4-1-2010

Development and Testing of a New C-Based Algorithm to Control a 9-Degree-ofFreedom Wheelchair-Mounted-Robotic-Arm System

Ana Catalina Torres Rocco
University of South Florida

Development and Testing of a New C-Based Algorithm to Control a 9-Degree-of-Freedom Wheelchair-Mounted-Robotic-Arm System

by

Ana Catalina Torres Rocco

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.
Redwan Alqasemi, Ph.D.
Susana Lai-Yuen, Ph.D.

Date of Approval:
April 1, 2010

Keywords: rehabilitation robotics, programming environment, control algorithm, GUI, WMRA

Dedication

To all of the people who encouraged me every day; who enjoy my happiness and success. I love you.

Acknowledgments

I would like to express my gratitude to all of those who helped me through this long way. I will always be thankful with my mother for all her love, support, advice and, and for making me who I am today. I thank my sister for believing in me and always making me laugh. My father for his support and for giving me the courage to succeed. Thanks to my aunt Lolita and uncle Fernando for their support and love.

I thank my advisor, Dr. Rajiv Dubey for giving me the opportunity to accomplish this success. I especially would like to thank Dr. Redwan Alqasemi for all of the time, patience and advice he has given me. Dr. Susana Lai-Yuen for her valuable opinion that helped me improve my project.

I would like to acknowledge the people from the Center for Assistive Rehabilitation Engineering at the University of South Florida, especially Dr. Kathryn De Laurentis and Dr. Stephanie Carey who supported me along the way. Thanks go to the members of the Rehabilitation Robotics group, including Karan Khokar, Mario Simoes, John Capille, Michelle Smith, Elijah Klay, Punya Basnayaka, and special thanks to Garret Pence, for their collaboration and for making me laugh in stressful moments. I would like to thank the Mechanical Engineering Department, the College of Engineering, and the University of South Florida for giving me this wonderful opportunity.

Thanks to my friends and people who have made me feel like at home, and helped me out through this process. Finally, I would like to thank Fabian, for never letting me down, and for his loving, unconditional support. Thank you very much.

Table of Contents

List of Tables

List of Figures

Development and Testing of a New C-Based Algorithm to Control a 9 -Degree-of-Freedom Wheelchair-Mounted-Robotic-Arm System

Ana Catalina Torres

ABSTRACT

A Wheelchair-Mounted Robotic Arm (WMRA) was designed to aid people with limited or no upper-limb usage to accomplish activities of daily living (ADLs). The primary objective of this research was to enhance the performance of the WMRA by improving the communication protocols and functions between the hardware and software used for its control.

Previously, the control algorithm of the robotic arm was tested in simulation and in the physical arm. These implementations required a combination of Matlab and C++ language and introduced some software instability under Windows operating system. To improve the performance of the WMRA, the programs for hardware control were separated from the ones intended for simulation. The control algorithm of the arm was rewritten using C++ language to facilitate the communication with the controller boards and to make the system more stable and reliable. As a result, the communication delays were decreased since the interfaces between different programs is no longer needed. Preliminary tests were performed to demonstrate the stability and reliability of the new control algorithm. The overall response of the control implementation was enhanced and

the algorithm routines and optimization procedures achieved the same goals with more efficiency. Accuracy and repeatability tests were performed, and data was collected and analyzed.

Chapter 1: Introduction

1.1. Motivation

The latest data from the US Census Bureau of 2009 [1], reveals that more than 19% of the American community suffer from disability, and around 20% of these people endure limitations to perform activities of daily living (ADLs) [2]. Robotic assistive devices aid people with disabilities to complete tasks. These devices not only augment self-sufficiency, but also confidence and self-esteem [3].

The Wheelchair-Mounted Robotic Arm (WMRA) was designed to provide aid to people with limited or no upper-limb usage enhancing their manipulation capabilities. The WMRA project has been a success, but it still has room for improvement. Some of the weaknesses of the system are due to the lack of a secure and reliable control program. When the WMRA was programmed, several communication problems between hardware and software, and software and software emerged. Alqasemi [4] addressed the problem as a communication issue: since the WMRA uses functions from complex DLL libraries, functions in C++ had to be recreated and compiled into DLL files in a data structure compatible with Matlab, and then these functions have to be called and used in Matlab with the PIC Servo SC controller boards used for controlling the WMRA. The arrangement worked but occasionally the link between Matlab and the DLL library failed. The system also presented undesirable delays due to the combination of the WMRA system control and the simulation control. The use of Matlab's code to control

the physical and the Virtual Reality Simulation simultaneously, slowed down the system, and sometimes froze it.

As mentioned, the main problem was due to the language used to program the control algorithm of the system. The communication between Matlab and the C++ applications created to control the hardware was not solid and, although Matlab is a powerful programming tool, its complexity creates a large and slow program that decreases the system's reliability.

The desire behind this project is to enhance the performance of the WMRA by improving the communication protocols and functions between the hardware and software used for its control. The WMRA's performance would be optimized by rewriting the control code in a language with similar capabilities to the one implemented in the hardware of the system. By improving the WMRA's performance, users could employ it more efficiently and it could be more powerful for future testing and development of new implementations.

1.2. Thesis Objectives

The main objective of this thesis is to enhance the performance of the WMRA by improving its control algorithm and the communication protocols between its programs. Rewriting the code in the same language the hardware uses and discarding the Matlab algorithm from the system's control would eliminate the interplatform interfacing, and make the control program more efficient. Delays concerning communication in the system would no longer be expected. According to Biggs [5], the generic language (C++) meets the needs of research projects such as the WMRA.

Secondary objectives are:

1. Rewrite the control algorithm of the WMRA in C++ language.
2. Redesign the WMRA's GUI in C++ to avoid the linkage between C++ and Matlab.
3. Integrate the SpaceBall interface into the system.
4. Evaluate the stability and performance of the rewritten code.
5. Identify areas for improvement and recommend future modifications for performance enhancement.

1.3. Thesis Outline

A review on rehabilitation robotics and wheelchair-mounted-robotic-arms is provided in Chapter 2. Information about the language and characteristics of the new control code for the WMRA are described in Chapter 3. Chapter 4 presents the new Graphical User Interface and its improvements. Testing results and analysis are discussed in Chapter 5. Chapter 6 and Chapter 7 provide conclusions and future work, respectively.

Chapter 2: Background

2.1. Introduction

The robotics field has been around ever since the early 1960's when industrialization caused the need for robot implementation for manufacturing plants and production facilities [6]. In the last few decades, new applications that go from maintenance to rehabilitation have been emerging. For rehabilitation, the main application over the last 30 years has been robots that aid manipulation capabilities of persons with disabilities.

2.2. Workstation Rehabilitation Robotics

A pioneering work was done in 1969 at Rancho Los amigos Hospital in Downy, California, where they developed the Rancho Arm [7], one of the first attempts at rehabilitation robotics. The Rancho-Arm has 6 degrees of freedom and is controlled by tongue switches.

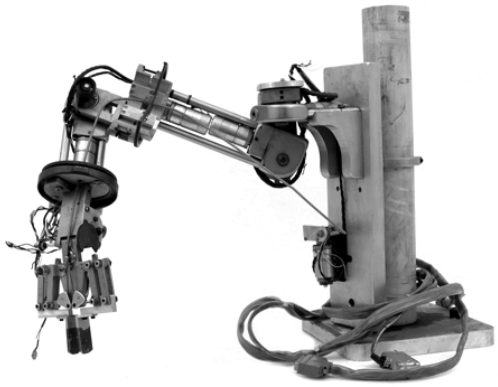

Figure 1. Rancho Arm [7]

The Rancho Arm is an example of what is called a workstation. A workstation is a system used to interact within a known structured environment and it can complete specific tasks. This means that the robotic arm is attached to a fixed base. Other examples of workstations are the Handy-1 [8], the RAID project [9], ProVAR [10] and the design conducted by Gunnar Bolmsjo, et al [11].

Handy-1 is a robotic arm designed to assist in very specific activities of daily living (ADL) within a small workspace around its five-degrees-of-freedom. It initially provided persons with severe disabilities assistance at mealtimes; but along the years, the Handy-1 capabilities have been extended to assisting a person with personal hygiene, eating, drinking, and the application of make-up.

The RAID workstation (Robot for Assisting the Integration of the Disabled) was designed to aid individuals with little or no upper limb motion in office environments. Its robotic arm is mounted onto a linear track in a controlled environment which allows a larger workspace.

The ProVAR system provides a person with disabilities with assistance in a semi-structured office workspace setting [12]. ProVAR's system uses a web-based virtual environment to model the functionality of the manipulator. The actions of the arm and its interactions within its workspace can be seen before any action is taken.

2.3. Mobile Rehabilitation Robotics

In general, workstations aid the manipulation capabilities of the user in a well known environment, and they can process complex task efficiently; but their limited workspace limits their applications as well. This was the main motivation for the development of the mobile systems.

2.3.1. Mobile Base Robots

Mobile systems are capable of assisting individuals with disabilities to execute tasks on a high level of abstraction. These systems include a mobile platform with a mounted manipulator, and various sensors. All sensors as well as the platform and the manipulator itself are connected to a computer system that processes the user commands and controls all peripheral components.

Early versions of these systems are the Mobile Vocational Assistant Robot (MoVAR), MOVAID, and URMAD (Mobile Robotic Unit for the Assistance of the Disabled) [13]. The MoVAR system uses a PUMA-250 robotic arm mounted on an omni-directional mobile platform, and several sensors including a remote viewing camera, and force and gripper proximity sensors.

The MOVAID system, an advanced version of MoVAR, is a modular mobile robotic assistance system that interacts with activity workstations. The modular robotic system features a mobile base with an eight degree of freedom robot arm with low level controller, a gripper, and sensory systems for navigation and obstacle avoidance. URMAD consists of an eight degree of freedom arm mounted on a mobile base and controlled from a static workstation via a radio link.

In general, mobile manipulators have shown great improvements in planning and control in decision level, processing level and execution. In [14] a study on coordinated control for a mobile manipulator was performed. This work implements a coordinated control of the wheeled base and the mounted manipulator. A figure of the system hardware is shown below.

Figure 2. Mobile Manipulator

This robot presents expandability to many different applications. However, combined mobility and manipulation is not implemented. This attains problems with the ease of use of the system since it requires several inputs from the final user. This will eventually limit its applications.

2.3.2. Wheelchair-Mounted Robotic Arms

The wheelchair mounted robotic arm (WMRA) combines the aid of the robotic arm with the mobility of the wheelchair, which provides an extension of the workspace of the system. The most important design consideration when deciding where to mount a robotic arm in a power wheelchair is the safety of the operator [15]. Some considerations such as the weight of the arm and the side in which the arm will be mounted are important when designing such a system. The Weston arm is an early example of a WMRA [16]. It has a vertical actuator that moves the arm base. This design adds only about 10 cm to the chair width.

There are two commercially available WMRAs: the Manus (ARM, Assistive Robotic Manipulator, and iARM, intelligent Assistive Robotic Manipulator), manufactured by Exact Dynamics; and the Raptor, manufactured by Applied Resources.

Figure 3. Weston Arm

The Manus is a six-degree-of-freedom robot arm with servomotors all housed in a cylindrical base, and a linear vertical lift and a gripper which are driven by a series of cables that run throughout the structure of the arm. It can be used as a table top system or can be attached to a wheelchair for WMRA usage. Two main control modes can be used to operate the arm: Cartesian control mode and Joint control mode. The Manus is controlled with a key pad comprised by 16 buttons. Other controllers include a joystick, a mini keypad, and a chin switch. Research is being conducted on the improvement of the human-robot interface. The Manus manipulator arm is programmed similar to industrial robotic manipulators. It has been under development since the 1980s and it entered into production in the early 1990s [17].

The Raptor [18], which is mounted to the right side of a wheelchair, has 4 degrees of freedom and a gripper as the end-effector. The arm joints are controlled with a joystick or a 10-button controller.

The joystick that controls the manipulator arm is located on the opposite to the joystick that controls the steering of the wheelchair. The lack of encoders makes it impossible to control the arm in Cartesian coordinates. This was done to minimize the overall system cost.

Figure 4. Manus-Arm

Figure 5. Raptor Arm

The care-providing robotic system FRIEND (Functional Robot arm with user-frIENdly interface for Disabled people) and FRIEND II [19] consist of a power

wheelchair and a mounted MANUS robotic arm, controlled by a computer, also mounted on the backside of the wheelchair. It integrates an LC display for interaction, a stereo pan-tilt-zoom camera system, and a smart tray. Besides using a joystick or a keyboard, the user can also control the arm through verbal commands using an integrated voice recognition system.

Figure 6. FRIEND Robotic System

FRIEND II was developed to overcome the hardware shortcomings of its predecessor. FRIEND-II has a dexterous seven degrees-of-freedom robotic arm with a humanlike kinematical structure. The arm is mounted on a linear axis that allows it to drive in a specific home position and reduce visibility if not in use.

2.4. USF WMRA

Previous work has shown the design and development of the University of South Florida USF WMRA system combines mobility and manipulation with a single controller [20, 21, 22]. The WMRA consists of a seven degree of freedom robotic arm, a gripper, and a power wheelchair. The WMRA provides three-degrees of kinematic redundancy. A single control structure is used to control the WMRA system, which gives much more

flexibility to the system. The combination of mobility and manipulation expands the workspace that a mobile base attains to a manipulator. The system uses Matlab to control the arm and the chair motion with a single Graphical User Interface (GUI). Other user interfaces include a touch screen, a SpaceBall with 3-D input capabilities and a Brain Computer Interface (BCI) that uses the stimulated P-300 signal.

Recent enhancements of the USF WMRA include improvements of its control structure to include algorithms for optimized task execution, making the WMRA a modular task oriented mobile manipulator [23].

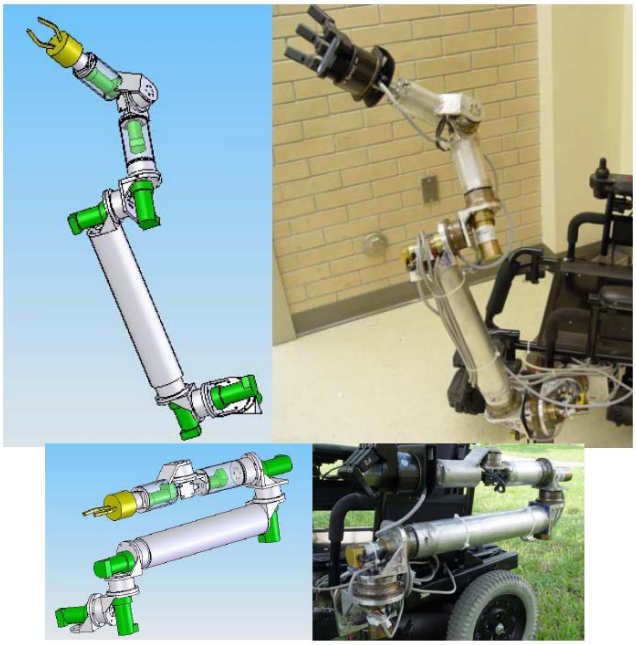

Figure 7. USF WMRA

Chapter 3: Control Algorithm Programming Language

3.1. Language Environments

This thesis refers to two programming environments: C++ and Matlab 2006b from Mathworks. High-level language is user-friendly and more portable across platforms, isolating the semantics of computer architecture from the specification of the program [24]. Low-level language is closer to hardware language (for example machine and assembly language). C++ is programming language that combines both, high-level and low-level language features, enabling communication at hardware level at satisfactory speed, especially in multi-threading environments. Matlab, on the other hand, is a high-level language and interactive environment for performing computationally intensive tasks.

The previous WMRA control algorithm was divided into two programs. One of them, the main control code, was written in Matlab to make use of the language's numerical capabilities. The communication protocols and functions that send the commands and receive the sensory information from the controller boards used C++ with certain DLL library functions [25].

The shortcomings of this configuration came when the program was used to control the physical arm. Due to the complex communication between the functions in Matlab and the PIC Servo SC controller boards used to control the arm, the interplatform interface sometimes failed and the system became unresponsive.

3.2. Programming in C++

C++ is a general purpose programming language [26] that supports data abstraction and object-oriented programming. C++ has been applied to most branches of systems programming including compiler construction, data base management, graphics, image processing, music synthesis, networking, numerical software, programming environments, robotics, simulation, and switching [27].

Coding in C++ has several advantages over other programming languages; its: flexibility, efficiency, availability, portability, and standardization have made it a feature-rich language [28]. C++ provides direct access to a system at hardware level, and control over almost all aspects of software, both suitable benefits for the needs of the WMRA control system.

Among the limitations of the C++ language is the shortage of a numerical computing environment [29]. Unlike Matlab, which allows matrix manipulation, and data and functions plotting, C++ requires the use of libraries that are not built-in to allow the creation and handling of matrices and matrices as variables, and combining them through syntax that closely resembles linear algebra. Graphical plotting within the framework of C++ is possible with the integration of a graphical library or software. Plots can be displayed on a screen or display in a Web browser through a Web server, or saved as an image file in different file formats. As with matrix libraries, some graphical libraries can be used for free for educational purposes.

3.3. New Control Algorithm

Since the communication with the controller boards of the Wheelchair-Mounted Robotic-Arm (WMRA) is made through DLL libraries programmed in C++ as functions,

the new control algorithm was also programmed in C++ to avoid compatibility issues and to make the system more stable and reliable. The new algorithm is divided into the main script and several other functions created for different purposes. The algorithm was designed to be modular so it can be easily modified for future changes.

The main script basically contains the core code of the control algorithm. Three major pieces of code can be identified in the main script. First, the main script includes the libraries used to run certain functions. The following portion of code contains the different options that are displayed on screen so the user can select to control the WMRA according to his/her preferences. When each option is selected, it returns a value that is used on the third and final part of the script, where computations are done and the calculated position is sent to the WMRA. While using teleoperation control (either SpaceBall or touch-screen), the movements are mapped to a six element vector of the main script. When the user decides to teleoperate the physical arm, every time he/she selects a new movement, by clicking the touch-screen or manipulating the SpaceBall, the third part of the main script updates the information that is sent to the WMRA. When the user finishes manipulating the arm by clicking the EXIT button of these interfaces, the main script will stop sending commands to the arm.

In the following sections, the different libraries used for the new control algorithm and their functions are going to be explained.

3.3.1. Libraries

The algorithm uses both, built-in and external libraries. The standard library contains functions, constants, classes, objects and templates that provide basic functionality to perform certain tasks [30]. The headers included in the main script to

declare some of the different elements provided by the standard library are: math, time, limits, iostream, and stdlib. Other non-standard library headers included in the main script are matix and vector.

3.3.1.1. Matrix.h

Since the standard library does not support matrix manipulation, an open source matrix library had to be used. The matrix class defines the majority of the matrix operations as overloaded operato rs or methods. All the instances of matrices are created implicitly by the compiler [30]. This library allows the following:

1. Obtain and set individual matrix elements.
2. Compute the inverse, transpose, and power of a matrix.
3. Read a matrix from input stream.
4. Write a matrix to output stream.
5. Change the size of an existing matrix.
6. Create a matrix of zeros or the identity matrix.
7. Addition, subtraction, scalar multiplication, matrix multiplication and scalar division.
8. Calculate the determinant of a matrix.

The library also permits other functions that are not mentioned here. For further information on the matrix library or the standard libraries, see Appendix A.

3.3.1.2. Vector.h

The header vector.h contains a simple class template that had to be designed to handle the creation of vectors and cross product operation.

3.3.2. Functions

The new algorithm was structured in a modular way. Modularity was introduced by performing each computation independently and storing the physical characteristics of the system separately in different files. Each function contains statements that are executed when they are called from the main script according to the user's preferences.

The functions are divided into two files each. One is a header file that declares classes, subroutines, variables and identifiers. Each function's header encloses the arguments of the function and the type of return value if they have one. The header also includes libraries and other header files required to execute the function. The other file (with cpp extension) contains the necessary statements to manipulate the functions parameters and variables.

Functions that return data specify the type of data they return (matrices, vectors or scalars). The data type of each parameter of the function is specified followed by an identifier. These parameters act within the function as local variables. They allow passing copies of the arguments to the function when it is called.

The other type of function used in this project is functions with no type. In this type of functions the value of an external variable is manipulated inside the function, allowing passing arguments by reference. Any modification done to the local variables will affect their counterpart variables passed as arguments when calling the function. When declaring the function's parameters, the variables that are passed to the function are announced stating the type of each argument followed by an ampersand sign (&). This sign specifies that the argument is passed by reference. This type of function is used when two or more values need to be returned. Since C++ does not allow returning more

than one value, by using this method the user can have access to two or more variables inside the function. Because of the way variables can be accessed using this type of function, their use was restricted to avoid referring them in different parts in the code.

Figure 8 shows two functions of this project that follow both formats. As mentioned previously, functions without type were used in a restricted way to avoid having several variables accessed and modified in different files.

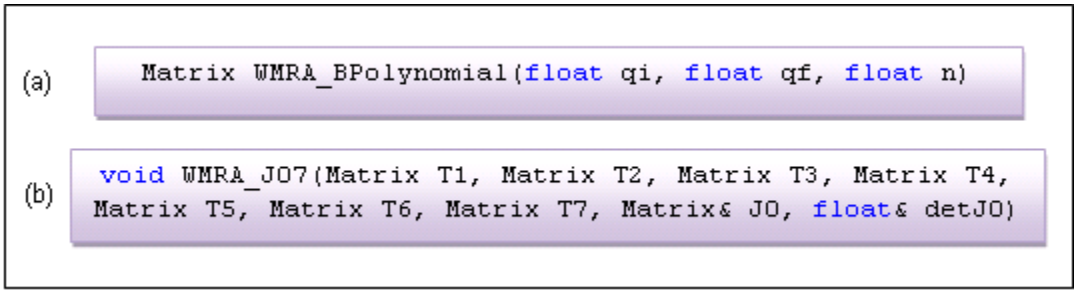

Figure 8. Function Examples (a) Matrix Return Function (b) Void Function

The functions above are examples of the types of functions used in this project. Figure 8 (a) computes smooth trajectory points of a variable “q” given the initial (qi) and final (qf) variable values and the number of trajectory points (n), using a 3rd order Polynomial with a Blending factor. The function in figure 8 (b) allows access to the Jacobian Matrix (Matrix& J0) and its determinant (float& detJ0) given the transformation matrices of each link (T1 to T7).

The control algorithm has, in total, 52 files, 25 of which have the cpp extension. The remaining are header files. Besides the function header files, and the matrix library header, there is one header file that defines and declares the global variables of the algorithm. All of the other files have access to these global variables and they can be modified in any of the files.

3.3.3. Global Variables

A global variable is a variable that is accessible from any file or anywhere in a code, even inside functions. Usually, global variables are defined at the beginning of a code, but since this project requires modularity, the global variables are declared in a header file created especially for this purpose. All the files that include the file *var_included.h* have access to the variables defined on it. The global variables declared for this project are: dHo (7 elements vector; gradient of the optimization function to avoid joint limits), DH (7x4 matrix; DH parameters of the robotic arm), e2r1, e2r2, e2r3, e2r4, e2r5, e2r6, e2r7, e2r8, e2r9, e2d (encoder readings), and 34 mxn matrices that are modified in the plotting function, and that store values that are sent to text files for plotting.

3.3.4. Physical Dimensions and Constraints of the System

The physical characteristics of the WMRA are stored in separate files that functions can access by including their header files. Any physical change in the system can be easily accommodated in the control software. The files storing the dimensions are:

1. WMRA_WCD.h: contains a five-element vector with the physical dimensions of the wheelchair and the mounting location of the robotic arm on the wheelchair.
2. WMRA_DH.h: carries a 7x4 matrix with the D-H parameters of the robotic arm.
3. WMRA_Jlimit.h: stores two seven-element vectors with the maximum and minimum joint limits of the robotic arm.

Any modification of the physical system needs to be updated in the previously mentioned files. The main script, as well as any other function that reads these dimensions, will not need to be modified.

3.3.5. Arm Motion File

The arm motion file allows the communication with the physical WMRA system. It reads the encoder readings and sends commands to the arm to be executed. The major difference between the former control algorithm of the WMRA and the new one resides in this file. Through this file, position, velocity, and acceleration are sent to the controller board so the physical arm moves. The controller board is interfaced with the PC through a DLL library programmed in C++ as functions; it was created for the former algorithm (controlMotor.dll). The Matlab code has to first, load the DLL library, open it, and then, it creates a pointer for the 10 joints to be used to read or set the encoders. The interface between Matlab (using the pointer) and the DLL library sometimes fails, and the system becomes unresponsive. Since the new algorithm code is programmed in C++, the interface between Matlab and the DLL library is no longer needed.

The arm motion file of the new control algorithm for the WMRA communicates directly with the DLL library of the PIC Servo SC controller board using the same language. The DLL library is no longer loaded, but embedded in the executable file of the new control algorithm. The advantage of this configuration is that the system is responsive all the time.

3.3.6. Difference between Previous and New Control Algorithm

As discussed, the previous control algorithm of the system was written in two different languages. The main script and functions that compute the trajectory of the

wheelchair and/or the arm were coded in Matlab. Since Matlab is a numerical computing program, matrix manipulation is possible. To achieve the same goal using the C++ language, it was necessary to resort to an open source matrix library. The library allows doing most of the operations Matlab does, but it has big differences in the way they have to be done. In the following lines, these differences will be explained.

