
        dq=dqi;
        dq*=(-0.01);
}
// Collision with the wheelchair's rear left wheel:
if (T04(0,3) <= -100 && T04(1,3) <= 100 && T04(2,3) <= 120){
        dq=dqi;
        dq*=(-0.01);
}

// 3- Collision of frame 5 using T05:
Matrix T05(1,1);
T05=T04*T45;
// Collision with the ground:
if (T05(2,3) <= gr){
        dq=dqi;
        dq*=(-0.01);
}
```

Appendix A. (Continued)

```
        if (T05(0,3) <= -100 && T05(0,3) >= -550 && T05(1,3) <= 150){
                dq=dqi;
                dq*=(-0.01);
        }
        // Collision with the wheelchair driver's lap:
        if (T05(0,3) <= 400 && T05(0,3) >= -100 && T05(1,3) <= 0 && T05(2,3) <= 470){
                dq=dqi;
                dq*=(-0.01);
        }
        // Collision with the wheelchair's battery pack:
        if (T05(0,3) <= -430 && T05(0,3) >= -630 && T05(1,3) <= 100 && T05(2,3) <= 50){
                dq=dqi;
                dq*=(-0.01);
        }

        // 4- Collision of frame 7 using T07:
        Matrix T07(1,1);
        T07=T05*T56*T67;
        // Collision with the ground:
        if (T07(2,3) <= gr){
                dq=dqi;
                dq*=(-0.01);
        }
        // Collision with the wheelchair driver's left shoulder:
        if (T07(0,3) <= -50 && T07(0,3) >= -600 && T07(1,3) <= 200){
                dq=dqi;
                dq*=(-0.01);
        }
        // Collision with the wheelchair driver's lap:
        if (T07(0,3) <= 450 && T07(0,3) >= -50 && T07(1,3) <= 50 && T07(2,3) <= 520){
                dq=dqi;
                dq*=(-0.01);
        }
        // Collision with the wheelchair's battery pack:
        if (T07(0,3) <= -480 && T07(0,3) >= -680 && T07(1,3) <= 50 && T07(2,3) <= 100){
                dq=dqi;
                dq*=(-0.01);
        }

        // 5- Collision of the arm and itself using T37:
        Matrix T37(1,1);
        T37=T34*T45*T56*T67;
        // Collision between the forearm and the upper arm:
        if (T37(0,3) <= 170 && T37(0,3) >= -170 && T37(1,3) >= -100 && T37(2,3) <= 0){
                dq=dqi;
                dq*=(-0.01);
        }
        return dq;
        }

/* This function gives the WMRA's errors from the current position to the required
trajectory position.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%Modified By:Ana Catalina Torres%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "Delta.h"
using namespace std;
using namespace math;
```

Appendix A. (Continued)

```
Matrix WMRA_delta(Matrix Tid, Matrix Tdd){

        Matrix delta(6,1), ep(3,1), eo(3,1);
        int i,j;
        for (i=0; i < 3; i++){
                ep(i,0) = Tdd(i,3)-Tid(i,3);
        }

        vector temp3 (0, 0, 0);
        for (j=0; j<3; j++){
                vector temp1 (Tid(0,j), Tid(1,j), Tid(2,j));
                vector temp2 (Tdd(0,j), Tdd(1,j), Tdd(2,j));
                vector crossed = crossProduct (temp1, temp2);
                temp3.x=temp3.x+crossed.x;
                temp3.y=temp3.y+crossed.y;
                temp3.z=temp3.z+crossed.z;
        }
        eo(0,0)=temp3.x;
        eo(1,0)=temp3.y;
        eo(2,0)=temp3.z;
        eo*=(0.5);

        // delta definition
        for (i=0;i<3;i++){
                delta(i,0)=ep(i,0);
        }
        for (i=3;i<6;i++){
                delta(i,0)=eo(i-3,0);
        }

/*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.*/
        return delta;
        }



/* This function gives the DH-Parameters matrix to be used in the program.
Modifying the parameters on this file is sufficient to change these dimention in all
related programs.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%Modified By:Ana Catalina Torres%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "DH.h"

using namespace math;
using namespace std;

Matrix WMRA_DH(Matrix q){

        Matrix DH1(7,4);
```

Appendix A. (Continued)

```
//Inputting the D-H Parameters in a Matrix form, dimensions are in millimeters and
radians:

// Dimentions based on the actual physical arm:
        float DHtemp[7][4]={{-PI/2, 0,        102,  q(0,0)},
                                        { PI/2, 0,      133,  q(1,0)},
                                        {-PI/2, 0,  502,  q(2,0)},
                                        { PI/2, 0,  90,  q(3,0)},
                                        {-PI/2, 0,  375,  q(4,0)},
                                        { PI/2, 0,   0,   q(5,0)},
                                        {-PI/2, 0, 161+143, q(6,0)}};

// Dimentions based on the Virtual Reality arm model:
/*  float DH[7][4]={{-PI/2, 0, 109.72, q(0,0)},
                                        { PI/2, 0, 118.66, q(1,0)},
                                        {-PI/2, 0, 499.67, q(2,0)},
                                        { PI/2, 0, 121.78, q(3,0)},
                                        {-PI/2, 0, 235.67, q(4,0)},
                                        { PI/2, 0,   0,    q(5,0)},
                                        {-PI/2, 0, 276.68, q(6,0)}};
*/
        int i,j;
        for (j=0; j < 4; j++){
                for (i=0; i < 7; i++){
                        DH1(i,j) = DHtemp[i][j];
                }
        }
        return DH1;
}


/* 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 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%Modified By:Ana Catalina Torres%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "vector.h"
#include "J07.h"

using namespace std;
using namespace math;

void WMRA_J07(Matrix T1, Matrix T2, Matrix T3, Matrix T4, Matrix T5, Matrix T6, Matrix
T7, Matrix& J0, float& detJ0 ){
        Matrix T(4,4), J0temp(6,6), J0trans(7,6);
        T.Unit(4);
        J0.SetSize(6,7);
        J0(0,6) = -T(0,0)*T(1,3)+T(1,0)*T(0,3);
        J0(1,6) = -T(0,1)*T(1,3)+T(1,1)*T(0,3);
        J0(2,6) = -T(0,2)*T(1,3)+T(1,2)*T(0,3);
        J0(3,6) = T(2,0);
        J0(4,6) = T(2,1);
        J0(5,6) = T(2,2);

        T = T7 * T;
//      cout << "\nT is:\n" << T << endl;
```

Appendix A. (Continued)

```
        J0(0,5) = -T(0,0)*T(1,3)+T(1,0)*T(0,3);
        J0(1,5) = -T(0,1)*T(1,3)+T(1,1)*T(0,3);
        J0(2,5) = -T(0,2)*T(1,3)+T(1,2)*T(0,3);
        J0(3,5) = T(2,0);
        J0(4,5) = T(2,1);
        J0(5,5) = T(2,2);

        T = T6 * T;
//      cout << "\nT is:\n" << T << endl;

        J0(0,4) = -T(0,0)*T(1,3)+T(1,0)*T(0,3);
        J0(1,4) = -T(0,1)*T(1,3)+T(1,1)*T(0,3);
        J0(2,4) = -T(0,2)*T(1,3)+T(1,2)*T(0,3);
        J0(3,4) = T(2,0);
        J0(4,4) = T(2,1);
        J0(5,4) = T(2,2);

        T = T5 * T;
//      cout << "\nT is:\n" << T << endl;

        J0(0,3) = -T(0,0)*T(1,3)+T(1,0)*T(0,3);
        J0(1,3) = -T(0,1)*T(1,3)+T(1,1)*T(0,3);
        J0(2,3) = -T(0,2)*T(1,3)+T(1,2)*T(0,3);
        J0(3,3) = T(2,0);
        J0(4,3) = T(2,1);
        J0(5,3) = T(2,2);

        T = T4 * T;
//      cout << "\nT is:\n" << T << endl;

        J0(0,2) = -T(0,0)*T(1,3)+T(1,0)*T(0,3);
        J0(1,2) = -T(0,1)*T(1,3)+T(1,1)*T(0,3);
        J0(2,2) = -T(0,2)*T(1,3)+T(1,2)*T(0,3);
        J0(3,2) = T(2,0);
        J0(4,2) = T(2,1);
        J0(5,2) = T(2,2);

        T = T3 * T;
//      cout << "\nT is:\n" << T << endl;

        J0(0,1) = -T(0,0)*T(1,3)+T(1,0)*T(0,3);
        J0(1,1) = -T(0,1)*T(1,3)+T(1,1)*T(0,3);
        J0(2,1) = -T(0,2)*T(1,3)+T(1,2)*T(0,3);
        J0(3,1) = T(2,0);
        J0(4,1) = T(2,1);
        J0(5,1) = T(2,2);

        T = T2 * T;
//      cout << "\nT is:\n" << T << endl;
        J0(0,0) = -T(0,0)*T(1,3)+T(1,0)*T(0,3);
        J0(1,0) = -T(0,1)*T(1,3)+T(1,1)*T(0,3);
        J0(2,0) = -T(0,2)*T(1,3)+T(1,2)*T(0,3);
        J0(3,0) = T(2,0);
        J0(4,0) = T(2,1);
        J0(5,0) = T(2,2);

        T = T1 * T;
//      cout << "\nT is:\n" << T << endl;
        J0temp.Null(6,6);
        int i, j;
        for ( i=0 ; i < 3 ; i++ ) {
                for ( j = 0 ; j < 3 ; j++ ) {
                        J0temp(i,j)=T(i,j);
                }
```

Appendix A. (Continued)

```
}
        for ( i=3 ; i < 6 ; i++ ) {
                for ( j = 3 ; j < 6 ; j++ ) {
                        J0temp(i,j)=T(i-3,j-3);
                }
        }

        J0 = J0temp * J0;
        J0trans = ~J0;
        J0temp = J0 * J0trans;
        detJ0= sqrt(J0temp.Det());
}



/* 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 %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "WCD.h"
#include "matrix.h"
#include "Jga.h"
using namespace std;
using namespace math;

Matrix WMRA_Jga(int ind, float p, float X, float Y){

        Matrix Jga(6,2);
        Matrix L(1,1);
        // Reading the Wheelchair's constant dimentions, all dimentions are converted in
millimeters:
        L=WMRA_WCD();
        int i;
        // Deciding if the motion is in reference to the arm base (1) or the wheel axle
center (0):
        if (ind==0){
                for (i=1; i<4;i++){
                        L(0,i)=0;
                }
        }

        // Calculating the Jacobian:
        Matrix Jtemp1(6,6),Jtemp2(6,2),Jtemp3(2,2);
        Jtemp1.Unit(6);
        Jtemp1(0,5)=-(X*sin(p)+Y*cos(p));
        Jtemp1(1,5)=X*cos(p)-Y*sin(p);
        Jtemp2.Null(6,2);
        Jtemp2(0,0)=cos(p)+2*(L(0,1)*sin(p)+L(0,2)*cos(p))/L(0,0);
        Jtemp2(0,1)=cos(p)-2*(L(0,1)*sin(p)+L(0,2)*cos(p))/L(0,0);
        Jtemp2(1,0)=sin(p)-2*(L(0,1)*cos(p)-L(0,2)*sin(p))/L(0,0);
        Jtemp2(1,1)=sin(p)+2*(L(0,1)*cos(p)-L(0,2)*sin(p))/L(0,0);
        Jtemp2(5,0)=-2/L(0,0);
        Jtemp2(5,1)=2/L(0,0);
        Jtemp2*=(L(0,4)/2);
```

Appendix A. (Continued)

```
        Jtemp3(0,0)=1;
        Jtemp3(0,1)=-L(0,0)/(2*L(0,4));
        Jtemp3(1,0)=1;
        Jtemp3(1,1)=L(0,0)/(2*L(0,4));

        Jga = Jtemp1 * Jtemp2 * Jtemp3;
        return Jga;
        }

/* 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 %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "Jlimit.h"
using namespace std;
using namespace math;

void WMRA_Jlimit(Matrix& qmin, Matrix& qmax){

        float qmintemp[7]= {-170*PI/180,-170*PI/180,-170*PI/180,-170*PI/180,-170*PI/180,-
79*PI/180,-200*PI/180};
        float qmaxtemp[7] =
{170*PI/180,170*PI/180,170*PI/180,170*PI/180,170*PI/180,79*PI/180,200*PI/180};
        int i;
        for (i=0; i < 7; i++){
                qmin(0,i) = qmintemp[i];
                qmax(0,i) = qmaxtemp[i];
        }
        }



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Function Declaration:*/

#include "matrix.h"
#include "Linear.h"
using namespace std;
using namespace math;

Matrix WMRA_Linear(float qi, float qf, float n){
        Matrix qt(2,1);
        int i;
        float dq;
        dq =(qf-qi)/(n-1);

        float *qttemp;
```

Appendix A. (Continued)

```
        qttemp = new float[n];
        for (i=1; i<n+1; i++){
                qttemp[i-1]=qi+dq*(i-1);
        }

        qt.SetSize(n,1);
        for (i=0; i < n; i++){
                qt(i,0) = qttemp[i];
        }
        delete [] qttemp;

        return qt;
}


/* 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 %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres & Punya A.Basnayaka%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

#include "matrix.h"
#include "var_included.h"
#include "any2ready.h"
#include "ArmMotion.h"
#include "BPolynomial.h"
#include "Collide.h"
#include "Delta.h"
#include "DH.h"
#include "J07.h"
#include "Jga.h"
#include "Jlimit.h"
#include "Linear.h"
#include "Opt.h"
#include "p2T.h"
#include "plots2.h"
#include "park2ready.h"
#include "Polynomial.h"
#include "q2T.h"
#include "ready2any.h"
#include "ready2park.h"
#include "Rotx.h"
#include "Roty.h"
#include "Rotz.h"
#include "sign.h"
#include "T2rpy.h"
#include "Tall.h"
#include "Traj.h"
#include "Transl.h"
#include "w2T.h"
#include "WCD.h"
#include <math.h>
#include <time.h>
#include <iostream>
#include <fstream>
#include <tchar.h>
#include <string.h>

using namespace std;
using namespace math;
using namespace System;
using namespace System::Windows::Forms;
```

Appendix A. (Continued)

```
using namespace System::Drawing;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Data;
#define PI 3.14159265
#define d2r PI/180 //Conversions from Degrees to Radians.
#define r2d 180/PI //Conversions from Radians to Degrees.
// Creating an intnce of object to be used in all the programs
static Galil g("COM4 19200");
int main(){
	Matrix L(1,1);
	// Reading the Wheelchair's constant dimentions, all dimentions are converted in
millimeters:
	L=WMRA_WCD();
	exitvar=0;
	// User input prompts:
	int choice000, choice00000, choice0000, choice5, choice50, choice500, choice0,
choice00, choice1, choice10, choice2, choice3, choice4, choice6, choice7, choice8;
	int WCA, coord, cart, optim, JLA, JLO, cont, trajf, vr, ml, arm, ini, plt;
	int j, k;
	int i;
	int port1, ts;
	float v;
	Matrix Td(4,4), Vd(3,1), qi(7,1), WCi(3,1);

topchoice:;
	cout << "\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\n ";
	cin >> choice000;
	if (choice000==2){
		WCA=2;
top1:;
		cout << "\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\n ";
		cin >> choice00000;
		if (choice00000==2){
			coord=2;
		}
		else if (choice00000==3){
			coord=3;
		}
		else if (choice00000==1){
			coord=1;
		}
		else {
			cout << "\n Invalid choice. Try again. \n\n ";
			goto top1;
		}
top2:;
		cout << "\n Choose the cartesian coordinates to be controlled: \n   For
Position and Orientation, press '1', \n   For Position only, press '2'. \n\n ";
		cin >> choice0000;
		if (choice0000==2){
			cart=2;
		}
		else if (choice0000==1) {
			cart=1;
		}
		else {
			cout << "\n Invalid choice. Try again. \n\n ";
			goto top2;
		}
top3:;
```

Appendix A. (Continued)

```
cout << "\n Please enter the desired optimization method: (1= SR-I & WLN, 2= P-I & WLN,
3= SR-I & ENE, 4= P-I & ENE) \n\n ";
                cin >> choice5;
                if (choice5==2){
                        optim=2;
                }
                else if (choice5==3){
                        optim=3;
                }
                else if (choice5==4){
                        optim=4;
                }
                else if (choice5==1){
                        optim=1;
                }
                else {
                        cout << "\n Invalid choice. Try again. \n\n ";
                        goto top3;
                }
top4:;
                cout << "\n Do you want to include Joint Limit Avoidance? (1= Yes, 2= No)
\n\n ";
                cin >> choice50;
                if (choice50==2){
                        JLA=0;
                }
                else if (choice50==1){
                        JLA=1;
                }
                else {
                        cout << "\n Invalid choice. Try again. \n\n ";
                        goto top4;
                }
top5:;
                cout << "\n Do you want to include Joint Limit/Velocity and Obstacle
Safety Stop? (1= Yes, 2= No) \n\n ";
                cin >> choice500;
                if (choice500==2){
                        JLO=0;
                }
                else if (choice500==1){
                        JLO=1;
                }
                else {
                        cout << "\n Invalid choice. Try again. \n\n ";
                        goto top5;
                }
        }
        else if (choice000==3){
                WCA=3;
top6:;
                cout << "\n Choose whose frame to base the control on: \n   For Ground
Frame, press '1', \n   For Wheelchair Frame, press '2'. \n\n ";
                cin >> choice00000;
                if (choice00000==2){
                        coord=2;
                }
                else if (choice00000==1){
                        coord=1;
                }
                else {
                        cout << "\n Invalid choice. Try again. \n\n ";
                        goto top6;
                }
top7:;
```

Appendix A. (Continued)

```
cout << "\n Do you want to include Joint Limit/Velocity and Obstacle Safety Stop? (1=
Yes, 2= No) \n\n ";

        cin >> choice500;
                if (choice500==2){
                        JLO=0;
                }
                else if (choice500==1){
                        JLO=1;
                }
                else {
                        cout << "\n Invalid choice. Try again. \n\n ";
                        goto top7;
                }

                cart=2;
                optim=0;
                JLA=0;
                }
        else if (choice000==1) {
                WCA=1;
top8:;
                cout << "\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\n ";
                cin >> choice00000;
                if (choice00000==2){
                        coord=2;
                }
                else if (choice00000==3){
                        coord=3;
                }
                else if (choice00000==1){
                        coord=1;
                }
                else {
                        cout << "\n Invalid choice. Try again. \n\n ";
                        goto top8;
                }
top9:;
                cout << "\n Choose the cartesian coordinates to be controlled: \n   For
Position and Orientation, press '1', \n   For Position only, press '2'. \n\n ";
                cin >> choice0000;
                if (choice0000==2){
                        cart=2;
                }
                else if (choice0000==1){
                        cart=1;
                }
                else {
                        cout << "\n Invalid choice. Try again. \n\n ";
                        goto top9;
                }
top10:;
                cout << "\n Please enter the desired optimization method: (1= SR-I & WLN,
2= P-I & WLN, 3= SR-I & ENE, 4= P-I & ENE) \n\n ";
                cin >> choice5;
                if (choice5==2){
                        optim=2;
                }
                else if (choice5==3){
                        optim=3;
                }
                else if (choice5==4){
                        optim=4;
```

Appendix A. (Continued)

```
                }
                else if (choice5==1){
                        optim=1;
                }
                else {

cout << "\n Invalid choice. Try again. \n\n ";
                        goto top10;
                }
top11:;
                cout << "\n Do you want to include Joint Limit Avoidance? (1= Yes, 2= No)
\n\n ";
                cin >> choice50;
                if (choice50==2){
                        JLA=0;
                }
                else if (choice50==1){
                        JLA=1;
                }
                else {
                        cout << "\n Invalid choice. Try again. \n\n ";
                        goto top11;
                }
top12:;
                cout << "\n Do you want to include Joint Limit/Velocity and Obstacle
Safety Stop? (1= Yes, 2= No) \n\n ";
                cin >> choice500;
                if (choice500==2){
                        JLO=0;
                }
                else if (choice500==1){
                        JLO=1;
                }
                else {
                        cout << "\n Invalid choice. Try again. \n\n ";
                        goto top12;
                }
        }
        else {
                cout << "\n Invalid choice. Try again. \n\n ";
                goto topchoice;
        }
topchoice2:;
        cout << "\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\n ";
        cin >> choice0;
        if (choice0==1){
                cont=1;
                printf ("\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\n");
                cin >> Td;
                cout << "\n Please enter the desired linear velocity of the gripper in
mm/s (e.g. 50) \n\n ";
                cin >> v;
top13:;
                cout << "\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\n ";
                cin >> choice00;
                if (choice00==2) {
                        trajf=2;
                }
                else if (choice00==3){
```

Appendix A. (Continued)