1. Matlab can dynamically allocate variables, while in C++ all variables have to be declared before using them. This might make the code longer in extension, but ensures no variable is used more than once for different purposes.
2. Matrix indexing is not the same. When a matrix is created using C++, its size is the same as in Matlab, but its indexing starts at 0, not 1. Figure 9 shows the indexing of a 2x2 matrix in C++ and Matlab.

a) $\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$

b) $\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix}$

Figure 9. Matrix Indexing in (a) Matlab and (b) C++

3. The notation for multiplying a matrix times a scalar in C++ is: $M^{*}=(a)$, where *M* is the matrix to be multiplied, and *a* is the multiplying scalar. Meanwhile, Matlab does not require special notation: $M= a \times M$.
4. To set all the elements of a matrix at once using C++, an array with the same size of the matrix needs to be created first. The elements are set in the array and then transferred to the matrix using *for* loops. The number of loops needed depends on

the dimensions of the matrix: a vector requires one for loop, while a 2-dimension matrix requires two for loops (see figure 10).

```
float DHtemp[7][4]= {{-PI/2, 0,     110,  q(0,0)},
                     { PI/2, 0,     146,  q(1,0)},
                     {-PI/2, 0,     549,  q(2,0)},
                     { PI/2, 0,     130,  q(3,0)},
                     {-PI/2, 0,     241,  q(4,0)},
                     { PI/2, 0,       0,  q(5,0)},
                     {-PI/2, 0, 179+131,  q(6,0)}};

int i,j;
Matrix DH(7,4);
for (j=0; j < 4; j++){
       for (i=0; i < 7; i++){
              DH(i,j) = DHtemp[i][j];
}
}
```

Figure 10. Setting Elements in a Matrix Using C++

In Matlab, setting the elements of a vector or a matrix is an easy task. Each element of a matrix is entered separated by a space or a comma, and each row is separated by a semicolon. Matrices or vectors are delimited by brackets and are set equal to a variable. Figure 11 shows an example of matrix setting in Matlab.

```
a = [-PI/2,0,110,q(0,0);PI/2,0,146,q(1,0);-PI/2,0,549,q(2,0)]
```

Figure 11. Setting Elements in a Matrix Using Matlab

5. The matrix library of C++ does not support 3-dimensional matrices as Matlab does, therefore, for trajectory planning along a line (Traj.h and Traj.cpp) it was necessary to use 3-dimensional arrays and pointers along 4x4 matrices. The code for trajectory planning using C++ is longer than the Matlab one, but its processing time is shorter.

As seen before, Matlab manipulates matrices in a more efficient manner than C++ while programming; however, the computational requirements for Matlab to process the

code are larger than that in C++. Although C++ code is larger in extension, it is more efficient while using fewer resources from the system.

As a final remark, the new algorithm will discriminate whether the user inputs the correct choice to control the system, i.e. if a particular option has four alternatives, and the user selects one other than these, the program will rerun the inquiry for the user input. The previous algorithm assumed a default value if an invalid option was selected and kept on running. This was included as a preventive measure to avoid involuntary typing mistakes.

3.4. Algorithm Advantages

As with any programming language, C++ has its advantages and disadvantages for programming a new control algorithm for the 9-degrees-of-freedom-system over the previous programming language. The new algorithm shortens the processing time of the system because the speed of a program is higher when coded into C++ than into Matlab since the interpretation time of Matlab is higher. In the testing results chapter, it will be shown that motion is more reliable because of the embedded arm motion functions, and that accuracy has been improved. The program runs smoothly and the CPU usage is less using C++. Also, the new control algorithm has expandability for real time applications, while Matlab stays short. Moreover, the new control algorithm is portable and executable from any operating system without needing parenting software to run.

3.5. Algorithm Disadvantages

As mentioned before, Matlab has certain features that C++ does not have that add some disadvantages for the new algorithm control, especially when programming it. The matrix manipulation in C++ was done using an open source library, and it does not

support all the features Matlab supports. Because of this, some arrangements in the new code made it larger due to the extra steps needed for matrix handling. In the case of C++, a larger code does not imply more computational time than Matlab, but for a programmer it might make it tiresome to add future changes.

Another disadvantage of the new control algorithm compared to the previous one is that every variable or constant has to be declared specifying its data type. If, during the computations the type of a variable is converted, memory loss can occur. While programming the new control algorithm, this issue was minimized by keeping data type conversions to a minimum.

Other deficiencies of the new code are the use of global variables that lead to unpredictable side effects, which can directly contribute to lowering the quality of the code. The primary reason for global variables to be avoided is because they can be modified in any function that is called. For example, if the dHo variable used in the main script is modified in the arm motion file without considering the repercussion of this modification on the main script, the programmer might not know the final output of the main script.

Another disadvantage of the code is the lack of an embedded plotting application. To be able to plot the results of the computations made in the algorithm, values of variables had to be stored in a text file that can be read later using a plotting software. For testing the new algorithm, the stored values were opened and plotted using Microsoft Excel. When no plots are required, the algorithm runs faster and consumes fewer resources from the CPU since no values are stored in a text file.

Another disadvantage of the new algorithm is the control flow. The main script runs and the subroutines are executed in a logical order. While a subroutine is performed, no other subroutine or statement is executed, meaning no two statements can be executed at the same time. This is not a disadvantage while commanding the robotic arm to move in autonomous mode because it is guaranteed that the code and the movement of the arm are synchronized; however, when the arm is controlled using the available teleoperation interfaces, a stopping mechanism had to be created that executed behind the main script and not along with it. In the previous algorithm, when the system was controlled using one of the teleoperation interfaces, a window with an EXIT button would pop up as soon as the program started; the window would always be visible, and the user could stop using the system by clicking the exit button at any time. In the new control algorithm, the window is an external application that has to be opened before the arm starts moving. The exit window application of the new algorithm communicates with the main script by text file. When the EXIT button is pressed, the value of the text file is changed and immediately read by the main script. The need of having two applications opened when controlling the system with the SpaceBall and the touch-screen would cease by implementing threads.

3.6. Summary

This chapter presents the environment used for writing the new control algorithm, which was fully described. Unlike the previous algorithm, the new one handles different types of functions more efficiently. Different libraries are included in each file for execution. An open-source matrix library was used to manipulate matrices. The

differences between the new code and the old one were presented. The advantages and disadvantages of the new code were discussed.

Chapter 4: Graphical User Interface (GUI)

4.1. Graphical User Interface

The Wheelchair-Mounted-Robotic-Arm was designed to be controlled using different user interfaces, which were created considering the manipulation capabilities of the user so the robotic arm is employed effectively. The users can select an interface based on their preferences and abilities. In the case of the WMRA system, a single user interface is sufficient to control both the arm and the wheelchair using the same control algorithm.

The main control program of the WMRA can be accessed by a command prompt window where the user can manipulate the system by selecting desired options. However, using the command prompt to select options is a time consuming, exhausting, and demanding task to complete by the user, especially by a person with disabilities. To ease the process of options selection a main Graphical User Interface (GUI) was previously designed using Matlab. A similar GUI was redesigned in C++ to avoid having to communicate between different software and to achieve the goal of removing Matlab from the control loop.

The new GUI contains the main control algorithm but displays and stores default values that the user can apply as soon as the software opens, dispensing the user from the task of selecting options to run the system, though these default values can be changed according to the user's preferences. Also, different options will be enabled and disabled

according to the user's selections. Figure 12 shows the main GUI designed using Visual Studio C++ and Windows Forms. The flexibility of VS C++ allows further modifications to this GUI given the necessity.

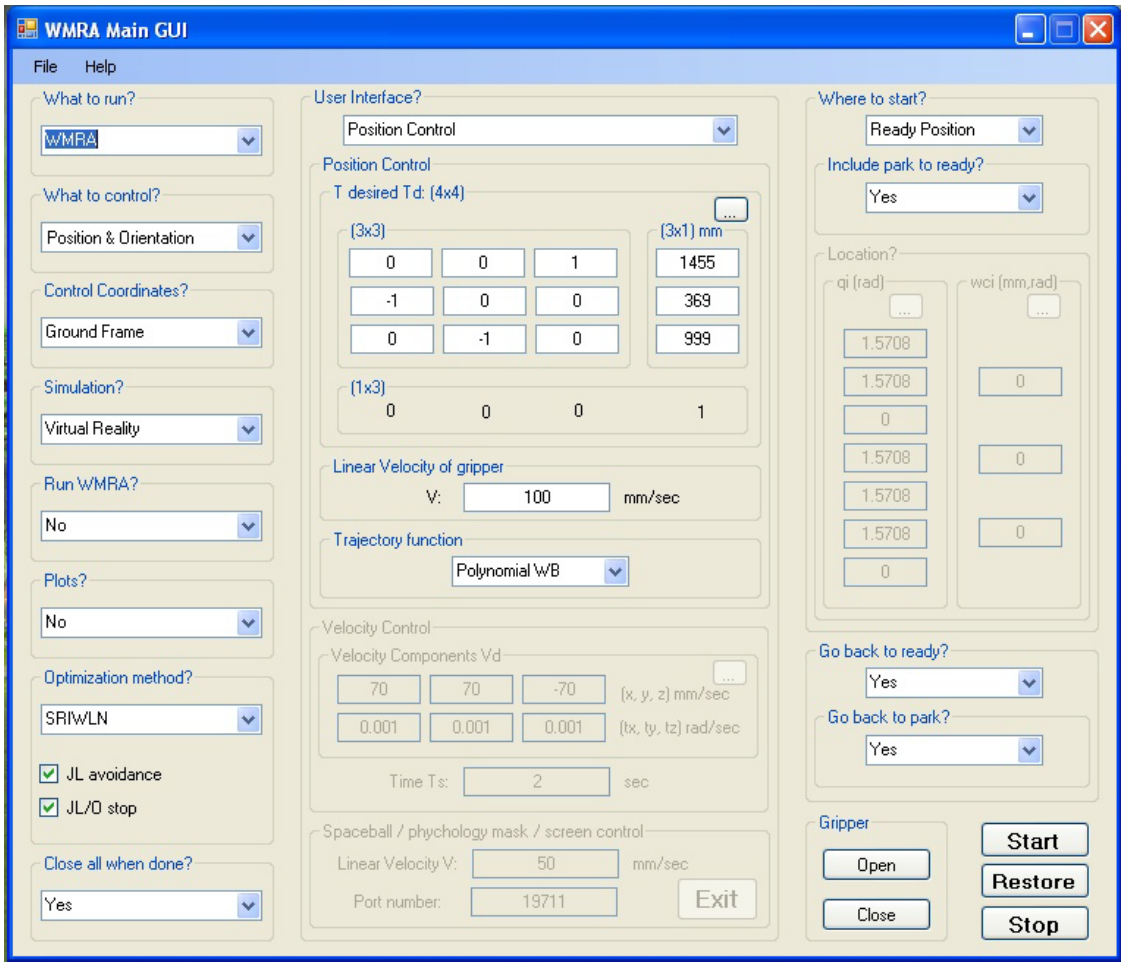

Figure 12. WMRA Main GUI in C++

The WMRA Main GUI is built with the same functions created for the main control code. The major changes reside in the subsections of the WRMA Main GUI cpp file. Before each calculation, the algorithm verifies that the user still wants to manipulate the robotic arm by inspecting a variable designed for this purpose. As soon as the user clicks the stop button of the main GUI, the variable changes, and the algorithm stops.

The main difference between the previous GUI and the new one is that the latter does not create other windows forms as functions.

4.2. Graphical User Interface Components

The new Graphical User Interface was designed using the GUI toolkit embedded in Microsoft Visual C++ since it facilitates the development of Windows Forms. A form contains controls that allow the user to interact with the system. For this specific application, five different controls were applied. Each control allows different ways to manipulate the data: by selecting options from a list, by writing values in a text box, or by clicking buttons to display different windows or to select events.

Unlike the previous GUI created in Matlab, the C++ GUI does not require the use of unfriendly techniques and is more flexible. The new GUI did not need the creation of extra functions for it to work. Most of the functions are inherited from C++ language.

4.2.1. Group Box

Group boxes are objects that delimit other control objects inside a form by displaying a frame and caption around them. They logically group the controls. One of its properties sets groups mutually exclusive. When a groupbox is disabled, all the controls contained on it are automatically disabled. Most of the properties applied to a group box are passed to the controls on it.

4.2.2. Combo Box

A combo box is a control that contains a list of objects. The list can be always displayed or displayed in a drop-down. For this application, all the combo boxes have a drop-down style, and text editing is disabled. To access the options of the combo box the user has to click it and the list shows up.

The list of options of a combo box is indexed in a vector. When an option is selected, the index of the combo box changes and that value can be stored and later used. The combo box features enable the user to add or remove items from it dynamically, without the control being repainted each time.

4.2.3. Text Box

Text box controls are used to either display text or to receive text. The multiline property of a text box enables the control to have multiple lines of text to be displayed or entered. Text in a text box is contained in a string, and to display a value coming from the program, it has to be converted into a human-readable string using the ToString method. Similarly, data read from a text box has to be converted from string representation to its number equivalent. For this system, all the string representations were converted to double-precision floating point numbers using the Double::Parse method.

A text box does not indicate its function. Therefore, in this application, they are accompanied by labels.

4.2.4. Labels

Labels are used to present useful information to the user about controls. Labels can also display images. The text contained in a label can be dynamically modified. None of the labels used on the WMRA GUI have an event associated to them and they cannot be modified by the user.

4.2.5. Buttons

Command buttons allow the user to trigger actions by clicking the control. The control algorithm of the WMRA system is encapsulated inside the OnClick event of the START button of the Main GUI. The properties of each control inside the Main GUI

were customized according to the necessities of the system. All of the properties can be easily modified for future improvements.

4.3. User Interfaces

To accomplish the desired modularity of the system, all the user interfaces are linked to a main Graphical User Interface. This GUI allows the user to select an interface to work with. All interfaces are converted into corresponding vectors that are linked to the main program through global variables; therefore, modifying or adding new interfaces is easy to do while the vector is properly formatted.

One of the main interfaces is a Touch-Screen on a Tablet PC (Fujitsu Lifebook tablet PC equipped with a 12-inch active digitizer). The tablet is already a part of the control hardware of the system. When the touch-screen user interface control is selected, a window pops up with the functions and directions that the user can choose by pressing the button by mouse or the equipped touch pad. Figure 13 shows the touch-screen form.

The touch-screen is linked to the main algorithm through a seven element vector, with each element relating to a joint in the arm. When one button of the touch-screen is selected, the value of the corresponding element vector changes and the arm will be commanded to move in the commanded direction. The touch-screen interface was tested to confirm that it is working properly, and at the same time, to ensure that the new control algorithm runs smoothly and can be used to teleoperate the WMRA system. While using the interface with the new control implementation, the system responded effectively and accurately.

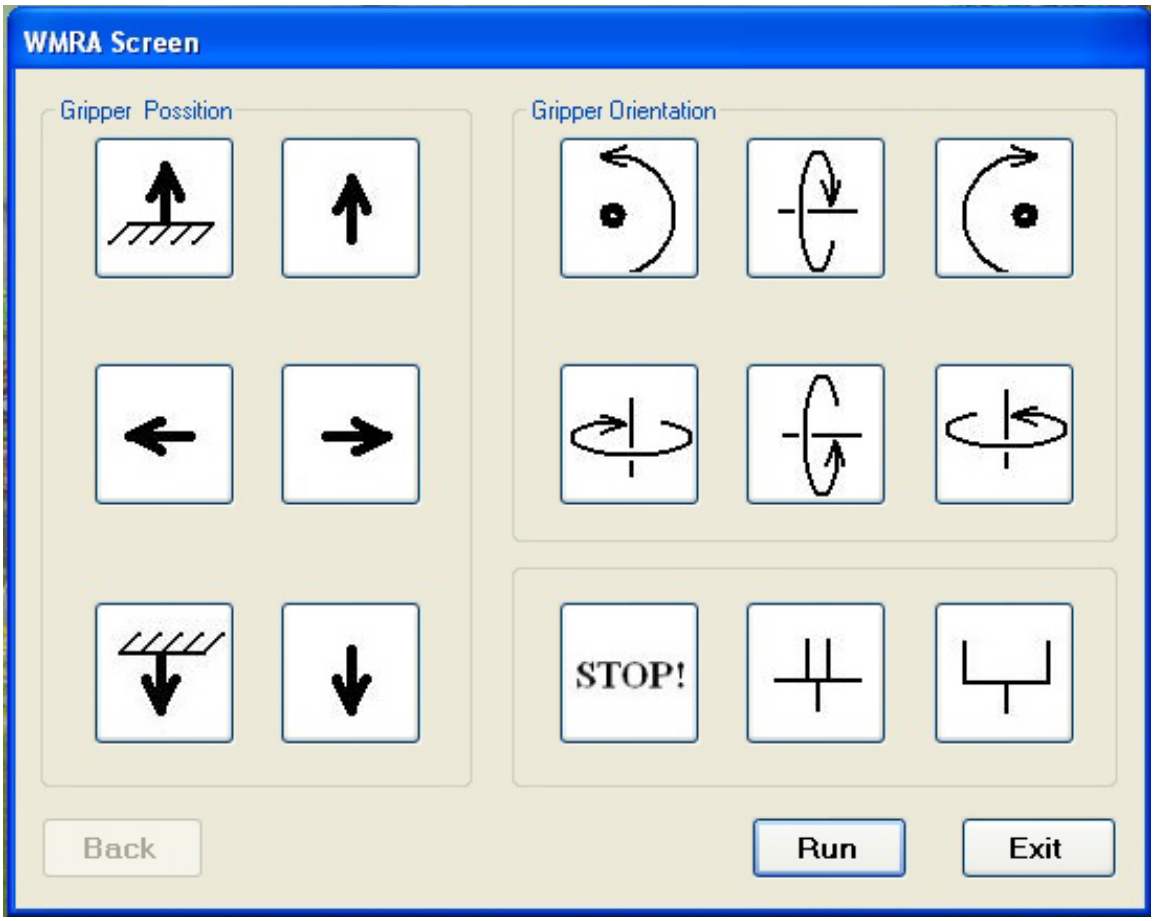

Figure 13. WMRA Touch Screen

4.3.1. SpaceBall

One of the main interfaces of the WMRA system is a six axis, twelve-way SpaceBall. This user interface makes a three dimensional motion that corresponds to the six Cartesian space variables used in the WMRA. For the SpaceBall to be implemented and integrated to the former control algorithm, a program was designed in C++ to run the driver of the SpaceBall, collect the user inputs from the device and send them to a Matlab environment as a vector variable that was changed constantly as the user moved the SpaceBall.

However, the SpaceBall interface was never able to continuously move the physical arm because the software crashed right after commanding the arm to move. The arrangement for the SpaceBall never worked for several reasons. First, the original code of the SpaceBall program was designed in C. The data coming from the SpaceBall program into Matlab was not coming correctly; it was sending the size of the variables for the vector and then indirectly sending the actual vector. Matlab used a wrapper function that used a global variable to share the data, so the reliability of the SpaceBall program depended on the reliability of the wrapper program.

The new program for the SpaceBall uses a simple demo program (3DxTest32) that the manufacturers provided in a software development kit (SDK). The demo basically opens the SpaceBall and creates a window, connects to the SpaceBall, and then when the ball is moved, a motion event function is called and data from the SpaceBall is delivered to the window display. The new code was written under the motion event function. This way, whenever motion is detected and data is being retrieved from the SpaceBall, the data is sent to a text file. This text file is written live in write mode only, so the file can be simultaneously opened for reading with minimum delay. Meanwhile, in the main script of the control algorithm (or the WMRA GUI), the data from the text file is extracted into 6 values in character form, and then converted to numeric variable types for use in the program. Since the control algorithm opens the text file in read mode, it can be simultaneously read from the file with minimum delays.

In the future, threads might be used to run both the SpaceBall data function and the WMRA function, so that a text file is not necessary, which will eliminate any present

delay. Unfortunately the 3D Connexion driver is very limited in its capabilities and is designed for use in GUI-based design software and not console programs.

The implementation and integration of the SpaceBall to the new control algorithm has widened the interface options for controlling the WMRA system. The SpaceBall was tested with the physical arm, and it responded accurately and smoothly. However, the physical conditions of the SpaceBall are not near factory settings. For future improvements, a newer SpaceBall might be implemented.

4.4. Summary

In this chapter, the new Graphical User Interface of the WMRA system was described. It was developed making use of the inherit functions of C++. The linkage between the GUI and the control algorithm was relatively easy to make since both programs run under the same language. The implementation of the SpaceBall, an interface of three-dimensional motion, was described. Its integration enhanced the modularity and expanded the user interface capabilities of the WMRA system. The code behind the interface is in Appendix B, and the codes for the SpaceBall as well as the original un-modified 3DxTest32 demo are in Appendix C for further information.

Chapter 5: Testing and Results

5.1. Testing C++ vs. Matlab Functions

In order to prove the efficiency of the new algorithm, several tests were performed. Most of them were done to compare the capabilities of the new control algorithm against the previous one.

Every time a function of the control algorithm was programmed in C++, a test was run to prove the accuracy of the function. The results of each function were compared against the results obtained in Matlab. The comparison was made through plots of the results. To be able to plot the results of the new algorithm, the data had to be sent to a text file and was later plotted using Microsoft Excel.

The following case was executed using both the previous and the new control algorithm. The graphs obtained from this execution prove the similarity between the calculation done in Matlab and in C++.

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

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

Figure 14. Test Specifications

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

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

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

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

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

 Please enter the transformation matrix of the desired position and orientation from the control-
based frame
 (e.g. [0 0 1 1455;-1 0 0 369;0 -1 0 999; 0 0 0 1])
[0 0 1 2455;0 1 0 369;0 -1 0 999; 0 0 0 1]

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

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

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

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

Figure 14. Test Specifications (Continued)

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

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

Figure 14. Test Specifications (Continued)

5.1.1. Joint Angular Velocities and Displacements

Figures 15, 16, 17, and 18 are graphs of each joint angular velocities and displacements against time calculated using Matlab's and C++'s algorithm. At a glance, both graphs are similar, but to ensure the new algorithm is not deviating too much from the calculations of the previous one, the sum of the square error was calculated for each calculation.

$$SSE = \sum \left(X_{C++} - X_{Matlab} \right)^2 \tag{1}$$

In (1), X_{c++} and X_{Matlab} take the value of any variable that is being compared (i.e., joint angular displacements, velocities, etc). Figure 19 shows the sum of the square error of each joint angular velocity and displacement. In this case, the calculations of angular displacement of joints one and seven present the biggest deviation from the computations done using Matlab. Figure 19 also proves the sum of the square error for joint angular velocities and displacements to be very small, meaning the calculations of angular velocities and displacements done by the new control algorithm are acceptable.

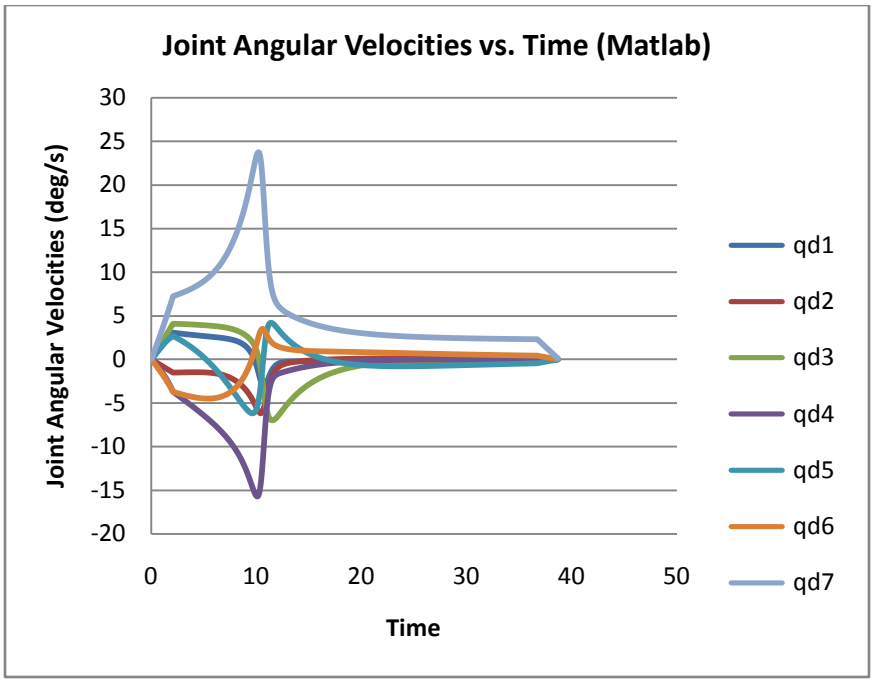

Figure 15. Joint Angular Velocities vs. Time, Matlab Algorithm

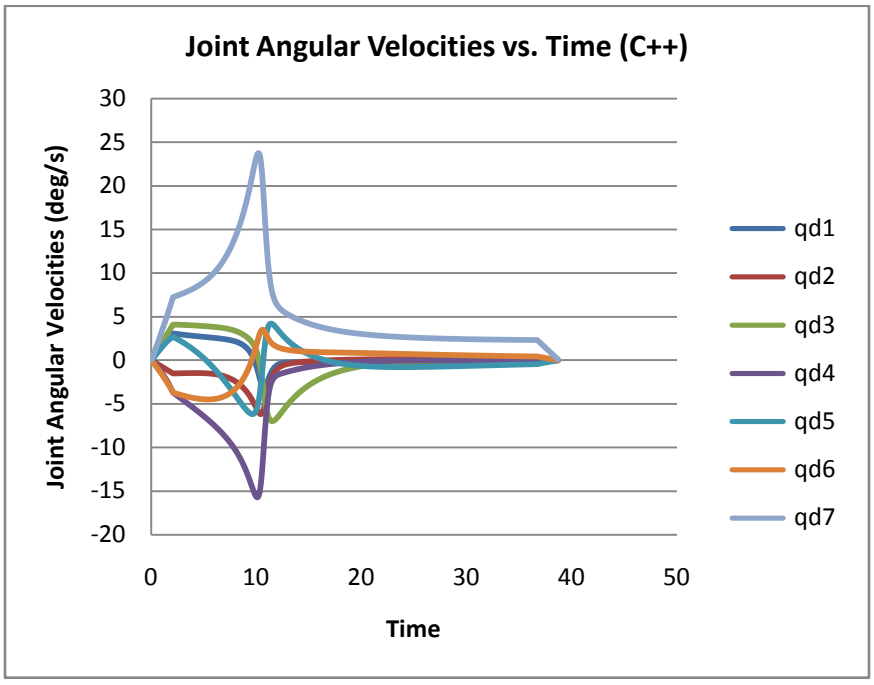

Figure 16. Joint Angular Velocities vs. Time, C++Algorithm

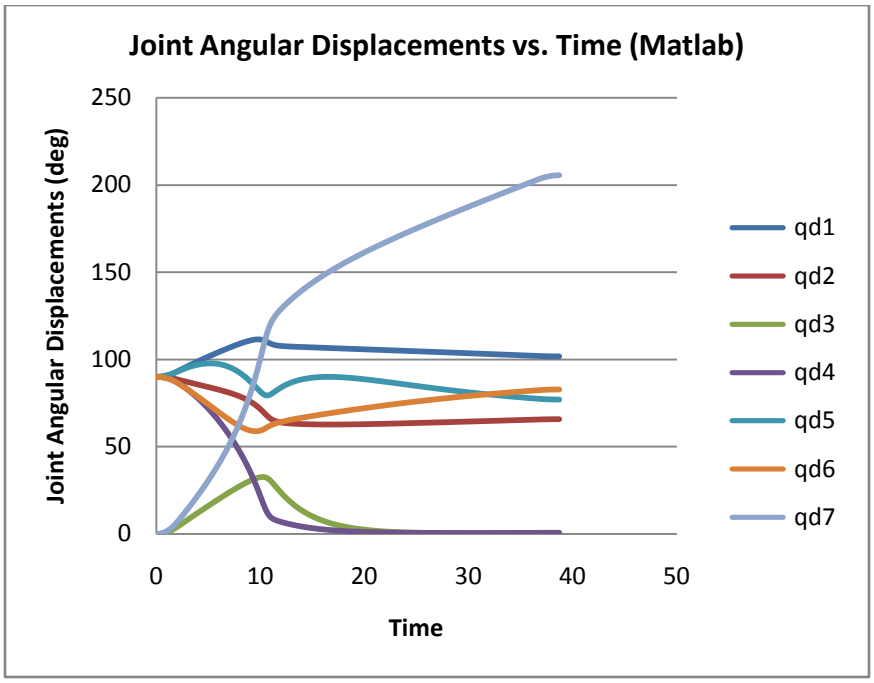

Figure 17. Joint Angular Displacements vs. Time, Matlab Algorithm

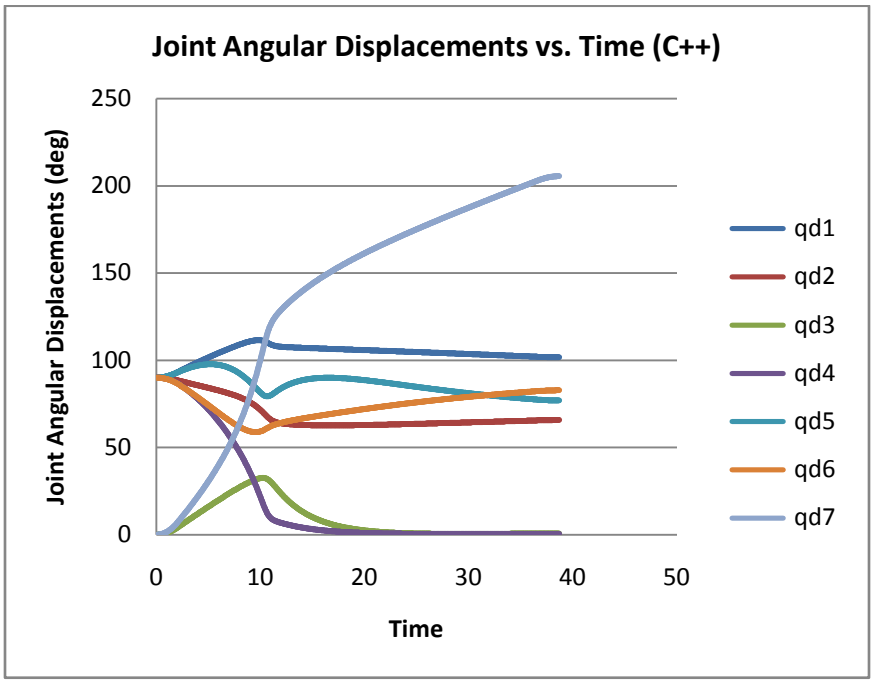

Figure 18. Joint Angular Displacements vs. Time, C++Algorithm

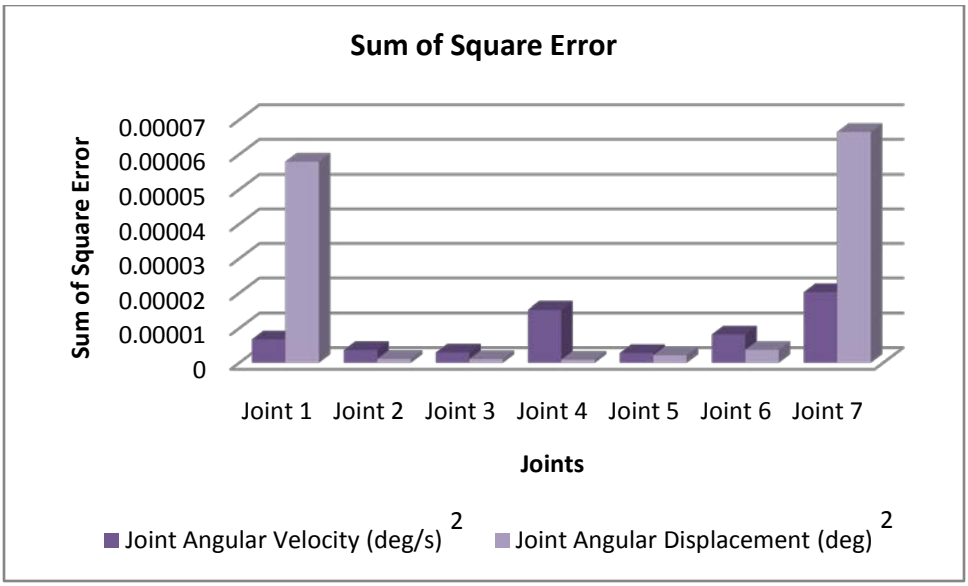

Figure 19. Sum of Square Errors for Joint Angular Velocities and Displacements

5.1.2. Wheel Track Velocities and Displacements

The following graphs illustrate the velocity and displacement calculations of each wheel track against time, made using both, Matlab and C++. In the same manner as the joint angular velocities and displacements values were treated, the sum of the square errors was calculated and plotted in figure 24.

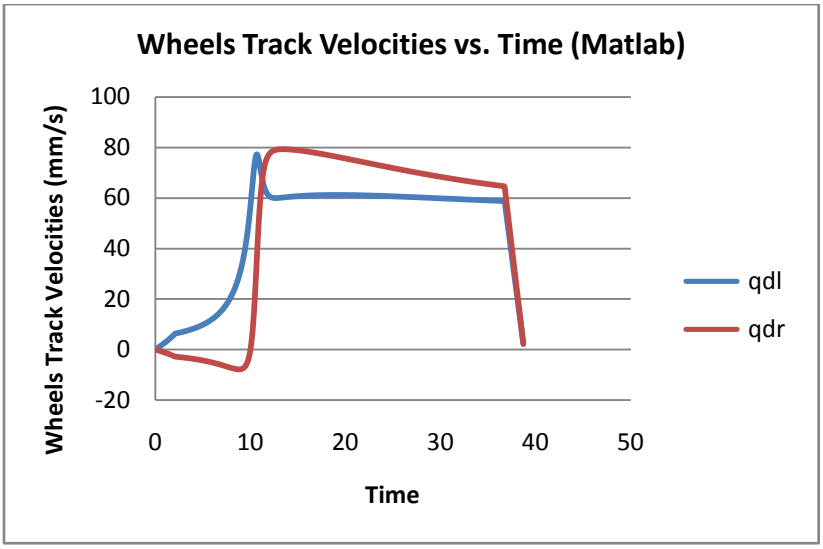

Figure 20. Wheels Track Velocities vs. Time, Matlab Algorithm

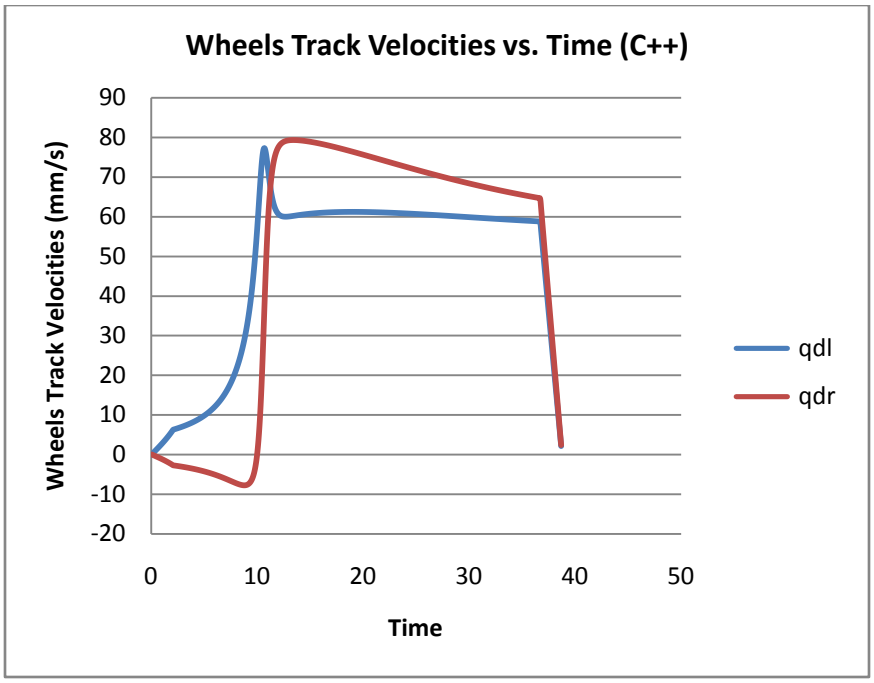

Figure 21. Wheels Track Velocities vs. Time, C++ Algorithm

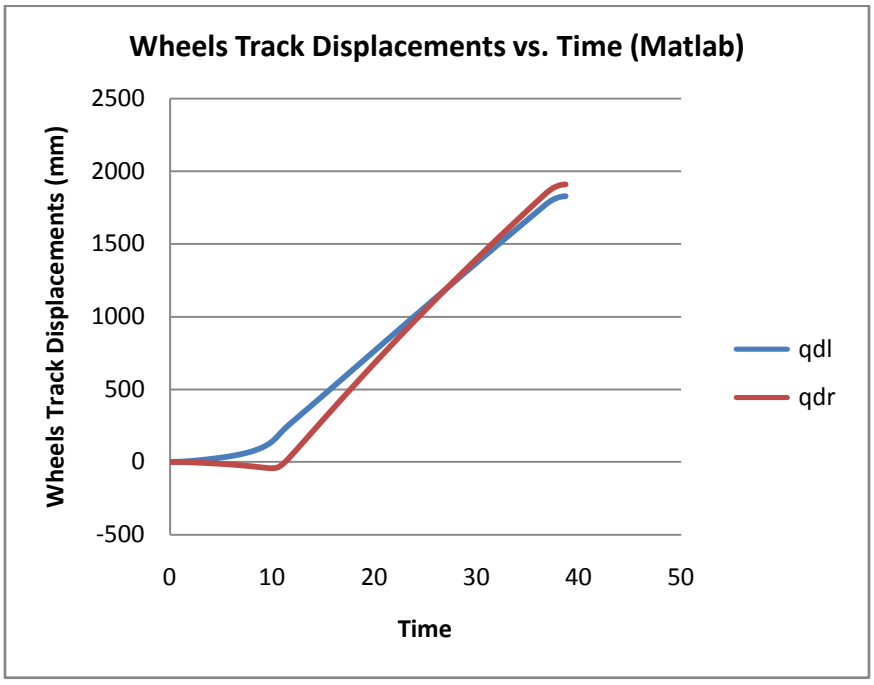

Figure 22. Wheels Track Displacements vs. Time, Matlab Algorithm

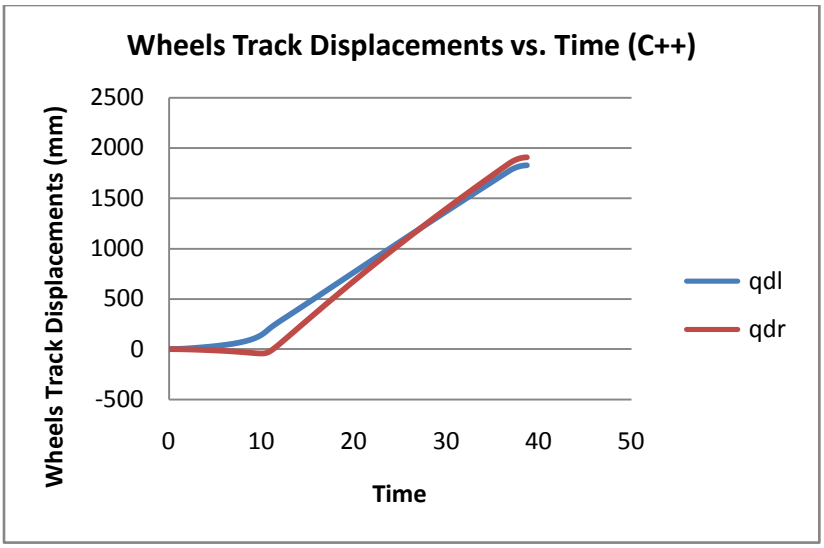

Figure 23. Wheels Track Displacements vs. Time, C++ Algorithm

The sum of the square error of the wheels track velocities and displacements was calculated using (1). As figure 24 shows, the sum of square error is higher in the calculation of track displacement than in track velocities, specifically for the right wheel. As it can be appreciated in the figure, the error is very small compared to the magnitude of the values computed for the wheels track velocities and displacements.

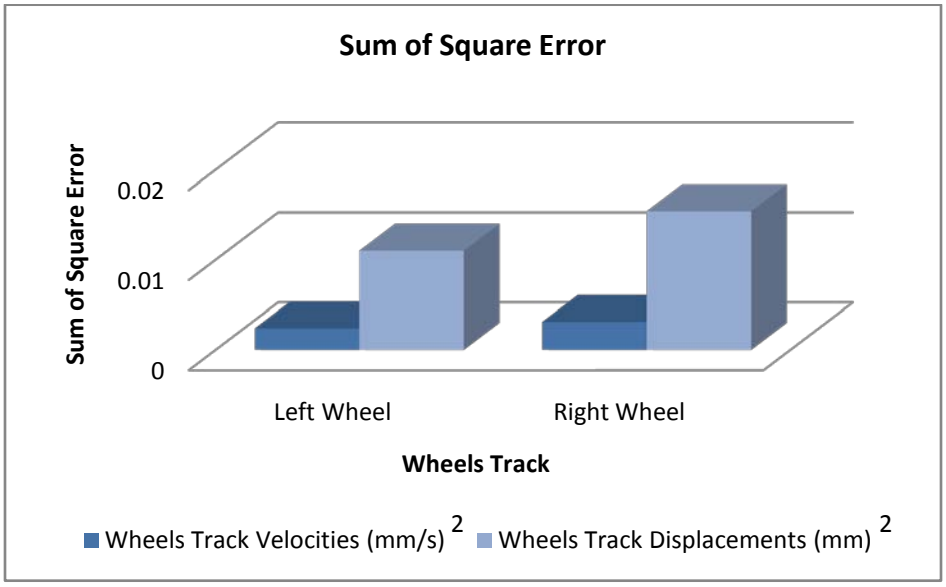

Figure 24. Sum of Square Errors for Wheels Track Velocities and Displacements

5.1.3. Cartesian Position and Orientations

Figures 26, 27, 28, and 29 illustrate the position and orientation of the end-effector of the robotic arm versus time. For the case tested, the gripper's orientation only changed in one direction when using Matlab's code; it remained almost constant in the other two directions. However, while using the new control algorithm the orientation of the gripper changed about half a degree in the roll and yaw direction (25) during the last iteration. This error can be noticed in the sum of square error graph of figure 30. This error is introduced by the loss of data due to the conversion of variable types (i.e. from float to doable, integer to float, etc) that was done internally by the program when combining the data type of the vector that stores the values of the angles, and the data type of the standard library of C++ that calculates trigonometric functions. Figure 30 shows the sum of the square errors is higher for both, rolling and yawing, however, these errors are negligible since they are very small compared to the actual calculated values of the orientation angles done by Matlab and C++.

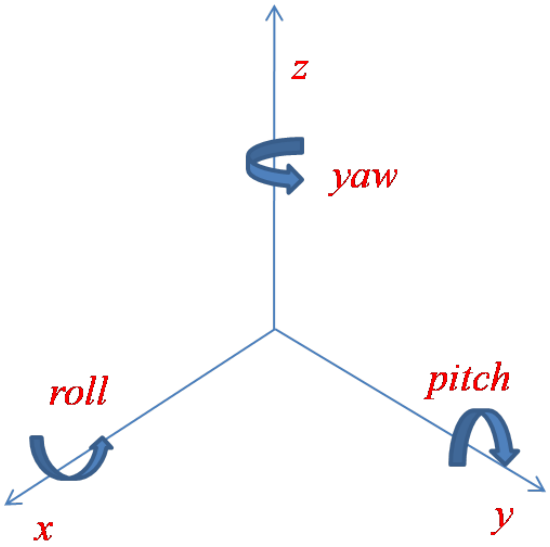

Figure 25. Roll, Pitch and Yaw Angles

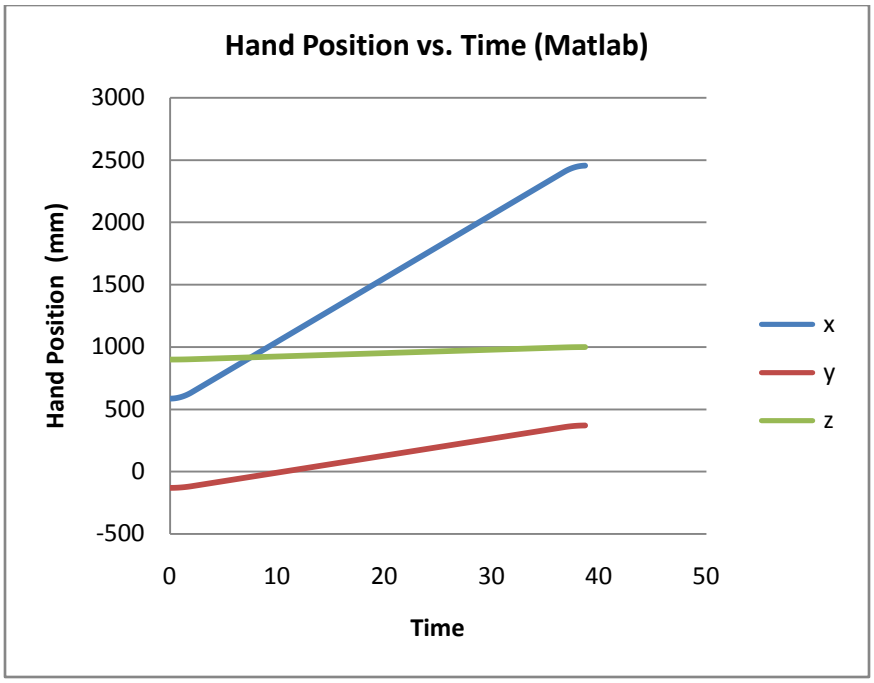

Figure 26. Hand Position vs. Time, Matlab Algorithm

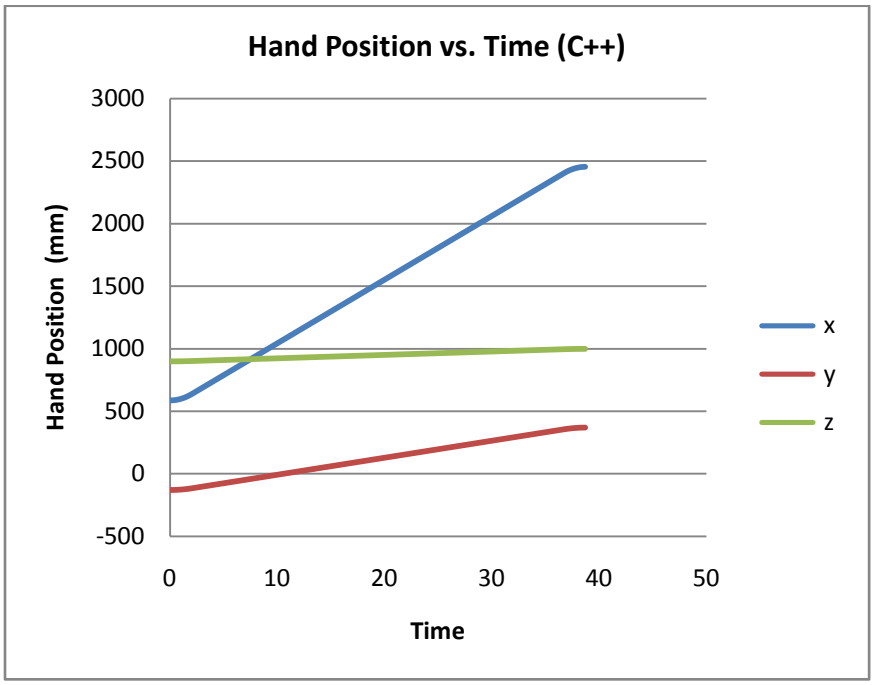

Figure 27. Hand Position vs. Time, C++ Algorithm

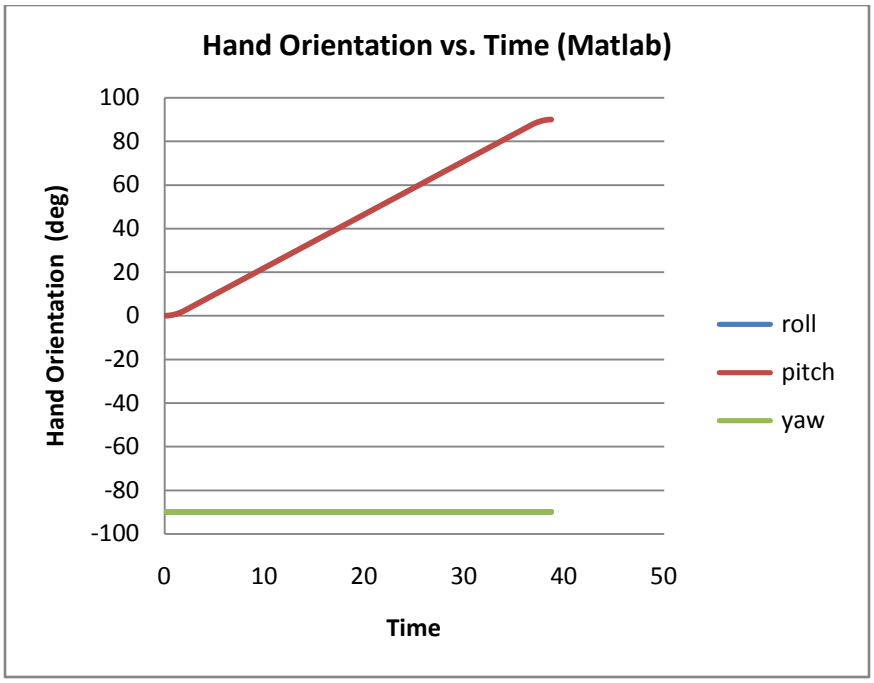

Figure 28. Hand Orientation vs Time, Matlab Algorithm

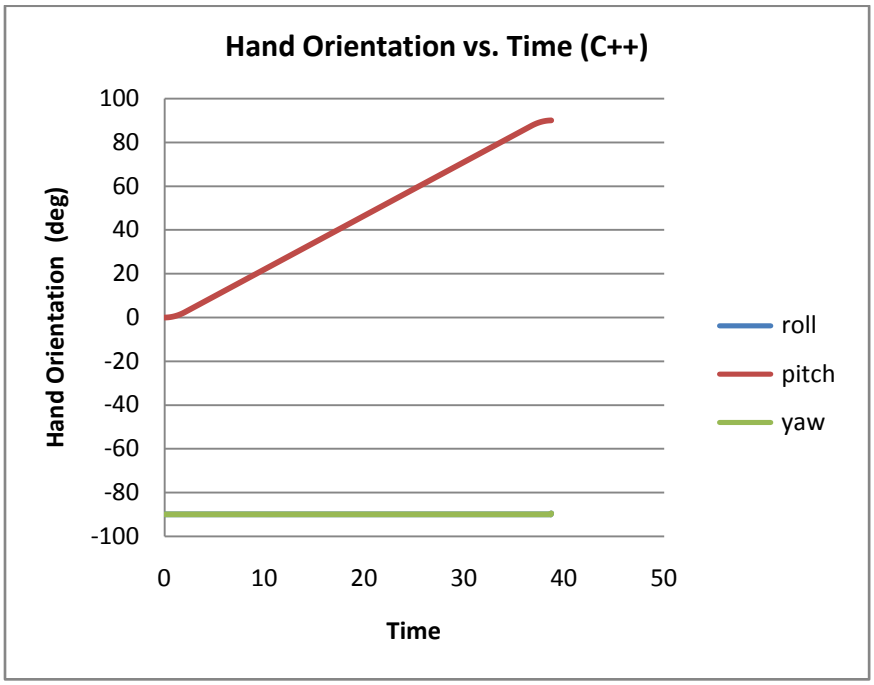

Figure 29. Hand Orientation vs. Time, C++ Algorithm

As mentioned earlier, the sum of the square error is higher when calculating the orientation of the gripper, especially for the roll and yaw angle. However, as it can be seen in figures 27 and 28, the error is introduced in the last iteration. The exact difference between the calculation in Matlab and C++ is 0.45 degrees, a negligible amount.