```
trajf=3;
                }
                else if (choice00==1){
                        trajf=1;
                }
                else {
                cout << "\n Invalid choice. Try again. \n\n ";

goto top13;
                }
        }
        else if (choice0==2){
                cont=2;
                cout << "\n Please enter the desired simulation time in seconds (e.g. 2)
\n\n ";
                cin >> ts;
                if (cart==2){
                        cout << "\n Please enter the desired 3x1 cartesian velocity vector
of the gripper (in mm/s) (e.g. [70;70;-70]) \n\n ";
                        cin >> Vd;
                }
                else {
                        cout << "\n Please enter the desired 6x1 cartesian velocity vector
of the gripper (in mm/s, radians/s) (e.g. [70;70;-70;0.001;0.001;0.001]) \n\n ";
                        Vd.SetSize(6,1);
                        cin >> Vd;
                }
        }
        else if (choice0==3){
                cont=3;
                //Space Ball will be used for control.
                cout << "\n Please enter the desired linear velocity of the gripper in
mm/s (e.g. 50) \n\n ";
                cin >> v;
        }
        else if (choice0==4){
                cont=4;
                //BCI 2000 Psychology Mask will be used for control.
                cout << "\n Please enter the desired linear velocity of the gripper in
mm/s (e.g. 50) \n\n ";
                cin >> v;
                cout << "\n Please enter the desired port number (e.g. 19711) \n\n ";
                cin >> port1;
        }
        else if (choice0==5){
                cont=5;
                //Touch Screen will be used for control.
                cout << "\n Please enter the desired linear velocity of the gripper in
mm/s (e.g. 50) \n\n ";
                cin >> v;
        }
        else {
                cout << "\n Invalid choice. Try again. \n\n ";
                goto topchoice2;
        }

topchoice3:;
        cout << "\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\n ";
        cin >> choice1;
        if (choice1==2) {
                vr=0; ml=1;
        }
        else if (choice1==3){
```

Appendix A. (Continued)

```
vr=1; ml=1;
        }
        else if (choice1==4){
                vr=0; ml=0;
        }
        else if (choice1==1){
                vr=1; ml=0;
        }
        else {
                cout << "\n Invalid choice. Try again. \n\n ";
goto topchoice3;
        }

topchoice4:;
        cout << "\n Would you like to run the actual WMRA? \n For yes, press '1', \n For
no, press '2'.\n\n ";
        cin >> choice10;
        if (choice10==1) {
                arm=1;
        }
        else if (choice10==2){
                arm=0;
        }
        else {
                cout << "\n Invalid choice. Try again. \n\n ";
                goto topchoice4;
        }

topchoice5:;
        cout <<"\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\n ";
        cin >> choice2;
        if (choice2==2) {
                cout <<"\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\n ";
                cin >> qi;
                cout <<"\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\n ";
                cin >> WCi;
                ini = 0;
        }
        else if (choice2==1){
                float qi2 [7]= {90, 90, 0, 90, 90, 90, 0};
                for ( j = 0 ; j < 7 ; j++ ) {
                        qi(j,0)=qi2[j]*d2r;
                }
                WCi.Null(3,1);
                if (vr==1 || ml==1 || arm==1) {
top14:;
                        cout <<"\n Press '1' if you want to include 'park' to 'ready'
motion, \n or press '2' if not.\n\n ";
                        cin >> choice3;
                        if (choice3==2){
                                ini = 0;
                        }
                        else if (choice3==1){
                                ini = 1;
                        }
                        else {
                                cout << "\n Invalid choice. Try again. \n\n ";
                                goto top14;
                        }
                }
        }
        else {
```

Appendix A. (Continued)

```
cout << "\n Invalid choice. Try again. \n\n ";
                goto topchoice5;
        }

topchoice6:;
        cout <<"\n Press '1' if you do NOT want to plot the simulation results, \n or
press '2' if do.\n\n ";
        cin >> choice4;
        if (choice4==2) {
                plt=2;
        }
        else if (choice4==1) {

plt=1;
        }
        else {
                cout << "\n Invalid choice. Try again. \n\n ";
                goto topchoice6;
        }

        // Calculating the Transformation Matrix of the initial position of the WMRA's
base:
        Matrix Tiwc(4,4), qiwc(2,1);
        Tiwc = WMRA_p2T(WCi(0,0),WCi(1,0),WCi(2,0));
        //cout<<"Tiwc\n"<<Tiwc<<"\n\n";
        //Calculating the initial Wheelchair Variables:
        qiwc(0,0)=sqrt(pow(WCi(0,0),2)+pow(WCi(1,0),2));
        qiwc(1,0)=WCi(2,0);
        //cout<<"qiwc\n"<<qiwc<<"\n\n";

        //Calculating the initial transformation matrices:
        Matrix dq(1,1), Ti(4,4), Tia(4,4), Tiwco(4,4), T01(4,4), T12(4,4), T23(4,4),
T34(4,4), T45(4,4), T56(4,4), T67(4,4), Unit(1,1);
        Unit.Null(2,1);
        WMRA_Tall(1, qi, Unit, Tiwc, Ti, Tia, Tiwco, T01, T12, T23, T34, T45, T56, T67);

        FILE * fid;
        fid = fopen("main.txt","w");
        fprintf(fid," Tiwc is: \n\n");
        for (j=0;j<Tiwc.RowNo();j++){
                for (k=0;k<Tiwc.ColNo();k++){
                        fprintf(fid," %4.2f ",  Tiwc(j,k));
                }
                fprintf(fid," \n");
        }
        fprintf(fid," \n\n\n");

        Tiwc=Tiwco;

        fprintf(fid," qiwc is: \n\n");
        for (j=0;j<qiwc.RowNo();j++){
                fprintf(fid," %4.2f ",  qiwc(j,0));
        }
        fprintf(fid," \n\n\n");
        fprintf(fid," Ti is: \n\n");
        for (j=0;j<Ti.RowNo();j++){
                for (k=0;k<Ti.ColNo();k++){
                        fprintf(fid," %4.2f ",  Ti(j,k));
                }
                fprintf(fid," \n");
        }
        fprintf(fid," \n\n\n");
        fprintf(fid," Tia is: \n\n");
        for (j=0;j<Tia.RowNo();j++){
                for (k=0;k<Tia.ColNo();k++){
```

Appendix A. (Continued)

```
fprintf(fid," %4.2f ",  Tia(j,k));
                }
                fprintf(fid," \n");
        }
        fprintf(fid," \n\n\n");
        fprintf(fid," Tiwc is: \n\n");
        for (j=0;j<Tiwc.RowNo();j++){
                for (k=0;k<Tiwc.ColNo();k++){
                        fprintf(fid," %4.2f ",  Tiwc(j,k));
                }
                fprintf(fid," \n");
        }
        fprintf(fid," \n\n\n");
        fprintf(fid," T01 is: \n\n");
        for (j=0;j<T01.RowNo();j++){
                for (k=0;k<T01.ColNo();k++){
                        fprintf(fid," %4.2f ",  T01(j,k));
                }
                fprintf(fid," \n");
        }
        fprintf(fid," \n\n\n");
        fprintf(fid," T12 is: \n\n");
        for (j=0;j<T12.RowNo();j++){
                for (k=0;k<T12.ColNo();k++){
                        fprintf(fid," %4.2f ",  T12(j,k));
                }
                fprintf(fid," \n");
        }
        fprintf(fid," \n\n\n");
        fprintf(fid," T23 is: \n\n");
        for (j=0;j<T23.RowNo();j++){
                for (k=0;k<T23.ColNo();k++){
                        fprintf(fid," %4.2f ",  T23(j,k));
                }
                fprintf(fid," \n");
        }
        fprintf(fid," \n\n\n");
        fprintf(fid," T34 is: \n\n");
        for (j=0;j<T34.RowNo();j++){
                for (k=0;k<T34.ColNo();k++){
                        fprintf(fid," %4.2f ",  T34(j,k));
                }
                fprintf(fid," \n");
        }
        fprintf(fid," \n\n\n");
        fprintf(fid," T45 is: \n\n");
        for (j=0;j<T45.RowNo();j++){
                for (k=0;k<T45.ColNo();k++){
                        fprintf(fid," %4.2f ",  T45(j,k));
                }
                fprintf(fid," \n");
        }
        fprintf(fid," \n\n\n");
        fprintf(fid," T56 is: \n\n");
        for (j=0;j<T56.RowNo();j++){
                for (k=0;k<T56.ColNo();k++){
                        fprintf(fid," %4.2f ",  T56(j,k));
                }
                fprintf(fid," \n");
        }
        fprintf(fid," \n\n\n");
        fprintf(fid," T67 is: \n\n");
        for (j=0;j<T67.RowNo();j++){
                for (k=0;k<T67.ColNo();k++){
                        fprintf(fid," %4.2f ",  T67(j,k));
```

Appendix A. (Continued)

```
        }
                fprintf(fid," \n");
        }
        fprintf(fid," \n\n\n");

        float D, dt, total_time, n, dg;
        Matrix dx(1,1);
        float ***Tt;
        if (cont==1){
                //Calculating the linear distance from the initial position to the desired
position and the linear velocity:
                if (coord==2){
                        D=sqrt(pow(Td(0,3)-Tia(0,3),2) + pow(Td(1,3)-Tia(1,3),2) +
pow(Td(2,3)-Tia(2,3),2));
                        fprintf(fid," D is: %g \n\n", D);
                        fprintf(fid," \n\n\n");
                }

else if (coord==3){
                        D=sqrt(pow(Td(0,3),2) + pow(Td(1,3),2) + pow(Td(2,3),2));
                        fprintf(fid," D is: %g \n\n", D);
                        fprintf(fid," \n\n\n");
                }
                else {
                        D=sqrt(pow(Td(0,3)-Ti(0,3),2) + pow(Td(1,3)-Ti(1,3),2) +
pow(Td(2,3)-Ti(2,3),2));
                        fprintf(fid," D is: %g \n\n", D);
                        fprintf(fid," \n\n\n");
                }
                //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=Math::Round(total_time/dt); // Number of iterations rounded up.
                dt=total_time/n;    // Adjusted time increment in seconds.
                fprintf(fid," Total time is: %4.2f \n\n", total_time);
                fprintf(fid," n is: %g \n\n", n);
                fprintf(fid," dt is: %g \n\n", dt);
                fprintf(fid," \n\n\n");

                // Calculating the Trajectory of the end effector, and once the trajectory
is calculated, we should redefine "Td" based on the ground frame:
                FILE * fid2;
                fid2 = fopen("traj.txt","w");
                if (coord==2){
                        Tt=WMRA_traj(trajf,Tia,Td,n+1);
                        Td=Tiwc*Td;
                }
                else if (coord==3){
                        Unit.Unit(4);
                        Tt=WMRA_traj(trajf,Unit,Td,n+1);
                        Td=Ti*Td;
                }
                else {
                        Tt=WMRA_traj(trajf,Ti,Td,n+1);
                }
                fprintf(fid2," Trajectory is \n\n");
                        for (i=0;i<n+1;i++){
                                for (j=0;j<4;j++){
                                        for (k=0;k<4;k++){
                                                fprintf(fid2," %4.2f ",  Tt[i][j][k]);
                                        }
                                        fprintf(fid2," \n");
                                }
                                fprintf(fid2," \n\n\n");
```

Appendix A. (Continued)

```
}
                        fprintf(fid2," \n\n\n");
                        fclose (fid2);
        }
        else if (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=Math::Round(total_time/dt); // Number of iterations rounded up.
                dt=total_time/n;    // Adjusted time increment in seconds.
                dx=Vd;
                dx*=(dt);
                Td=Ti;
                fprintf(fid," Total time is: %4.2f \n\n", total_time);
                fprintf(fid," n is: %d \n\n", n);
                fprintf(fid," dt is: %g \n\n", dt);
                fprintf(fid," \n\n\n");
                fprintf(fid," dx is: \n\n");
                for (j=0;j<dx.RowNo();j++){
        fprintf(fid," %4.2f ",  dx(j,0));
                }
        }
        else if (cont==3){
                //WMRA_exit(); // This is to stop the simulation in SpaceBall control when
the user presses the exit key.

                FILE * fus;
                fus = fopen("output.txt","w");
                fprintf(fus,"0\t0\t0\t0\t0\t0\t\n");
                fclose(fus);
                n=1;
        }
        else if (cont==4){

myForm2.ShowDialog() == System::Windows::Forms::DialogResult::OK;
        }
        else {
                //WMRA_screen('0');   // This is to start the screen controls. Argument:
'0'=BACK button disabled, '1'=BACK button enabled.
                dt=0.05;

                n=1;
        }

        // Initializing the joint angles, the Transformation Matrix, and time:
        Matrix qo(9,1), To(4,4), Towc(4,4), Toa(4,4), Jo(1,1);
        float tt, ddt;
        dq.Null(9,1);
        dg=0;
        for (j=0; j<7; j++){
                qo(j,0)=qi(j,0);
        }
        for (j=7; j<9; j++){
                qo(j,0)=qiwc(j-7,0);
        }

        To=Ti;
        Toa=Tia;
        Towc=Tiwc;
        tt=0;
        i=0;
        dHo.Null(7,1);
```

Appendix A. (Continued)

```
fprintf(fid," dq is: \n\n");
        for (j=0;j<dq.RowNo();j++){
                fprintf(fid," %4.2f ",  dq(j,0));
        }
        fprintf(fid," \n\n\n");
        fprintf(fid," qo is: \n\n");
        for (j=0;j<qo.RowNo();j++){
                fprintf(fid," %4.2f ",  qo(j,0));
        }
        fprintf(fid," \n\n\n");
        fprintf(fid," To is: \n\n");
        for (j=0;j<To.RowNo();j++){
                for (k=0;k<To.ColNo();k++){
                        fprintf(fid," %4.2f ",  To(j,k));
                }
                fprintf(fid," \n");
        }
        fprintf(fid," \n\n\n");
        fprintf(fid," Toa is: \n\n");
        for (j=0;j<Toa.RowNo();j++){
                for (k=0;k<Toa.ColNo();k++){
                        fprintf(fid," %4.2f ",  Toa(j,k));
                }
                fprintf(fid," \n");
        }
        fprintf(fid," \n\n\n");
        fprintf(fid," Towc is: \n\n");
        for (j=0;j<Towc.RowNo();j++){
                for (k=0;k<Towc.ColNo();k++){
                        fprintf(fid," %4.2f ",  Towc(j,k));
                }
                fprintf(fid," \n");
        }
        fprintf(fid," \n\n\n");

        // Initializing the WMRA:
        if (ini==0){    // When no "park" to "ready" motion required.
                                if (vr==1){
                        //WMRA_VR_Animation(1, Towc, qo);
                }
                // Initializing Robot Animation in Matlab Graphics:
                if (ml==1){
                        //WMRA_ML_Animation(1, To, Td, Towc, T01, T12, T23, T34, T45, T56,
T67);
                }
                                if (arm==1){
                        Matrix qotemp(1,1);
                        qotemp.Null(10,1);
                        for (j=0; j<9; j++){
                                qotemp(j,0)=qo(j,0);
                        }
                        qotemp(9,0)=1;
                        WMRA_ARM_Motion(1, 2, qotemp, 0, g);
                        ddt=0;
                }
        }
        else if (ini==1 && (vr==1 || ml==1 || arm==1)){       // When "park" to "ready"
motion is required.
                Matrix qotemp(2,1);
                qotemp(0,0)=qo(7,0);
                qotemp(1,0)=qo(8,0);
                WMRA_park2ready(1, vr, ml, arm, Towc, qotemp, g);
                if (arm==1){
                        ddt=0;
                }
```

Appendix A. (Continued)

```
}
        wait(1);

                // Re-Drawing the Animation:
                Matrix Joa(6,7), Jowc(1,1), Jowctemp(1,1), Joatemp(1,1);
                float detJoa, detJo, phi;
                if (vr==1 || ml==1){
                        //drawnow;
                }
                // Starting a timer:
                clock_t startt, start2, endt, endf, endtt;
                //time_t startt, endt, endf;
                //time(&startt);
                start2=clock()/ CLOCKS_PER_SEC;
                startt=clock()*CLK_TCK/1000;
                cout<<"\nStart is "<<startt<<"\n\n";
                double timedif;
                /*Stopwatch ^ timePerParse;
                timePerParse = Stopwatch::StartNew();
        //      Stopwatch::StartNew();*/

        /*      float ***Ttnew;
                Ttnew = new float**[n];
                for (j = 0; j < n; ++j) {
                        Ttnew[j] = new float*[4];
                        for (k = 0; k < 4; ++k){
                                Ttnew[j][k] = new float[4];
                        }
                }*/

                Matrix qn(1,1), Tn(4,4), Tna(4,4), Tnwc(4,4);
                Matrix q1(1,1), T1(4,4), Ta1(4,4), Twc1(4,4);
                Matrix dq1(1,1);

                // Starting the Iteration Loop:
                //for (i=0;i<n;i++){
                int flagspace=0;
                Matrix space(6,1);
                while (i<=n){
                        if (i==0){
                                q1=qo;
                                T1=To;
                                Ta1=Toa;
                                Twc1=Towc;
                        }
                        else{
                                q1=qn;
                                T1=Tn;
                                Ta1=Tna;
                                Twc1=Tnwc;
                                dq=dq1;
                        }

                        // Calculating the 6X7 Jacobian of the arm in frame 0:
                        WMRA_J07(T01, T12, T23, T34, T45, T56, T67, Joa, detJoa);
                        fprintf(fid," i is: %d\n\n",i);
                        fprintf(fid," Joa is: \n\n");
                        for (j=0;j<Joa.RowNo();j++){
                                for (k=0;k<Joa.ColNo();k++){
                                        fprintf(fid," %4.2f ",  Joa(j,k));
                                }
                                fprintf(fid," \n");
                        }
                        fprintf(fid," \n\n\n");
                        fprintf(fid," detJoa is: %g \n\n", detJoa);
```

Appendix A. (Continued)

```
// Calculating the 6X2 Jacobian based on the WMRA's base in the ground frame:
                        phi=atan2(Twc1(1,0),Twc1(0,0));
                        Jowc=WMRA_Jga(1, phi, Ta1(0,3), Ta1(1,3));
                        fprintf(fid," Jowc is: \n\n");
                        for (j=0;j<Jowc.RowNo();j++){
                                for (k=0;k<Jowc.ColNo();k++){
                                        fprintf(fid," %4.2f ",  Jowc(j,k));
                                }
                                fprintf(fid," \n");
                        }
                        fprintf(fid," \n\n\n");
                        fprintf(fid," phi is: %g \n\n", phi);
                        // 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;
                                Jowctemp.Null(6,6);
                                for (j=0; j<3; j++){
                                        for (k=0; k<3; k++){
                                                Jowctemp(j,k)=Twc1(k,j);
                                        }
                                }
                                for (j=3; j<6; j++){
                                        for (k=3; k<6; k++){
                                                Jowctemp(j,k)=Twc1(k-3,j-3);
                                        }
                                }
                                Jowctemp=Jowctemp*Jowc;
                                Jowc=Jowctemp;
                        }
                        else if (coord==3){
                                Joatemp.Null(6,6);
                                for (j=0; j<3; j++){
                                        for (k=0; k<3; k++){
                                                Joatemp(j,k)=Ta1(k,j);
                                        }
                                }
                                for (j=3; j<6; j++){
                                        for (k=3; k<6; k++){
                                                Joatemp(j,k)=Ta1(k-3,j-3);
                                        }
                                }
                                Joatemp=Joatemp*Joa;
                                Joa=Joatemp;
                                Jowctemp.Null(6,6);
                                for (j=0; j<3; j++){
                                        for (k=0; k<3; k++){
                                                Jowctemp(j,k)=T1(k,j);
                                        }
                                }
                                for (j=3; j<6; j++){
                                        for (k=3; k<6; k++){
                                                Jowctemp(j,k)=T1(k-3,j-3);
                                        }
                                }
                                Jowctemp=Jowctemp*Jowc;
                                Jowc=Jowctemp;
                        }
                        else if (coord==1){
                                Joatemp.Null(6,6);
```

Appendix A. (Continued)

```
                        for (j=0; j<3; j++){
                        for (k=0; k<3; k++){
                                Joatemp(j,k)=Twc1(j,k);