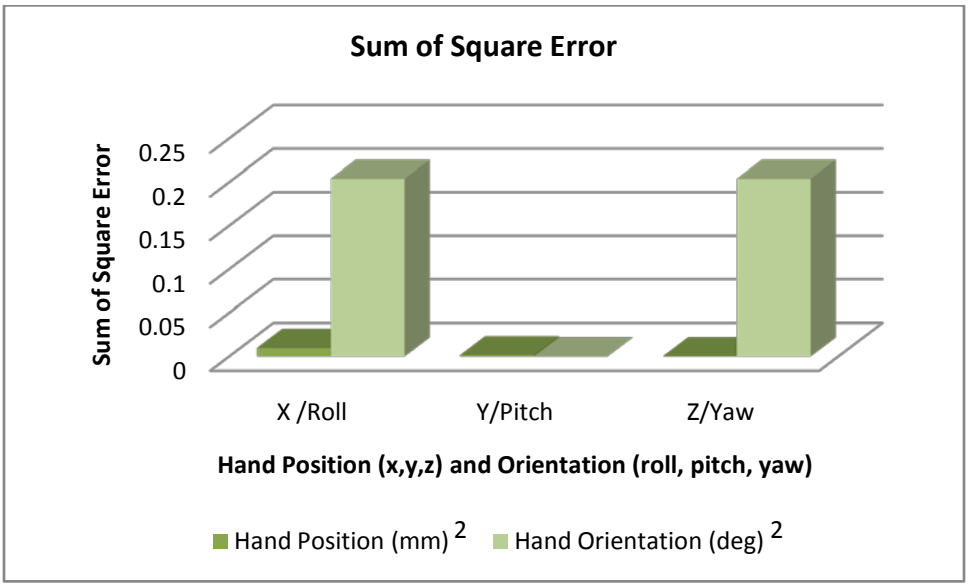

Figure 30. Sum of Square Errors for Hand Position and Orientation

5.1.4. Arm Base Positions and Orientations

Figures 31, 32, 33, and 34 represent the calculations of the arm base position and orientation against time made by the previous and the new control algorithm. For this specific case, pitch and yaw rotations do not change through time and remained zero. It can be appreciated in figure 35 that although the roll movement does not remain zero, nor constant, the error between calculations is nearly zero. For the arm base, the maximum sum of square error appears in the x position, where its value is close to 0.01mm when the values of x position of the arm base are in the range of thousands of millimeters.

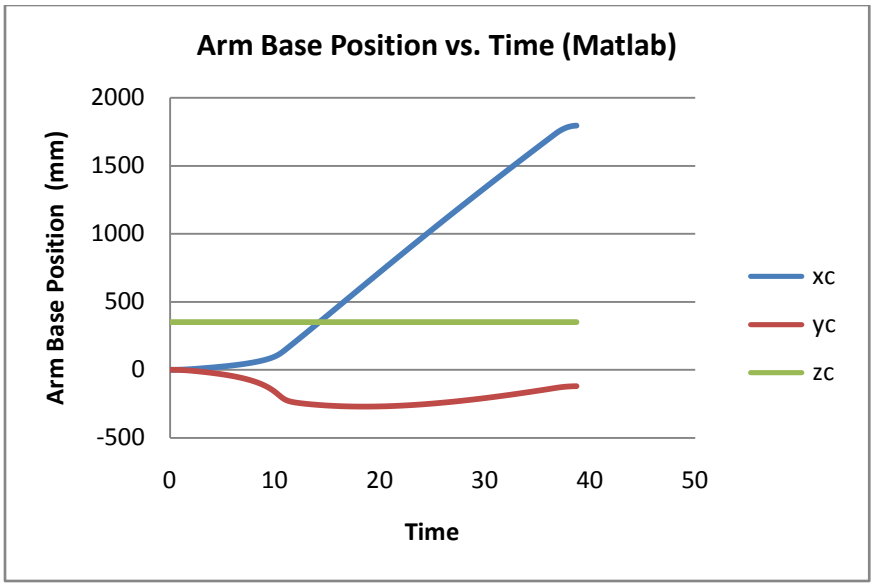

Figure 31. Arm Base Position vs. Time, Matlab Algorithm

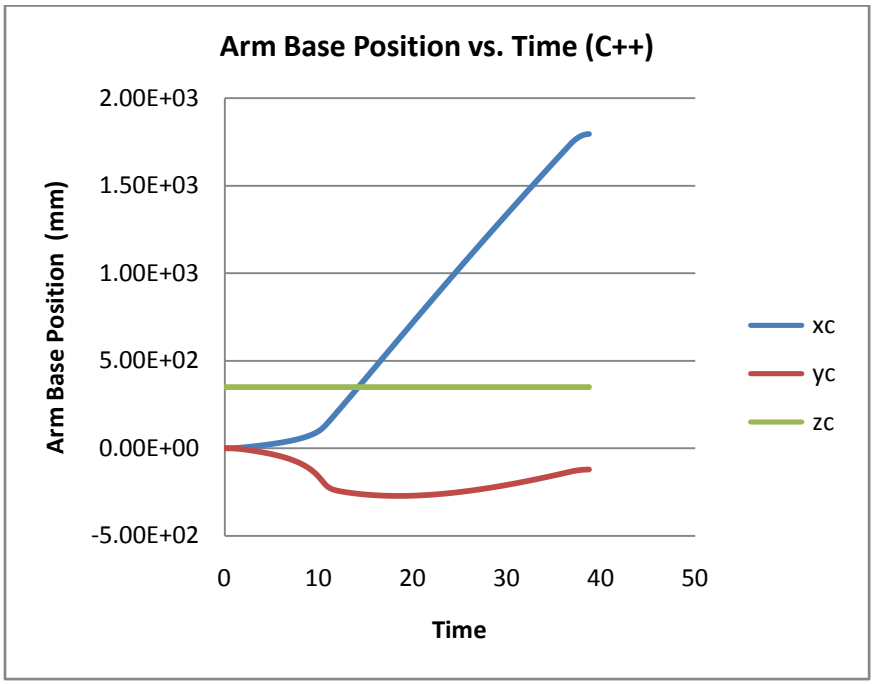

Figure 32. Arm Base Position vs. Time, C++ Algorithm

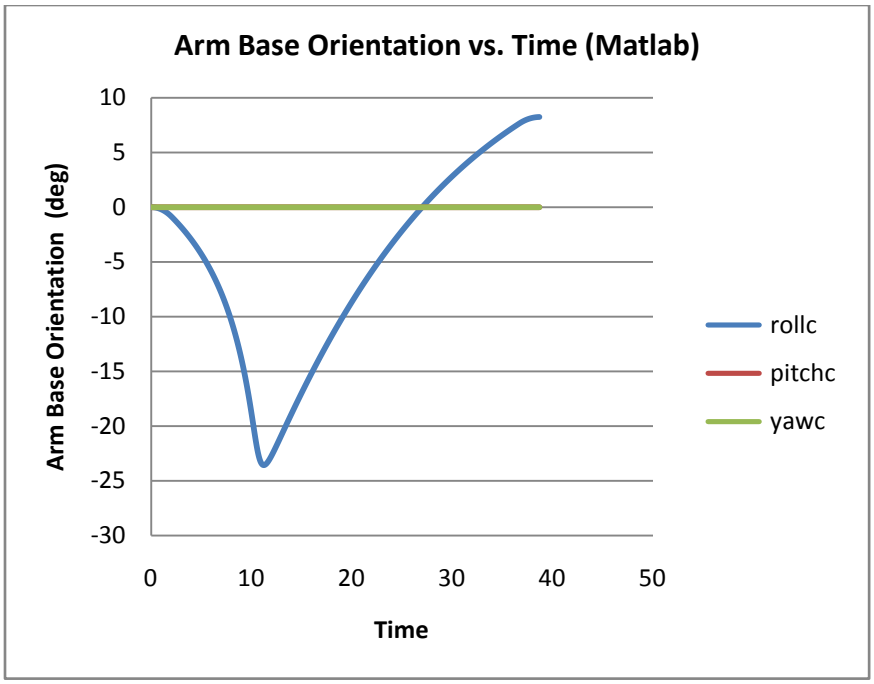

Figure 33. Arm Base Orientation vs Time, Matlab Algorithm

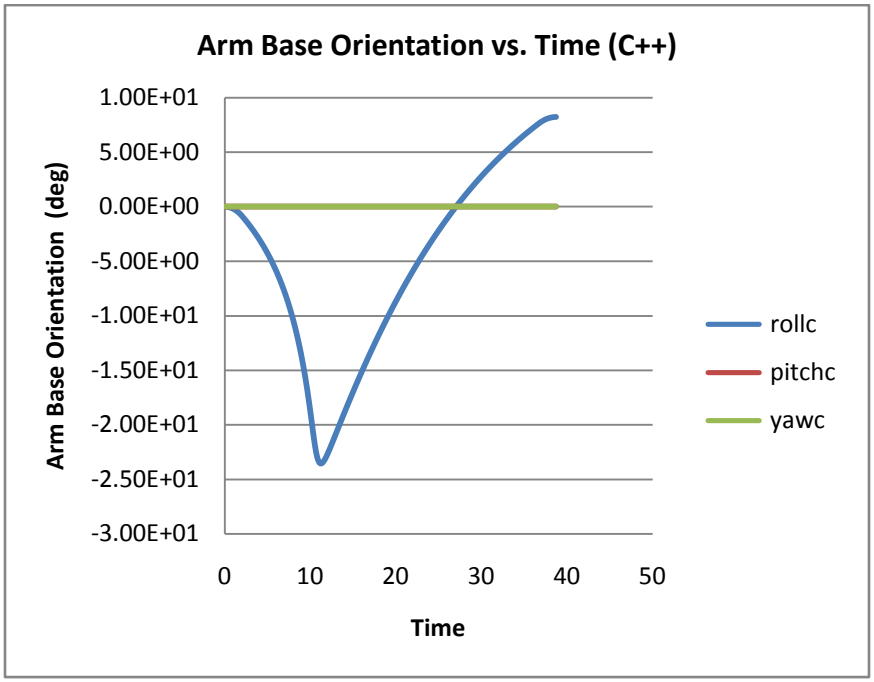

Figure 34. Arm Base Orientation vs Time, C++ Algorithm

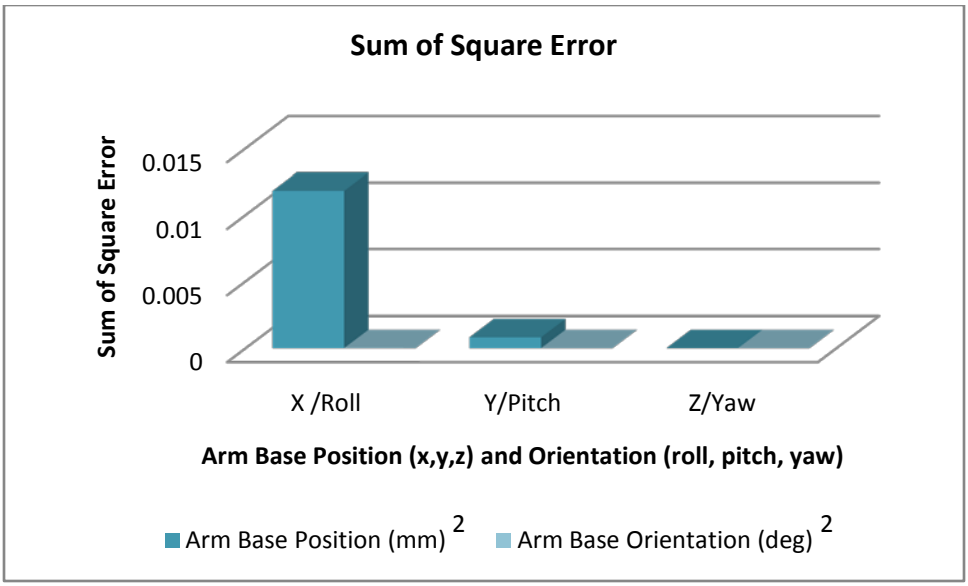

Figure 35. Sum of Square Errors for Arm Base Position and Orientation

5.1.5. Manipulability Index

Manipulability was measured against time for both arm only and the combined WMRA system using both algorithms. Figures 36 and 37 present the normalized results of the measurement. The results were normalized by multiplying the values by a factor of 10^{-9}. Figure 38 shows that the error between the calculations made by the new algorithm for the manipulability index is close to the computations of the previous algorithm. The average percentage error shows the calculations done using the new algorithm are less than one percent off from the calculation made using the old algorithm.

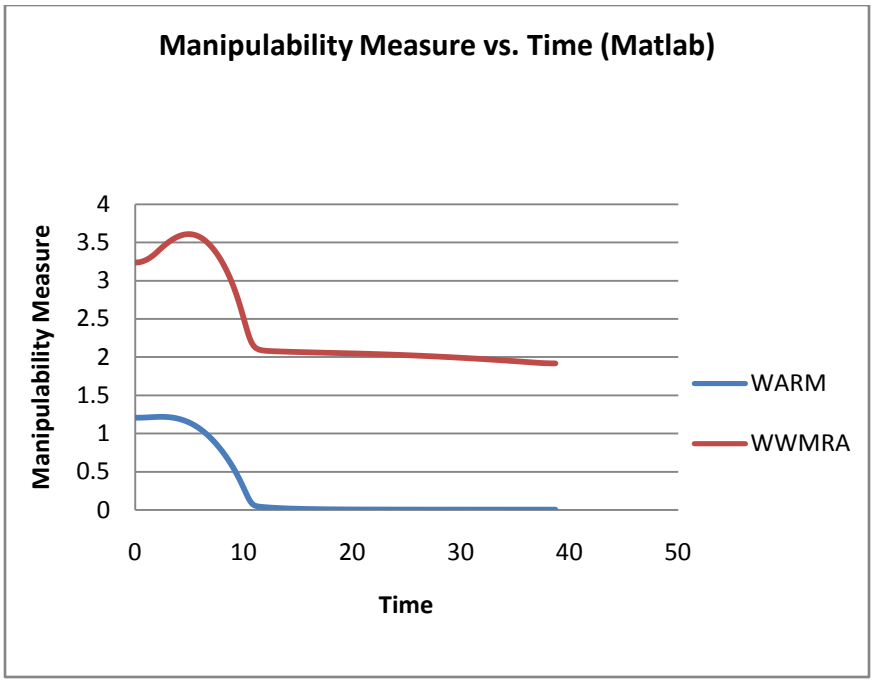

Figure 36. Manipulability Measure vs. Time, Matlab Algorithm

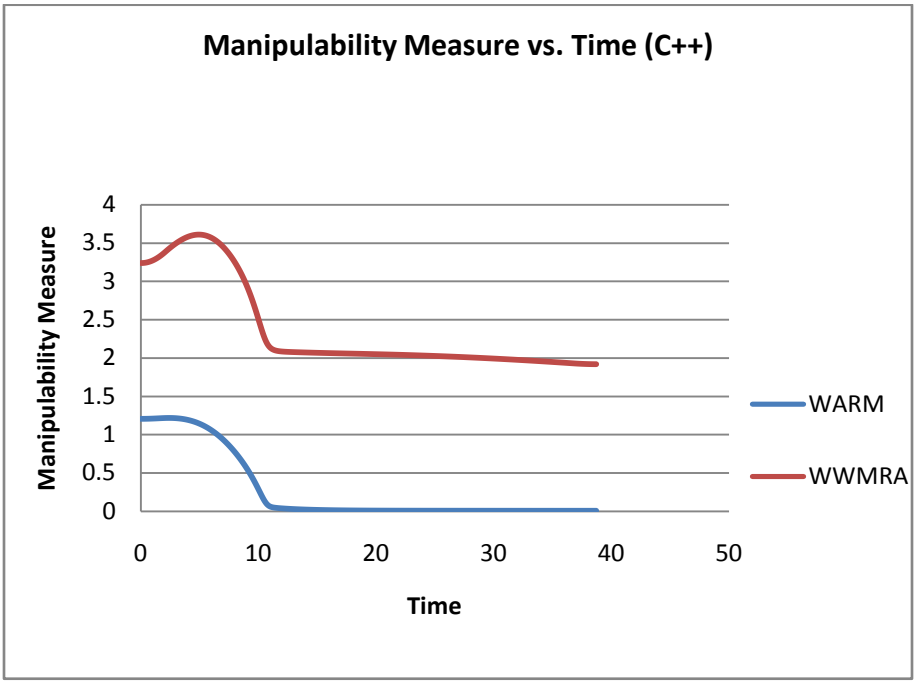

Figure 37. Manipulability Measure vs. Time, C++ Algorithm

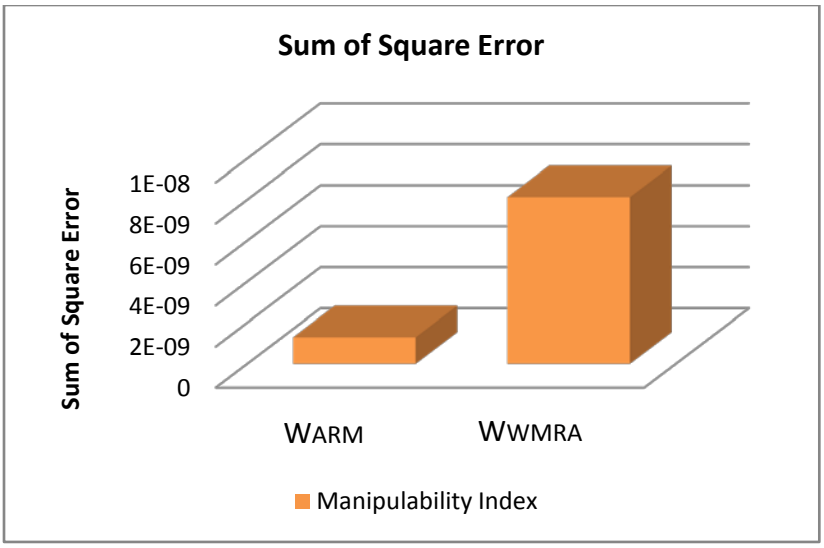

Figure 38. Sum of Square Error for Manipulability Index

In general, testing each function of the new control algorithm against its equivalent proved the new code to be as accurate as the old one in terms of calculations of variables. The sum of square error that was calculated for each function demonstrates that the values computed by the functions of the new control algorithm are close to the ones in Matlab. In most of the cases, the errors were introduced by the possible data loss that occurs while using the new code because of the manipulation of different variable types (i.e. float, char, double, etc).

Although comparing the calculations made by the control algorithms verified the proximity of the calculations, it is necessary to prove the overall accuracy and effectiveness of the system by confirming that WMRA behaves as commanded.

5.2. Processing Times

One of the goals for rewriting the control algorithm of the WMRA system was to reduce the processing times. Matlab is a powerful tool designed to perform computationally intensive tasks, but the speed of the computations is slower than that in a

low-level programming language. Also, due to the complex communication between the software and the hardware in the previous control algorithm, the time it took for the algorithm to process the computations and control the robotic arm at the same time was high.

By rewriting the control algorithm of the WMRA the processing times were expected to be significantly reduced. To prove that the speed of the program was, in fact, increased, the time it takes the program to run one iteration loop, as well as the total time it takes the algorithm to complete a task, were recorded. Nine trials were performed using each control algorithm; the average times are shown in table 1. For the trials, the arm was commanded to move 20 cm in any direction from an initial position.

Table 1. Average Computational Times

	C++	Matlab
Loop iteration time	12 milliseconds	32 milliseconds
Total time	6.4 seconds	11.15 seconds

The loop iteration times vary depending on the calculations done inside the loop. In the new control algorithm, the highest loop iteration time takes place when the calculated position is sent to the controller board of the WMRA system. The communication between the physical arm and the software lasts between 12 and 14 milliseconds approximately. A loop iteration that does not require communicating with the arm lasts less than 5 milliseconds, especially, when the joint limit avoidance and the joint obstacle avoidance options are disabled. When using the previous control algorithm of the WMRA, the average total time of the system to reach a desired position is almost double than that in C++. One loop iteration takes between 30 and 35 milliseconds in Matlab when communicating with the physical arm; otherwise, it lasts approximately 11

milliseconds. As seen, the new control algorithm is faster than its predecessor. It has been shown that the calculation times have been improved by the new control implementation.

5.3. New Algorithm's Accuracy and Repeatability

In the previous section, it was demonstrated that the new control algorithm is capable of computing the same variables as the previous algorithm with minimum error. Now, to prove the accuracy and repeatability of the real system, a series of tests was performed. In this section, the tests that were executed are going to be described and their results are going to be analyzed.

5.3.1. Repeatability Test

To measure repeatability, the arm was commanded to go from the ready position to a desired destination using position control. The ready position is assumed to have the joint angles as shown in table 2.

Table 2. Joint Angles for Ready Position in Radians

Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6	Joint 7
$\frac{\pi}{2}$	$\frac{\pi}{2}$	0	$\frac{\pi}{2}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$	0

The desired destination employed for testing was a point in space defined by the following transformation matrix. The coordinate frame used was based on the wheelchair's coordinate frame.

$$\begin{bmatrix} 0 & 0 & 1 & 700 \\ -1 & 0 & 0 & 200 \\ 0 & -1 & 0 & 700 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The first ready position was taken as the reference position and every time the arm was commanded to go to ready position again, the x, y and z position of the end-

effector were recorded, as well as an overall value of repeatability of the system. The ground zero position was chosen arbitrarily to measure all the trials with respect to the same origin. Figure 39 shows the ground reference used for the trials. The location of the end-effector at the ready position for the first trial was taken as the ground zero with an offset in the z direction. For measuring an estimate of the overall repeatability of the system, a laser was side mounted to the gripper. A blank white board was placed in front of the WMRA, where the laser point could be spotted and its position could be measured. The arrangement shown in figure 40 allowed calculating a value of the overall repeatability of the system between each trial.

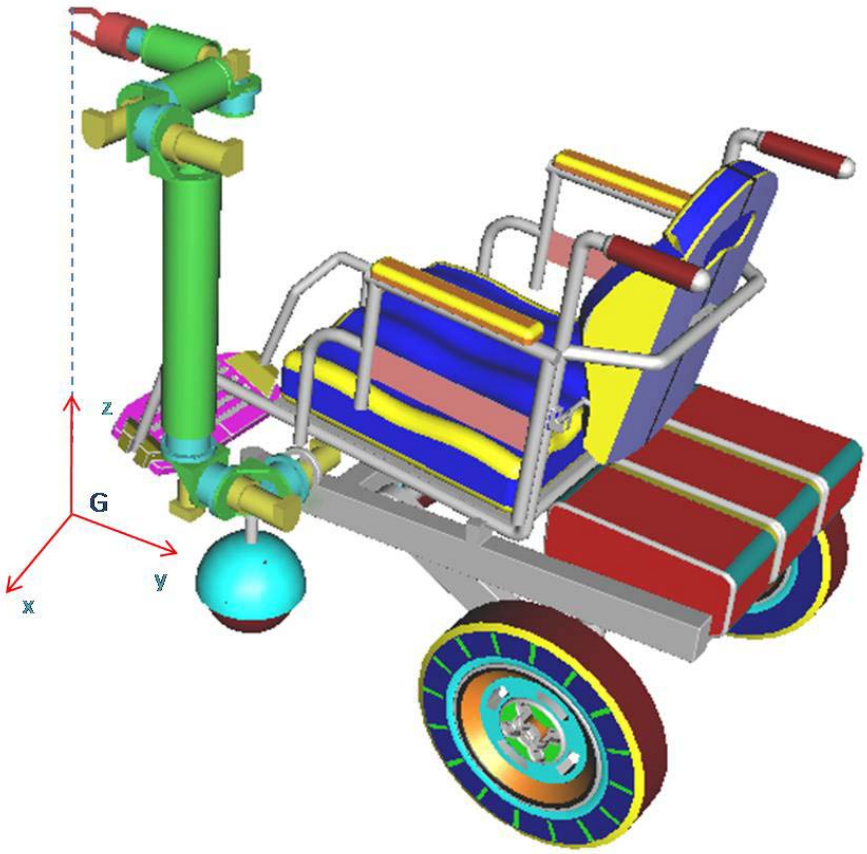

Figure 39. Ground Definition for Trials

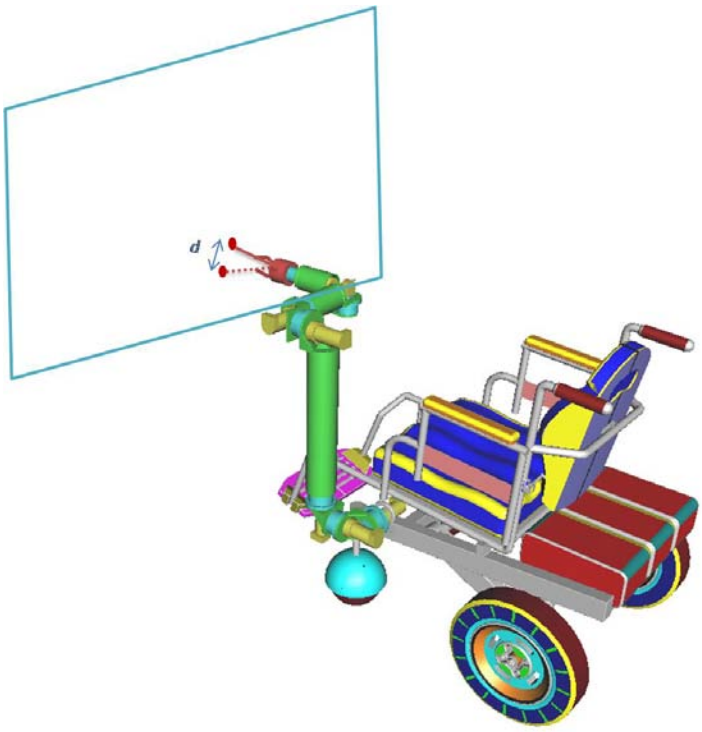

Figure 40. End-Effector's Laser Experiment

The measurements were divided in two sections. First, the arm was commanded to go from ready position to a target location, and then go back to ready. The first ready position was taken as reference for the following measurements related to ready arrangement precision. The first target position of the WMRA system was taken as reference for the following measurements relating target accuracy. The experiment had five trials. Table 3 presents the data recorded for the ready position.

d is the distance measured between two separate trials spotting dots with the side-mounted laser on the white board. This value does not represent any variable of the system; however, it represents an overall repeatability of the system. Since the system is commanded to go to the same position every time, the laser should always mark the same point. If this is not the case, this translates to an error in any of the six Cartesian variables (x, y, z, α, β, γ).

Table 3. X, Y, and Z Position of End-Effector at Ready Position

Trial	X Position	Y Position	Z Position	d
1	0"	0"	36 1/8"	0.00"
2	0"	0"	36 1/8"	1.13"
3	0"	0"	36 3/4"	3.00"
4	0"	0"	37 1/16"	3.75"
5	0"	0"	36 3/4"	3.00"
Average	**0"**	**0"**	**36.672"**	**2.72"**

As it can be noticed in table 3, the position of the end effector did not change along the x and y directions. However, the end-effector did not always return to the same location in the z direction. The error in the ready position was only introduced in the z direction because of the wobbling of the fifth joint of the arm. The movement of this joint is responsible for the position of the end-effector in the z direction in this particular position. This joint could have also introduced a small error in the x and y direction, but these are considerably smaller, and are not within the measurement resolution of this test (1/32”). Table 4 shows the absolute percentage error calculated using (3) for the z position of the end-effector. The average of the absolute percentage error indicates the WMRA is returning to the ready position within a two-percent error.

$$\%\epsilon = \frac{|v - v_{approx}|}{|v|} \times 100 \tag{3}$$

Table 4. Z Position Absolute Percentage Error

Trial	Z Position Absolute Percentage Error
1	Reference
2	0.00%
3	1.73%
4	2.60%
5	1.73%
Average	1.21%

Since the desired *d* is zero, an absolute comparison is not possible. Therefore, a Z-test was performed to determine the confidence interval in which this error will be. The interval is calculated using the confidence interval (4), where σ is the standard deviation, α is the confidence level and the *z* value for alpha equal 99% is 2.576.

$$\bar{x} - z_{\alpha/2} \cdot \frac{\sigma}{\sqrt{n}} \leq \mu \leq \bar{x} + z_{\alpha/2} \cdot \frac{\sigma}{\sqrt{n}} \tag{4}$$

The values of average and standard deviation of *d* at ready position are shown in table 5. These values were used in (4) to calculate the confidence interval of error for *d* at ready position.

Table 5. Average and Standard Deviation of d at Ready Position

	S (in)
Average	2.7188
STDV	1.5552

This is the confidence interval for *d* in inches at the ready position:

$$0.9271 \leq d \leq 4.3018$$

The previous interval indicates that the error for *d* will be between 0.5 and 3.1 inches with a 99% certainty. As said before, the repeatability at the ready position is affected by the wobbling of the fifth joint. To measure the repeatability of the system while moving towards a target position, a similar test was done; but the overall repeatability measurement that was calculated with the laser for the ready position was not taken into account for this test. The first target position was taken as the reference position (0, 0, 0). Table 6 shows the results obtained.

Table 6. X, Y, and Z Position of End-Effector at Target Position

Trial	X Position	Y Position	Z Position
1	0	0	0
2	5/16"	-1/16"	4/16"
3	4/16"	3/16"	-1/16"
4	5/16"	1/16"	-1/16"
Average	0.292"	0.063"	0.042"

Table 7. Standard Deviation at of X, Y, and Z Position at Target Position

	X Position	Y Position	Z Position
STDV	0.036	0.125	0.180

These are the confidence intervals for x, y, and z position in inches at the target position:

$$0.2380 \le x \le 0.3453$$

$$-0.1234 \le y \le 0.2484$$

$$-0.2267 \le z \le 0.3100$$

The errors between the reference position and the rest of the measurements, and the interval of confidence for the x, y, and z position, are considerably small. However, these small differences are mainly due to miscalculations done while measuring. Also, the wobbling of the fifth joint at the target position might considerably affect the errors in all of the directions. In spite of these errors, the new control algorithm of the WMRA system can be considered reasonably repeatable.

The repeatability test was performed using the old control algorithm to ensure the control of the system has been enhanced; however, the tests could not take place since the system crashed after one or two trials, making the tests inaccurate or the results impossible to compare.

5.3.2. Accuracy Test

In order to measure accuracy, the arm was commanded to go from the ready position to three different targets using position control. The ready position is assumed to have the joint angles shown in table 2. The arm was commanded to move 20 cm in each direction (forward, sideward, and upward; x, y, and z direction respectively) separately.

The transformation matrix represents the desired position and orientation of the end-effector from the control-based frame. The ready position of the system is determined by the following initial transformation matrix (in mm):

$$\begin{bmatrix} 0 & 0 & 1 & 586 \\ -1 & 0 & 0 & -131 \\ 0 & -1 & 0 & 549 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In order to move 20 cm forward (x direction), the target position was defined by:

$$\begin{bmatrix} 0 & 0 & 1 & 786 \\ -1 & 0 & 0 & -131 \\ 0 & -1 & 0 & 549 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To move 20 cm sideward (y direction) and upward (z direction), the final transformation matrices respectively were:

$$\begin{bmatrix} 0 & 0 & 1 & 586 \\ -1 & 0 & 0 & 69 \\ 0 & -1 & 0 & 549 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 & 586 \\ -1 & 0 & 0 & -131 \\ 0 & -1 & 0 & 749 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The difference between the final transformation matrix and the initial one gives the distance and direction the arm should move. The test was performed three times for each direction. The system is considered accurate if the physical traveled distance by the end-effector corresponds to the commanded one. The commanded distance is calculated by the algorithm, and as said before, it is determined by the difference between the final transformation matrix and the initial one. For all the trials, the algorithm calculated a travel distance of 200 mm (20 cm) in each direction respectively. To measure the physical traveled distance, a laser was side mounted to the gripper. Figures 41 and 42 show the set up for the accuracy test.

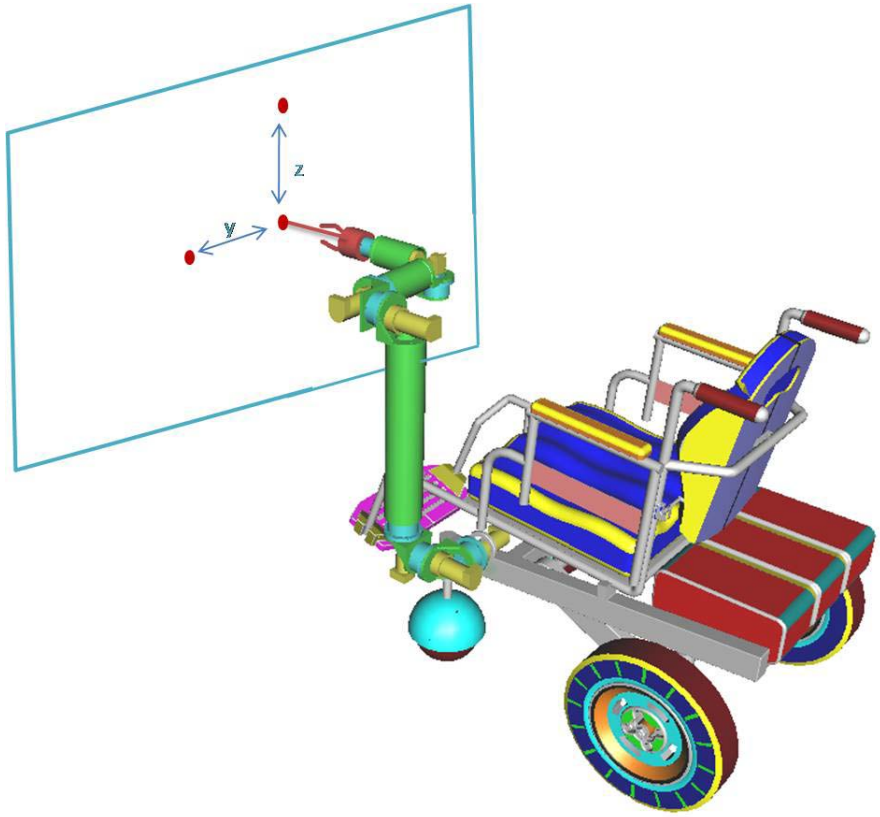

Figure 41. Setup of Accuracy Test on Y and Z Directions

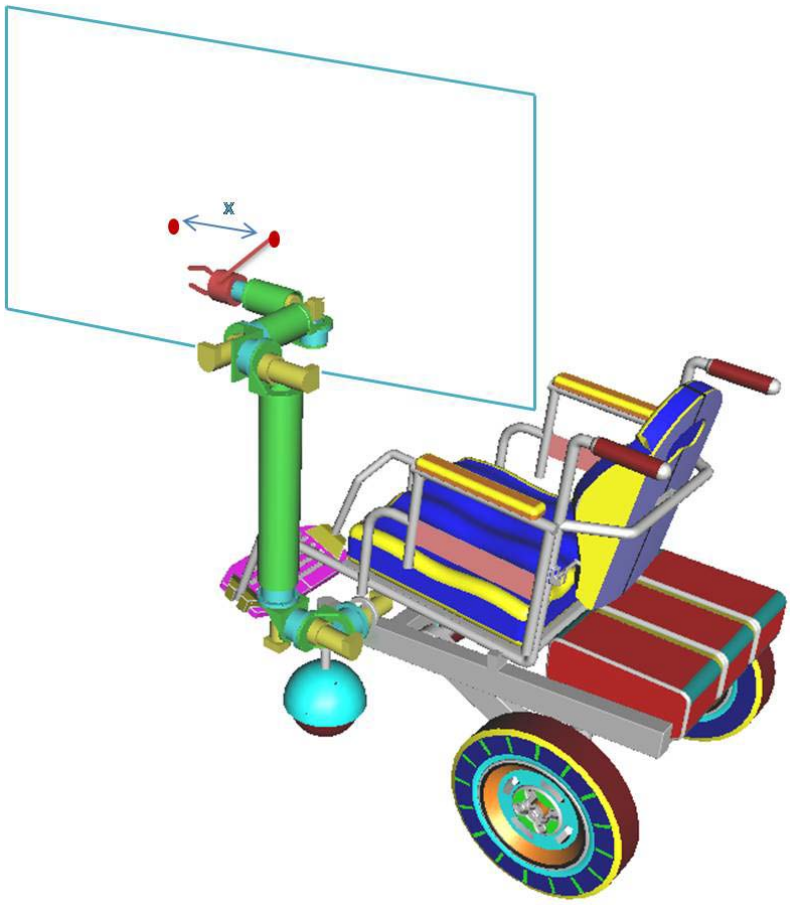

Figure 42. Setup of Accuracy Test on X Direction

The initial position (ready position) was marked on the white board. When the arm reached the desired position, it was also marked on the white board. The distance within the two marks was measured with a measuring tape. This distance should be 20 cm in x, or y, or z direction, depending on the commanded task. The recorded physical traveled distances are shown in table 8.

Table 8. Physical Traveled Distances Using Both Algorithms in cm

C++				Matlab			
Trial	x (cm)	y (cm)	z (cm)	Trial	x (cm)	y (cm)	z (cm)
1	20	19	20	1	19.5	22	22
2	20	20	21	2	20.5	19.5	24
3	19.5	19.5	21	3	19.5	19	25.5
Average	19.833	19.5	20.667	Average	19.833	20.167	23.833

The test was performed using both control algorithms to verify their accuracy. The following figures illustrate the data from the table above.

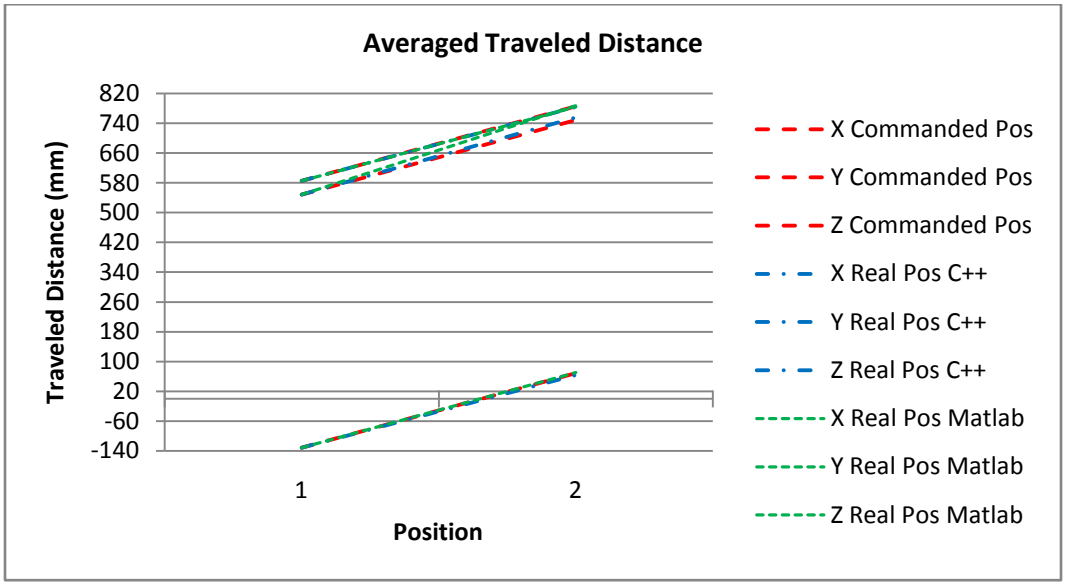

Figure 43. Averaged Traveled Distance

Figure 43 compares the averaged traveled distance using C++ and Matlab, plotted against the commanded distance. Position one is the distance of the initial position, and the final location of the gripper is the second position. Since the traveled distances are close to the commanded values, the error cannot be easily noticed in the figure. Therefore, figures 44, 45, and 46 present the averaged traveled distances in each direction.

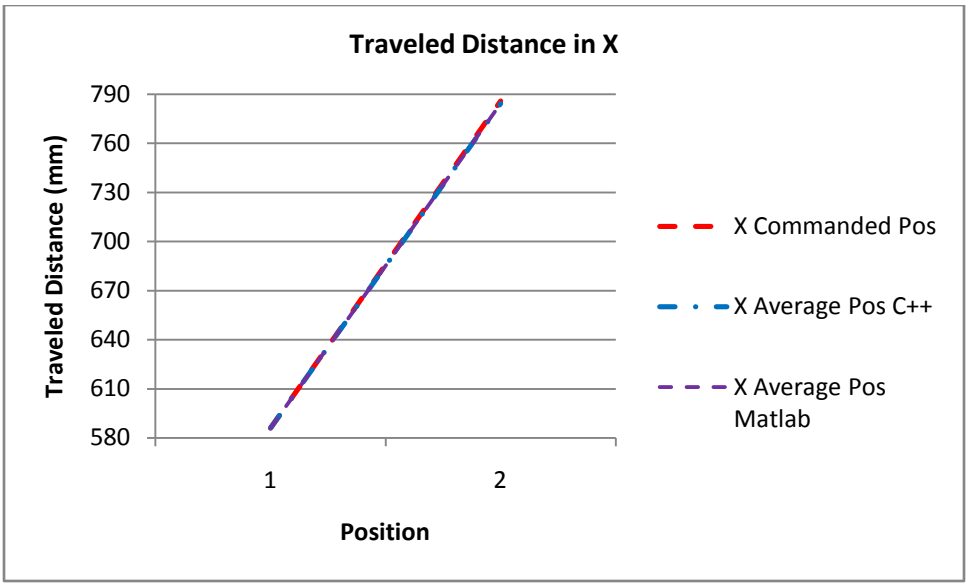

Figure 44. Traveled Distance in X Direction

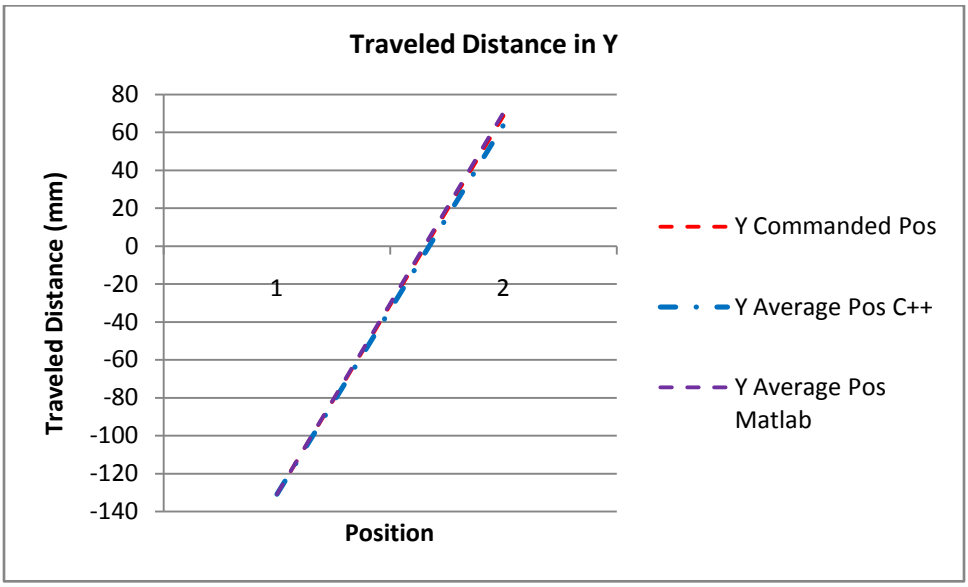

Figure 45. Traveled Distance in Y Direction

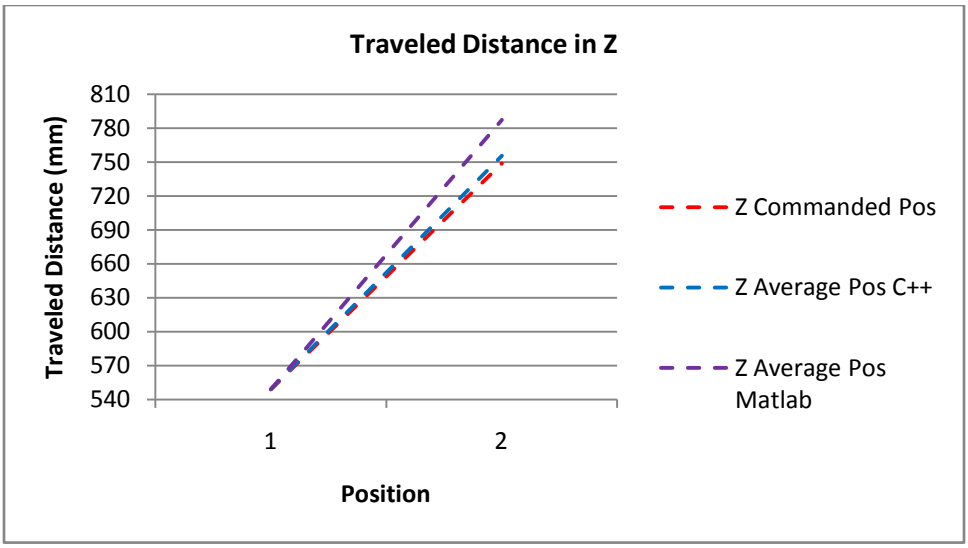

Figure 46. Traveled Distance in Z Direction

As the above figures show, the WMRA system is very accurate in the x and y directions while using both algorithms. Meanwhile, the new control algorithm is more accurate than its predecessor in the z direction. Although this test was performed in both algorithms to verify their accuracy, the goal of this test was not to compare accuracy between the two algorithms, but to prove the new control algorithm of the WMRA is accurate enough for the overall system to be considered accurate, precise, repeatable, and reliable.

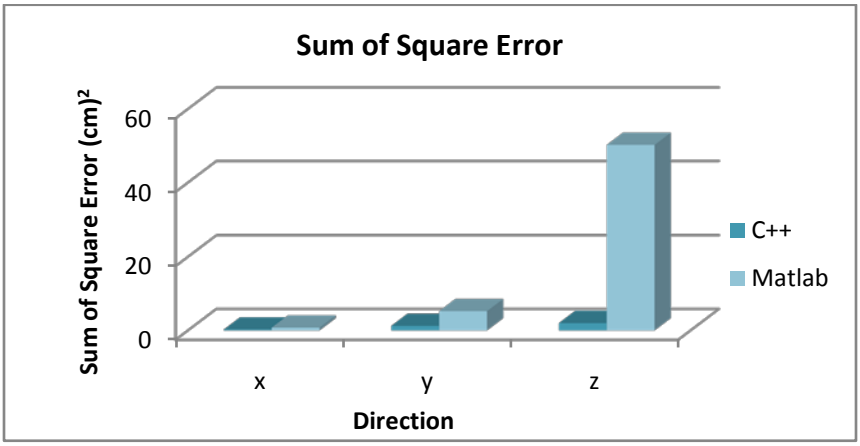

Figure 47. Sum of Square Errors for Traveled Distances

Figure 47 shows the sum of square error for the traveled distances. It can be appreciated that the error of the new control algorithm is small. The accuracy test demonstrated the new control algorithm of the WMRA to be precise.

5.4. Summary

This chapter discusses the different experiments made to confirm that the overall response of the control implementation was enhanced and that the algorithm routines and optimization procedures achieved the same goals with less computing requirements. Accuracy and repeatability tests were performed, and data was collected and presented. Results prove the new control algorithm to be reasonably accurate and the system to be responsive and reliable.

Chapter 6: Conclusions

A new control algorithm based on a generic language was developed for the Wheelchair-Mounted-Robotic-Arm. The algorithm was tested and analyzed to demonstrate that it improves upon the previous code in stability, reliability and efficiency.

The simulation control was separated from the physical arm control system for the latter to be converted to C++ language. By rewriting the control code of the WMRA, the communication delays were decreased since the interfaces between different programs and platforms to send data were no longer needed. The implementation and integration of the SpaceBall, a previously available but not properly working interface, expanded the interface capabilities of the system.

Preliminary tests were performed to demonstrate the stability and reliability of the new control algorithm. The overall response of the control implementation was enhanced and the algorithm routines and optimization procedures achieved the same goals with less computing requirements. The newly developed code was tested for accuracy and repeatability.

The new code, like its predecessor, is modular and easy to understand. These features allow future modifications of the code for improvement.

Chapter 7: Future Work

7.1. Testing Operating Systems and Real-Time Control

The experiments showed that new algorithm made the system more reliable and stable. The CPU usage and the time required to execute certain tasks were measured. However, to prove the code's versatility for future improvements, the algorithm might be tested using a different operating system. Since the code is written in a generic language, it might be used under an operating system other than Windows to control the physical arm. Real-time operating systems, such as QNX, allow the programmer to set priority rules to control the WMRA as well as any other software or hardware used by the computer. This would enhance the response time of the system without compromising its accuracy [4].

7.2. Implementation of the New Control Algorithm in the New WMRA Prototype

The new control algorithm might be implemented in the future in the new WMRA prototype which is still in development. Since the new code is an executable file that does not need any program to be launched, the new algorithm code is suitable for implementation in the new WMRA prototype. Future work would require developing a new DLL library for the new controller board. Besides the DLL library of the controller board, all of the functions and files of the new control algorithm can be implemented in the new prototype.

7.3. Human Testing

Human testing should be conducted using the new control algorithm to ensure the proper operation of the system. Both, human subjects with and without disabilities should be tested, to validate the aiding capabilities of the system. All the user interfaces might be tested as well, and the cognitive load should be addressed based on the different users and the type of interface they use.

7.4. Python

Python is a high-level programming language that features a dynamic-type system, automatic memory management, and supports multiple programming paradigms (object oriented, imperative, and functional) [31]. Python provides the required formatting to make programs more readable, it is easy to embed in and extend with other languages, and its programs are 2 to 10 times smaller than their C, C++ or JAVA counterparts [31].

Python is being used for GUI applications (it has bindings for many GUI toolkits) and vision recognition. Converting the control algorithm and the GUI to the Python language would allow better, faster, and more versatile control code that can be extended for vision recognition in the system, and other sensory fusion.

References

[1] US Census Bureau (2009) “Americans with Disabilities Act: July 26” http://www.census.gov/Press-Release/www/releases/pdf/cb09ff-13_disabilityact.pdf

[2] US Census Bureau (2002) “Americans with Disabilities: 2002 - Table 2” http://www.census.gov/hhes/www/disability/sipp/disab02/ds02t2.html, 2002

[3] R. Mahoney. “Robotic Products for Rehabilitation: Status and Strategy” In: Proceedings of ICORR ’97 – International Conference on Rehabilitation Robotics. Bath, UK, pp 12–22. http://www.bath.ac.uk/bime/icorrproc/mahoney1.pdf

[4] R. Alqasemi. “Maximizing Manipulation Capabilities of Persons with Disabilities Using a Smart 9-Degree-of-Freedom Wheelchair-Mounted Robotic Arm System”. University of South Florida, 2007.

[5] G. Biggs and B. MacDonald 2003. “A survey of robot programming systems”. In Proceedings of the Australasian Conference on Robotics and Automation. Brisbane, Australia.

[6] R. Murphy, “Introduction to AI Robotics”, MIT Press, 2nd edition, 2002.

[7] J. Allen, A. Karchak, and E. Bontrager, “Design and Fabrication of a Pair of Rancho Anthropomorphic Arms”, Technical Report of the Attending Staff Association of the Rancho Los Amigos Hospital Inc, 1972.

[8] Topping, M.J., “The development of handy-1, a robotic system to assist the severely disabled,” Proc ICORR ’99, pp. 244-249, 1999

[9] Topping, M., Heck, H., Bolmsjo, G., and Weightman, D., 1998, “The development of RAIL (Robotic Aid to Independent Living),” Proceedings of the third TIDE Congress.

[10] N. Katevas (Ed), “Mobile robotics in health care services,” IOS Press , pp. 227-251, Amsterdam, 2000.