                }
                        }
                        for (j=3; j<6; j++){
                                for (k=3; k<6; k++){
                                        Joatemp(j,k)=Twc1(j-3,k-3);
                                }
                        }
                        Joatemp=Joatemp*Joa;
                        Joa=Joatemp;
                        Jowc=Jowc;
                }
                        fprintf(fid," Joa is: \n\n");
                        for (j=0;j<Joa.RowNo();j++){
                                for (k=0;k<Joa.ColNo();k++){
                                        fprintf(fid," %4.2f ",  Joa(j,k));
                                }
                                fprintf(fid," \n");
                        }
                        fprintf(fid," \n\n\n");
                        fprintf(fid," Jowc is: \n\n");
                        for (j=0;j<Jowc.RowNo();j++){
                                for (k=0;k<Jowc.ColNo();k++){
                                        fprintf(fid," %4.2f ",  Jowc(j,k));
                                }
                                fprintf(fid," \n");
                        }
                        fprintf(fid," \n\n\n");


                //Calculating the 3X9 or 6X9 augmented Jacobian of the WMRA system
based on the ground frame:
                if (cart==2){
                        Joa.SetSize(3,7);
                        Joatemp.Null(7,3);
                        Joatemp=~Joa;
                        Joatemp=Joa*Joatemp;
                        detJoa=sqrt(Joatemp.Det());
                        Jowc.SetSize(3,2);
                        Jo.SetSize(3,9);
                        for (j=0; j<3; j++){
                                for (k=0; k<7; k++){
                                        Jo(j,k)=Joa(j,k);
                                }
                                for (k=7; k<9; k++){
                                        Jo(j,k)=Jowc(j,k-7);
                                }
                        }
                        Joatemp.Null(9,3);
                        Joatemp=~Jo;
                        Joatemp=Jo*Joatemp;
                        detJo=sqrt(Joatemp.Det());
                }
                else {
                        Jo.SetSize(6,9);
                        for (j=0; j<6; j++){
                                for (k=0; k<7; k++){
                                        Jo(j,k)=Joa(j,k);
                                }
                                for (k=7; k<9; k++){
                                        Jo(j,k)=Jowc(j,k-7);
                                }
```

Appendix A. (Continued)

```
}
                                Joatemp=~Jo;
                                Joatemp=Jo*Joatemp;
                                detJo=sqrt(Joatemp.Det());
                        }
                        fprintf(fid," Jo is: \n\n");

for (j=0;j<Jo.RowNo();j++){
                                for (k=0;k<Jo.ColNo();k++){
                                        fprintf(fid," %4.2f ",  Jo(j,k));
                                }
                                fprintf(fid," \n");
                        }
                        fprintf(fid," \n\n\n");
                        fprintf(fid," detJo is: %g \n\n", detJo);


                        // Finding the Cartesian errors of the end effector:
                        Matrix invTowc(1,1), invTo(1,1), Towctemp(1,1), Towctemp2(1,1),
Tttemp(4,4), Ttnew(4,4);
                        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.Unit(4);
                                        Towctemp=Twc1;
                                        Towctemp.SetSize(3,3);
                                        Towctemp2=~Towctemp;
                                        Towctemp2*=(-1);
                                        Towctemp.Null(3,1);
                                        for (j=0; j<3; j++){
                                                Towctemp(j,0)=Twc1(j,3);
                                        }
                                        Towctemp=Towctemp2*Towctemp;
                                        for (j=0; j<3; j++){
                                                for (k=0; k<3; k++){
                                                        invTowc(j,k)=Twc1(k,j);
                                                }
                                                invTowc(j,3)=Towctemp(j,0);
                                        }

                                        Tttemp.Null(4,4);
                                        for (j=0; j<4; j++){
                                                for (k=0; k<4; k++){
                                                        Tttemp(j,k)=Tt[i][j][k];

                                                }
                                        }
                                        Ttnew=invTowc*Tiwc*Tttemp;
                                        dx=WMRA_delta(Ta1, Ttnew);
                                }
                                else if (coord==3){
                                        invTo.Unit(4);
                                        Towctemp=T1;
                                        Towctemp.SetSize(3,3);
                                        Towctemp2=~Towctemp;
                                        Towctemp2*=(-1);
                                        Towctemp.Null(3,1);
                                        for (j=0; j<3; j++){
                                                Towctemp(j,0)=T1(j,3);
                                        }
                                        Towctemp=Towctemp2*Towctemp;
                                        for (j=0; j<3; j++){
                                                for (k=0; k<3; k++){
                                                        invTo(j,k)=T1(k,j);
```

Appendix A. (Continued)

```
                                        }
                                        invTo(j,3)=Towctemp(j,0);
                                }

                                Tttemp.Null(4,4);
                                for (j=0; j<4; j++){
                                        for (k=0; k<4; k++){
                                                Tttemp(j,k)=Tt[i][j][k];


}
                                }
                                Ttnew=invTo*Ti*Tttemp;
                                Unit.Unit(4);
                                dx=WMRA_delta(Unit, Ttnew);
                        }
                        else {
                                for (j=0; j<4; j++){
                                        for (k=0; k<4; k++){
                                                Tttemp(j,k)=Tt[i][j][k];
                                        }
                                }
                                dx=WMRA_delta(T1, Tttemp);
                        }
                }
                else if (cont==2) {
                }
                else if (cont==3) {
                        char tx1[8], ty1[8], tz1[8], rx1[8], ry1[8], rz1[8],
varscreenop[8], varexit1[9], varrun1[8];
                        double tx, ty, tz, rx, ry, rz;
                        double varexit, varscree;

                        fstream results1 ("outputexit.txt", ios::in);
                        if (!results1){
                                cout<<"Can not open";
                        }
                        if (!results1.eof())
                        {
                                results1.getline(varscreenop, 8, '\t');
                                results1.getline(varexit1, 8, '\t');
                                results1.close();
                        }
                        varscreenopn = atoi (varscreenop);
                        varexit = atof (varexit1);
                        varscree = atof (varscreenop);

                        if (varexit==0){
                                MessageBox::Show("The system has been stopped","WMRA
STOP", MessageBoxButtons::OK,MessageBoxIcon::Exclamation);
                                varexit=1;
                                FILE * ex;
                                ex = fopen("outputexit.txt","w");
                                fprintf(ex,"%1.f\t%1.f\t\n",varscree, varexit);
                                fclose(ex);
                        }

                        fstream results ("output.txt", ios::in);
                        if (!results){
                                cout<<"Can not open";
                        }
                        if (!results.eof())
                        {
                                results.getline(ty1, 8, '\t');
                                results.getline(tz1, 8, '\t');
```

Appendix A. (Continued)

```
                                        results.getline(tx1, 8, '\t');
                                        results.getline(ry1, 8, '\t');
                                        results.getline(rz1, 8, '\t');
                                        results.getline(rx1, 8, '\t');

                                        results.close();
                                }

                                tx = atoi (tx1);
                                ty = atoi (ty1);
                                tz = atoi (tz1);
                                rx = atoi (rx1);
                                ry = atoi (ry1);
                                rz = atoi (rz1);
                                Matrix spdata(6,1);
                                cout<<"tx = "<<tx<<endl;
                                cout<<"ty = "<<ty<<endl;
                                cout<<"tz = "<<tz<<endl;
                                cout<<"rx = "<<rx<<endl;
                                cout<<"ry = "<<ry<<endl;
                                cout<<"rz = "<<rz<<endl;
                                spdata(0,0)=tx/20;
                                spdata(1,0)=-ty/40;
                                spdata(2,0)=tz/30;
                                spdata(3,0)=rx/1500;
                                spdata(4,0)=-ry/900;
                                spdata(5,0)=rz/1300;
                                dt=0.05;
                                dx=spdata;
                                dx*=(v*dt);
                                dg=0;//spdata(6,0);

                        }
                        else if (cont==4) {
                                //dx=WMRA_psy(port1);
                                //dx*=(v*dt);
                                //dg=dx(6,0);
                                //dx.SetSize(6,1);
                        }
                        else {
                                char dx1[8], dx2[8], dx3[8], dx4[8], dx5[8], dx6[8],
dgx[8], varscreenop[8];
                                dx.Null(6,1);

                                fstream results ("outputtouch.txt", ios::in);
                                if (!results){
                                        cout<<"Can not open";
                                }

                                if (!results.eof())
                                {
                                        results.getline(dx1, 8, '\t');
                                        results.getline(dx2, 8, '\t');
                                        results.getline(dx3, 8, '\t');

                                        results.getline(dx4, 8, '\t');
                                        results.getline(dx5, 8, '\t');
                                        results.getline(dx6, 8, '\t');
                                        results.getline(dgx, 8, '\t');
                                        results.getline(varscreenop, 8, '\t');

                                        results.close();
                                }
                                dx(0,0) = atoi (dx1);
                                cout<<"dx1: "<<dx(0,0)<<"\n";
```

Appendix A. (Continued)

```
                        dx(1,0) = atoi (dx2);
                        cout<<"dx2: "<<dx(1,0)<<"\n";
                        dx(2,0) = atoi (dx3);
                        cout<<"dx3: "<<dx(2,0)<<"\n";
                        dx(3,0) = atof (dx4);
                        cout<<"dx4: "<<dx(3,0)<<"\n";
                        dx(4,0) = atof (dx5);
                        cout<<"dx5: "<<dx(4,0)<<"\n";
                        dx(5,0) = atof (dx6);
                        cout<<"dx6: "<<dx(5,0)<<"\n";
                        dg = atoi (dgx);
                        cout<<"dx7: "<<dg<<"\n";
                        varscreenopn = atoi (varscreenop);
                        cout<<"varscreen is: "<<varscreenopn<<"\n\n";
                        dx*=(v*dt);
                }
                fprintf(fid," dx is: \n\n");
                for (j=0;j<dx.RowNo();j++){
                        fprintf(fid," %4.2f ",  dx(j,0));
                }
                fprintf(fid," \n\n\n");
        //      cout<<"trying2"<<endl;

                // 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(0,0)=-dx(1,0);
                        dx(1,0)=-dx(2,0);
                        dx(2,0)=dx(0,0);
                        dx(3,0)=-dx(4,0);
                        dx(4,0)=-dx(5,0);
                        dx(5,0)=dx(3,0);
        //              cout<<"trying3"<<endl;
                }
                if (cart==2) {
                        dx.SetSize(3,1);
        //              cout<<"trying4"<<endl;
                }

                // 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:
                fprintf(fid," dq is: \n\n");
                for (j=0;j<dq.RowNo();j++){
                        fprintf(fid," %4.2f ",  dq(j,0));
                }
                fprintf(fid," \n\n\n");

                if (WCA==2) {
        //              cout<<"tryingbef52"<<endl;
                        dq.SetSize(7,1);
                        dq1.Null(7,1);
                        dq1=WMRA_Opt(optim, JLA, JLO, Joa, detJoa, dq, dx, dt, q1);
                        dq1.SetSize(9,1);
                //      cout<<"trying5"<<endl;
                }
                else if (WCA==3) {

                        Matrix dqtemp(2,1), dxtemp(2,1);
                        dqtemp(0,0)=dq(7,0);
                        dqtemp(1,0)=dq(8,0);
                        dxtemp(0,0)=dx(0,0);
                        dxtemp(1,0)=dx(1,0);
                        Unit.Unit(1);
                        dq1.Null(2,1);
```

Appendix A. (Continued)

```
        dq1=WMRA_Opt(optim, JLA, JLO, Jowc, 1, dqtemp, dxtemp, dt, Unit);
                        dqtemp.Null(9,1);
                        dqtemp(7,0)=dq1(0,0);
                        dqtemp(8,0)=dq1(1,0);
                        dq1=dqtemp;
                }
                else {
                        dq1.Null(9,1);
                        dq1=WMRA_Opt(optim, JLA, JLO, Jo, detJo, dq, dx, dt, q1);
                }
                fprintf(fid," dq is: \n\n");
                for (j=0;j<dq1.RowNo();j++){
                        fprintf(fid," %4.2f ",  dq1(j,0));
                }
                fprintf(fid," \n\n\n");

                // Calculating the new Joint Angles:
                qn=q1+dq1;
                fprintf(fid," qn is: \n\n");
                for (j=0;j<qn.RowNo();j++){
                        fprintf(fid," %4.2f ",  qn(j,0));
                }
                fprintf(fid," \n\n\n");

        //              cout<<"try"<<endl;

                // Calculating the new Transformation Matrices:
                Matrix dqtemp(2,1);
                dqtemp(0,0)=dq1(7,0);
                dqtemp(1,0)=dq1(8,0);
                WMRA_Tall(2, qn, dqtemp, Twc1, Tn, Tna, Tnwc, T01, T12, T23, T34,
T45, T56, T67);
                fprintf(fid," Tn is: \n\n");
                for (j=0;j<Tn.RowNo();j++){
                        for (k=0;k<Tn.ColNo();k++){
                                fprintf(fid," %4.2f ",  Tn(j,k));
                        }
                        fprintf(fid," \n");
                }
                fprintf(fid," \n\n\n");
                fprintf(fid," Tna is: \n\n");
                for (j=0;j<Tna.RowNo();j++){
                        for (k=0;k<Tna.ColNo();k++){
                                fprintf(fid," %4.2f ",  Tna(j,k));
                        }
                        fprintf(fid," \n");
                }
                fprintf(fid," \n\n\n");
                fprintf(fid," Tnwc is: \n\n");
                for (j=0;j<Tnwc.RowNo();j++){
                        for (k=0;k<Tnwc.ColNo();k++){
                                fprintf(fid," %4.2f ",  Tnwc(j,k));
                        }
                        fprintf(fid," \n");
                }
                fprintf(fid," \n\n\n");

// 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){
                        Matrix dqtemp2(2,1), dq2(7,1),dq3(7,1);
                        dqtemp2(0,0)=dq1(7,0);
                        dqtemp2(1,0)=dq1(8,0);
                        for (j=0;j<7;j++){
                                dq2(j,0)=dq1(j,0);
```

Appendix A. (Continued)

```
                        }
                        dq3=WMRA_collide(dq2, T01, T12, T23, T34, T45, T56, T67);
                        // Re-calculating the new Joint Angles:
                        dq1.Null(9,1);
                        for (j=0;j<7;j++){
                                dq1(j,0)=dq3(j,0);
                        }
                        for (j=7;j<9;j++){
                                dq1(j,0)=dqtemp2(j-7,0);
                        }
                        qn=q1+dq1;

                        // Re-calculating the new Transformation Matrices:

                        WMRA_Tall(2, qn, dqtemp2, Twc1, Tn, Tna, Tnwc, T01, T12,
T23, T34, T45, T56, T67);
                }

                fprintf(fid," qn is: \n\n");
                for (j=0;j<qn.RowNo();j++){
                        fprintf(fid," %4.2f ",  qn(j,0));
                }
                fprintf(fid," \n\n\n");
                fprintf(fid," Tn is: \n\n");
                for (j=0;j<Tn.RowNo();j++){
                        for (k=0;k<Tn.ColNo();k++){
                                fprintf(fid," %4.2f ",  Tn(j,k));
                        }
                        fprintf(fid," \n");
                }
                fprintf(fid," \n\n\n");
                fprintf(fid," Tna is: \n\n");
                for (j=0;j<Tna.RowNo();j++){
                        for (k=0;k<Tna.ColNo();k++){
                                fprintf(fid," %4.2f ",  Tna(j,k));
                        }
                        fprintf(fid," \n");
                }
                fprintf(fid," \n\n\n");
                fprintf(fid," Tnwc is: \n\n");
                for (j=0;j<Tnwc.RowNo();j++){
                        for (k=0;k<Tnwc.ColNo();k++){
                                fprintf(fid," %4.2f ",  Tnwc(j,k));
                        }
                        fprintf(fid," \n");
                }
                fprintf(fid," \n\n\n");

                // Saving the plot data in case plots are required:
                if (plt==2){
                        WMRA_Plots(1, L, dt, i, tt, qn, dq1, Tn, Tnwc, detJoa,
detJo);
                }

                // Updating Physical Arm:
                if (arm==1){
                        ddt=ddt+dt;
                        if (ddt>=0.5 || i>=(n+1)){
                                Matrix dqtemp(10,1);

                                for (j=0; j<9; j++){
                                        dqtemp(j,0)=qn(j,0);
                                }
                                dqtemp(9,0)=dg;
```

Appendix A. (Continued)

```
                        WMRA_ARM_Motion(2, 1, dqtemp, ddt, g);
                                cout<<"dqtemp(9,0)"<<dqtemp(9,0)<<endl;
                                wait(.5);
                                ddt=0;
                                cout<<"waitfdd= "<<endl;
                        }
                }

                // Updating Virtual Reality Animation:
                if (vr==1){
                        //WMRA_VR_Animation(2, Tnwc, qn);
                }
                // Updating Matlab Graphics Animation:
                if (ml==1){
                        //WMRA_ML_Animation(2, Ti, Td, Tnwc, T01, T12, T23, T34,
T45, T56, T67);
                }
                // Re-Drawing the Animation:
                if (vr==1 || ml==1){
                        //drawnow;
                }

                // Updating the old values with the new values for the next
iteration:
                tt=tt+dt;
                fprintf(fid," tt is %g ",  tt);
                fprintf(fid," \n\n\n");

                // Stopping the simulation when the exit button is pressed:
                if (cont==3 || cont==4 || cont==5){
                        if (cont==3 || cont==4 ){
                        //      Exit myForm2;
                        //      myForm2.ShowDialog() ==
System::Windows::Forms::DialogResult::OK;
                        }
        //              cout<<"Sale o no: "<<varscreenopn<<"\n\n";
                        if (varscreenopn == 1){
                                n=n+1;
                        }
                        else {
                                break;
                        }
                }
                // Delay to comply with the required speed:
                endt=0;
                //time(&endt);
                endt=clock()/ CLOCKS_PER_SEC;
                //endt=clock()*CLK_TCK;
                timedif=0;
                //timedif=difftime(endt,startt);
                timedif=(endt-start2);
                if (timedif < tt){
                        //wait((tt-timedif));
                }
                //endtt=clock()*CLK_TCK;
                //cout<<"\n"<<endtt;
                i++;
                if (cont==3){i=1;}
        }

        fclose(fid);

// Reading the elapsed time and printing it with the simulation time:
        if (cont==1 || cont==2){
                printf("\nSimula. time is %6.6f seconds.\n", total_time);
```

Appendix A. (Continued)

```
		}
		//time(&endf);
		//endf=clock()/ CLOCKS_PER_SEC;
		endf=clock()*CLK_TCK/1000;
		cout<<"\nEnd is "<<endf<<"\n\n";
		//cout<<"\nElapsed time is "<<difftime(endf,startt)*1000<<"\n\n";
		cout<<"\nElapsed time is "<<(endf-startt)<<"\n\n";

		// Plotting:
		if (plt==2){
			WMRA_Plots(2, L, dt, i, tt, qn, dq1, Tn, Tnwc, detJoa, detJo);
		}

		if (vr==1 || ml==1 || arm==1){
			// Going back to the ready position:
			cout << "\n Do you want to go back to the 'ready' position? \n
Press '1' for Yes, or press '2' for No. \n\n ";
			cin >> choice6;
			if (choice6==1){
				WMRA_any2ready(2, vr, ml, arm, Tnwc, qn, g);
				// Going back to the parking position:
				cout << "\n Do you want to go back to the 'parking'
position? \n Press '1' for Yes, or press '2' for No. \n\n ";
				cin >> choice7;
					if (choice7==1){
						Matrix temp(2,1);
						temp(0,0)=qn(7,0);
						temp(1,0)=qn(8,0);
						WMRA_ready2park(2, vr, ml, arm, Tnwc, temp,
g);
					}
			}
						// Closing the Arm library and Matlab
Graphics Animation and Virtual Reality Animation and Plots windows:
			cout << "\n Do you want to close all simulation windows and arm
controls? \n Press '1' for Yes, or press '2' for No. \n\n ";
			cin >> choice8;
			if (choice8==1){
				Matrix temp(1,1);
				temp.Null(1,1);
				if (arm==1){
					WMRA_ARM_Motion(3, 0, temp, 0, g);
				}
				if (vr==1){
					//WMRA_VR_Animation(3, 0, 0);
				}
				if (ml==1){
					//WMRA_ML_Animation(3, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0);
				}
				if (plt==2){
					//close
(figure(2),figure(3),figure(4),figure(5),figure(6),figure(7),figure(8),figure(9),figure(1
0));
				}
			}
		}
		return 0;
}
```

Appendix A. (Continued)

```
/* 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 %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "vector.h"
#include "Opt.h"
#include "WCD.h"
#include "Jlimit.h"
#include "sign.h"
#include <limits>

using namespace std;
using namespace math;