[11] G. Bolmsjö, M. Olsson, P. Hedenborn, U. Lorentzon, F. Charnier, H. Nasri, “Modular robotics design - system integration of a robot for disabled people,” Proceedings of the EURISCON’98, Athens, 1998.

[12] H.F.M. Van der Loos, J.J. Wagne, N. Smaby, K. Chang. O. Madrigal, L.J. Leife, and O. Khatib. “ProVAR Assistive Robot System Architecture”. Proceedings of the 1999 IEEE International Conference on Robotics & Automation, Michigan, May 1999

[13] John L. Dallaway, Robin D. Jackson, and Paul H. A. Timmers. “Rehabilitation Robotics in Europe”. IEEE Transactions on Rehabilitation Engineering, Vol. 3, No. 1**,** March 1995

[14] Y. Chen, L. Liu, M. Zhang, and H. Rong. “Study on Coordinated Control and Hardware System of a Mobile Manipulator”. Proceedings of the 6th World Congress on Intelligent Control and Automation. 2006.

[15] Holly A. Yanco., “Integrating robotic research: a survey of robotic wheelchair development,” AAAI Spring Symposium on Integrating Robotic Research, Stanford, California, March 1998.

[16] M. Hamilton, “The Weston Wheelchair Mounted Assistive Robot - The Design Story”, Robotica, V. 20, pp. 125-132, 2002.

[17] H.Eftring, K.Boschian, “Technical results from manus user trials,” Proc. ICORR ’99, 136-141, 1999

[18] M. Hillman, A. Gammie, “The Bath institute of medical engineering assistive robot”, Proc. ICORR ’94, 211-212, 1994.

[19] C. Martens, O. Prenzel, and A. Gräser. “The Rehabilitation Robots FRIEND-I & II: Daily Life Independency through Semi-Autonomous Task-Execution”. I-Tech Education and Publishing. Vienna, Austria, pp: 137–162, 2007

[20] R. Alqasemi, E. McCaffrey, K. Edwards and R. Dubey, “Analysis, Evaluation and Development of Wheelchair-Mounted Robotic Arms.” Proceedings of the 2005 IEEE 9th International Conference on Rehabilitation Robotics, Chicago, IL, June 2005.

[21] R. Alqasemi and R. Dubey. “Maximizing Manipulation Capabilities for People with Disabilities Using a 9-DoF Wheelchair-Mounted Robotic Arm System”. Proceedings of the 2007 IEEE 10th International Conference on Rehabilitation Robotics.

[22] K. Edwards, R. Alqasemi and R. Dubey, “Wheelchair-Mounted Robotic Arms: Design and Development”. The First IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics 2006.

[23]] F. Farelo. “Task Oriented Simulation and Control of a Wheelchair Mounted Robotic Arm”. University of South Florida, 2009.

[24] P. Surana “Meta-Compilation of Language Abstractions”. Northwestern University, 2006.

[25] Stroustrup, Bjarne. The C++ Programming Language.Addison-Wesley, 1986.

[26] An Overview of C++. Bjarne Stroustrup. AT&T Bell Laboratories. Murray Hill, New Jersey 07974

[27] H.Schildt, “C++ From the Ground Up”, Second Edition, Osborne McGraw-Hill, 1998.

[28] J. R. Berryhill. “C++ Scientific Programming”. Wiley InterScience, 2001.

[29] C. H. Pappas and W. H. Murray. “Visual C++.Net: The Complete Reference” Third Edition. McGraw-Hill, 2001

[30] “Matrix TCL Pro *Matrix Algebra Made Easy”*. http://www.techsoftpl.com/matrix/

[31] V. Arora, “Python - Why Settle For Snake Oil When You Can Have The Whole Snake?” IEEE NTU Student Chapter. Available at: http://ewh.ieee.org/reg/10/epub/exclamations/articles_IEEE_NTU.pdf

Appendices

Appendix A: C++ Functions

A.1 Functions Listed Alphabetically

```
/* This new "USF WMRA" function commands the arm to go from any position to the ready
position. 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 --> Initialization, ini=2 or any --> Update, ini=3 --> Close.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Modified By: Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 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){
Matrix zero(1,1);
zero.Null(1,1);
	if (ini==3){
		if (arm==1){
			try {
		WMRA_ARM_Motion(ini, 0, zero, 0);
			}
			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:
	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];
	}
	ts=10;  // (5 or 10 or 20) 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; // Sending position to text file
        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);
                }
                WMRA_ARM_Motion(ini, 2, zero, dt);
                ddt=0;
        }
        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");

        // 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));
```

Appendix A (Continued)

```
                //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) {
                        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");

        // Joint angle change at every time step.
        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++){
                        fprintf(fel," %4.2f ",  qo(j,k));
                }
                fprintf(fel," \n");
        }
        fprintf(fel," \n\n\n");
```

Appendix A (Continued)

```
        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);
                                }
                                WMRA_ARM_Motion(2, 1, zero, ddt2);
                                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
                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");
        }
        fclose(fel);
}
```

```
/* 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 (.DLL) file that contains the used functions must be in the directory containing this
program.
config=0: Set the current encoder readings to zeros.
config=1: Read the encoder readings from the configuration txt file.
config=2: Change the configuration file to the initial values provided by (qo), then read
the encoder readings from the configuration txt file.

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

Function Declaration:*/

#include "matrix.h"
#include "WCD.h"
#include "ArmMotion.h"
#include "controlMotor.h"
#include <math.h>
using namespace std;
using namespace math;
using namespace System;

Matrix WMRA_ARM_Motion(int ind, int config, Matrix qo, float dt){
        long * ptr;
        long ptrtemp[10]={0.0};
        ptr = ptrtemp;

        extern float e2r1, e2r2, e2r3, e2r4, e2r5, e2r6, e2r7, e2r8, e2r9, e2d;

        FILE * fad;
        fad = fopen("qn.txt","a");
```

```
        Matrix qn(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){
                int com, baud, Kp, Kd, Ki;
                // Serial Communication properties:
                com=4;
                baud=19200;
                // PID controller gains:
                Kp=100;
                Kd=1000;
                Ki=0;

                //Conversion of encoder readings to radians: Note that the encoder
readings are negative of the kinematic arrangements in the control code.
                e2r1=-PI/900000;
                e2r2=-PI/900000;
                e2r3=-PI/950000;
                e2r4=-PI/710000;
                e2r5=-PI/580000;
                e2r6=-PI/440000;
                e2r7=-PI/630000;
                e2r8=1; // Change this to forward motion when wheelchair controllers are
installed (Only when reading the encoders).
                e2r9=1; // Change this to rotation motion when wheelchair controllers are
installed (Only when reading the encoders).
                e2d =-0.00001;

                // 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)=qo(9,0)/e2d;

                        // Changing the configuration file to qo:
                        FILE * fid;
                        fid = fopen("configuration.txt","w");
                        int j;
                        for (j=0;j<7;j++){
                                fprintf(fid," %g ",qo(j,0));
                        }
                        fclose (fid);
                        config=1;
                }
                // Establishing the connections, and setting the encoders to the current
configuration:
                int check;
                check=init(com, baud, config);
                if (check == 0){
                        printf("\nWMRA initialization has failed, Please check your
communications.\n");
                }

                cout<<"\n Conections checked\n";
```

```
                // Setting the PID controller gains (All motors the same gains in this
 case. Use 'setParamsPID' command to set each individual motor to different PID gains:
                setParamsPIDAll(Kp, Kd, Ki);
                cout<<"\n Parameters set\n";

                // Reading the current positions and converting them to radians:
                getPosAll(ptr);
                long qc[10];
                for (int e=0; e<10; e++)
                {
                        qc[e] = *ptr;
                        ptr++;
                }

                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;
                // Printing the current positions:
                fprintf(fad," qn is: \n\n");
                for (int j=0;j<qn.RowNo();j++){
                        fprintf(fad," %g ",  qn(j,0));
                        fprintf(fad," \n\n");
                }
        }
        // Closing the Arm library:
        else if (ind==3){
                // Reading the current positions to be saved in the configuration file:
                getPosAll(ptr);
                // Reporting the function output to be zero (This value will not be used):
                qn.Null(10,1);
        }

        // Updating the Arm:
        else {
                // Reading the current positions:
                getPosAll(ptr);
                long qc[10];
                for (int e=0; e<10; e++)
                {
                        qc[e] = *ptr;
                        ptr++;
                }
                // 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)=qo(9,0)/e2d;

                // Finding the needed velocities for the arm, note that a factor of 33.8
is needed for encoder velocities and position conversion:
                Matrix qdo(1,10), qddo(10,1);
                int j;
```

```
for (j=0;j<7;j++){
        qdo(0,j)=33.8*abs(qo(j,0)-qc[j])/dt;
}

// Finding the needed velocities for the wheels':
qdo(0,7)=33.8*abs(qo(7,0)-qc[7])/dt;
qdo(0,8)=33.8*abs(qo(8,0)-qc[8])/dt;

// Calculating the gripper's commanded position and velocity:
qo(9,0)=qo(9,0)+qc[9];
qdo(0,9)=33.8*abs(qo(9,0)-qc[9]);

// Setting the system's commanded accelerations:
qddo(0,0)=500;
qddo(1,0)=500;
qddo(2,0)=500;
qddo(3,0)=500;
qddo(4,0)=500;
qddo(5,0)=500;
qddo(6,0)=500;
qddo(7,0)=0;
qddo(8,0)=0;
qddo(9,0)=5000;

// Splitting the negative sign to be used in the DLL functions:

int dir[10]={0.0};  // Add two more zeros when wheelchair controllers are
installed.

for (j=0;j<10;j++){ // Change 8 to 10 when wheelchair controllers are
installed.
        if (Math::Sign(qo(j,0)) == -1){
                qo(j,0) = -qo(j,0);
                dir[j] = 1;
        }
}

unsigned long qotemp[10];
unsigned long qdotemp[10];
unsigned long qddotemp[10];
for (j=0;j<10;j++){
        qotemp[j] = qo(j,0);
        qdotemp[j] = qdo(0,j);
        qddotemp[j] = qddo(j,0);
}

int temp[11]={1,1,1,1,1,1,1,0,0,1,-1};

// Sending the commanded angles to the controller boards:
posSelect (temp, qotemp, qdotemp, qddotemp, dir);

// Reading the current positions and converting them to radians:
long ptrtemp2[10]={0.0};
ptr = ptrtemp2;
getPosAll (ptr);
for (int e=0; e<10; e++)
{
        qc[e] = *ptr;
        ptr++;
}
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;
                // Printing the current positions:
                fprintf(fad," qn is: \n\n");
                for (int j=0;j<qn.RowNo();j++){
                        fprintf(fad," %g ",  qn(j,0));
                        fprintf(fad," \n\n");
                }
        }
        fclose (fad);
        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 introduction to Robotics by John J. Craig

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% 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);
                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 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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){

        // Reading the Wheelchair's constant dimentions, all dimentions are converted in
millimeters:
        Matrix L(1,1);
```

Appendix A (Continued)

```
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);
}
// Collision with the wheelchair driver's left shoulder:
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 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

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

```
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 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function Declaration:*/

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

using namespace math;
using namespace std;

Matrix WMRA_DH(Matrix q){

        Matrix DH1(7,4);

//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,       110,  q(0,0)},
```

```
                                      { PI/2, 0,     146,  q(1,0)},
                                      {-PI/2, 0,  549,  q(2,0)},
                                      { PI/2, 0,  130,  q(3,0)},
                                      {-PI/2, 0,  241,  q(4,0)},
                                      { PI/2, 0,   0,   q(5,0)},
                                      {-PI/2, 0, 179+131, 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 returns the Jacobian Matrix and its determinant based on frame
0 of the USF WMRA, given the Transformation Matrices of each link.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Modified By: Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% 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;
        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);
```

Appendix A (Continued)

```
        T = T6 * T;
        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;
        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;
        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;
        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;
        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;

        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);
                }
        }
        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 returns 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);

	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 returns 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,-
100*PI/180,-200*PI/180};
	float qmaxtemp[7] =
{170*PI/180,170*PI/180,170*PI/180,170*PI/180,170*PI/180,100*PI/180,200*PI/180};
	int i;
	for (i=0; i < 7; i++){
		qmin(0,i) = qmintemp[i];
		qmax(0,i) = qmaxtemp[i];
	}
}
```

```
/* This function uses a linear function to find an equally-spaced trajectory points of a
variable "q" along a straight line, given the initial and final variable values and the
number of trajectory points.
The output is the variable position.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. 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;
	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 controls the WMRA system. Plots for 9 DOF are availabe. All
angles are in Radians.

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

#include "matrix.h"
#include "var_included.h" // Global variables header
#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;
#define PI 3.14159265
#define d2r PI/180 //Conversions from Degrees to Radians.
#define r2d 180/PI //Conversions from Radians to Degrees.

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;
```

Appendix A (Continued)

```
        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:;
                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:;
```

Appendix A (Continued)

```
                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:;
                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:;
```

Appendix A (Continued)

```
                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;
                }
                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){
                        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){
                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 {
		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));

	//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);
```

Appendix A (Continued)

```
        //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);


        Tiwc=Tiwco;

        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));
                }
                else if (coord==3){
                        D=sqrt(pow(Td(0,3),2) + pow(Td(1,3),2) + pow(Td(2,3),2));
                }
                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));
                }
                //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.

                // Calculating the Trajectory of the end effector, and once the trajectory
is calculated, we should redefine "Td" based on the ground frame:
                if (coord==2){
                        Tt=WMRA_traj(trajf,Tia,Td,n+1);
                        Td=Tiwc*Td;
                }
                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);
                }
        }
        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;
        }
        else if (cont==3){
                n=1;
        }
        else if (cont==4){
                //dt=0.05;
                //dx=WMRA_psy(port1);
                //dx*=(v*dt);
```

```
                //dg=dx(7);
                //dx.SetSize(6,1);
                //Td=Ti;
                //n=1;
        }
        else {
                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);

        // Initializing the WMRA:
        if (ini==0){   // When no "park" to "ready" motion required.
                // Initializing Virtual Reality Animation:
                if (vr==1){
                        //WMRA_VR_Animation(1, Towc, qo);
                }
                // Initializing Robot Animation in Matlab Graphics:
                if (ml==1){
                        //WMRA_ML_Animation(1, To, Td, Towc, T01, T12, T23, T34, T45, T56,
T67);
                }
                // Initializing the Physical Arm:
                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)=dg;
                        WMRA_ARM_Motion(1, 2, qotemp, 0);
                        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);
                if (arm==1){
                        ddt=0;
                }
        }


                // Re-Drawing the Animation:
                Matrix Joa(6,7), Jowc(1,1), Jowctemp(1,1), Joatemp(1,1);
                float detJoa, detJo, phi;
```

Appendix A (Continued)

```
		if (vr==1 || ml==1){
			//drawnow;
		}
		// Starting a timer:
		clock_t start, start2, end, end2;
		start=clock();
		start2=clock()/ CLOCKS_PER_SEC;

		cout<<"\nStart time in milliseconds is "<<start<<"\n\n";
		cout<<"\nStart time in seconds is "<<start2<<"\n\n";
		double timedif;

		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:
		while (i<=n+1){
			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);

			// 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));
			// 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);
					}
```

Appendix A (Continued)

```
                        }
                        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);
                        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;
                }

                //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);
                                }
                        }
                        Joatemp=~Jo;
                        Joatemp=Jo*Joatemp;
                        detJo=sqrt(Joatemp.Det());
                }

                // 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);
                                        }
                                        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];
                        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 paused","WMRA
PAUSED", 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');
                                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);
                        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;
                }
                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);
                        dx(1,0) = atoi (dx2);
                        dx(2,0) = atoi (dx3);
                        dx(3,0) = atof (dx4);
                        dx(4,0) = atof (dx5);
                        dx(5,0) = atof (dx6);
                        dg = atoi (dgx);
                        varscreenopn = atoi (varscreenop);
                        dx*=(v*dt);
                }

                // 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);
                }
                if (cart==2) {
                        dx.SetSize(3,1);
                }

                // Calculating the resolved rate with optimization:
                // Index input values for "optim": 1= SR-I & WLN, 2= P-I & WLN, 3=
SR-I & ENE, 4= P-I & ENE:

                if (WCA==2) {
                        dq.SetSize(7,1);
                        dq1.Null(7,1);
                        dq1=WMRA_Opt(optim, JLA, JLO, Joa, detJoa, dq, dx, dt, q1);
                        dq1.SetSize(9,1);
                }
                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);
                        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);
                }

                // Calculating the new Joint Angles:
                qn=q1+dq1;

                // 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);

                // 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);
                        }
                        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;
```

Appendix A (Continued)

```
                                // Re-calculating the new Transformation Matrices:

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

                        // 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;
                                        WMRA_ARM_Motion(2, 1, dqtemp, ddt);
                                        ddt=0;
                                }
                        }

                        // 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){}

                        // Updating the old values with the new values for the next
iteration:
                        tt=tt+dt;

                        // Stopping the simulation when the exit button is pressed:
                        if (cont==3 || cont==4 || cont==5){
                                if (varscreenopn == 1){
                                        n=n+1;
                                }
                                else { break; }
                        }
                        i++;
                }

                fclose(fid);
                end=clock();
                end2=clock()/ CLOCKS_PER_SEC;
                cout<<"\nEnd time in milliseconds is: "<<end<<"\n\n";
                cout<<"\nEnd time in seconds is: "<<end2<<"\n\n";
                cout<<"\nElapsed time in milliseconds is "<<(end-start)<<"\n\n";
                cout<<"\nElapsed time in seconds is "<<(end2-start2)<<"\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);
                                // 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);
                                        }
                        }

                        // 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);
                                }
                        }
                }
                return 0;
}
```

```
/* 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 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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);
                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;
                // 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);
```

Appendix A (Continued)

```
                                                // 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++){
                                        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;
                        mul1=Winv*Jotrans;
                        mul2=Jo*mul1;
                        mul=mul2;
                        detJo= sqrt(mul.Det());
                }
                dof=dx.RowNo();
        }

        // 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);
                }
                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;
                // 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);
                }
        }

        // Pseudo Inverse and Weighted Least Norm Optimization:
        else if (i==2){
                // 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);
                }
                else {
                        sf=0;
                }
                // Calculating the SR-Inverse of the Jacobian:
                Matrix sfm(2,2);
                sfm.Unit(dof);
                sfm*=(sf);
                mul= Jo * Jotrans;
                mul= mul + sfm;
                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;
                        mul*=(0.001);
                        dq=pinvJo*dx+mul*dH;
                }
                else {
                        unit.Unit(9);
                        mul=unit-pinvJo*Jo;
                        mul*=(0.001);
```

Appendix A (Continued)

```
                        dH.SetSize(9,1);
                        dq=pinvJo*dx+mul*dH;
                        dq(7,0)=dq(7,0)*L(0,4);
                }
        }
                // 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;
                        mul*=(0.001);
                        dq=pinvJo*dx+mul*dH;
                }
                else {
                        unit.Unit(9);
                        mul=unit-pinvJo*Jo;
                        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;
                                }
                        }
                }

                // 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;
                        }
                        dqmax(7,0)=100;
                        dqmax(8,0)=0.15;
                        dqmax*=(dt);     // Joint velocity limits when the time increment
is dt second.
                }
                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);
```

```
                        }
                }
        }
        return dq;
}
```

```
/* This function returns 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;
}
```

```
/* This new “USF WMRA” function commands the arm to go from the parking position to the
ready position. All angles are in Radians.
Parking position is assumed to be qi=[0;pi/2;0;pi;0;0;0] (radians).
Ready position is assumed to be qd=[pi/2;pi/2;0;pi/2;pi/2;pi/2;0] (radians).
ini=1 --> Initialization, ini=2 or any --> Update, ini=3 --> Close.

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

Function Declaration:*/

#include "matrix.h"
#include "park2ready.h"
#include "ArmMotion.h"
```

Appendix A (Continued)

```
#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){
Matrix zero(1,1);
zero.Null(1,1);
	if (ini==3){
		if (arm==1){
			try {
		    WMRA_ARM_Motion(ini, 0, zero, 0);
			}
			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;
	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.

	// 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);
		ddt=0;
	}

	// Initializing Virtual Reality Animation:
	if (vr==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);
        }

        // 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);

        qn.Null(7,1);
        while (tt <= ts) {

                // 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);
                        }
                }

                // 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);
					ddt2=0;
				}
				ddt=0;
				ddt=ddt2;
			}

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

			// 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;
		}
}
```

```
/* This function stores data from the USF WMRA with 9 DOFto be plotted.

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

Function Declaration:*/
```

Appendix A (Continued)

```
#include "matrix.h"
#include "Plots.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;

		// 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("Joint Angular Velocities.cvs","w");
                fprintf(fid," time \t qd1 \t qd2 \t qd3 \t qd4 \t qd5 \t qd6 \t qd7 \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 \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)
);
                }
                fclose(fid);

                FILE * fid1;
                fid1 = fopen("Wheels Track Velocities.cvs","w");
                fprintf(fid1," time \t qdl \t qdr \n");
                for (j=0; j<timep.RowNo(); j++){
                        fprintf(fid1," %g \t %g \t %g \n", timep(j,0) , qdl(j,0) ,
qdr(j,0));
                }
                fclose(fid1);

                FILE * fid2;
                fid2 = fopen("Joint Angular Displacements.cvs","w");
                fprintf(fid2," time \t q1 \t q2 \t q3 \t q4 \t q5 \t q6 \t q7 \n");
                for (j=0; j<timep.RowNo(); j++){
                        fprintf(fid2," %g \t %g \t %g \t %g \t %g \t %g \t %g \t %g \n",
timep(j,0) , q1(j,0) , q2(j,0) , q3(j,0) , q4(j,0) , q5(j,0) , q6(j,0) , q7(j,0) );
                }
                fclose(fid2);

                FILE * fid3;
                fid3 = fopen("Wheels Track Distances.cvs","w");
                fprintf(fid3," time \t qll \t qrr \n");
                for (j=0; j<timep.RowNo(); j++){
                        fprintf(fid3," %g \t %g \t %g \n", timep(j,0) , qll(j,0) ,
qrr(j,0));
                }
                fclose(fid3);

                FILE * fid4;
                fid4 = fopen("Hand Position.cvs","w");
                fprintf(fid4," time \t x \t y \t z \n");
                for (j=0; j<timep.RowNo(); j++){
                        fprintf(fid4," %g \t %g \t %g \t %g \n", timep(j,0) , x(j,0) ,
y(j,0), z(j,0));
                }
                fclose(fid4);

                FILE * fid5;
```

```
                fid5 = fopen("Hand Orientation.cvs","w");
                fprintf(fid5," time \t roll \t pitch \t yaw \n");
                for (j=0; j<timep.RowNo(); j++){
                        fprintf(fid5," %g \t %g \t %g \t %g \n", timep(j,0) , roll(j,0) ,
pitch(j,0), yaw(j,0));
                }
                fclose(fid5);

                FILE * fid6;
                fid6 = fopen("Arm Base Position.cvs","w");
                fprintf(fid6," time \t xc \t yc \t zc \n");
                for (j=0; j<timep.RowNo(); j++){
                        fprintf(fid6," %g \t %g \t %g \t %g \n", timep(j,0) , xc(j,0) ,
yc(j,0), zc(j,0));
                }
                fclose(fid6);

                FILE * fid7;
                fid7 = fopen("Arm Orientation.cvs","w");
                fprintf(fid7," time \t rollc \t pitchc \t yawc \n");
                for (j=0; j<timep.RowNo(); j++){
                        fprintf(fid7," %g \t %g \t %g \t %g \n", timep(j,0) , rollc(j,0) ,
pitchc(j,0), yawc(j,0));
                }
                fclose(fid7);

                FILE * fid8;
                fid8 = fopen("Manipulability Measure.cvs","w");
                fprintf(fid8," time \t detJoa \t detJo \n");
                for (j=0; j<timep.RowNo(); j++){
                        fprintf(fid8," %g \t %g \t %g \n", timep(j,0) , detJoap(j,0) ,
detJop(j,0));
                }
                fclose(fid8);
        }
}
```

```
/*  This function uses a 3rd order Polynomial with no Blending factor to find a smooth
trajectory points of a variable "q" along a straight 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;
}
```

```
/* This function returns 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;
}
```

```
/* This new “USF WMRA” function commands the arm to go from the ready position to any
position. All angles are in Radians.
Ready position is assumed to be qd=[pi/2;pi/2;0;pi/2;pi/2;pi/2;0] (radians).
ini=1 --> Initialization, ini=2 or any --> Update, ini=3 --> Close.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Modified By: Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% 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){
Matrix zero(1,1);
zero.Null(1,1);
        if (ini==3){
                if (arm==1){
                        try {
             WMRA_ARM_Motion(ini, 0, zero, 0);
                        }
                        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];
        }
        ts=10;  // (5 or 10 or 20) 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);
                ddt=0;
                qi.SetSize(7,1);
        }

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

                // 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);
                                ddt=0;
                                qn.SetSize(7,1);
                        }
                        ddt=0;
                        ddt=ddt2;
                }

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

                // 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;
        }
}
```

```
/* This new "USF WMRA" function commands the arm to go from the ready position to the 
parking position. All angles are in Radians.
Parking position is assumed to be qi=[0;pi/2;0;pi;0;0;0] (radians).
Ready position is assumed to be qd=[pi/2;pi/2;0;pi/2;pi/2;pi/2;0]] (radians).
ini=1 --> Initialization, ini=2 or any --> Update, ini=3 --> Close.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Redwan M. Alqasemi %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Modified By: Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% 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){
Matrix zero(1,1);
zero.Null(1,1);
        if (ini==3){
                if (arm==1){
                        try {
            WMRA_ARM_Motion(ini, 0, zero, 0);
```

```
                        }
                        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=10;  // (5 or 10 or 20) 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.

        // 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);
                ddt=0;
        }

        // Initializing Virtual Reality Animation:
        if (vr==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
```

Appendix A (Continued)

```
                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;

        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) {

                // 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);
                        }
                }
                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);
                }

                // 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);
                                ddt2=0;
                        }
```

```
                        ddt=0;
                        ddt=ddt2;
                }

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

                // 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;
        }
}
```

```
/* This function returns 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);
        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 returns 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 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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 returns 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 returns the Roll, Pitch, and Yaw 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:

        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 returns 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
                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;
        }
}
```

```
/* This function finds the trajectory points along a straight 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){
                        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;
        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);
                }
        }
}

/*//Rotational Trajectory:
// Single-angle Change:

da=2*PI/(n-1);
kx=1;
ky=0;
kz=0;
s=sin(da);
c=cos(da);
v=1-c;

//Rotation and Position Change:

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

Matrix Tti1(3,3),Tti(4,4);
for (i=1; i<n; i++){
        Titemp=~Titemp;
        dR= dR ^ (i)
        Tti1 = Titemp * dR;
        for ( m=0 ; m < 3 ; m++ ) {
                for ( j = 0 ; j < 3 ; j++ ) {
                        Tti(m,j)=Tti1(m,j);
                }
        }
        Tti(0,3)=Ti(0,3)+2000*cos(i*da);
        Tti(1,3)=Ti(1,3)+2000*sin(i*da);
        Tti(2,3)=Ti(2,3);
        Tti(3,0)=0;
        Tti(3,1)=0;
        Tti(3,2)=0;
```

```
                Tti(3,3)=1;

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

```
/* This function returns 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;
}
```

```
/* This function returns the cross product of two vectors.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Thanks to: W. Garret Pence %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Modified By: Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#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 returns 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 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Thanks to: W. Garret Pence %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%% 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);
        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;
}
```

```
/* This function returns 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 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Thanks to: W. Garret Pence %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% 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] = {560,440,230,182,168};
        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).
```

Appendix A (Continued)

```
        //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;
}
```

A.2 Global Variables Header File

```
/* This header file contains all the global variables.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Thanks to: W. Garret Pence %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Modified By: Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

// Global Variables

#ifndef VAR_INCLUDED
#define VAR_INCLUDED

#include "matrix.h"

using namespace std;
using namespace math;

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

Matrix dHo(7,1);
Matrix DH(7,4);
float e2r1, e2r2, e2r3, e2r4, e2r5, e2r6, e2r7, e2r8, e2r9, e2d;
Matrix timep(1,1), q1(1,1), q2(1,1), q3(1,1), q4(1,1), q5(1,1), q6(1,1), q7(1,1),
qll(1,1), qrr(1,1), qd1(1,1), qd2(1,1), qd3(1,1), qd4(1,1), qd5(1,1), qd6(1,1), qd7(1,1),
qdl(1,1), qdr(1,1), x(1,1), y(1,1), z(1,1), roll(1,1), pitch(1,1), yaw(1,1), xc(1,1),
yc(1,1), zc(1,1), rollc(1,1), pitchc(1,1), yawc(1,1), detJoap(1,1), detJop(1,1);
int varscreenopn;
int exitvar;

#endif
```

Appendix A (Continued)

A.3 Matrix Library

```
////////////////////////////////
// Matrix TCL Lite v1.13
// Copyright (c) 1997-2002 Techsoft Pvt. Ltd. (See License.Txt file.)
//
// Matrix.h: Matrix C++ template class include file
// Web: http://www.techsoftpl.com/matrix/
// Email: matrix@techsoftpl.com
//

////////////////////////////////
// Installation:
//
// Copy this "matrix.h" file into include directory of your compiler.
//

////////////////////////////////
// Note: This matrix template class defines majority of the matrix
// operations as overloaded operators or methods. It is assumed that
// users of this class is familiar with matrix algebra. We have not
// defined any specialization of this template here, so all the instances
// of matrix will be created implicitly by the compiler. The data types
// tested with this class are float, double, long double, complex<float>,
// complex<double> and complex<long double>. Note that this class is not
// optimized for performance.
//
// Since implementation of exception, namespace and template are still
// not standardized among the various (mainly old) compilers, you may
// encounter compilation error with some compilers. In that case remove
// any of the above three features by defining the following macros:
//
//  _NO_NAMESPACE:  Define this macro to remove namespace support.
//
//  _NO_EXCEPTION:  Define this macro to remove exception handling
//                  and use old style of error handling using function.
//
//  _NO_TEMPLATE:   If this macro is defined matrix class of double
//                  type will be generated by default. You can also
//                  generate a different type of matrix like float.
//
//  _SGI_BROKEN_STL: For SGI C++ v.7.2.1 compiler.
//
//  Since all the definitions are also included in this header file as
//  inline function, some compiler may give warning "inline function
//  can't be expanded". You may ignore/disable this warning using compiler
//  switches. All the operators/methods defined in this class have their
//  natural meaning except the followings:
//
//  Operator/Method                          Description
//  ---------------                          -----------
//   operator ()   :   This function operator can be used as a
//                     two-dimensional subscript operator to get/set
//                     individual matrix elements.
//
//   operator !    :   This operator has been used to calculate inversion
//                     of matrix.
//
//   operator ~    :   This operator has been used to return transpose of
//                     a matrix.
//
//   operator ^    :   It is used calculate power (by a scalar) of a matrix.
//                     When using this operator in a matrix equation, care
//                     must be taken by parenthesizing it because it has
```

Appendix A (Continued)

```
//                    lower precedence than addition, subtraction,
//                    multiplication and division operators.
//
//   operator >>   :  It is used to read matrix from input stream as per
//                    standard C++ stream operators.
//
//   operator <<   :  It is used to write matrix to output stream as per
//                    standard C++ stream operators.
//
// Note that professional version of this package, Matrix TCL Pro 2.11
// is optimized for performance and supports many more matrix operations.
// It is available from our web site at <http://www.techsoftpl.com/matrix/>.
//

#ifndef __cplusplus
#error Must use C++ for the type matrix.
#endif

#if !defined(__STD_MATRIX_H)
#define __STD_MATRIX_H

////////////////////////////
// First deal with various shortcomings and incompatibilities of
// various (mainly old) versions of popular compilers available.
//

#if defined(__BORLANDC__)
#pragma option -w-inl -w-pch
#endif

#  include <stdio.h>
#  include <stdlib.h>
#  include <math.h>
#  include <iostream>
#  include <string.h>


#if defined(_MSC_VER) && _MSC_VER <= 1000
#  define _NO_EXCEPTION        // stdexception is not fully supported in MSVC++ 4.0
typedef int bool;
#  if !defined(false)
#    define false  0
#  endif
#  if !defined(true)
#    define true   1
#  endif
#endif

#if defined(__BORLANDC__) && !defined(__WIN32__)
#  define _NO_EXCEPTION        // std exception and namespace are not fully
#  define _NO_NAMESPACE        // supported in 16-bit compiler
#endif

#if defined(_MSC_VER) && !defined(_WIN32)
#  define _NO_EXCEPTION
#endif

#if defined(_NO_EXCEPTION)
#  define _NO_THROW
#  define _THROW_MATRIX_ERROR
#else
#  if defined(_MSC_VER)
#    if _MSC_VER >= 1020
#      include <stdexcept>
#    else
#      include <stdexcpt.h>
```

```
#    endif
#  elif defined(__MWERKS__)
#      include <stdexcept>
#  elif (__GNUC__ >= 2 || (__GNUC__ == 2 && __GNUC_MINOR__ >= 8))
#     include <stdexcept>
#  else
#     include <stdexcep>
#  endif
#  define _NO_THROW               throw ()
#  define _THROW_MATRIX_ERROR     throw (matrix_error)
#endif

#ifndef __MINMAX_DEFINED
#  define max(a,b)    (((a) > (b)) ? (a) : (b))
#  define min(a,b)    (((a) < (b)) ? (a) : (b))
#endif

#if defined(_MSC_VER)
#undef _MSC_EXTENSIONS      // To include overloaded abs function definitions!
#endif

/*#if ( defined(__BORLANDC__) || _MSC_VER ) && !defined( __GNUG__ )
inline float abs (float v) { return (float)fabs( v); }
inline double abs (double v) { return fabs( v); }
inline long double abs (long double v) { return fabsl( v); }
#endif*/

#if defined(__GNUG__) || defined(__MWERKS__) || (defined(__BORLANDC__) && (__BORLANDC__
>= 0x540))
#define FRIEND_FUN_TEMPLATE <>
#else
#define FRIEND_FUN_TEMPLATE
#endif

#if defined(_MSC_VER) && _MSC_VER <= 1020   // MSVC++ 4.0/4.2 does not
#  define _NO_NAMESPACE                     // support "std" namespace
#endif

#if !defined(_NO_NAMESPACE)
#if defined( _SGI_BROKEN_STL )              // For SGI C++ v.7.2.1 compiler
namespace std { }
#endif
using namespace std;
#endif

#ifndef _NO_NAMESPACE
namespace math {
#endif

#if !defined(_NO_EXCEPTION)
class matrix_error : public logic_error
{
    public:
       matrix_error (const string& what_arg) : logic_error( what_arg) {}
};
#define REPORT_ERROR(ErrormMsg)  throw matrix_error( ErrormMsg);
#else
inline void _matrix_error (const char* pErrMsg)
{
    cout << pErrMsg << endl;
    exit(1);
}
#define REPORT_ERROR(ErrormMsg)  _matrix_error( ErrormMsg);
#endif

#if !defined(_NO_TEMPLATE)
```

```
#  define MAT_TEMPLATE  template <class T>
#  define matrixT  matrix<T>
#else
#  define MAT_TEMPLATE
#  define matrixT  matrix
#  ifdef MATRIX_TYPE
     typedef MATRIX_TYPE T;
#  else
     typedef double T;
#  endif
#endif


MAT_TEMPLATE
class matrix
{
public:
   // Constructors
   matrix (const matrixT& m);
   matrix (size_t row = 6, size_t col = 6);

   // Destructor
   ~matrix ();

   // Assignment operators
   matrixT& operator = (const matrixT& m) _NO_THROW;

   // Value extraction method
   size_t RowNo () const { return _m->Row; }
   size_t ColNo () const { return _m->Col; }

   // Subscript operator
   T& operator () (size_t row, size_t col) _THROW_MATRIX_ERROR;
   T  operator () (size_t row, size_t col) const _THROW_MATRIX_ERROR;

   // Unary operators
   matrixT operator + () _NO_THROW { return *this; }
   matrixT operator - () _NO_THROW;

   // Combined assignment - calculation operators
   matrixT& operator += (const matrixT& m) _THROW_MATRIX_ERROR;
   matrixT& operator -= (const matrixT& m) _THROW_MATRIX_ERROR;
   matrixT& operator *= (const matrixT& m) _THROW_MATRIX_ERROR;
   matrixT& operator *= (const T& c) _NO_THROW;
   matrixT& operator /= (const T& c) _NO_THROW;
   matrixT& operator ^= (const size_t& pow) _THROW_MATRIX_ERROR;

   // Miscellaneous -methods
   void Null (const size_t& row, const size_t& col) _NO_THROW;
   void Null () _NO_THROW;
   void Unit (const size_t& row) _NO_THROW;
   void Unit () _NO_THROW;
   void SetSize (size_t row, size_t col) _NO_THROW;

   // Utility methods
   matrixT Solve (const matrixT& v) const _THROW_MATRIX_ERROR;
   matrixT Adj () _THROW_MATRIX_ERROR;
   matrixT Inv () _THROW_MATRIX_ERROR;
   T Det () const _THROW_MATRIX_ERROR;
   T Norm () _NO_THROW;
   T Cofact (size_t row, size_t col) _THROW_MATRIX_ERROR;
   T Cond () _NO_THROW;

   // Type of matrices
   bool IsSquare () _NO_THROW { return (_m->Row == _m->Col); }
   bool IsSingular () _NO_THROW;
```

Appendix A (Continued)

```
   bool IsDiagonal () _NO_THROW;
   bool IsScalar () _NO_THROW;
   bool IsUnit () _NO_THROW;
   bool IsNull () _NO_THROW;
   bool IsSymmetric () _NO_THROW;
   bool IsSkewSymmetric () _NO_THROW;
   bool IsUpperTriangular () _NO_THROW;
   bool IsLowerTriangular () _NO_THROW;

private:
    struct base_mat
    {
        T **Val;
        size_t Row, Col, RowSiz, ColSiz;
        int Refcnt;

        base_mat (size_t row, size_t col, T** v)
        {
            Row = row; RowSiz = row;
            Col = col; ColSiz = col;
            Refcnt = 1;

            Val = new T* [row];
            size_t rowlen = col * sizeof(T);

            for (size_t i=0; i < row; i++)
            {
                Val[i] = new T [col];
                if (v) memcpy( Val[i], v[i], rowlen);
            }
        }
        ~base_mat ()
        {
            for (size_t i=0; i < RowSiz; i++)
                delete [] Val[i];
            delete [] Val;
        }
    };
    base_mat *_m;

    void clone ();
    void realloc (size_t row, size_t col);
    int pivot (size_t row);
};

#if defined(_MSC_VER) && _MSC_VER <= 1020
#  undef  _NO_THROW               // MSVC++ 4.0/4.2 does not support
#  undef  _THROW_MATRIX_ERROR     // exception specification in definition
#  define _NO_THROW
#  define _THROW_MATRIX_ERROR
#endif

// constructor
MAT_TEMPLATE inline
matrixT::matrix (size_t row, size_t col)
{
  _m = new base_mat( row, col, 0);
}

// copy constructor
MAT_TEMPLATE inline
matrixT::matrix (const matrixT& m)
{
    _m = m._m;
    _m->Refcnt++;
}
```

Appendix A (Continued)

```
// Internal copy constructor
MAT_TEMPLATE inline void
matrixT::clone ()
{
    _m->Refcnt--;
    _m = new base_mat( _m->Row, _m->Col, _m->Val);
}

// destructor
MAT_TEMPLATE inline
matrixT::~matrix ()
{
   if (--_m->Refcnt == 0) delete _m;
}

// assignment operator
MAT_TEMPLATE inline matrixT&
matrixT::operator = (const matrixT& m) _NO_THROW
{
    m._m->Refcnt++;
    if (--_m->Refcnt == 0) delete _m;
    _m = m._m;
    return *this;
}

//  reallocation method
MAT_TEMPLATE inline void
matrixT::realloc (size_t row, size_t col)
{
   if (row == _m->RowSiz && col == _m->ColSiz)
   {
      _m->Row = _m->RowSiz;
      _m->Col = _m->ColSiz;
      return;
   }

   base_mat *m1 = new base_mat( row, col, NULL);
   size_t colSize = min(_m->Col,col) * sizeof(T);
   size_t minRow = min(_m->Row,row);

   for (size_t i=0; i < minRow; i++)
      memcpy( m1->Val[i], _m->Val[i], colSize);

   if (--_m->Refcnt == 0)
       delete _m;
   _m = m1;

   return;
}

// public method for resizing matrix
MAT_TEMPLATE inline void
matrixT::SetSize (size_t row, size_t col) _NO_THROW
{
   size_t i,j;
   size_t oldRow = _m->Row;
   size_t oldCol = _m->Col;

   if (row != _m->RowSiz || col != _m->ColSiz)
      realloc( row, col);

   for (i=oldRow; i < row; i++)
      for (j=0; j < col; j++)
        _m->Val[i][j] = T(0);

   for (i=0; i < row; i++)
```

```
      for (j=oldCol; j < col; j++)
        _m->Val[i][j] = T(0);

   return;
}

// subscript operator to get/set individual elements
MAT_TEMPLATE inline T&
matrixT::operator () (size_t row, size_t col) _THROW_MATRIX_ERROR
{
   if (row >= _m->Row || col >= _m->Col)
      REPORT_ERROR( "matrixT::operator(): Index out of range!");
   if (_m->Refcnt > 1) clone();
   return _m->Val[row][col];
}

// subscript operator to get/set individual elements
MAT_TEMPLATE inline T
matrixT::operator () (size_t row, size_t col) const _THROW_MATRIX_ERROR
{
   if (row >= _m->Row || col >= _m->Col)
      REPORT_ERROR( "matrixT::operator(): Index out of range!");
   return _m->Val[row][col];
}

// input stream function
MAT_TEMPLATE inline istream&
operator >> (istream& istrm, matrixT& m)
{
   for (size_t i=0; i < m.RowNo(); i++)
      for (size_t j=0; j < m.ColNo(); j++)
      {
         T x;
         istrm >> x;
         m(i,j) = x;
      }
   return istrm;
}

// output stream function
MAT_TEMPLATE inline ostream&
operator << (ostream& ostrm, const matrixT& m)
{
   for (size_t i=0; i < m.RowNo(); i++)
   {
      for (size_t j=0; j < m.ColNo(); j++)
      {
         T x = m(i,j);
         ostrm << x << '\t';
      }
      ostrm << endl;
   }
   return ostrm;
}


// logical equal-to operator
MAT_TEMPLATE inline bool
operator == (const matrixT& m1, const matrixT& m2) _NO_THROW
{
   if (m1.RowNo() != m2.RowNo() || m1.ColNo() != m2.ColNo())
      return false;

   for (size_t i=0; i < m1.RowNo(); i++)
      for (size_t j=0; j < m1.ColNo(); j++)
             if (m1(i,j) != m2(i,j))
```

```
                return false;

   return true;
}

// logical no-equal-to operator
MAT_TEMPLATE inline bool
operator != (const matrixT& m1, const matrixT& m2) _NO_THROW
{
    return (m1 == m2) ? false : true;
}

// combined addition and assignment operator
MAT_TEMPLATE inline matrixT&
matrixT::operator += (const matrixT& m) _THROW_MATRIX_ERROR
{
   if (_m->Row != m._m->Row || _m->Col != m._m->Col)
      REPORT_ERROR( "matrixT::operator+= : Inconsistent matrix sizes in addition!");
   if (_m->Refcnt > 1) clone();
   for (size_t i=0; i < m._m->Row; i++)
      for (size_t j=0; j < m._m->Col; j++)
         _m->Val[i][j] += m._m->Val[i][j];
   return *this;
}

// combined subtraction and assignment operator
MAT_TEMPLATE inline matrixT&
matrixT::operator -= (const matrixT& m) _THROW_MATRIX_ERROR
{
   if (_m->Row != m._m->Row || _m->Col != m._m->Col)
      REPORT_ERROR( "matrixT::operator-= : Inconsistent matrix sizes in subtraction!");
   if (_m->Refcnt > 1) clone();
   for (size_t i=0; i < m._m->Row; i++)
      for (size_t j=0; j < m._m->Col; j++)
         _m->Val[i][j] -= m._m->Val[i][j];
   return *this;
}

// combined scalar multiplication and assignment operator
MAT_TEMPLATE inline matrixT&
matrixT::operator *= (const T& c) _NO_THROW
{
    if (_m->Refcnt > 1) clone();
    for (size_t i=0; i < _m->Row; i++)
       for (size_t j=0; j < _m->Col; j++)
           _m->Val[i][j] *= c;
    return *this;
}

// combined matrix multiplication and assignment operator
MAT_TEMPLATE inline matrixT&
matrixT::operator *= (const matrixT& m) _THROW_MATRIX_ERROR
{
   if (_m->Col != m._m->Row)
      REPORT_ERROR( "matrixT::operator*= : Inconsistent matrix sizes in
multiplication!");

   matrixT temp(_m->Row,m._m->Col);

   for (size_t i=0; i < _m->Row; i++)
      for (size_t j=0; j < m._m->Col; j++)
      {
         temp._m->Val[i][j] = T(0);
         for (size_t k=0; k < _m->Col; k++)
            temp._m->Val[i][j] += _m->Val[i][k] * m._m->Val[k][j];
      }
```

```
   *this = temp;

   return *this;
}

// combined scalar division and assignment operator
MAT_TEMPLATE inline matrixT&
matrixT::operator /= (const T& c) _NO_THROW
{
    if (_m->Refcnt > 1) clone();
    for (size_t i=0; i < _m->Row; i++)
        for (size_t j=0; j < _m->Col; j++)
            _m->Val[i][j] /= c;

    return *this;
}

// combined power and assignment operator
MAT_TEMPLATE inline matrixT&
matrixT::operator ^= (const size_t& pow) _THROW_MATRIX_ERROR
{
       matrixT temp(*this);

       for (size_t i=2; i <= pow; i++)
      *this = *this * temp;

       return *this;
}

// unary negation operator
MAT_TEMPLATE inline matrixT
matrixT::operator - () _NO_THROW
{
   matrixT temp(_m->Row,_m->Col);

   for (size_t i=0; i < _m->Row; i++)
      for (size_t j=0; j < _m->Col; j++)
        temp._m->Val[i][j] = - _m->Val[i][j];

   return temp;
}

// binary addition operator
MAT_TEMPLATE inline matrixT
operator + (const matrixT& m1, const matrixT& m2) _THROW_MATRIX_ERROR
{
   matrixT temp = m1;
   temp += m2;
   return temp;
}

// binary subtraction operator
MAT_TEMPLATE inline matrixT
operator - (const matrixT& m1, const matrixT& m2) _THROW_MATRIX_ERROR
{
   matrixT temp = m1;
   temp -= m2;
   return temp;
}

// binary scalar multiplication operator
MAT_TEMPLATE inline matrixT
operator * (const matrixT& m, const T& no) _NO_THROW
{
   matrixT temp = m;
   temp *= no;
```

```
   return temp;
}


// binary scalar multiplication operator
MAT_TEMPLATE inline matrixT
operator * (const T& no, const matrixT& m) _NO_THROW
{
   return (m * no);
}

// binary matrix multiplication operator
MAT_TEMPLATE inline matrixT
operator * (const matrixT& m1, const matrixT& m2) _THROW_MATRIX_ERROR
{
   matrixT temp = m1;
   temp *= m2;
   return temp;
}

// binary scalar division operator
MAT_TEMPLATE inline matrixT
operator / (const matrixT& m, const T& no) _NO_THROW
{
    return (m * (T(1) / no));
}


// binary scalar division operator
MAT_TEMPLATE inline matrixT
operator / (const T& no, const matrixT& m) _THROW_MATRIX_ERROR
{
    return (!m * no);
}

// binary matrix division operator
MAT_TEMPLATE inline matrixT
operator / (const matrixT& m1, const matrixT& m2) _THROW_MATRIX_ERROR
{
    return (m1 * !m2);
}

// binary power operator
MAT_TEMPLATE inline matrixT
operator ^ (const matrixT& m, const size_t& pow) _THROW_MATRIX_ERROR
{
   matrixT temp = m;
   temp ^= pow;
   return temp;
}

// unary transpose operator
MAT_TEMPLATE inline matrixT
operator ~ (const matrixT& m) _NO_THROW
{
   matrixT temp(m.ColNo(),m.RowNo());

   for (size_t i=0; i < m.RowNo(); i++)
      for (size_t j=0; j < m.ColNo(); j++)
      {
         T x = m(i,j);
             temp(j,i) = x;
      }
   return temp;
}
```

Appendix A (Continued)

```
// unary inversion operator
MAT_TEMPLATE inline matrixT
operator ! (const matrixT m) _THROW_MATRIX_ERROR
{
   matrixT temp = m;
   return temp.Inv();
}

// inversion function
MAT_TEMPLATE inline matrixT
matrixT::Inv () _THROW_MATRIX_ERROR
{
   size_t i,j,k;
   T a1,a2,*rowptr;

   if (_m->Row != _m->Col)
      REPORT_ERROR( "matrixT::operator!: Inversion of a non-square matrix");

   matrixT temp(_m->Row,_m->Col);
   if (_m->Refcnt > 1) clone();


   temp.Unit();
   for (k=0; k < _m->Row; k++)
   {
      int indx = pivot(k);
      if (indx == -1)
             REPORT_ERROR( "matrixT::operator!: Inversion of a singular matrix");

      if (indx != 0)
      {
             rowptr = temp._m->Val[k];
             temp._m->Val[k] = temp._m->Val[indx];
             temp._m->Val[indx] = rowptr;
      }
      a1 = _m->Val[k][k];
      for (j=0; j < _m->Row; j++)
      {
             _m->Val[k][j] /= a1;
             temp._m->Val[k][j] /= a1;
      }
      for (i=0; i < _m->Row; i++)
          if (i != k)
          {
             a2 = _m->Val[i][k];
             for (j=0; j < _m->Row; j++)
             {
                _m->Val[i][j] -= a2 * _m->Val[k][j];
                temp._m->Val[i][j] -= a2 * temp._m->Val[k][j];
             }
          }
   }
   return temp;
}

// solve simultaneous equation
MAT_TEMPLATE inline matrixT
matrixT::Solve (const matrixT& v) const _THROW_MATRIX_ERROR
{
   size_t i,j,k;
   T a1;

   if (!(_m->Row == _m->Col && _m->Col == v._m->Row))
      REPORT_ERROR( "matrixT::Solve():Inconsistent matrices!");

   matrixT temp(_m->Row,_m->Col+v._m->Col);
```

```
   for (i=0; i < _m->Row; i++)
   {
      for (j=0; j < _m->Col; j++)
         temp._m->Val[i][j] = _m->Val[i][j];
      for (k=0; k < v._m->Col; k++)
         temp._m->Val[i][_m->Col+k] = v._m->Val[i][k];
   }
   for (k=0; k < _m->Row; k++)
   {
      int indx = temp.pivot(k);
      if (indx == -1)
         REPORT_ERROR( "matrixT::Solve(): Singular matrix!");

      a1 = temp._m->Val[k][k];
      for (j=k; j < temp._m->Col; j++)
         temp._m->Val[k][j] /= a1;

      for (i=k+1; i < _m->Row; i++)
      {
         a1 = temp._m->Val[i][k];
         for (j=k; j < temp._m->Col; j++)
           temp._m->Val[i][j] -= a1 * temp._m->Val[k][j];
      }
   }
   matrixT s(v._m->Row,v._m->Col);
   for (k=0; k < v._m->Col; k++)
      for (int m=int(_m->Row)-1; m >= 0; m--)
      {
         s._m->Val[m][k] = temp._m->Val[m][_m->Col+k];
         for (j=m+1; j < _m->Col; j++)
            s._m->Val[m][k] -= temp._m->Val[m][j] * s._m->Val[j][k];
      }
   return s;
}

// set zero to all elements of this matrix
MAT_TEMPLATE inline void
matrixT::Null (const size_t& row, const size_t& col) _NO_THROW
{
    if (row != _m->Row || col != _m->Col)
        realloc( row,col);

    if (_m->Refcnt > 1)
        clone();

    for (size_t i=0; i < _m->Row; i++)
        for (size_t j=0; j < _m->Col; j++)
            _m->Val[i][j] = T(0);
    return;
}

// set zero to all elements of this matrix
MAT_TEMPLATE inline void
matrixT::Null() _NO_THROW
{
    if (_m->Refcnt > 1) clone();
    for (size_t i=0; i < _m->Row; i++)
        for (size_t j=0; j < _m->Col; j++)
                _m->Val[i][j] = T(0);
    return;
}

// set this matrix to unity
MAT_TEMPLATE inline void
matrixT::Unit (const size_t& row) _NO_THROW
{
```