Matrix WMRA_Opt(int i, float JLA, float JLO, Matrix Jo, float detJo, Matrix dq, Matrix
dx, float dt, Matrix q){

        extern Matrix dHo;
        dHo.Null(7,1);

        // Reading the Wheelchair's constant dimentions, all dimentions are converted in
millimeters:
        Matrix L(1,1);
        L=WMRA_WCD();
        Matrix qmin(1,7), qmax(1,7);
        WMRA_Jlimit(qmin, qmax);

        double inf = std::numeric_limits<double>::infinity();
        int WCA, wo, ko, j, k;
        Matrix pinvJo(7,7);
        Matrix dH(7,1);
        Matrix mul(1,1),  mul1(1,1), mul2(1,1), Jotrans(7,7);
        Matrix W(7,7), dia(7,1), Winv(7,7);
        float dof;
        // The case when wheelchair-only control is required with no arm motion:
        if (i==0){
                WCA=3;
                // Calculating the Inverse of the Jacobian, which is always non-singular:
                Matrix Jotemp(2,2);
                for (j=0; j<2; j++){
                        for (k=0; k<2; k++){
                                Jotemp(j,k)=Jo(j,k);
                        }
                }
                pinvJo=!Jotemp;
                // 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(0,0)=dq(0,0)*L(0,4);
        }
        else {
                // Creating the gradient of the optimization function to avoid joint
limits:
                dH.Null(7,1);
                //cout<<"dHo is \n\n"<<dHo<<"\n\n";
```

Appendix A. (Continued)

```
                        if (JLA==1){
                for (j=0; j<7; j++){
                        dH(j,0)=-0.25*pow((qmax(0,j)-qmin(0,j)),2)*(2*q(j,0)-
qmax(0,j)-qmin(0,j))/(pow((qmax(0,j)-q(j,0)),2)*pow((q(j,0)-qmin(0,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,0)) < abs(dHo(j,0)) && q(j,0) < qmax(0,j) &&
q(j,0) > qmin(0,j)){
                                dH(j,0)=0;
                        }
                        else if (abs(dH(j,0)) < abs(dHo(j,0)) && (q(j,0) >=
qmax(0,j) || q(j,0) <= qmin(0,j))){
                                dH(j,0)=inf;
                        }
                        else if (abs(dH(j,0)) > abs(dHo(j,0)) && (q(j,0) >=
qmax(0,j) || q(j,0) <= qmin(0,j))){
                                dH(j,0)=0;
                        }
                }
        }
        dHo = dH;
        //cout<<"dHo is \n\n"<<dHo<<"\n\n";
        // The case when arm-only control is required with no wheelchair motion:
        if (dq.RowNo()==7){
                W.Null(7,7);
                Winv.Null(7,7);
                WCA=2;
                wo=20000000;
                ko=350000;
                // The weight matrix to be used for the Weighted Least Norm
Solution with Joint Limit Avoidance:
                for (j=0; j < 7; j++){
                        for (k=0; k < 7; k++){
                                if (j==k){
                                        W(j,k)=1+abs(dH(j,0));
                                        dia(j,0)=W(j,k);
                                        // The inverse of the diagonal weight
matrix:
                                        Winv(j,k)=1/(dia(j,0));

                                }
                        }
                }
        }
        // The case when wheelchair-and-arm control is required:
        else {
                WCA=1;
                wo=34000000;
                ko=13;
                // The weight matrix to be used for the Weighted Least Norm
Solution:
                W.Null(9,9);
                W(7,7)=10;
                W(8,8)=10;
                for (j=0; j < 7; j++){
                        for (k=0; k < 7; k++){
                                if (j==k){
                                        W(j,k)=1+abs(dH(j,0));
                                }
                        }
                }
                dia.SetSize(9,1);
                Winv.Null(9,9);
                for (j=0; j < 9; j++){
                        for (k=0; k < 9; k++){
```

Appendix A. (Continued)

```
                                if (j==k){
                                        dia(j,0)=W(j,k);
                                        // The inverse of the diagonal weight
matrix:
                                        Winv(j,k)=1/(dia(j,0));
                                }

                }
                        }
                }

                // Redefining the determinant based on the weight:
                if (i==1 || i==2){
                        Jotrans=~Jo;
        //              cout<<"Jo' is\n\n"<<Jotrans<<"\n\n";
                        mul1=Winv*Jotrans;
        //              cout<<"mul1 is\n\n"<<mul1<<"\n\n";
                        mul2=Jo*mul1;
        //              cout<<"mul2 is\n\n"<<mul2<<"\n\n";
                        mul=mul2;
                        //mul= Jo * Winv * Jotrans;
        //              cout<<"mul is\n\n"<<mul<<"\n\n";
                        detJo= sqrt(mul.Det());
        //              cout<<"detJo is\n\n"<<detJo<<"\n\n";
                }
                dof=dx.RowNo();
//              cout<<"dof is\n\n"<<dof<<"\n\n";
        }

        // SR-Inverse and Weighted Least Norm Optimization:
        float sf;
        if (i==1){
                // Calculating the variable scale factor, sf:
                if (detJo<wo){
                        sf=ko*pow((1-detJo/wo),2);        // from eq. 9.79 page 268 of
Nakamura's book.
                }
                else {
                        sf=0;
                }
                        // Calculating the SR-Inverse of the Jacobian:
                Matrix sfm(2,2);
                sfm.Unit(dof);
                sfm*=(sf);
                mul= mul + sfm;
                mul=!mul;
                pinvJo=Winv*Jotrans*mul;
                //cout<<"pinvJo is\n\n"<<pinvJo<<"\n\n";
                // 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;
        //              cout<<"dq is\n\n"<<dq<<"\n\n";
                        dq(7,0)=dq(7,0)*L(0,4);
                }
        //      cout<<"dq is\n\n"<<dq<<"\n\n";
        }

        // Pseudo Inverse and Weighted Least Norm Optimization:
        else if (i==2){
```

Appendix A. (Continued)

```
// Calculating the Pseudo Inverse of the Jacobian:
                mul=!mul;
                pinvJo=Winv*Jotrans*mul;
                // 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(7,0)=dq(7,0)*L(0,4);
                }
        }

        // SR-Inverse and Projection Gradient Optimization based on Euclidean norm of
errors:
        else if (i==3){
                Jotrans=~Jo;
                // Calculating the variable scale factor, sf:
                if (detJo<wo){
                        sf=ko*pow((1-detJo/wo),2);        // from eq. 9.79 page 268 of
Nakamura's book.
                }
                else {
                        sf=0;
                }
                // Calculating the SR-Inverse of the Jacobian:
                Matrix sfm(2,2);
                sfm.Unit(dof);
                sfm*=(sf);
        //      cout<<"\n sfm  is: "<<sfm<<"\n";
        //      cout<<"\n Jo is: "<<Jo<<"\n";
        //      cout<<"\n Jotrans is: "<<Jotrans<<"\n";
                mul= Jo * Jotrans;
        //      cout<<"\n mul is: "<<mul<<"\n";
                mul= mul + sfm;
        //      cout<<"\n mul 2 is: "<<mul<<"\n";
                mul=!mul;
        //      cout<<"\n mul 3 is: "<<mul<<"\n";
                pinvJo=Jotrans * mul;
        //      cout<<"\n pinvJo is: "<<pinvJo<<"\n";
                // 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.
                Matrix unit(7,7);
                if (WCA==2){
                        unit.Unit(7);
                        mul=unit-pinvJo*Jo;
                        //dq=pinvJo*dx+mul*dq;
                        mul*=(0.001);
                        dq=pinvJo*dx+mul*dH;
        //              cout<<"\n dq is: "<<pinvJo<<"\n";

                }
                else {
                        //dq(7,0)=dq(7,0)/L(0,4);
                        unit.Unit(9);
                        mul=unit-pinvJo*Jo;
                        //dq=pinvJo*dx+mul*dq;
                        mul*=(0.001);
```

Appendix A. (Continued)

```
                        dH.SetSize(9,1);
                        dq=pinvJo*dx+mul*dH;
                        dq(7,0)=dq(7,0)*L(0,4);
        //              cout<<"\n dq is: "<<pinvJo<<"\n";
                }
        }
                // Pseudo Inverse and Projection Gradient Optimization based on Euclidean
norm of errors:
        else if (i==4){


                Jotrans=~Jo;
                // Calculating the Pseudo Inverse of the Jacobian:
                mul= Jo * Jotrans;
                mul=!mul;
                pinvJo=Jotrans * mul;
                // 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.
                Matrix unit(7,7);
                if (WCA==2){
                        unit.Unit(7);
                        mul=unit-pinvJo*Jo;
                        //dq=pinvJo*dx+mul*dq;
                        mul*=(0.001);
                        dq=pinvJo*dx+mul*dH;
                }
                else {
                        //dq(7,0)=dq(7,0)/L(0,4);
                        unit.Unit(9);
                        mul=unit-pinvJo*Jo;
                        //dq=pinvJo*dx+mul*dq;
                        mul*=(0.001);
                        dH.SetSize(9,1);
                        dq=pinvJo*dx+mul*dH;
                        dq(7,0)=dq(7,0)*L(0,4);
                }
        }
        if (JLO==1){
                // A safety condition to stop the joint that reaches the joint limits in
the arm:
                if (WCA!=3){
                        for (k=0; k<7; k++){
                                if (q(k,0) >= qmax(0,k) || q(k,0) <= qmin(0,k)){
                                        dq(k,0)=0;
                                }
                        }
                //      cout<<"dq is\n\n"<<dq<<"\n\n";
                }

                // A safety condition to slow the joint that exceeds the velocity limits
in the WMRA:
                Matrix dqmax(2,1);
                if (WCA==3){
                        dqmax(0,0)=100;
                        dqmax(1,0)=0.15;
                        dqmax*=(dt);     // Joity velocity limits when the time increment
is dt second.
                }
                else {
                        dqmax.Null(9,1);
                        for (k=0; k<7; k++){
                                dqmax(k,0)=0.5;
                        }
```

Appendix A. (Continued)

```
                        dqmax(7,0)=100;
                        dqmax(8,0)=0.15;
        //              cout<<"dqmax is\n\n"<<dqmax<<"\n\n";
        //              cout<<"dt is\n\n"<<dt<<"\n\n";
                        dqmax*=(dt);     // Joint velocity limits when the time increment
is dt second.
        //              cout<<"dqmax is\n\n"<<dqmax<<"\n\n";
                }
                for (k=0; k<dq.RowNo(); k++){
                        if (abs(dq(k,0)) >= dqmax(k,0)){
                                dq(k,0)=sign(dq(k,0))*dqmax(k,0);
                        }
                }

//      cout << "\ndq is\n\n" << dq  ;
        }

        return dq;
        }



/* 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 %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "vector.h"
#include "WCD.h"
#include "p2T.h"
using namespace std;
using namespace math;

Matrix WMRA_p2T(float x, float y, float a){

        Matrix T(4,4);

        //Reading the Wheelchair's constant dimentions, all dimentions are converted in
millimeters:
        Matrix L(1,5);
        L=WMRA_WCD();
        //Defining the Transformation Matrix:

        T.Unit(4);
        T(0,0)= cos(a);
        T(0,1)= -sin(a);
        T(0,3)= x;
        T(1,0)= sin(a);
        T(1,1)= cos(a);
        T(1,3)= y;
        T(2,3)= L(0,3)+L(0,4);
        return T;
        }
```

Appendix A. (Continued)

```
/* 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;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 %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres & Punya Basnayaka%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


Function Declaration:*/

#include "matrix.h"
#include "park2ready.h"
#include "ArmMotion.h"
#include "DH.h"
#include "q2T.h"
#include <time.h>
using namespace std;
using namespace math;

void WMRA_park2ready(int ini, int vr, int ml, int arm, Matrix Tiwc, Matrix qiwc, Galil
&g){
Matrix zero(1,1);
zero.Null(1,1);
        if (ini==3){
                if (arm==1){
                        try {
             WMRA_ARM_Motion(ini, 0, zero, 0, g);
                        }
                        catch (...) {
                                cout << "Exception 1 occurred";
                        }
                }
                if (vr==1){
                        /*try {
                                WMRA_VR_Animation(ini, 0, 0);
                        }
                        catch (...) {
                                cout << "Exception 2 occurred";
                        }*/
                }
                if (ml==1){
                        /*try {
                                WMRA_ML_Animation(ini, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
                        }
                        catch (...) {
                                cout << "Exception 3 occurred";
                        }*/
                }
                return;
        }

        // Defining the used conditions:
        float qi2 [7]= {0, PI/2, 0, PI, 0, 0, 0}; // Initial joint angles (Parking
Position).
        float qd2 [7]= {PI/2, PI/2, 0, PI/2, PI/2, PI/2, 0}; // Final joint angles (Ready
Position).
        Matrix qd(7,1), qi(7,1);
        float ts, dt, ddt;
```

Appendix A. (Continued)

```
                int n, j;
        for ( j = 0 ; j < 7 ; j++ ) {
                qi(j,0)=qi2[j];
                qd(j,0)=qd2[j];
        }
        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.
        Matrix dq(1,1);
        dq=qd-qi;
        dq/=(0.5*n+5);  // Joint angle change at every time step.

        FILE * fil;
        fil = fopen("Park2ready.txt","w");
        fprintf(fil," qi is: \n\n");

                int k;
        for (j=0;j<qi.RowNo();j++){
                for (k=0;k<qi.ColNo();k++){
                        fprintf(fil," %4.2f ",  qi(j,k));
                }
                fprintf(fil," \n");
        }
        fprintf(fil," \n\n\n");
        fprintf(fil," qd is: \n\n");
        for (j=0;j<qd.RowNo();j++){
                for (k=0;k<qd.ColNo();k++){
                        fprintf(fil," %4.2f ",  qd(j,k));
                }
                fprintf(fil," \n");
        }
        fprintf(fil," \n\n\n");
        fprintf(fil," ts is %4.2f ",  ts);
        fprintf(fil," \n\n\n");
        fprintf(fil," n is %d ",  n);
        fprintf(fil," \n\n\n");
        fprintf(fil," dt is %4.2f ",  dt);
        fprintf(fil," \n\n\n");
        fprintf(fil," dq is: \n\n");
        for (j=0;j<dq.RowNo();j++){
                for (k=0;k<dq.ColNo();k++){
                        fprintf(fil," %4.2f ",  dq(j,k));
                }
                fprintf(fil," \n");
        }
        fprintf(fil," \n\n\n");

        // Initializing the physical Arm:
        if (arm==1){
                zero.Null(10,1);
                for ( j = 0 ; j < 7 ; j++ ) {
                        zero(j,0)=qi(j,0);
                }
                zero(7,0)=qiwc(0,0);
                zero(8,0)=qiwc(1,0);
                WMRA_ARM_Motion(ini, 2, zero, dt, g);
                ddt=0;
        }

        fprintf(fil," qi is: \n\n");
        for (j=0;j<qi.RowNo();j++){
                for (k=0;k<qi.ColNo();k++){
                        fprintf(fil," %4.2f ",  qi(j,k));
```

Appendix A. (Continued)

```
        }
                fprintf(fil," \n");
        }
        fprintf(fil," \n\n\n");
        fprintf(fil," qiwc is: \n\n");
        for (j=0;j<qiwc.RowNo();j++){
                for (k=0;k<qiwc.ColNo();k++){
                        fprintf(fil," %4.2f ",  qiwc(j,k));
                }
                fprintf(fil," \n");
        }
        fprintf(fil," \n\n\n");

        // Initializing Virtual Reality Animation:
        if (vr==1){

//      WMRA_VR_Animation(ini, Tiwc, qi);
        }

        // Initializing Robot Animation in Matlab Graphics:
        Matrix DH(1,1);
        Matrix T01(4,4), T12(4,4), T23(4,4), T34(4,4), T45(4,4), T56(4,4), T67(4,4);
        Matrix Ti(4,4), Td(4,4);

        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(0,0))*WMRA_transl(DH(0,1),0,0)*WMRA_rotz(DH(0,3))*WMRA_transl(0,0,DH(0,2));
                //T2
                T12 =
WMRA_rotx(DH(1,0))*WMRA_transl(DH(1,1),0,0)*WMRA_rotz(DH(1,3))*WMRA_transl(0,0,DH(1,2));
                //T3
                T23 =
WMRA_rotx(DH(2,0))*WMRA_transl(DH(2,1),0,0)*WMRA_rotz(DH(2,3))*WMRA_transl(0,0,DH(2,2));
                //T4
                T34 =
WMRA_rotx(DH(3,0))*WMRA_transl(DH(3,1),0,0)*WMRA_rotz(DH(3,3))*WMRA_transl(0,0,DH(3,2));
                //T5
                T45 =
WMRA_rotx(DH(4,0))*WMRA_transl(DH(4,1),0,0)*WMRA_rotz(DH(4,3))*WMRA_transl(0,0,DH(4,2));
                //T6
                T56 =
WMRA_rotx(DH(5,0))*WMRA_transl(DH(5,1),0,0)*WMRA_rotz(DH(5,3))*WMRA_transl(0,0,DH(5,2));
                //T7
                T67 =
WMRA_rotx(DH(6,0))*WMRA_transl(DH(6,1),0,0)*WMRA_rotz(DH(6,3))*WMRA_transl(0,0,DH(6,2));
                // 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);
        }

        // Initialization:
        Matrix qo(1,1);
        float tt;
        qo=qi;
        tt=0;

        fprintf(fil," qo is: \n\n");
        for (j=0;j<qo.RowNo();j++){
                for (k=0;k<qo.ColNo();k++){
```

Appendix A. (Continued)

```
        fprintf(fil," %4.2f ",  qo(j,k));
                }
                fprintf(fil," \n");
        }
        fprintf(fil," \n\n\n");
        fprintf(fil," tt is %4.2f ",  tt);
        fprintf(fil," \n\n\n");

        Matrix qn(1,1);
        clock_t startt, endt, endf;
        double timedif;
        Matrix T1a(1,1), T2a(1,1), T3a(1,1), T4a(1,1), T5a(1,1), T6a(1,1), T7a(1,1);

        qn.Null(7,1);
        while (tt <= ts) {
                // Starting a timer:
                startt=0;

startt=clock()/ CLOCKS_PER_SEC;


                // Calculating the new Joint Angles:
                if (tt==0){
                        qn=qo;
                }
                else if (tt < (dt*(0.5*n-5))){
                        qn(0,0)=qo(0,0)+dq(0,0);
                }
                else if (tt < (dt*(0.5*n+5))){
                        qn=qo+dq;
                }
                else if (tt < (dt*(n-1))){
                        for (j=1;j<7;j++){
                                qn(j,0)=qo(j,0)+dq(j,0);
                        }
                }
                fprintf(fil," qn is: \n\n");
                for (j=0;j<qn.RowNo();j++){
                        for (k=0;k<qn.ColNo();k++){
                                fprintf(fil," %4.2f ",  qn(j,k));
                        }
                        fprintf(fil," \n");
                }
                fprintf(fil," \n\n\n");
                fprintf(fil," ddt is %4.2f ",  ddt);
                fprintf(fil," \n\n\n");
                // Updating the physical Arm:
                if (arm==1) {
                        float ddt2=0;
                        ddt2=ddt+dt;
                        if (ddt2>=0.5 || tt>=ts){
                                zero.Null(10,1);
                                for ( j = 0 ; j < 7 ; j++ ) {
                                        zero(j,0)=qn(j,0);
                                }
                                zero(7,0)=qiwc(0,0);
                                zero(8,0)=qiwc(1,0);
                                WMRA_ARM_Motion(2, 1, zero, ddt2, g);
                                ddt2=0;
                        }
                        ddt=0;
                        ddt=ddt2;
                }

                // Updating Virtual Reality Animation:
```

Appendix A. (Continued)

```
                        if (vr==1){
                        /*zero.Null(9,1);
                        for ( j = 0 ; j < 7 ; j++ ) {
                                zero(j,0)=qn(j,0);
                        }
                        zero(7,0)=qiwc(0,0);
                        zero(8,0)=qiwc(1,0);*/
                        //WMRA_VR_Animation(2, Tiwc, zero);
                }

                // Updating Matlab Animation:
                if (ml==1){
                        // Calculating the new Transformation Matrix:
                        T1a =
WMRA_rotx(DH(0,0))*WMRA_transl(DH(0,1),0,0)*WMRA_rotz(qn(0,0))*WMRA_transl(0,0,DH(0,2));
                //T2
                T2a =
WMRA_rotx(DH(1,0))*WMRA_transl(DH(1,1),0,0)*WMRA_rotz(qn(1,0))*WMRA_transl(0,0,DH(1,2));
                //T3
                T3a =
WMRA_rotx(DH(2,0))*WMRA_transl(DH(2,1),0,0)*WMRA_rotz(qn(2,0))*WMRA_transl(0,0,DH(2,2));
                //T4
                T4a =
WMRA_rotx(DH(3,0))*WMRA_transl(DH(3,1),0,0)*WMRA_rotz(qn(3,0))*WMRA_transl(0,0,DH(3,2));
                //T5
                T5a =
WMRA_rotx(DH(4,0))*WMRA_transl(DH(4,1),0,0)*WMRA_rotz(qn(4,0))*WMRA_transl(0,0,DH(4,2));
                //T6
                T6a =
WMRA_rotx(DH(5,0))*WMRA_transl(DH(5,1),0,0)*WMRA_rotz(qn(5,0))*WMRA_transl(0,0,DH(5,2));
                //T7
                T7a =
WMRA_rotx(DH(6,0))*WMRA_transl(DH(6,1),0,0)*WMRA_rotz(qn(6,0))*WMRA_transl(0,0,DH(6,2));

                //WMRA_ML_Animation(2, Ti, Td, Tiwc, T1a, T2a, T3a, T4a, T5a, T6a, T7a);
                }

                // Updating the old values with the new values for the next iteration:

                qo.Null(7,1);
                qo=qn;
                tt=tt+dt;

                fprintf(fil," qo is: \n\n");
                for (j=0;j<qo.RowNo();j++){
                        for (k=0;k<qo.ColNo();k++){
                                fprintf(fil," %4.2f ",  qo(j,k));
                        }
                        fprintf(fil," \n");
                }
                fprintf(fil," \n\n\n");
                fprintf(fil," tt is %4.2f ",  tt);
                fprintf(fil," \n\n\n");

                // Pausing for the speed sync:
                endt=0;
                endt=clock()/ CLOCKS_PER_SEC;
                timedif=0;
                timedif=endt-startt;
                wait((dt-timedif));

        }
        fclose(fil);
}
```

Appendix A. (Continued)

```
/* This function plots different animation variables for USF WMRA with 9 DOF.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/
#include "matrix.h"
#include "plots2.h"
#include "T2rpy.h"
#include <math.h>
using namespace std;
using namespace math;
#define PI 3.14159265
#define r2d 180/PI //Conversions from Radians to Degrees.

void WMRA_Plots(int st, Matrix L, float dt, int i, float tt, Matrix qn, Matrix dq, Matrix
Tn, Matrix Tnwc, float detjoa, float detjo){

// Calling global variables:
       extern Matrix timep, q1, q2, q3, q4, q5, q6, q7, qll, qrr, qd1, qd2, qd3, qd4,
qd5, qd6, qd7, qdl, qdr, x, y, z, roll, pitch, yaw, xc, yc, zc, rollc, pitchc, yawc,
detJoap, detJop;

       timep.SetSize(i+2,1); q1.SetSize(i+2,1); q2.SetSize(i+2,1); q3.SetSize(i+2,1);
q4.SetSize(i+2,1); q5.SetSize(i+2,1); q6.SetSize(i+2,1); q7.SetSize(i+2,1);
qll.SetSize(i+2,1); qrr.SetSize(i+2,1); qd1.SetSize(i+2,1); qd2.SetSize(i+2,1);
qd3.SetSize(i+2,1); qd4.SetSize(i+2,1); qd5.SetSize(i+2,1); qd6.SetSize(i+2,1);
qd7.SetSize(i+2,1); qdl.SetSize(i+2,1); qdr.SetSize(i+2,1); x.SetSize(i+2,1);
y.SetSize(i+2,1); z.SetSize(i+2,1); roll.SetSize(i+2,1); pitch.SetSize(i+2,1);
yaw.SetSize(i+2,1); xc.SetSize(i+2,1); yc.SetSize(i+2,1); zc.SetSize(i+2,1);
rollc.SetSize(i+2,1); pitchc.SetSize(i+2,1); yawc.SetSize(i+2,1); detJoap.SetSize(i+2,1);
detJop.SetSize(i+2,1);


// Data collection at every iteration:
       if (st==1){

              // Generating a time vector for plotting:
              timep(i,0)=tt;
              // Joint Angles:
              q1(i,0)=qn(0,0)*r2d;
              q2(i,0)=qn(1,0)*r2d;
              q3(i,0)=qn(2,0)*r2d;
              q4(i,0)=qn(3,0)*r2d;
              q5(i,0)=qn(4,0)*r2d;
              q6(i,0)=qn(5,0)*r2d;
              q7(i,0)=qn(6,0)*r2d;
              qll(i,0)=qn(7,0)-L(0,0)*qn(8,0)/2;
              qrr(i,0)=qn(7,0)+L(0,0)*qn(8,0)/2;

              // Joint Velocities:
              qd1(i,0)=r2d*dq(0,0)/dt;
              qd2(i,0)=r2d*dq(1,0)/dt;
              qd3(i,0)=r2d*dq(2,0)/dt;
              qd4(i,0)=r2d*dq(3,0)/dt;
              qd5(i,0)=r2d*dq(4,0)/dt;
              qd6(i,0)=r2d*dq(5,0)/dt;
              qd7(i,0)=r2d*dq(6,0)/dt;
              qdl(i,0)=(dq(7,0)-L(0,0)*dq(8,0)/2)/dt;
              qdr(i,0)=(dq(7,0)+L(0,0)*dq(8,0)/2)/dt;
```

Appendix A. (Continued)

```
// Hand Position and Orientation:
                x(i,0)=Tn(0,3);
                y(i,0)=Tn(1,3);
                z(i,0)=Tn(2,3);
                Matrix or(1,1);
                or.Null(1,1);
                or=WMRA_T2rpy(Tn);
                roll(i,0)=or(0,0)*r2d;
                pitch(i,0)=or(1,0)*r2d;
                yaw(i,0)=or(2,0)*r2d;

                // Arm Base Position and Orientation on the Wheelchair:
                xc(i,0)=Tnwc(0,3);
                yc(i,0)=Tnwc(1,3);
                zc(i,0)=Tnwc(2,3);
                Matrix orc(1,1);
                orc.Null(1,1);
                orc=WMRA_T2rpy(Tnwc);
                rollc(i,0)=orc(0,0)*r2d;
                pitchc(i,0)=orc(1,0)*r2d;
                yawc(i,0)=orc(2,0)*r2d;

                // Manipulability Measure:
                detJoap(i,0)=detjoa;
                detJop(i,0)=detjo;
        }
                // Plotting the data in graphas:
        else {
                int j;

                FILE * fid;
                fid = fopen("plots.cvs","w");
                fprintf(fid," time \t qd1 \t qd2 \t qd3 \t qd4 \t qd5 \t qd6 \t qd7 \t\t
time \t qdl \t qdr \t\t time \t q1 \t q2 \t q3 \t q4 \t q5 \t q6 \t q7 \t\t time \t qll
\t qrr \t\t time \t x \t y \t z \t\t time \t roll \t pitch \t yaw \t\t time \t xc \t yc
\t zc \t\t time \t rollc \t pitchc \t yawc \t\t time \t detJoa \t detJo \n");
                for (j=0; j<timep.RowNo(); j++){
                        fprintf(fid," %g \t %g \t %g \t %g \t %g \t %g \t %g \t %g \t\t %g
\t %g \t %g \t\t %g \t %g \t %g \t %g \t %g \t %g \t %g \t %g \t\t %g \t %g \t %g \t\t %g
\t %g \t %g \t %g \t\t %g \t %g \t %g \t %g \t\t %g \t %g \t %g \t %g \t\t %g \t %g \t %g
\t %g \t\t %g \t %g \t %g\n", timep(j,0) , qd1(j,0) , qd2(j,0) , qd3(j,0) , qd4(j,0) ,
qd5(j,0) , qd6(j,0) , qd7(j,0), timep(j,0) , qdl(j,0) , qdr(j,0), timep(j,0) , q1(j,0) ,
q2(j,0) , q3(j,0) , q4(j,0) , q5(j,0) , q6(j,0) , q7(j,0), timep(j,0) , qll(j,0) ,
qrr(j,0), timep(j,0) , x(j,0) , y(j,0), z(j,0), timep(j,0) , roll(j,0) , pitch(j,0),
yaw(j,0), timep(j,0) , xc(j,0) , yc(j,0), zc(j,0), timep(j,0) , rollc(j,0) , pitchc(j,0),
yawc(j,0), timep(j,0) , detJoap(j,0) , detJop(j,0) );
                }
                fclose(fid);

        }
        }
```

Appendix A. (Continued)

```
/*  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 %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "vector.h"
#include "Polynomial.h"
using namespace std;
using namespace math;

#ifndef _NO_TEMPLATE
typedef matrix<double> Matrix;
#else
typedef matrix Matrix;
#endif

Matrix WMRA_Polynomial(float qi, float qf, float n){

        Matrix qtp(2,1);

        float tt, tf, dt;
        int i;
        tt=0;
        tf=abs(qf-qi);
        dt=tf/(n-1);

        float *qttemp;
        qttemp = new float[n];

        for (i=0; i<n; i++){
                if (tf<=0.001){
                        qttemp[i]=qi;
                }
                else {
                        qttemp[i]=qi+(qf-qi)*3*pow(tt,2)/pow(tf,2)-(qf-
qi)*2*pow(tt,3)/pow(tf,3); //From Eq.7.3 and 7.6 page 204,205 of Craig Book
                }
                tt = tt + dt;
        }
        qtp.SetSize(n,1);
        for (i=0; i < n; i++){
                qtp(i,0) = qttemp[i];
        }
        delete [] qttemp;

        return qtp;
        }
```

Appendix A. (Continued)

```
/* 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 %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "vector.h"
#include "DH.h"
#include "Rotx.h"
#include "Rotz.h"
#include "Transl.h"
#include "q2T.h"
#include <time.h>

using namespace std;
using namespace math;

void wait ( int seconds )
{
        clock_t endwait;
        endwait = clock () + seconds * CLOCKS_PER_SEC ;
        while (clock() < endwait) {}
}

Matrix WMRA_q2T(Matrix q){

        Matrix T(4,4);
        // Inputting the D-H Parameters in a Matrix form:
        Matrix DH(1,1);
        DH=WMRA_DH(q);
        // Calculating the transformation matrices of each link:
        Matrix T1(4,4),T2(4,4),T3(4,4),T4(4,4),T5(4,4),T6(4,4),T7(4,4),Tttemp(4,4);
        //T1
        T1 =
WMRA_rotx(DH(0,0))*WMRA_transl(DH(0,1),0,0)*WMRA_rotz(DH(0,3))*WMRA_transl(0,0,DH(0,2));
        //T2
        T2 =
WMRA_rotx(DH(1,0))*WMRA_transl(DH(1,1),0,0)*WMRA_rotz(DH(1,3))*WMRA_transl(0,0,DH(1,2));
        //T3
        T3 =
WMRA_rotx(DH(2,0))*WMRA_transl(DH(2,1),0,0)*WMRA_rotz(DH(2,3))*WMRA_transl(0,0,DH(2,2));
        //T4
        T4 =
WMRA_rotx(DH(3,0))*WMRA_transl(DH(3,1),0,0)*WMRA_rotz(DH(3,3))*WMRA_transl(0,0,DH(3,2));
        //T5
        T5 =
WMRA_rotx(DH(4,0))*WMRA_transl(DH(4,1),0,0)*WMRA_rotz(DH(4,3))*WMRA_transl(0,0,DH(4,2));
        //T6
        T6 =
WMRA_rotx(DH(5,0))*WMRA_transl(DH(5,1),0,0)*WMRA_rotz(DH(5,3))*WMRA_transl(0,0,DH(5,2));
        //T7
        T7 =
WMRA_rotx(DH(6,0))*WMRA_transl(DH(6,1),0,0)*WMRA_rotz(DH(6,3))*WMRA_transl(0,0,DH(6,2));
        //Calculating the total Transformation Matrix of the given arm position:
        T=T1*T2*T3*T4*T5*T6*T7;

        return T;
}
```

Appendix A. (Continued)

```
/* 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;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 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres & Punya A. Basnayaka%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "ready2any.h"
#include "ArmMotion.h"
#include "DH.h"
#include "q2T.h"
#include <time.h>
using namespace std;
using namespace math;

void WMRA_ready2any(int ini, int vr, int ml, int arm, Matrix Tiwc, Matrix qd, Galil &g){
Matrix zero(1,1);
zero.Null(1,1);
        if (ini==3){
                if (arm==1){
                        try {
             WMRA_ARM_Motion(ini, 0, zero, 0, g);
                        }
                        catch (...) {
                                cout << "Exception 1 occurred";
                        }
                }
                if (vr==1){
                        /*try {
                                WMRA_VR_Animation(ini, 0, 0);
                        }
                        catch (...) {
                                cout << "Exception 2 occurred";
                        }*/
                }
                if (ml==1){
                        /*try {
                                WMRA_ML_Animation(ini, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
                        }
                        catch (...) {
                                cout << "Exception 3 occurred";
                        }*/
                }
                return;
        }

        // Defining the used conditions:
        float qi2 [7]= {PI/2, PI/2, 0, PI/2, PI/2, PI/2, 0}; // Initial joint angles
(Ready Position).
        Matrix qi(7,1);
        float ts, dt, ddt;
        int n, j;
        for ( j = 0 ; j < 7 ; j++ ) {
                qi(j,0)=qi2[j];
        }
```

Appendix A. (Continued)

```
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){
                qi.SetSize(10,1);
                WMRA_ARM_Motion(ini, 2, qi, dt, g);
                ddt=0;
                qi.SetSize(7,1);
        }

        // Initializing Virtual Reality Animation:
        if (vr==1){
                //qi.SetSize(9,1);
        //      WMRA_VR_Animation(ini, Tiwc, qi);
                //qi.SetSize(7,1);
        }

        // Initializing Robot Animation in Matlab Graphics:
        Matrix DH(1,1);
        Matrix T01(4,4), T12(4,4), T23(4,4), T34(4,4), T45(4,4), T56(4,4), T67(4,4);
        Matrix Ti(4,4), Td(4,4);

        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(0,0))*WMRA_transl(DH(0,1),0,0)*WMRA_rotz(DH(0,3))*WMRA_transl(0,0,DH(0,2));
                //T2
                T12 =
WMRA_rotx(DH(1,0))*WMRA_transl(DH(1,1),0,0)*WMRA_rotz(DH(1,3))*WMRA_transl(0,0,DH(1,2));
                //T3
                T23 =
WMRA_rotx(DH(2,0))*WMRA_transl(DH(2,1),0,0)*WMRA_rotz(DH(2,3))*WMRA_transl(0,0,DH(2,2));
                //T4
                T34 =
WMRA_rotx(DH(3,0))*WMRA_transl(DH(3,1),0,0)*WMRA_rotz(DH(3,3))*WMRA_transl(0,0,DH(3,2));
                //T5
                T45 =
WMRA_rotx(DH(4,0))*WMRA_transl(DH(4,1),0,0)*WMRA_rotz(DH(4,3))*WMRA_transl(0,0,DH(4,2));
                //T6
                T56 =
WMRA_rotx(DH(5,0))*WMRA_transl(DH(5,1),0,0)*WMRA_rotz(DH(5,3))*WMRA_transl(0,0,DH(5,2));
                //T7
                T67 =
WMRA_rotx(DH(6,0))*WMRA_transl(DH(6,1),0,0)*WMRA_rotz(DH(6,3))*WMRA_transl(0,0,DH(6,2));
                // 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);
        }

        // Check for the shortest route:
        Matrix diff(7,1);
        for ( j = 0 ; j < 7 ; j++ ) {
                diff(j,0)=qd(j,0)-qi(j,0);
        }
        for ( j = 0 ; j < 7 ; j++ ) {
                if (diff(j,0) > PI) {
```

Appendix A. (Continued)

```
diff(j,0)=diff(j,0)-2*PI;
                }
                else if (diff(j,0) < (-PI)) {
                        diff(j,0)=diff(j,0)+2*PI;
                }
        }


// Joint angle change at every time step.
        diff/=(n);
        Matrix dq(7,1);
        for ( j = 0 ; j < 7 ; j++ ) {
                dq(j,0)=diff(j,0);
        }

        // Initialization:
        Matrix qo(1,1);
        float tt;
        qo=qi;
        tt=0;

        Matrix qn(1,1);
        clock_t startt, endt, endf;
        double timedif;
        Matrix T1a(1,1), T2a(1,1), T3a(1,1), T4a(1,1), T5a(1,1), T6a(1,1), T7a(1,1);

        while (tt <= (ts-dt)) {
                // Starting a timer:
                startt=0;
                startt=clock()/ CLOCKS_PER_SEC;


                // Calculating the new Joint Angles:
                qn.Null(7,1);
                qn=qo+dq;

                // Updating the physical Arm:
                if (arm==1) {
                        float ddt2;
                        ddt2=0;
                        ddt2=ddt+dt;
                        if (ddt>=0.5 || tt>=(ts-dt)){
                                qn.SetSize(10,1);
                                WMRA_ARM_Motion(2, 1, qn, ddt2, g);
                                ddt=0;
                                qn.SetSize(7,1);
                        }
                        ddt=0;
                        ddt=ddt2;
                }

                // Updating Virtual Reality Animation:
                if (vr==1){
                        //qn.SetSize(9,1);
                        //WMRA_VR_Animation(2, Tiwc, qn);
                }

                // Updating Matlab Animation:
                if (ml==1){
                        // Calculating the new Transformation Matrix:
                        T1a =
WMRA_rotx(DH(0,0))*WMRA_transl(DH(0,1),0,0)*WMRA_rotz(qn(0,0))*WMRA_transl(0,0,DH(0,2));
                //T2
                T2a =
WMRA_rotx(DH(1,0))*WMRA_transl(DH(1,1),0,0)*WMRA_rotz(qn(1,0))*WMRA_transl(0,0,DH(1,2));
```

Appendix A. (Continued)

```
                //T3
                T3a =
WMRA_rotx(DH(2,0))*WMRA_transl(DH(2,1),0,0)*WMRA_rotz(qn(2,0))*WMRA_transl(0,0,DH(2,2));
                //T4
                T4a =
WMRA_rotx(DH(3,0))*WMRA_transl(DH(3,1),0,0)*WMRA_rotz(qn(3,0))*WMRA_transl(0,0,DH(3,2));
                //T5
                T5a =
WMRA_rotx(DH(4,0))*WMRA_transl(DH(4,1),0,0)*WMRA_rotz(qn(4,0))*WMRA_transl(0,0,DH(4,2));
                //T6
                T6a =
WMRA_rotx(DH(5,0))*WMRA_transl(DH(5,1),0,0)*WMRA_rotz(qn(5,0))*WMRA_transl(0,0,DH(5,2));
                //T7
                T7a =
WMRA_rotx(DH(6,0))*WMRA_transl(DH(6,1),0,0)*WMRA_rotz(qn(6,0))*WMRA_transl(0,0,DH(6,2));

                //WMRA_ML_Animation(2, Ti, Td, Tiwc, T1a, T2a, T3a, T4a, T5a, T6a, T7a);
                }

                // Updating the old values with the new values for the next iteration:

                qo.Null(7,1);
                qo=qn;
                tt=tt+dt;

                // Pausing for the speed sync:
                endt=0;
                endt=clock()/ CLOCKS_PER_SEC;
                timedif=0;
                timedif=endt-startt;
                wait((dt-timedif));
        }
}


/* 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 %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres & Punya A. Basnayaka%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "ready2park.h"
#include "ArmMotion.h"
#include "DH.h"
#include "q2T.h"
#include <time.h>
using namespace std;
using namespace math;

void WMRA_ready2park(int ini, int vr, int ml, int arm, Matrix Tiwc, Matrix qiwc, Galil
&g){

Matrix zero(1,1);
zero.Null(1,1);
```

Appendix A. (Continued)

```
            if (ini==3){
            if (arm==1){
                    try {
          WMRA_ARM_Motion(ini, 0, zero, 0, g);
                    }
                    catch (...) {
                            cout << "Exception 1 occurred";
                    }
            }
            if (vr==1){

                    /*try {
                            WMRA_VR_Animation(ini, 0, 0);
                    }
                    catch (...) {
                            cout << "Exception 2 occurred";
                    }*/
            }
            if (ml==1){
                    /*try {
                            WMRA_ML_Animation(ini, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
                    }
                    catch (...) {
                            cout << "Exception 3 occurred";
                    }*/
            }
            return;
    }

    // Defining the used conditions:
    float qi2 [7]= {PI/2, PI/2, 0, PI/2, PI/2, PI/2, 0}; // Initial joint angles
(Parking Position).
    float qd2 [7]= {0, PI/2, 0, PI, 0, 0, 0}; // Final joint angles (Ready Position).
    Matrix qd(7,1), qi(7,1);
    float ts, dt, ddt;
    int n, j;
    for ( j = 0 ; j < 7 ; j++ ) {
            qi(j,0)=qi2[j];
            qd(j,0)=qd2[j];
    }
    ts=20;  // (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.
    Matrix dq(1,1);
    dq=qd-qi;
    dq/=(0.5*n+5);  // Joint angle change at every time step.

    FILE * fil;
    fil = fopen("Ready2park.txt","w");
    fprintf(fil," qi is: \n\n");
    int k;
    for (j=0;j<qi.RowNo();j++){
            for (k=0;k<qi.ColNo();k++){
                    fprintf(fil," %4.2f ",  qi(j,k));
            }
            fprintf(fil," \n");
    }
    fprintf(fil," \n\n\n");
    fprintf(fil," qd is: \n\n");
    for (j=0;j<qd.RowNo();j++){
            for (k=0;k<qd.ColNo();k++){
                    fprintf(fil," %4.2f ",  qd(j,k));
            }
```

Appendix A. (Continued)

```
        fprintf(fil," \n");
        }
        fprintf(fil," \n\n\n");
        fprintf(fil," ts is %4.2f ",  ts);
        fprintf(fil," \n\n\n");
        fprintf(fil," n is %d ",  n);
        fprintf(fil," \n\n\n");
        fprintf(fil," dt is %4.2f ",  dt);
        fprintf(fil," \n\n\n");
        fprintf(fil," dq is: \n\n");
        for (j=0;j<dq.RowNo();j++){
                for (k=0;k<dq.ColNo();k++){
                fprintf(fil," %4.2f ",  dq(j,k));
                }
                fprintf(fil," \n");
        }
        fprintf(fil," \n\n\n");
        // Initializing the physical Arm:
        if (arm==1){
                zero.Null(10,1);
                for ( j = 0 ; j < 7 ; j++ ) {
                        zero(j,0)=qi(j,0);
                }
                zero(7,0)=qiwc(0,0);
                zero(8,0)=qiwc(1,0);
                WMRA_ARM_Motion(ini, 2, zero, dt, g);
                ddt=0;
        }
        fprintf(fil," qi is: \n\n");
        for (j=0;j<qi.RowNo();j++){
                for (k=0;k<qi.ColNo();k++){
                        fprintf(fil," %4.2f ",  qi(j,k));
                }
                fprintf(fil," \n");
        }
        fprintf(fil," \n\n\n");
        fprintf(fil," qiwc is: \n\n");
        for (j=0;j<qiwc.RowNo();j++){
                for (k=0;k<qiwc.ColNo();k++){
                        fprintf(fil," %4.2f ",  qiwc(j,k));
                }
                fprintf(fil," \n");
        }
        fprintf(fil," \n\n\n");
        // Initializing Virtual Reality Animation:
        if (vr==1){
                /*zeros.Null(9,1);
                for ( j = 0 ; j < 7 ; j++ ) {
                        zeros(j,0)=qi(j,0);
                }
                zeros(7,0)=qiwc(0,0);
                zeros(8,0)=qiwc(1,0);*/
        //      WMRA_VR_Animation(ini, Tiwc, qi);
        }
        // Initializing Robot Animation in Matlab Graphics:
        Matrix DH(1,1);
        Matrix T01(4,4), T12(4,4), T23(4,4), T34(4,4), T45(4,4), T56(4,4), T67(4,4);
        Matrix Ti(4,4), Td(4,4);
                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(0,0))*WMRA_transl(DH(0,1),0,0)*WMRA_rotz(DH(0,3))*WMRA_transl(0,0,DH(0,2));
```

Appendix A. (Continued)

```
                //T2
                T12 =
WMRA_rotx(DH(1,0))*WMRA_transl(DH(1,1),0,0)*WMRA_rotz(DH(1,3))*WMRA_transl(0,0,DH(1,2));
                //T3
                T23 =
WMRA_rotx(DH(2,0))*WMRA_transl(DH(2,1),0,0)*WMRA_rotz(DH(2,3))*WMRA_transl(0,0,DH(2,2));
                //T4
                T34 =
WMRA_rotx(DH(3,0))*WMRA_transl(DH(3,1),0,0)*WMRA_rotz(DH(3,3))*WMRA_transl(0,0,DH(3,2));
                //T5


                T45=
WMRA_rotx(DH(4,0))*WMRA_transl(DH(4,1),0,0)*WMRA_rotz(DH(4,3))*WMRA_transl(0,0,DH(4,2));
                //T6
                T56 =
WMRA_rotx(DH(5,0))*WMRA_transl(DH(5,1),0,0)*WMRA_rotz(DH(5,3))*WMRA_transl(0,0,DH(5,2));
                //T7
                T67 =
WMRA_rotx(DH(6,0))*WMRA_transl(DH(6,1),0,0)*WMRA_rotz(DH(6,3))*WMRA_transl(0,0,DH(6,2));
                // 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);
        }

        // Initialization:
        Matrix qo(1,1);
        float tt;
        qo=qi;
        tt=0;

        fprintf(fil," qo is: \n\n");
        for (j=0;j<qo.RowNo();j++){
                for (k=0;k<qo.ColNo();k++){
                        fprintf(fil," %4.2f ",  qo(j,k));
                }
                fprintf(fil," \n");
        }
        fprintf(fil," \n\n\n");
        fprintf(fil," tt is %4.2f ",  tt);
        fprintf(fil," \n\n\n");

        Matrix qn(1,1);
        clock_t startt, endt, endf;
        double timedif;
        Matrix T1a(1,1), T2a(1,1), T3a(1,1), T4a(1,1), T5a(1,1), T6a(1,1), T7a(1,1);

        qn.Null(7,1);
        while (tt <= ts) {
                // Starting a timer:
                startt=0;
                startt=clock()/ CLOCKS_PER_SEC;


                // Calculating the new Joint Angles:

                if (tt==0){
                        qn=qo;
                }
                else if (tt < (dt*(0.5*n-5))){
                        for (j=1;j<7;j++){
                                qn(j,0)=qo(j,0)+dq(j,0);
```

Appendix A. (Continued)

```
		}
		}
		else if (tt < (dt*(0.5*n+5))){
			qn=qo+dq;
		}
		else if (tt < (dt*(n-1))){
			qn(0,0)=qo(0,0)+dq(0,0);
		}

		fprintf(fil," qn is: \n\n");
		for (j=0;j<qn.RowNo();j++){
			for (k=0;k<qn.ColNo();k++){
				fprintf(fil," %4.2f ",  qn(j,k));
			}
			fprintf(fil," \n");


		}
		fprintf(fil," \n\n\n");
		fprintf(fil," ddt is %4.2f ",  ddt);
		fprintf(fil," \n\n\n");

		// Updating the physical Arm:
		if (arm==1) {
			float ddt2=0;
			ddt2=ddt+dt;
			if (ddt>=0.5 || tt>=ts){
				zero.Null(10,1);
				for ( j = 0 ; j < 7 ; j++ ) {
					zero(j,0)=qn(j,0);
				}
				zero(7,0)=qiwc(0,0);
				zero(8,0)=qiwc(1,0);
				WMRA_ARM_Motion(2, 1, zero, ddt2, g);
				ddt2=0;
			}
			ddt=0;
			ddt=ddt2;
		}

		// Updating Virtual Reality Animation:
		if (vr==1){
			/*zero.Null(9,1);
			for ( j = 0 ; j < 7 ; j++ ) {
				zero(j,0)=qn(j,0);
			}
			zero(7,0)=qiwc(0,0);
			zero(8,0)=qiwc(1,0);*/
			//WMRA_VR_Animation(2, Tiwc, zero);
		}

		// Updating Matlab Animation:
		if (ml==1){
			// Calculating the new Transformation Matrix:
			T1a =
WMRA_rotx(DH(0,0))*WMRA_transl(DH(0,1),0,0)*WMRA_rotz(qn(0,0))*WMRA_transl(0,0,DH(0,2));
		//T2
		T2a =
WMRA_rotx(DH(1,0))*WMRA_transl(DH(1,1),0,0)*WMRA_rotz(qn(1,0))*WMRA_transl(0,0,DH(1,2));
		//T3
		T3a =
WMRA_rotx(DH(2,0))*WMRA_transl(DH(2,1),0,0)*WMRA_rotz(qn(2,0))*WMRA_transl(0,0,DH(2,2));
		//T4
```

Appendix A. (Continued)

```
                T4a =
        WMRA_rotx(DH(3,0))*WMRA_transl(DH(3,1),0,0)*WMRA_rotz(qn(3,0))*WMRA_transl(0,0,DH(
        3,2));
                //T5
                T5a =
WMRA_rotx(DH(4,0))*WMRA_transl(DH(4,1),0,0)*WMRA_rotz(qn(4,0))*WMRA_transl(0,0,DH(4,2));
                //T6
                T6a =
WMRA_rotx(DH(5,0))*WMRA_transl(DH(5,1),0,0)*WMRA_rotz(qn(5,0))*WMRA_transl(0,0,DH(5,2));
                //T7
                T7a =
WMRA_rotx(DH(6,0))*WMRA_transl(DH(6,1),0,0)*WMRA_rotz(qn(6,0))*WMRA_transl(0,0,DH(6,2));

                //WMRA_ML_Animation(2, Ti, Td, Tiwc, T1a, T2a, T3a, T4a, T5a, T6a, T7a);
                }
                // Updating the old values with the new values for the next iteration:
                qo.Null(7,1);
                qo=qn;
                tt=tt+dt;
                fprintf(fil," qo is: \n\n");
                for (j=0;j<qo.RowNo();j++){
                        for (k=0;k<qo.ColNo();k++){
                                fprintf(fil," %4.2f ",  qo(j,k));
                        }
                        fprintf(fil," \n");
                }
                fprintf(fil," \n\n\n");
                fprintf(fil," tt is %4.2f ",  tt);
                fprintf(fil," \n\n\n");
                // Pausing for the speed sync:
                endt=0;
                endt=clock()/ CLOCKS_PER_SEC;
                timedif=0;
                timedif=endt-startt;
                wait((dt-timedif));
        }
        fclose(fil);
}


/* This function gives the homogeneous transformation matrix, given the rotation angle
about the X axis.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "vector.h"
#include "Rotx.h"
using namespace std;
using namespace math;

Matrix WMRA_rotx(float t){

        Matrix T(4,4);

        float c, s;
        c=cos(t);
```

Appendix A. (Continued)

```
        s=sin(t);
        T.Unit(4);
        T(1,1)= c;
        T(1,2)= -s;
        T(2,1)= s;
        T(2,2)= c;

        return T;
}

/* This function gives the homogeneous transformation matrix, given the rotation angle
about the Y axis.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2007 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "vector.h"
#include "Roty.h"
using namespace std;
using namespace math;

Matrix WMRA_roty(float t){

        Matrix T(4,4);

        float c, s;
        c=cos(t);
        s=sin(t);
        T.Unit(4);
        T(0,0)= c;
        T(0,2)= s;
        T(2,0)= -s;
        T(2,2)= c;

        return T;
}


/* This function gives the homogeneous transformation matrix, given the rotation angle
about the Z axis.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "vector.h"
#include "Rotz.h"
using namespace std;
using namespace math;

Matrix WMRA_rotz(float t){
```

```
        Matrix T(4,4);

        float c, s;
        c=cos(t);
        s=sin(t);
        T.Unit(4);
        T(0,0)= c;
        T(0,1)= -s;
        T(1,0)= s;
        T(1,1)= c;

        return T;

}

        /* This function returns the sign of a variable.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#include <iostream>
#include <math.h>
#include "sign.h"
using namespace std;

int sign(float x){
        int y;
        if (x>0){
                y=1;
        }
        else if (x==0){
                y=0;
        }
        else {
                y=-1;
        }
        return y;
}
/* This function gives the Roll, Pitch, Taw angles, given the transformation matrix.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "vector.h"
#include "T2rpy.h"
using namespace std;
using namespace math;

Matrix WMRA_T2rpy(Matrix T){

        Matrix rpy(3,1);
        rpy.Null(3,1);
        float c, s;

        //Making sure there is no singularity:
```

Appendix A. (Continued)

```
                if (abs(T(0,0))<EPS && abs(T(1,0))<EPS){
                rpy(0,0)=0;
                rpy(1,0)=atan2(-T(2,0), T(0,0));
                rpy(2,0)=atan2(-T(1,2), T(1,1));
        }
        else{
                rpy(0,0)=atan2(T(1,0), T(0,0));
                s=sin(rpy(0,0));
                c=cos(rpy(0,0));
                rpy(1,0)=atan2(-T(2,0), c*T(0,0)+s*T(1,0));
                rpy(2,0)=atan2(s*T(0,2)-c*T(1,2), c*T(1,1)-s*T(0,1));
        }

        return rpy;
}


/* 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 %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "vector.h"
#include "DH.h"
#include "Rotx.h"
#include "Rotz.h"
#include "Transl.h"
#include "w2T.h"
#include "Tall.h"
using namespace std;
using namespace math;

void WMRA_Tall(int i, Matrix q, Matrix dq, Matrix Twc, Matrix& T, Matrix& Ta, Matrix&
Twco, Matrix& T1, Matrix& T2, Matrix& T3, Matrix& T4, Matrix& T5, Matrix& T6, Matrix&
T7){

        // Inputting the D-H Parameters in a Matrix form:
        extern Matrix DH;

        if (i==1){
                DH=WMRA_DH(q);
                //Calculating the transformation matrices of each link:
                //T1
                T1 =
WMRA_rotx(DH(0,0))*WMRA_transl(DH(0,1),0,0)*WMRA_rotz(DH(0,3))*WMRA_transl(0,0,DH(0,2));
                //T2
                T2 =
WMRA_rotx(DH(1,0))*WMRA_transl(DH(1,1),0,0)*WMRA_rotz(DH(1,3))*WMRA_transl(0,0,DH(1,2));
                //T3
                T3 =
WMRA_rotx(DH(2,0))*WMRA_transl(DH(2,1),0,0)*WMRA_rotz(DH(2,3))*WMRA_transl(0,0,DH(2,2));
                //T4
                T4 =
WMRA_rotx(DH(3,0))*WMRA_transl(DH(3,1),0,0)*WMRA_rotz(DH(3,3))*WMRA_transl(0,0,DH(3,2));
                //T5
```

Appendix A. (Continued)

```
                T5 =
        WMRA_rotx(DH(4,0))*WMRA_transl(DH(4,1),0,0)*WMRA_rotz(DH(4,3))*WMRA_transl(0,0,DH(
        4,2));
                //T6
                T6 =
WMRA_rotx(DH(5,0))*WMRA_transl(DH(5,1),0,0)*WMRA_rotz(DH(5,3))*WMRA_transl(0,0,DH(5,2));
                //T7
                T7 =
WMRA_rotx(DH(6,0))*WMRA_transl(DH(6,1),0,0)*WMRA_rotz(DH(6,3))*WMRA_transl(0,0,DH(6,2));

                //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:
                Twco=Twc;
                T=Twc*Ta;
        }

                else{

                //T1
                T1 =
WMRA_rotx(DH(0,0))*WMRA_transl(DH(0,1),0,0)*WMRA_rotz(q(0,0))*WMRA_transl(0,0,DH(0,2));
                //T2
                T2 =
WMRA_rotx(DH(1,0))*WMRA_transl(DH(1,1),0,0)*WMRA_rotz(q(1,0))*WMRA_transl(0,0,DH(1,2));
                //T3
                T3 =
WMRA_rotx(DH(2,0))*WMRA_transl(DH(2,1),0,0)*WMRA_rotz(q(2,0))*WMRA_transl(0,0,DH(2,2));
                //T4
                T4 =
WMRA_rotx(DH(3,0))*WMRA_transl(DH(3,1),0,0)*WMRA_rotz(q(3,0))*WMRA_transl(0,0,DH(3,2));
                //T5
                T5 =
WMRA_rotx(DH(4,0))*WMRA_transl(DH(4,1),0,0)*WMRA_rotz(q(4,0))*WMRA_transl(0,0,DH(4,2));
                //T6
                T6 =
WMRA_rotx(DH(5,0))*WMRA_transl(DH(5,1),0,0)*WMRA_rotz(q(5,0))*WMRA_transl(0,0,DH(5,2));
                //T7
                T7 =
WMRA_rotx(DH(6,0))*WMRA_transl(DH(6,1),0,0)*WMRA_rotz(q(6,0))*WMRA_transl(0,0,DH(6,2));

                //Calculating the Transformation Matrix of the initial arm position:

                Ta=T1*T2*T3*T4*T5*T6*T7;
                Twc = WMRA_w2T(1, Twc, dq);
                Twco=Twc;
                T=Twc*Ta;
        }
}
```

Appendix A. (Continued)

```
/* 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 %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/
#include "matrix.h"
#include "vector.h"
#include "Polynomial.h"
#include "Linear.h"
#include "BPolynomial.h"
#include "sign.h"
#include "Traj.h"
using namespace std;
using namespace math;

float ***WMRA_traj(int ind, Matrix Ti, Matrix Td, int n){

        float ***Tt;

        //Finding the rotation of the desired point based on the initial point:
        Matrix R(3,3), Titemp(3,3),Tdtemp(3,3);
        int i,j,m;
        for ( i=0 ; i < 3 ; i++ ) {
                for ( j = 0 ; j < 3 ; j++ ) {
                        Titemp(i,j)=Ti(i,j);

Tdtemp(i,j)=Td(i,j);
                }
        }
        Titemp=~Titemp;
        R=Titemp*Tdtemp;

        //Initial single-angle representation of the rotation:
        float a, s, c, v;
        a=atan2(sqrt(pow((R(2,1)-R(1,2)),2)+pow((R(0,2)-R(2,0)),2)+pow((R(1,0)-
R(0,1)),2)),(R(0,0)+R(1,1)+R(2,2)-1));
        s=sin(a);
        c=cos(a);
        v=1-c;

        //Finding the single-vector components for the rotation:
        float kx, ky, kz;
        if (a<0.001){
                kx=1;
                ky=0;
                kz=0;
        }
        else if (a<PI/2+0.001){
                kx=(R(2,1)-R(1,2))/(2*s);
                ky=(R(0,2)-R(2,0))/(2*s);
                kz=(R(1,0)-R(0,1))/(2*s);
        }
        else {
                kx=sign((R(2,1)-R(1,2)))*sqrt((R(0,0)-c)/v);
                ky=sign((R(0,2)-R(2,0)))*sqrt((R(1,1)-c)/v);
                kz=sign((R(1,0)-R(0,1)))*sqrt((R(2,2)-c)/v);
                if (kx>ky && kx>kz){
```

Appendix A. (Continued)

```
        ky=(R(1,0)+R(0,1))/(2*kx*v);
                kz=(R(0,2)+R(2,0))/(2*kx*v);
        }
        else if (ky>kx && ky>kz){
                kx=(R(1,0)+R(0,1))/(2*ky*v);
                kz=(R(2,1)+R(1,2))/(2*ky*v);
        }
        else {
                kx=(R(0,2)+R(2,0))/(2*kz*v);
                ky=(R(2,1)+R(1,2))/(2*kz*v);
        }
}

// Running the desired trajectory method:
// 1 == Polynomial with Blending function,
// 2 == Polynomial without Blending function,
// 3 == Linear function.
Matrix at(n,1), xt(n,1), yt(n,1), zt(n,1);
Titemp=~Titemp;
if (ind == 2){
        at=WMRA_Polynomial(0,a,n);
        xt=WMRA_Polynomial(Ti(0,3), Td(0,3), n);
        yt=WMRA_Polynomial(Ti(1,3), Td(1,3), n);
        zt=WMRA_Polynomial(Ti(2,3), Td(2,3), n);
}
else if (ind == 3) {
        at=WMRA_Linear(0,a,n);
        xt=WMRA_Linear(Ti(0,3), Td(0,3), n);
        yt=WMRA_Linear(Ti(1,3), Td(1,3), n);
        zt=WMRA_Linear(Ti(2,3), Td(2,3), n);
}
else {
        at=WMRA_BPolynomial(0,a,n);
        xt=WMRA_BPolynomial(Ti(0,3), Td(0,3), n);
        yt=WMRA_BPolynomial(Ti(1,3), Td(1,3), n);

zt=WMRA_BPolynomial(Ti(2,3), Td(2,3), n);
}

Tt = new float**[n];
for (i = 0; i < n; ++i) {
        Tt[i] = new float*[4];
        for (j = 0; j < 4; ++j){
                Tt[i][j] = new float[4];
        }
}
for ( i=0 ; i < 4 ; i++ ) {
        for ( j = 0 ; j < 4 ; j++ ) {
                Tt[0][i][j]=Ti(i,j);
        }
}

for (i=1; i<n; i++){
        // Single-angle Change:
        float da;
        da=at(i,0)-at(0,0);
        s=sin(da);
        c=cos(da);
        v=1-c;
        // Rotation and Position Change:

        Matrix dR(3,3);
        dR(0,0)=pow(kx,2)*v+c;
```

Appendix A. (Continued)

```
                dR(0,1)=kx*ky*v-kz*s;
                dR(0,2)=kx*kz*v+ky*s;
                dR(1,0)=kx*ky*v+kz*s;
                dR(1,1)=pow(ky,2)*v+c;
                dR(1,2)=ky*kz*v-kx*s;
                dR(2,0)=kx*kz*v-ky*s;
                dR(2,1)=ky*kz*v+kx*s;
                dR(2,2)=pow(kz,2)*v+c;

                //Finding the trajectory points along the trajectory line:
                Matrix Tti1(3,3),Tti(4,4);
                Tti1 = Titemp * dR;
                Tti.Unit(4);
                for ( m=0 ; m < 3 ; m++ ) {
                        for ( j = 0 ; j < 3 ; j++ ) {
                                Tti(m,j)=Tti1(m,j);
                        }
                }
                Tti(0,3)=xt(i,0);
                Tti(1,3)=yt(i,0);
                Tti(2,3)=zt(i,0);

                for ( m=0 ; m < 4 ; m++ ) {
                        for ( j = 0 ; j < 4 ; j++ ) {
                                Tt[i][m][j]=Tti(m,j);
                        }
                }
        }

                return Tt;
        }


/* This function gives the homogeneous transformation matrix, given the X, Y, Z cartesian
translation values.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "vector.h"
#include "Transl.h"
using namespace std;
using namespace math;

Matrix WMRA_transl(float x, float y, float z){

        Matrix T(4,4);

        T.Unit(4);
        T(0,3)= x;
        T(1,3)= y;
        T(2,3)= z;

        return T;
}
```

Appendix A. (Continued)

```
 * vector.cpp
 *
 *  Created on: Aug 28, 2009
 *%%%%% Developed by: Garret Pence%%%%%%%%%%
 */

#include <iostream>
#include "vector.h"

using namespace std;

vector::vector()
{
        x = 0;
        y = 0;
        z = 0;
}

vector::vector(double m, double n, double o)
{
        x = m;
        y = n;
        z = o;
}

vector crossProduct(vector a, vector b)
{
        double i, j, k;
        vector c;

        i = (a.y * b.z - a.z * b.y);
        j = (a.x * b.z - a.z * b.x) * -1;
        k = (a.x * b.y - a.y * b.x);

        c.x = i;
        c.y = j;
        c.z = k;

        return c;
        }

/* 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.
Dimentions are as supplies, angles are in radians.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "vector.h"
#include "WCD.h"
#include "w2T.h"
using namespace std;
using namespace math;

Matrix WMRA_w2T(int ind, Matrix Tp, Matrix q){

        Matrix T(4,4);
```

Appendix A. (Continued)

```
        Matrix L(1,1);
        // 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):
        int i;
        if (ind==0){
                for (i=1; i<4; i++){
                        L(0,i) = 0;
                }
        }

        // Defining the inverse of Transformation Matrix between the wheelchair center and
the WMRA's base:
        Matrix Twa(4,4), Twainv(4,4);
        Twa.Unit(4);
        for (i=0; i<3; i++){
                Twa(i,3) = L(0,i+1);
        }

        // The previous transformation matrix from the ground to the wheelchair center:
        Twainv = !Twa;
        Tp = Tp * Twainv;

        // Defining the Transformation Matrix between the ground and the wheelchair center
and WMRA's base:
        if (abs(q(1,0))<=EPS){          // Streight line motion.
                for (i=0; i<2; i++){
                        Tp(i,3)= Tp(i,3) + q(0,0)*Tp(i,0);
                }
                T = Tp * Twa;
        }
        else {
                float po, p, r;
                po=atan2(Tp(1,0),Tp(0,0));
                p=q(1,0);
                r=q(0,0)/p-L(0,0)/2;
                Matrix Tgw(4,4);
                Tgw.Unit(4);
                Tgw(0,0)= cos(po+p);
                Tgw(0,1)= -sin(po+p);
                Tgw(0,3)= Tp(0,3)+sin(PI/2+po+p/2)*(r+L(0,0)/2)*sin(p)/cos(p/2);
                Tgw(1,0)= sin(po+p);
                Tgw(1,1)= cos(po+p);
                Tgw(1,3)= Tp(1,3)-cos(PI/2+po+p/2)*(r+L(0,0)/2)*sin(p)/cos(p/2);
                Tgw(2,3)= Tp(2,3);

                T= Tgw * Twa;
        }

        return T;
}
```

Appendix A. (Continued)

```
/* 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 %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% Modified By:Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

#include "matrix.h"
#include "WCD.h"
using namespace std;
using namespace math;

Matrix WMRA_WCD(){

        Matrix L(1,5);
        int Ltemp[5] = {500,400,230,200,170};
        int i;
        for (i=0; i < 5; i++){
                L(0,i) = Ltemp[i];
        }

        // All dimentions are in millimeters.
        //L(0,0)=560;  // Distance between the two driving wheels.
        //L(0,1)=440;  // Horizontal distance between the wheels axix of rotation and the
arm mounting position (along x).
        //L(0,2)=230;  // Horizontal distance between the middle point between the two
driving wheels and the arm mounting position (along y).
        //L(0,3)=182;  // Vertical distance between the wheels axix of rotation and the
arm mounting position (along z).
        //L(0,4)=168;  // Radius of the driving wheels.

        return L;
}
```

Appendix B. C++ Touch-Screen GUI Program

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Punya A. Basnayaka & Ana Catalina Torres%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#pragma once

double VAR_DX[7]={0.0};

double varscreenopn;

using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;


namespace Touch {

	/// <summary>
	/// Summary for Form1
	///
	/// WARNING: If you change the name of this class, you will need to change the
	///          'Resource File Name' property for the managed resource compiler tool
	///          associated with all .resx files this class depends on.  Otherwise,
	///          the designers will not be able to interact properly with localized
	///          resources associated with this form.
	/// </summary>
	public ref class Form1 : public System::Windows::Forms::Form
	{
	public:
		Form1(void)
		{
			InitializeComponent();
			//
			//TODO: Add the constructor code here
			//
		}

	protected:
		/// <summary>
		/// Clean up any resources being used.
		/// </summary>
		~Form1()
		{
			if (components)
			{
				delete components;
			}
		}

	protected:

	private: System::Windows::Forms::GroupBox^  groupBox3;
	private: System::Windows::Forms::Button^  button3;

	private: System::Windows::Forms::GroupBox^  groupBox2;
	private: System::Windows::Forms::GroupBox^  groupBox1;
	private: System::Windows::Forms::CheckBox^  checkBox1;

	private: System::Windows::Forms::Button^  button4;
	private: System::Windows::Forms::Button^  button2;
	private: System::Windows::Forms::Button^  button1;
```

Appendix B. (Continued)

```
private: System::Windows::Forms::CheckBox^  checkBox6;
        private: System::Windows::Forms::CheckBox^  checkBox5;
        private: System::Windows::Forms::CheckBox^  checkBox4;
        private: System::Windows::Forms::CheckBox^  checkBox3;
        private: System::Windows::Forms::CheckBox^  checkBox2;
        private: System::Windows::Forms::CheckBox^  checkBox14;
        private: System::Windows::Forms::CheckBox^  checkBox13;
        private: System::Windows::Forms::CheckBox^  checkBox12;
        private: System::Windows::Forms::CheckBox^  checkBox11;
        private: System::Windows::Forms::CheckBox^  checkBox10;
        private: System::Windows::Forms::CheckBox^  checkBox9;
        private: System::Windows::Forms::CheckBox^  checkBox8;
        private: System::Windows::Forms::CheckBox^  checkBox7;

        private:
                /// <summary>
                /// Required designer variable.
                /// </summary>
                System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
                /// <summary>
                /// Required method for Designer support - do not modify
                /// the contents of this method with the code editor.
                /// </summary>
                void InitializeComponent(void)
                {
                        System::ComponentModel::ComponentResourceManager^  resources =
(gcnew System::ComponentModel::ComponentResourceManager(Form1::typeid));
                        this->groupBox3 = (gcnew System::Windows::Forms::GroupBox());
                        this->checkBox14 = (gcnew System::Windows::Forms::CheckBox());
                        this->checkBox13 = (gcnew System::Windows::Forms::CheckBox());
                        this->button3 = (gcnew System::Windows::Forms::Button());
                        this->groupBox2 = (gcnew System::Windows::Forms::GroupBox());
                        this->checkBox12 = (gcnew System::Windows::Forms::CheckBox());
                        this->checkBox11 = (gcnew System::Windows::Forms::CheckBox());
                        this->checkBox10 = (gcnew System::Windows::Forms::CheckBox());
                        this->checkBox9 = (gcnew System::Windows::Forms::CheckBox());
                        this->checkBox8 = (gcnew System::Windows::Forms::CheckBox());
                        this->checkBox7 = (gcnew System::Windows::Forms::CheckBox());
                        this->groupBox1 = (gcnew System::Windows::Forms::GroupBox());
                        this->checkBox6 = (gcnew System::Windows::Forms::CheckBox());
                        this->checkBox5 = (gcnew System::Windows::Forms::CheckBox());
                        this->checkBox4 = (gcnew System::Windows::Forms::CheckBox());
                        this->checkBox3 = (gcnew System::Windows::Forms::CheckBox());
                        this->checkBox2 = (gcnew System::Windows::Forms::CheckBox());
                        this->checkBox1 = (gcnew System::Windows::Forms::CheckBox());
                        this->button4 = (gcnew System::Windows::Forms::Button());
                        this->button2 = (gcnew System::Windows::Forms::Button());
                        this->button1 = (gcnew System::Windows::Forms::Button());
                        this->groupBox3->SuspendLayout();
                        this->groupBox2->SuspendLayout();
                        this->groupBox1->SuspendLayout();
                        this->SuspendLayout();
                        //
                        // groupBox3
                        //
                        this->groupBox3->Controls->Add(this->checkBox14);
                        this->groupBox3->Controls->Add(this->checkBox13);
                        this->groupBox3->Controls->Add(this->button3);
                        this->groupBox3->Location = System::Drawing::Point(237, 232);
                        this->groupBox3->Name = L"groupBox3";
                        this->groupBox3->Size = System::Drawing::Size(289, 111);
                        this->groupBox3->TabIndex = 9;
                        this->groupBox3->TabStop = false;
```

Appendix B. (Continued)

```
                    this->checkBox14->Appearance =
                System::Windows::Forms::Appearance::Button;
                    this->checkBox14->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"checkBox14.BackgroundImage")));
                    this->checkBox14->Location = System::Drawing::Point(201, 22);
                    this->checkBox14->Name = L"checkBox14";
                    this->checkBox14->Size = System::Drawing::Size(68, 69);
                    this->checkBox14->TabIndex = 1;
                    this->checkBox14->UseVisualStyleBackColor = true;
                    this->checkBox14->CheckedChanged += gcnew
System::EventHandler(this, &Form1::checkBox14_CheckedChanged);
                    //
                    // checkBox13
                    //
                    this->checkBox13->Appearance =
System::Windows::Forms::Appearance::Button;
                    this->checkBox13->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"checkBox13.BackgroundImage")));
                    this->checkBox13->Location = System::Drawing::Point(111, 22);
                    this->checkBox13->Name = L"checkBox13";
                    this->checkBox13->Size = System::Drawing::Size(68, 69);
                    this->checkBox13->TabIndex = 1;
                    this->checkBox13->UseVisualStyleBackColor = true;
                    this->checkBox13->CheckedChanged += gcnew
System::EventHandler(this, &Form1::checkBox13_CheckedChanged);
                    //
                    // button3
                    //
                    this->button3->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"button3.BackgroundImage")));
                    this->button3->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Zoom;
                    this->button3->Location = System::Drawing::Point(22, 22);
                    this->button3->Name = L"button3";
                    this->button3->Size = System::Drawing::Size(68, 69);
                    this->button3->TabIndex = 0;
                    this->button3->UseVisualStyleBackColor = true;
                    this->button3->Click += gcnew System::EventHandler(this,
&Form1::button3_Click);
                    //
                    // groupBox2
                    //
                    this->groupBox2->Controls->Add(this->checkBox12);
                    this->groupBox2->Controls->Add(this->checkBox11);
                    this->groupBox2->Controls->Add(this->checkBox10);
                    this->groupBox2->Controls->Add(this->checkBox9);
                    this->groupBox2->Controls->Add(this->checkBox8);
                    this->groupBox2->Controls->Add(this->checkBox7);
                    this->groupBox2->Location = System::Drawing::Point(237, 13);
                    this->groupBox2->Name = L"groupBox2";
                    this->groupBox2->Size = System::Drawing::Size(289, 213);
                    this->groupBox2->TabIndex = 8;
                    this->groupBox2->TabStop = false;
                    this->groupBox2->Text = L"Gripper Orientation";
                    //
                    // checkBox12
                    //
                    this->checkBox12->Appearance =
System::Windows::Forms::Appearance::Button;
                    this->checkBox12->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"checkBox12.BackgroundImage")));
```

Appendix B. (Continued)

```
                        this->checkBox12->Location = System::Drawing::Point(201, 127);
                        this->checkBox12->Name = L"checkBox12";
                        this->checkBox12->Size = System::Drawing::Size(68, 69);
                        this->checkBox12->TabIndex = 1;
                        this->checkBox12->UseVisualStyleBackColor = true;
                        this->checkBox12->CheckedChanged += gcnew
System::EventHandler(this, &Form1::checkBox12_CheckedChanged);
                        //
                        // checkBox11
                        //
                        this->checkBox11->Appearance =
System::Windows::Forms::Appearance::Button;
                        this->checkBox11->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"checkBox11.BackgroundImage")));
                        this->checkBox11->Location = System::Drawing::Point(111, 127);
                        this->checkBox11->Name = L"checkBox11";
                        this->checkBox11->Size = System::Drawing::Size(68, 69);
                        this->checkBox11->TabIndex = 1;
                        this->checkBox11->UseVisualStyleBackColor = true;
                        this->checkBox11->CheckedChanged += gcnew
System::EventHandler(this, &Form1::checkBox11_CheckedChanged);
                        //
                        // checkBox10
                        //
                        this->checkBox10->Appearance =
System::Windows::Forms::Appearance::Button;
                        this->checkBox10->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"checkBox10.BackgroundImage")));
                        this->checkBox10->Location = System::Drawing::Point(22, 127);
                        this->checkBox10->Name = L"checkBox10";
                        this->checkBox10->Size = System::Drawing::Size(68, 69);
                        this->checkBox10->TabIndex = 1;
                        this->checkBox10->UseVisualStyleBackColor = true;
                        this->checkBox10->CheckedChanged += gcnew
System::EventHandler(this, &Form1::checkBox10_CheckedChanged);
                        //
                        // checkBox9
                        //
                        this->checkBox9->Appearance =
System::Windows::Forms::Appearance::Button;
                        this->checkBox9->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"checkBox9.BackgroundImage")));
                        this->checkBox9->Location = System::Drawing::Point(201, 19);
                        this->checkBox9->Name = L"checkBox9";
                        this->checkBox9->Size = System::Drawing::Size(68, 69);
                        this->checkBox9->TabIndex = 1;
                        this->checkBox9->UseVisualStyleBackColor = true;
                        this->checkBox9->CheckedChanged += gcnew System::EventHandler(this,
&Form1::checkBox9_CheckedChanged);
                        //
                        // checkBox8
                        //
                        this->checkBox8->Appearance =
System::Windows::Forms::Appearance::Button;
                        this->checkBox8->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"checkBox8.BackgroundImage")));
                        this->checkBox8->Location = System::Drawing::Point(111, 19);
                        this->checkBox8->Name = L"checkBox8";
                        this->checkBox8->Size = System::Drawing::Size(68, 69);
                        this->checkBox8->TabIndex = 1;
                        this->checkBox8->UseVisualStyleBackColor = true;
```

Appendix B. (Continued)

```
                    this->checkBox8->CheckedChanged += gcnew System::EventHandler(this,
            &Form1::checkBox8_CheckedChanged);
                    //
                    // checkBox7

                    //
                    this->checkBox7->Appearance =
System::Windows::Forms::Appearance::Button;
                    this->checkBox7->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"checkBox7.BackgroundImage")));
                    this->checkBox7->Location = System::Drawing::Point(22, 19);
                    this->checkBox7->Name = L"checkBox7";
                    this->checkBox7->Size = System::Drawing::Size(68, 69);
                    this->checkBox7->TabIndex = 1;
                    this->checkBox7->UseVisualStyleBackColor = true;
                    this->checkBox7->CheckedChanged += gcnew System::EventHandler(this,
&Form1::checkBox7_CheckedChanged);
                    //
                    // groupBox1
                    //
                    this->groupBox1->Controls->Add(this->checkBox6);
                    this->groupBox1->Controls->Add(this->checkBox5);
                    this->groupBox1->Controls->Add(this->checkBox4);
                    this->groupBox1->Controls->Add(this->checkBox3);
                    this->groupBox1->Controls->Add(this->checkBox2);
                    this->groupBox1->Controls->Add(this->checkBox1);
                    this->groupBox1->Location = System::Drawing::Point(12, 13);
                    this->groupBox1->Name = L"groupBox1";
                    this->groupBox1->Size = System::Drawing::Size(206, 330);
                    this->groupBox1->TabIndex = 7;
                    this->groupBox1->TabStop = false;
                    this->groupBox1->Text = L"Gripper  Possition";
                    //
                    // checkBox6
                    //
                    this->checkBox6->Appearance =
System::Windows::Forms::Appearance::Button;
                    this->checkBox6->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"checkBox6.BackgroundImage")));
                    this->checkBox6->Location = System::Drawing::Point(114, 241);
                    this->checkBox6->Name = L"checkBox6";
                    this->checkBox6->Size = System::Drawing::Size(68, 69);
                    this->checkBox6->TabIndex = 1;
                    this->checkBox6->UseVisualStyleBackColor = true;
                    this->checkBox6->CheckedChanged += gcnew System::EventHandler(this,
&Form1::checkBox6_CheckedChanged);
                    //
                    // checkBox5
                    //
                    this->checkBox5->Appearance =
System::Windows::Forms::Appearance::Button;
                    this->checkBox5->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"checkBox5.BackgroundImage")));
                    this->checkBox5->Location = System::Drawing::Point(25, 241);
                    this->checkBox5->Name = L"checkBox5";
                    this->checkBox5->Size = System::Drawing::Size(68, 69);
                    this->checkBox5->TabIndex = 1;
                    this->checkBox5->UseVisualStyleBackColor = true;
                    this->checkBox5->CheckedChanged += gcnew System::EventHandler(this,
&Form1::checkBox5_CheckedChanged);
                    //
                    // checkBox4
```

Appendix B. (Continued)

```
                        this->checkBox4->Appearance =
System::Windows::Forms::Appearance::Button;
                        this->checkBox4->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"checkBox4.BackgroundImage")));
                        this->checkBox4->Location = System::Drawing::Point(114, 127);

                        this->checkBox4->Name = L"checkBox4";
                        this->checkBox4->Size = System::Drawing::Size(68, 69);
                        this->checkBox4->TabIndex = 1;
                        this->checkBox4->UseVisualStyleBackColor = true;
                        this->checkBox4->CheckedChanged += gcnew System::EventHandler(this,
&Form1::checkBox4_CheckedChanged);
                        //
                        // checkBox3
                        //
                        this->checkBox3->Appearance =
System::Windows::Forms::Appearance::Button;
                        this->checkBox3->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"checkBox3.BackgroundImage")));
                        this->checkBox3->Location = System::Drawing::Point(25, 127);
                        this->checkBox3->Name = L"checkBox3";
                        this->checkBox3->Size = System::Drawing::Size(68, 69);
                        this->checkBox3->TabIndex = 1;
                        this->checkBox3->UseVisualStyleBackColor = true;
                        this->checkBox3->CheckedChanged += gcnew System::EventHandler(this,
&Form1::checkBox3_CheckedChanged);
                        //
                        // checkBox2
                        //
                        this->checkBox2->Appearance =
System::Windows::Forms::Appearance::Button;
                        this->checkBox2->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"checkBox2.BackgroundImage")));
                        this->checkBox2->Location = System::Drawing::Point(114, 19);
                        this->checkBox2->Name = L"checkBox2";
                        this->checkBox2->Size = System::Drawing::Size(68, 69);
                        this->checkBox2->TabIndex = 1;
                        this->checkBox2->UseVisualStyleBackColor = true;
                        this->checkBox2->CheckedChanged += gcnew System::EventHandler(this,
&Form1::checkBox2_CheckedChanged);
                        //
                        // checkBox1
                        //
                        this->checkBox1->Appearance =
System::Windows::Forms::Appearance::Button;
                        this->checkBox1->BackgroundImage =
(cli::safe_cast<System::Drawing::Image^  >(resources-
>GetObject(L"checkBox1.BackgroundImage")));
                        this->checkBox1->Location = System::Drawing::Point(25, 19);
                        this->checkBox1->Name = L"checkBox1";
                        this->checkBox1->Size = System::Drawing::Size(68, 69);
                        this->checkBox1->TabIndex = 1;
                        this->checkBox1->UseVisualStyleBackColor = true;
                        this->checkBox1->CheckedChanged += gcnew System::EventHandler(this,
&Form1::checkBox1_CheckedChanged);
                        //
                        // button4
                        //
                        this->button4->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 10, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
```

Appendix B. (Continued)

```
this->button4->Location = System::Drawing::Point(351, 357);
                        this->button4->Name = L"button4";
                        this->button4->Size = System::Drawing::Size(75, 30);
                        this->button4->TabIndex = 5;
                        this->button4->Text = L"Run";
                        this->button4->UseVisualStyleBackColor = true;
                        this->button4->Click += gcnew System::EventHandler(this,
&Form1::button4_Click);
                        //
                        // button2

                        //
                        this->button2->Enabled = false;
                        this->button2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 10, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
                        this->button2->Location = System::Drawing::Point(12, 357);
                        this->button2->Name = L"button2";
                        this->button2->Size = System::Drawing::Size(78, 30);
                        this->button2->TabIndex = 6;
                        this->button2->Text = L"Back";
                        this->button2->UseVisualStyleBackColor = true;
                        //
                        // button1
                        //
                        this->button1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 10, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
                        this->button1->Location = System::Drawing::Point(451, 357);
                        this->button1->Name = L"button1";
                        this->button1->Size = System::Drawing::Size(75, 30);
                        this->button1->TabIndex = 5;
                        this->button1->Text = L"Exit";
                        this->button1->UseVisualStyleBackColor = true;
                        this->button1->Click += gcnew System::EventHandler(this,
&Form1::button1_Click);
                        //
                        // Form1
                        //
                        this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                        this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                        this->ClientSize = System::Drawing::Size(539, 401);
                        this->ControlBox = false;
                        this->Controls->Add(this->groupBox3);
                        this->Controls->Add(this->groupBox2);
                        this->Controls->Add(this->groupBox1);
                        this->Controls->Add(this->button4);
                        this->Controls->Add(this->button2);
                        this->Controls->Add(this->button1);
                        this->MaximizeBox = false;
                        this->Name = L"Form1";
                        this->Text = L"WMRA Screen";
                        this->Load += gcnew System::EventHandler(this, &Form1::Form1_Load);
                        this->groupBox3->ResumeLayout(false);
                        this->groupBox2->ResumeLayout(false);
                        this->groupBox1->ResumeLayout(false);
                        this->ResumeLayout(false);

                }
#pragma endregion
        private: System::Void checkBox1_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                                switch(checkBox1->CheckState) {
                case CheckState::Checked:
                        // Code for checked state.
```

Appendix B. (Continued)

```
                        this->checkBox1->FlatStyle = FlatStyle::Flat;
                        this->checkBox1->FlatAppearance->BorderSize=2;
                        this->checkBox1->FlatAppearance->BorderColor=Color::Red;
                        this->checkBox5->Enabled=false;
                        VAR_DX[2]=1;
                        break;
         case CheckState::Unchecked:
                        // Code for unchecked state.
                        this->checkBox1->FlatStyle = FlatStyle::Standard;

                        this->checkBox5->Enabled=true;
                        VAR_DX[2]=0;
            break;


                        case CheckState::Indeterminate:
                        // Code for indeterminate state.
                        VAR_DX[2]=0;
                        this->checkBox1->FlatStyle = FlatStyle::Standard;
                       break;
      }
                        }
private: System::Void checkBox2_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                        switch(checkBox2->CheckState) {
         case CheckState::Checked:
                        // Code for checked state.
                        this->checkBox2->FlatStyle = FlatStyle::Flat;
                        this->checkBox2->FlatAppearance->BorderSize=2;
                        this->checkBox2->FlatAppearance->BorderColor=Color::Red;
                        this->checkBox6->Enabled=false;
                        VAR_DX[0]=1;
            break;
         case CheckState::Unchecked:
                        // Code for unchecked state.
                        this->checkBox2->FlatStyle = FlatStyle::Standard;

                        this->checkBox6->Enabled=true;
                        VAR_DX[0]=0;
            break;
         case CheckState::Indeterminate:
                        // Code for indeterminate state.
                        VAR_DX[0]=0;
                        this->checkBox2->FlatStyle = FlatStyle::Standard;
            break;
      }
                }
private: System::Void checkBox3_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                        switch(checkBox3->CheckState) {
         case CheckState::Checked:
                        // Code for checked state.
                        this->checkBox3->FlatStyle = FlatStyle::Flat;
                        this->checkBox3->FlatAppearance->BorderSize=2;
                        this->checkBox3->FlatAppearance->BorderColor=Color::Red;
                        this->checkBox4->Enabled=false;
                        VAR_DX[1]=1;
            break;
         case CheckState::Unchecked:
                        // Code for unchecked state.
                        this->checkBox3->FlatStyle = FlatStyle::Standard;

                        this->checkBox4->Enabled=true;
                        VAR_DX[1]=0;
            break;
```

Appendix B. (Continued)

```
                        case CheckState::Indeterminate:
                        // Code for indeterminate state.
                        VAR_DX[1]=0;
                        this->checkBox3->FlatStyle = FlatStyle::Standard;
                       break;
      }
                }
private: System::Void checkBox4_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                        switch(checkBox4->CheckState) {
         case CheckState::Checked:
                        // Code for checked state.
                        this->checkBox4->FlatStyle = FlatStyle::Flat;
                        this->checkBox4->FlatAppearance->BorderSize=2;
                        this->checkBox4->FlatAppearance->BorderColor=Color::Red;
                        this->checkBox3->Enabled=false;
                        VAR_DX[1]=-1;

               break;
         case CheckState::Unchecked:
                        // Code for unchecked state.
                        this->checkBox4->FlatStyle = FlatStyle::Standard;

                        this->checkBox3->Enabled=true;
                        VAR_DX[1]=0;
            break;
         case CheckState::Indeterminate:
                        // Code for indeterminate state.
                        VAR_DX[1]=0;
                        this->checkBox4->FlatStyle = FlatStyle::Standard;
                       break;
      }
                }
private: System::Void checkBox5_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                        switch(checkBox5->CheckState) {
                case CheckState::Checked:
                        // Code for checked state.
                        this->checkBox5->FlatStyle = FlatStyle::Flat;
                        this->checkBox5->FlatAppearance->BorderSize=2;
                        this->checkBox5->FlatAppearance->BorderColor=Color::Red;
                        this->checkBox1->Enabled=false;
                        VAR_DX[2]=-1;
            break;
         case CheckState::Unchecked:
                        // Code for unchecked state.
                        this->checkBox5->FlatStyle = FlatStyle::Standard;

                        this->checkBox1->Enabled=true;
                        VAR_DX[2]=0;
            break;
         case CheckState::Indeterminate:
                        // Code for indeterminate state.
                        VAR_DX[2]=0;
                        this->checkBox5->FlatStyle = FlatStyle::Standard;
            break;
      }
                }
private: System::Void checkBox6_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                        switch(checkBox6->CheckState) {
         case CheckState::Checked:
                        // Code for checked state.
                        this->checkBox6->FlatStyle = FlatStyle::Flat;
                        this->checkBox6->FlatAppearance->BorderSize=2;
```

Appendix B. (Continued)

```
                this->checkBox2->Enabled=false;
                        VAR_DX[0]=-1;
            break;
         case CheckState::Unchecked:
                        // Code for unchecked state.
                        this->checkBox6->FlatStyle = FlatStyle::Standard;

                        this->checkBox2->Enabled=true;
                        VAR_DX[0]=0;
            break;
         case CheckState::Indeterminate:
                        // Code for indeterminate state.
                        VAR_DX[0]=0;
                        this->checkBox6->FlatStyle = FlatStyle::Standard;
            break;
      }
                }
private: System::Void checkBox7_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {


                        switch(checkBox7->CheckState) {
         case CheckState::Checked:
                        // Code for checked state.
                        this->checkBox7->FlatStyle = FlatStyle::Flat;
                        this->checkBox7->FlatAppearance->BorderSize=2;
                        this->checkBox7->FlatAppearance->BorderColor=Color::Red;
                        this->checkBox9->Enabled=false;
                        VAR_DX[5]=0.003;
            break;
         case CheckState::Unchecked:
                        // Code for unchecked state.
                        this->checkBox7->FlatStyle = FlatStyle::Standard;

                        this->checkBox9->Enabled=true;
                        VAR_DX[5]=0;
            break;
         case CheckState::Indeterminate:
                        // Code for indeterminate state.
                        VAR_DX[5]=0;
                        this->checkBox7->FlatStyle = FlatStyle::Standard;
            break;
      }
                }
private: System::Void checkBox8_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                        switch(checkBox8->CheckState) {
         case CheckState::Checked:
                        // Code for checked state.
                        this->checkBox8->FlatStyle = FlatStyle::Flat;
                        this->checkBox8->FlatAppearance->BorderSize=2;
                        this->checkBox8->FlatAppearance->BorderColor=Color::Red;
                        this->checkBox11->Enabled=false;
                        VAR_DX[4]=0.003;
            break;
         case CheckState::Unchecked:
                        // Code for unchecked state.
                        this->checkBox8->FlatStyle = FlatStyle::Standard;

                        this->checkBox11->Enabled=true;
                        VAR_DX[4]=0;
            break;
         case CheckState::Indeterminate:
                        // Code for indeterminate state.
                        VAR_DX[4]=0;
```

Appendix B. (Continued)

```
                        this->checkBox8->FlatStyle = FlatStyle::Standard;
            break;
      }
                  }
private: System::Void checkBox9_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                        switch(checkBox9->CheckState) {
         case CheckState::Checked:
                        // Code for checked state.
                        this->checkBox9->FlatStyle = FlatStyle::Flat;
                        this->checkBox9->FlatAppearance->BorderSize=2;
                        this->checkBox9->FlatAppearance->BorderColor=Color::Red;
                        this->checkBox7->Enabled=false;
                        VAR_DX[5]=-0.003;
            break;
         case CheckState::Unchecked:
                        // Code for unchecked state.
                        this->checkBox9->FlatStyle = FlatStyle::Standard;

                        this->checkBox7->Enabled=true;
                        VAR_DX[5]=0;
            break;
         case CheckState::Indeterminate:
               // Code for indeterminate state.
                        VAR_DX[5]=0;
                        this->checkBox9->FlatStyle = FlatStyle::Standard;
            break;
      }
                  }
private: System::Void checkBox10_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                        switch(checkBox10->CheckState) {
         case CheckState::Checked:
                        // Code for checked state.
                        this->checkBox10->FlatStyle = FlatStyle::Flat;
                        this->checkBox10->FlatAppearance->BorderSize=2;
                        this->checkBox10->FlatAppearance->BorderColor=Color::Red;
                        this->checkBox12->Enabled=false;
                        VAR_DX[3]=-0.003;
            break;
         case CheckState::Unchecked:
                        // Code for unchecked state.
                        this->checkBox10->FlatStyle = FlatStyle::Standard;

                        this->checkBox12->Enabled=true;
                        VAR_DX[3]=0;
            break;
         case CheckState::Indeterminate:
                        // Code for indeterminate state.
                        VAR_DX[3]=0;
                        this->checkBox10->FlatStyle = FlatStyle::Standard;
            break;
      }
                  }
private: System::Void checkBox11_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                        switch(checkBox11->CheckState) {
         case CheckState::Checked:
                        // Code for checked state.
                        this->checkBox11->FlatStyle = FlatStyle::Flat;
                        this->checkBox11->FlatAppearance->BorderSize=2;
                        this->checkBox11->FlatAppearance->BorderColor=Color::Red;
                        this->checkBox8->Enabled=false;
                        VAR_DX[4]=-0.003;
            break;
```

Appendix B. (Continued)

```
          case CheckState::Unchecked:
                        // Code for unchecked state.
                        this->checkBox11->FlatStyle = FlatStyle::Standard;

                        this->checkBox8->Enabled=true;
                        VAR_DX[4]=0;
             break;
          case CheckState::Indeterminate:
                        // Code for indeterminate state.
                        VAR_DX[4]=0;
                        this->checkBox11->FlatStyle = FlatStyle::Standard;
             break;
      }
                }
private: System::Void checkBox12_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                        switch(checkBox12->CheckState) {
          case CheckState::Checked:
                        // Code for checked state.
                        this->checkBox12->FlatStyle = FlatStyle::Flat;
                        this->checkBox12->FlatAppearance->BorderSize=2;
                        this->checkBox12->FlatAppearance->BorderColor=Color::Red;
                        this->checkBox10->Enabled=false;
                        VAR_DX[3]=0.003;
             break;


  case CheckState::Unchecked:
                        // Code for unchecked state.
                        this->checkBox12->FlatStyle = FlatStyle::Standard;

                        this->checkBox10->Enabled=true;
                        VAR_DX[3]=0;
             break;
          case CheckState::Indeterminate:
                        // Code for indeterminate state.
                        VAR_DX[3]=0;
                        this->checkBox12->FlatStyle = FlatStyle::Standard;
             break;
      }
                }
private: System::Void button3_Click(System::Object^  sender, System::EventArgs^  e) {
                        VAR_DX[0]=0;
                        VAR_DX[1]=0;
                        VAR_DX[2]=0;
                        VAR_DX[3]=0;
                        VAR_DX[4]=0;
                        VAR_DX[5]=0;
                        VAR_DX[6]=0;
                        varscreenopn=1;

                        this->checkBox1->Enabled=true;
                        this->checkBox2->Enabled=true;
                        this->checkBox3->Enabled=true;
                        this->checkBox4->Enabled=true;
                        this->checkBox5->Enabled=true;
                        this->checkBox6->Enabled=true;
                        this->checkBox7->Enabled=true;
                        this->checkBox8->Enabled=true;
                        this->checkBox9->Enabled=true;
                        this->checkBox10->Enabled=true;
                        this->checkBox11->Enabled=true;
                        this->checkBox12->Enabled=true;
                        this->checkBox13->Enabled=true;
```

Appendix B. (Continued)

```
                        this->checkBox14->Enabled=true;
                        this->checkBox1->Checked = false;
                        this->checkBox2->Checked = false;
                        this->checkBox3->Checked = false;
                        this->checkBox4->Checked = false;
                        this->checkBox5->Checked = false;
                        this->checkBox6->Checked = false;
                        this->checkBox7->Checked = false;
                        this->checkBox8->Checked = false;
                        this->checkBox9->Checked = false;
                        this->checkBox10->Checked = false;
                        this->checkBox11->Checked = false;
                        this->checkBox12->Checked = false;
                        this->checkBox13->Checked = false;
                        this->checkBox14->Checked = false;
                        FILE * fid;
                        fid = fopen("outputtouch.txt","w");

fprintf(fid,"%1.f\t%1.f\t%1.f\t%1.3f\t%1.3f\t%1.3f\t%1.f\t%1.f\t\n",VAR_DX[0],VAR_DX[1],V
AR_DX[2],VAR_DX[3],VAR_DX[4],VAR_DX[5],VAR_DX[6],varscreenopn);
                        fclose(fid);
                }
private: System::Void checkBox13_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                         switch(checkBox13->CheckState) {
         case CheckState::Checked:
                        // Code for checked state.
                        this->checkBox13->FlatStyle = FlatStyle::Flat;
                        this->checkBox13->FlatAppearance->BorderSize=2;
                  this->checkBox13->FlatAppearance->BorderColor=Color::Red;
                        this->checkBox14->Enabled=false;
                        VAR_DX[6]=-1;
            break;
         case CheckState::Unchecked:
                        // Code for unchecked state.
                        this->checkBox13->FlatStyle = FlatStyle::Standard;

                        this->checkBox14->Enabled=true;
                        VAR_DX[6]=0;
            break;
         case CheckState::Indeterminate:
                        // Code for indeterminate state.
                        VAR_DX[6]=0;
                        this->checkBox12->FlatStyle = FlatStyle::Standard;
            break;
      }
                }
private: System::Void checkBox14_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
                           switch(checkBox14->CheckState) {
         case CheckState::Checked:
                        // Code for checked state.
                        this->checkBox14->FlatStyle = FlatStyle::Flat;
                        this->checkBox14->FlatAppearance->BorderSize=2;
                        this->checkBox14->FlatAppearance->BorderColor=Color::Red;
                        this->checkBox13->Enabled=false;
                        VAR_DX[6]=1;
            break;
         case CheckState::Unchecked:
                        // Code for unchecked state.
                        this->checkBox14->FlatStyle = FlatStyle::Standard;

                        this->checkBox13->Enabled=true;
                        VAR_DX[6]=0;
```

Appendix B. (Continued)

```
            break;
          case CheckState::Indeterminate:
                        // Code for indeterminate state.
                        VAR_DX[6]=0;
                        this->checkBox14->FlatStyle = FlatStyle::Standard;
             break;
      }
                }
public: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
                        varscreenopn = 0;
                        VAR_DX[0]=0;
                        VAR_DX[1]=0;
                        VAR_DX[2]=0;
                        VAR_DX[3]=0;
                        VAR_DX[4]=0;
                        VAR_DX[5]=0;
                        VAR_DX[6]=0;
                        FILE * fid;
                        fid = fopen("outputtouch.txt","w");

fprintf(fid,"%1.f\t%1.f\t%1.f\t%1.3f\t%1.3f\t%1.3f\t%1.f\t%1.f\t\n",VAR_DX[0],VAR_DX[1],V
AR_DX[2],VAR_DX[3],VAR_DX[4],VAR_DX[5],VAR_DX[6],varscreenopn);
                        fclose(fid);
                        Form1::Close();
                }
public: System::Void button4_Click(System::Object^  sender, System::EventArgs^  e) {
                        varscreenopn = 1;
                        FILE * fid;
                        fid = fopen("outputtouch.txt","w");

fprintf(fid,"%1.f\t%1.f\t%1.f\t%1.3f\t%1.3f\t%1.3f\t%1.f\t%1.f\t\n",VAR_DX[0],VAR_DX[1],V
AR_DX[2],VAR_DX[3],VAR_DX[4],VAR_DX[5],VAR_DX[6],varscreenopn);

fclose(fid);
               //      Form1::Hide();
               }
private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e) {
                         Form1::CenterToScreen();
                         varscreenopn = 1.0;
                         FILE * fid;
                         fid = fopen("outputtouch.txt","w");

fprintf(fid,"%1.f\t%1.f\t%1.f\t%1.3f\t%1.3f\t%1.3f\t%1.f\t%1.f\t\n",VAR_DX[0],VAR_DX[1],V
AR_DX[2],VAR_DX[3],VAR_DX[4],VAR_DX[5],VAR_DX[6],varscreenopn);
                         fclose(fid);

}*/
                }
};
}
```

About the Author

Punya A. Basnayaka has obtained her Bachelor degree in Engineering and graduated with honor from University of Pedadeniya, Sri Lanka in 2007. She was awarded the University Prize (Bartholomeuesz Price) for Engineering Mathematics. Write after her graduation she joined to the Department of Mechanical Engineering, University of Peradeniya, Sri Lanka as a lecturer.

In 2008, she was admitted to University of South Florida, Tampa, Florida for her Master's degree in Mechanical Engineering in the field of Rehabilitation Robotics. Since 2008 Fall she has been working as a Teaching Assistant and Research Assistant at University of South Florida. Punya is a student member of ASME. She completed her degree of Master of Science in Mechanical Engineering, successfully in 2010.