Appendix A (Continued)

```
    if (row != _m->Row || row != _m->Col)
        realloc( row, row);

    if (_m->Refcnt > 1)
        clone();

    for (size_t i=0; i < _m->Row; i++)
        for (size_t j=0; j < _m->Col; j++)
            _m->Val[i][j] = i == j ? T(1) : T(0);
    return;
}

// set this matrix to unity
MAT_TEMPLATE inline void
matrixT::Unit () _NO_THROW
{
    if (_m->Refcnt > 1) clone();
    size_t row = min(_m->Row,_m->Col);
    _m->Row = _m->Col = row;

    for (size_t i=0; i < _m->Row; i++)
        for (size_t j=0; j < _m->Col; j++)
            _m->Val[i][j] = i == j ? T(1) : T(0);
    return;
}

// private partial pivoting method
MAT_TEMPLATE inline int
matrixT::pivot (size_t row)
{
  int k = int(row);
  double amax,temp;

  amax = -1;
  for (size_t i=row; i < _m->Row; i++)
    if ( (temp = abs( _m->Val[i][row])) > amax && temp != 0.0)
     {
       amax = temp;
       k = i;
     }
  if (_m->Val[k][row] == T(0))
     return -1;
  if (k != int(row))
  {
     T* rowptr = _m->Val[k];
     _m->Val[k] = _m->Val[row];
     _m->Val[row] = rowptr;
     return k;
  }
  return 0;
}

// calculate the determinant of a matrix
MAT_TEMPLATE inline T
matrixT::Det () const _THROW_MATRIX_ERROR
{
   size_t i,j,k;
   T piv,detVal = T(1);

   if (_m->Row != _m->Col)
      REPORT_ERROR( "matrixT::Det(): Determinant a non-square matrix!");

   matrixT temp(*this);
   if (temp._m->Refcnt > 1) temp.clone();

   for (k=0; k < _m->Row; k++)
```

Appendix A (Continued)

```
   {
      int indx = temp.pivot(k);
      if (indx == -1)
         return 0;
      if (indx != 0)
         detVal = - detVal;
      detVal = detVal * temp._m->Val[k][k];
      for (i=k+1; i < _m->Row; i++)
      {
         piv = temp._m->Val[i][k] / temp._m->Val[k][k];
         for (j=k+1; j < _m->Row; j++)
            temp._m->Val[i][j] -= piv * temp._m->Val[k][j];
      }
   }
   return detVal;
}

// calculate the norm of a matrix
MAT_TEMPLATE inline T
matrixT::Norm () _NO_THROW
{
   T retVal = T(0);

   for (size_t i=0; i < _m->Row; i++)
      for (size_t j=0; j < _m->Col; j++)
         retVal += _m->Val[i][j] * _m->Val[i][j];
   retVal = sqrt( retVal);

   return retVal;
}

// calculate the condition number of a matrix
MAT_TEMPLATE inline T
matrixT::Cond () _NO_THROW
{
   matrixT inv = ! (*this);
   return (Norm() * inv.Norm());
}

// calculate the cofactor of a matrix for a given element
MAT_TEMPLATE inline T
matrixT::Cofact (size_t row, size_t col) _THROW_MATRIX_ERROR
{
   size_t i,i1,j,j1;

   if (_m->Row != _m->Col)
      REPORT_ERROR( "matrixT::Cofact(): Cofactor of a non-square matrix!");

   if (row > _m->Row || col > _m->Col)
      REPORT_ERROR( "matrixT::Cofact(): Index out of range!");

   matrixT temp (_m->Row-1,_m->Col-1);

   for (i=i1=0; i < _m->Row; i++)
   {
      if (i == row)
        continue;
      for (j=j1=0; j < _m->Col; j++)
      {
         if (j == col)
            continue;
         temp._m->Val[i1][j1] = _m->Val[i][j];
         j1++;
      }
      i1++;
   }
```

```
   T  cof = temp.Det();
   if ((row+col)%2 == 1)
      cof = -cof;

   return cof;
}


// calculate adjoin of a matrix
MAT_TEMPLATE inline matrixT
matrixT::Adj () _THROW_MATRIX_ERROR
{
   if (_m->Row != _m->Col)
      REPORT_ERROR( "matrixT::Adj(): Adjoin of a non-square matrix.");

   matrixT temp(_m->Row,_m->Col);

   for (size_t i=0; i < _m->Row; i++)
      for (size_t j=0; j < _m->Col; j++)
         temp._m->Val[j][i] = Cofact(i,j);
   return temp;
}

// Determine if the matrix is singular
MAT_TEMPLATE inline bool
matrixT::IsSingular () _NO_THROW
{
   if (_m->Row != _m->Col)
      return false;
   return (Det() == T(0));
}

// Determine if the matrix is diagonal
MAT_TEMPLATE inline bool
matrixT::IsDiagonal () _NO_THROW
{
   if (_m->Row != _m->Col)
      return false;
   for (size_t i=0; i < _m->Row; i++)
     for (size_t j=0; j < _m->Col; j++)
        if (i != j && _m->Val[i][j] != T(0))
          return false;
   return true;
}

// Determine if the matrix is scalar
MAT_TEMPLATE inline bool
matrixT::IsScalar () _NO_THROW
{
   if (!IsDiagonal())
     return false;
   T v = _m->Val[0][0];
   for (size_t i=1; i < _m->Row; i++)
     if (_m->Val[i][i] != v)
        return false;
   return true;
}

// Determine if the matrix is a unit matrix
MAT_TEMPLATE inline bool
matrixT::IsUnit () _NO_THROW
{
   if (IsScalar() && _m->Val[0][0] == T(1))
     return true;
   return false;
}
```

```
// Determine if this is a null matrix
MAT_TEMPLATE inline bool
matrixT::IsNull () _NO_THROW
{
   for (size_t i=0; i < _m->Row; i++)
      for (size_t j=0; j < _m->Col; j++)
         if (_m->Val[i][j] != T(0))
            return false;
   return true;
}

// Determine if the matrix is symmetric
MAT_TEMPLATE inline bool
matrixT::IsSymmetric () _NO_THROW
{
   if (_m->Row != _m->Col)
      return false;
   for (size_t i=0; i < _m->Row; i++)
      for (size_t j=0; j < _m->Col; j++)
         if (_m->Val[i][j] != _m->Val[j][i])
            return false;
   return true;
}

// Determine if the matrix is skew-symmetric
MAT_TEMPLATE inline bool
matrixT::IsSkewSymmetric () _NO_THROW
{
   if (_m->Row != _m->Col)
      return false;
   for (size_t i=0; i < _m->Row; i++)
      for (size_t j=0; j < _m->Col; j++)
         if (_m->Val[i][j] != -_m->Val[j][i])
            return false;
   return true;
}

// Determine if the matrix is upper triangular
MAT_TEMPLATE inline bool
matrixT::IsUpperTriangular () _NO_THROW
{
   if (_m->Row != _m->Col)
      return false;
   for (size_t i=1; i < _m->Row; i++)
      for (size_t j=0; j < i-1; j++)
         if (_m->Val[i][j] != T(0))
            return false;
   return true;
}

// Determine if the matrix is lower triangular
MAT_TEMPLATE inline bool
matrixT::IsLowerTriangular () _NO_THROW
{
   if (_m->Row != _m->Col)
      return false;

   for (size_t j=1; j < _m->Col; j++)
      for (size_t i=0; i < j-1; i++)
         if (_m->Val[i][j] != T(0))
            return false;

   return true;
}
```

Appendix A (Continued)

```
#ifndef _NO_NAMESPACE
}
#endif

#endif //__STD_MATRIX_H
```

Appendix B: Graphical User Interface

B.1 WMRA Main GUI

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


// 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;

	double Td00, Td01, Td02, Td03, Td10, Td11, Td12, Td13, Td20, Td21, Td22, Td23;
	double Vd00, Vd01, Vd02, Vd10, Vd11, Vd12;
	double qi00, qi01, qi02, qi03, qi04, qi05, qi06;
	double WCi00, WCi01, WCi02;
	int varstop, stop1, start, varmatrix;
	int varloop = 0;
	int varscreenopn = 0;
	Matrix varQi(7,1), varDX(7,1), varWCi(3,1), spdata1(7,1);
	#include "var_included.h"
	#include <iostream>
	#include <fstream>
	#include <tchar.h>
	#include <string.h>
	#include "Form2.h"
	#include "Form3.h"
	#include "Form4.h"
	#include "Form5.h"
	#include "Form6.h"

#pragma once

namespace WMRAGUI {

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

	/// <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;
                }
        }
private: System::Windows::Forms::GroupBox^  groupBox1;
private: System::Windows::Forms::GroupBox^  groupBox2;
private: System::Windows::Forms::GroupBox^  groupBox3;
private: System::Windows::Forms::GroupBox^  groupBox4;
private: System::Windows::Forms::GroupBox^  groupBox5;
private: System::Windows::Forms::GroupBox^  groupBox6;
private: System::Windows::Forms::GroupBox^  groupBox7;
private: System::Windows::Forms::GroupBox^  groupBox8;
private: System::Windows::Forms::GroupBox^  groupBox9;
private: System::Windows::Forms::GroupBox^  groupBox10;
private: System::Windows::Forms::GroupBox^  groupBox11;
private: System::Windows::Forms::GroupBox^  groupBox12;
private: System::Windows::Forms::GroupBox^  groupBox13;
private: System::Windows::Forms::GroupBox^  groupBox14;
private: System::Windows::Forms::GroupBox^  groupBox15;
private: System::Windows::Forms::GroupBox^  groupBox16;
private: System::Windows::Forms::GroupBox^  groupBox17;
private: System::Windows::Forms::GroupBox^  groupBox18;
private: System::Windows::Forms::GroupBox^  groupBox19;
private: System::Windows::Forms::GroupBox^  groupBox20;
private: System::Windows::Forms::GroupBox^  groupBox21;
private: System::Windows::Forms::GroupBox^  groupBox22;
private: System::Windows::Forms::GroupBox^  groupBox23;
private: System::Windows::Forms::GroupBox^  groupBox24;
private: System::Windows::Forms::GroupBox^  groupBox25;
private: System::Windows::Forms::GroupBox^  groupBox26;
private: System::Windows::Forms::GroupBox^  groupBox27;

private: System::Windows::Forms::ComboBox^  comboBox1;
private: System::Windows::Forms::ComboBox^  comboBox2;
private: System::Windows::Forms::ComboBox^  comboBox3;
private: System::Windows::Forms::ComboBox^  comboBox4;
private: System::Windows::Forms::ComboBox^  comboBox5;
private: System::Windows::Forms::ComboBox^  comboBox6;
private: System::Windows::Forms::ComboBox^  comboBox7;
private: System::Windows::Forms::ComboBox^  comboBox8;
private: System::Windows::Forms::ComboBox^  comboBox9;
private: System::Windows::Forms::ComboBox^  comboBox10;
private: System::Windows::Forms::ComboBox^  comboBox11;
private: System::Windows::Forms::ComboBox^  comboBox12;
private: System::Windows::Forms::ComboBox^  comboBox13;
private: System::Windows::Forms::ComboBox^  comboBox14;

private: System::Windows::Forms::CheckBox^  checkBox1;
private: System::Windows::Forms::CheckBox^  checkBox2;
```

```
        private: System::Windows::Forms::Label^  label1;
        private: System::Windows::Forms::Label^  label2;
        private: System::Windows::Forms::Label^  label3;
        private: System::Windows::Forms::Label^  label4;
        private: System::Windows::Forms::Label^  label5;
        private: System::Windows::Forms::Label^  label6;
        private: System::Windows::Forms::Label^  label7;
        private: System::Windows::Forms::Label^  label8;
        private: System::Windows::Forms::Label^  label9;
        private: System::Windows::Forms::Label^  label10;
        private: System::Windows::Forms::Label^  label11;
        private: System::Windows::Forms::Label^  label12;
        private: System::Windows::Forms::Label^  label13;

        private: System::Windows::Forms::TextBox^  textBox1;
        private: System::Windows::Forms::TextBox^  textBox2;
        private: System::Windows::Forms::TextBox^  textBox3;
        private: System::Windows::Forms::TextBox^  textBox4;
        private: System::Windows::Forms::TextBox^  textBox5;
        private: System::Windows::Forms::TextBox^  textBox6;
        private: System::Windows::Forms::TextBox^  textBox7;
        private: System::Windows::Forms::TextBox^  textBox8;
        private: System::Windows::Forms::TextBox^  textBox9;
        private: System::Windows::Forms::TextBox^  textBox10;
        private: System::Windows::Forms::TextBox^  textBox11;
        private: System::Windows::Forms::TextBox^  textBox12;
        private: System::Windows::Forms::TextBox^  textBox13;
        private: System::Windows::Forms::TextBox^  textBox14;
        private: System::Windows::Forms::TextBox^  textBox15;
        private: System::Windows::Forms::TextBox^  textBox16;
        private: System::Windows::Forms::TextBox^  textBox17;
        private: System::Windows::Forms::TextBox^  textBox18;
        private: System::Windows::Forms::TextBox^  textBox19;
        private: System::Windows::Forms::TextBox^  textBox20;
        private: System::Windows::Forms::TextBox^  textBox21;
        private: System::Windows::Forms::TextBox^  textBox22;
        private: System::Windows::Forms::TextBox^  textBox23;
        private: System::Windows::Forms::TextBox^  textBox24;
        private: System::Windows::Forms::TextBox^  textBox25;
        private: System::Windows::Forms::TextBox^  textBox26;
        private: System::Windows::Forms::TextBox^  textBox27;
        private: System::Windows::Forms::TextBox^  textBox28;
        private: System::Windows::Forms::TextBox^  textBox29;
        private: System::Windows::Forms::TextBox^  textBox30;
        private: System::Windows::Forms::TextBox^  textBox31;
        private: System::Windows::Forms::TextBox^  textBox32;

        private: System::Windows::Forms::Button^  button1;
        private: System::Windows::Forms::Button^  button2;
        private: System::Windows::Forms::Button^  button3;

        private: System::Windows::Forms::Button^  button6;
        private: System::Windows::Forms::Button^  button7;
        private: System::Windows::Forms::Button^  button8;
        private: System::Windows::Forms::Button^  button9;
        private: System::Windows::Forms::Button^  button10;

private: System::Windows::Forms::MenuStrip^  menuStrip1;
private: System::Windows::Forms::ToolStripMenuItem^  fileToolStripMenuItem;
private: System::Windows::Forms::ToolStripMenuItem^  helpToolStripMenuItem;
private: System::Windows::Forms::ToolStripMenuItem^  openToolStripMenuItem;
private: System::Windows::Forms::ToolStripMenuItem^  saveToolStripMenuItem;
private: System::Windows::Forms::ToolStripMenuItem^  saveAsToolStripMenuItem;
private: System::Windows::Forms::ToolStripMenuItem^  exitToolStripMenuItem;
private: System::Windows::Forms::Button^  button11;
private: System::Windows::Forms::Button^  button5;
```

Appendix B (Continued)

```
        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)
                {
                        this->groupBox11 = (gcnew System::Windows::Forms::GroupBox());
                        this->comboBox2 = (gcnew System::Windows::Forms::ComboBox());
                        this->groupBox1 = (gcnew System::Windows::Forms::GroupBox());
                        this->comboBox1 = (gcnew System::Windows::Forms::ComboBox());
                        this->groupBox2 = (gcnew System::Windows::Forms::GroupBox());
                        this->comboBox3 = (gcnew System::Windows::Forms::ComboBox());
                        this->groupBox3 = (gcnew System::Windows::Forms::GroupBox());
                        this->comboBox4 = (gcnew System::Windows::Forms::ComboBox());
                        this->groupBox4 = (gcnew System::Windows::Forms::GroupBox());
                        this->comboBox5 = (gcnew System::Windows::Forms::ComboBox());
                        this->groupBox5 = (gcnew System::Windows::Forms::GroupBox());
                        this->comboBox6 = (gcnew System::Windows::Forms::ComboBox());
                        this->groupBox6 = (gcnew System::Windows::Forms::GroupBox());
                        this->checkBox2 = (gcnew System::Windows::Forms::CheckBox());
                        this->checkBox1 = (gcnew System::Windows::Forms::CheckBox());
                        this->comboBox7 = (gcnew System::Windows::Forms::ComboBox());
                        this->groupBox7 = (gcnew System::Windows::Forms::GroupBox());
                        this->comboBox8 = (gcnew System::Windows::Forms::ComboBox());
                        this->groupBox9 = (gcnew System::Windows::Forms::GroupBox());
                        this->groupBox19 = (gcnew System::Windows::Forms::GroupBox());
                        this->button3 = (gcnew System::Windows::Forms::Button());
                        this->label12 = (gcnew System::Windows::Forms::Label());
                        this->textBox22 = (gcnew System::Windows::Forms::TextBox());
                        this->label13 = (gcnew System::Windows::Forms::Label());
                        this->textBox21 = (gcnew System::Windows::Forms::TextBox());
                        this->label11 = (gcnew System::Windows::Forms::Label());
                        this->groupBox18 = (gcnew System::Windows::Forms::GroupBox());
                        this->label10 = (gcnew System::Windows::Forms::Label());
                        this->groupBox20 = (gcnew System::Windows::Forms::GroupBox());
                        this->button2 = (gcnew System::Windows::Forms::Button());
                        this->label8 = (gcnew System::Windows::Forms::Label());
                        this->label7 = (gcnew System::Windows::Forms::Label());
                        this->textBox19 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox16 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox18 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox14 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox17 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox15 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox20 = (gcnew System::Windows::Forms::TextBox());
                        this->label9 = (gcnew System::Windows::Forms::Label());
                        this->groupBox8 = (gcnew System::Windows::Forms::GroupBox());
                        this->groupBox17 = (gcnew System::Windows::Forms::GroupBox());
                        this->comboBox10 = (gcnew System::Windows::Forms::ComboBox());
                        this->groupBox16 = (gcnew System::Windows::Forms::GroupBox());
                        this->label6 = (gcnew System::Windows::Forms::Label());
                        this->textBox13 = (gcnew System::Windows::Forms::TextBox());
                        this->label5 = (gcnew System::Windows::Forms::Label());
                        this->groupBox12 = (gcnew System::Windows::Forms::GroupBox());
                        this->button1 = (gcnew System::Windows::Forms::Button());
                        this->groupBox15 = (gcnew System::Windows::Forms::GroupBox());
```

Appendix B (Continued)

```
                        this->label4 = (gcnew System::Windows::Forms::Label());
                        this->label3 = (gcnew System::Windows::Forms::Label());
                        this->label2 = (gcnew System::Windows::Forms::Label());
                        this->label1 = (gcnew System::Windows::Forms::Label());
                        this->groupBox14 = (gcnew System::Windows::Forms::GroupBox());
                        this->textBox12 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox11 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox3 = (gcnew System::Windows::Forms::TextBox());
                        this->groupBox13 = (gcnew System::Windows::Forms::GroupBox());
                        this->textBox10 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox7 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox2 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox9 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox6 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox1 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox8 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox4 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox5 = (gcnew System::Windows::Forms::TextBox());
                        this->comboBox9 = (gcnew System::Windows::Forms::ComboBox());
                        this->groupBox10 = (gcnew System::Windows::Forms::GroupBox());
                        this->comboBox11 = (gcnew System::Windows::Forms::ComboBox());
                        this->groupBox22 = (gcnew System::Windows::Forms::GroupBox());
                        this->groupBox24 = (gcnew System::Windows::Forms::GroupBox());
                        this->button6 = (gcnew System::Windows::Forms::Button());
                        this->textBox23 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox24 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox25 = (gcnew System::Windows::Forms::TextBox());
                        this->groupBox25 = (gcnew System::Windows::Forms::GroupBox());
                        this->button5 = (gcnew System::Windows::Forms::Button());
                        this->textBox29 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox28 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox30 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox27 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox26 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox31 = (gcnew System::Windows::Forms::TextBox());
                        this->textBox32 = (gcnew System::Windows::Forms::TextBox());
                        this->groupBox21 = (gcnew System::Windows::Forms::GroupBox());
                        this->comboBox12 = (gcnew System::Windows::Forms::ComboBox());
                        this->groupBox23 = (gcnew System::Windows::Forms::GroupBox());
                        this->comboBox13 = (gcnew System::Windows::Forms::ComboBox());
                        this->groupBox27 = (gcnew System::Windows::Forms::GroupBox());
                        this->comboBox14 = (gcnew System::Windows::Forms::ComboBox());
                        this->groupBox26 = (gcnew System::Windows::Forms::GroupBox());
                        this->button10 = (gcnew System::Windows::Forms::Button());
                        this->button9 = (gcnew System::Windows::Forms::Button());
                        this->button7 = (gcnew System::Windows::Forms::Button());
                        this->button8 = (gcnew System::Windows::Forms::Button());
                        this->menuStrip1 = (gcnew System::Windows::Forms::MenuStrip());
                        this->fileToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());
                        this->openToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());
                        this->saveToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());
                        this->saveAsToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());
                        this->exitToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());
                        this->helpToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());
                        this->button11 = (gcnew System::Windows::Forms::Button());
                        this->groupBox11->SuspendLayout();
                        this->groupBox1->SuspendLayout();
                        this->groupBox2->SuspendLayout();
                        this->groupBox3->SuspendLayout();
                        this->groupBox4->SuspendLayout();
```

```
                        this->groupBox5->SuspendLayout();
                        this->groupBox6->SuspendLayout();
                        this->groupBox7->SuspendLayout();
                        this->groupBox9->SuspendLayout();
                        this->groupBox19->SuspendLayout();
                        this->groupBox18->SuspendLayout();
                        this->groupBox20->SuspendLayout();
                        this->groupBox8->SuspendLayout();
                        this->groupBox17->SuspendLayout();
                        this->groupBox16->SuspendLayout();
                        this->groupBox12->SuspendLayout();
                        this->groupBox15->SuspendLayout();
                        this->groupBox14->SuspendLayout();
                        this->groupBox13->SuspendLayout();
                        this->groupBox10->SuspendLayout();
                        this->groupBox22->SuspendLayout();
                        this->groupBox24->SuspendLayout();
                        this->groupBox25->SuspendLayout();
                        this->groupBox21->SuspendLayout();
                        this->groupBox23->SuspendLayout();
                        this->groupBox27->SuspendLayout();
                        this->groupBox26->SuspendLayout();
                        this->menuStrip1->SuspendLayout();
                        this->SuspendLayout();
                        //
                        // groupBox11
                        //
                        this->groupBox11->Controls->Add(this->comboBox2);
                        this->groupBox11->Location = System::Drawing::Point(12, 27);
                        this->groupBox11->Name = L"groupBox11";
                        this->groupBox11->RightToLeft =
System::Windows::Forms::RightToLeft::No;
                        this->groupBox11->Size = System::Drawing::Size(166, 60);
                        this->groupBox11->TabIndex = 0;
                        this->groupBox11->TabStop = false;
                        this->groupBox11->Text = L"What to run\?";
                        //
                        // comboBox2
                        //
                        this->comboBox2->FormattingEnabled = true;
                        this->comboBox2->Items->AddRange(gcnew cli::array< System::Object^
>(3) {L"WMRA", L"Arm Only", L"Wheelchair Only"});
                        this->comboBox2->Location = System::Drawing::Point(7, 25);
                        this->comboBox2->Name = L"comboBox2";
                        this->comboBox2->Size = System::Drawing::Size(151, 21);
                        this->comboBox2->TabIndex = 10;
                        this->comboBox2->SelectedIndex =0;
                        this->comboBox2->SelectedIndexChanged += gcnew
System::EventHandler(this, &Form1::comboBox2_SelectedIndexChanged);
                        //
                        // groupBox1
                        //
                        this->groupBox1->Controls->Add(this->comboBox1);
                        this->groupBox1->Location = System::Drawing::Point(12, 93);
                        this->groupBox1->Name = L"groupBox1";
                        this->groupBox1->RightToLeft =
System::Windows::Forms::RightToLeft::No;
                        this->groupBox1->Size = System::Drawing::Size(166, 60);
                        this->groupBox1->TabIndex = 0;
                        this->groupBox1->TabStop = false;
                        this->groupBox1->Text = L"What to control\?";
                        //
                        // comboBox1
                        //
                        this->comboBox1->DropDownStyle =
System::Windows::Forms::ComboBoxStyle::DropDownList;
```

```
                        this->comboBox1->FormattingEnabled = true;
                        this->comboBox1->Items->AddRange(gcnew cli::array< System::Object^
>(2) {L"Position & Orientation", L"Position Only"});
                        this->comboBox1->Location = System::Drawing::Point(7, 25);
                        this->comboBox1->Name = L"comboBox1";
                        this->comboBox1->Size = System::Drawing::Size(151, 21);
                        this->comboBox1->TabIndex = 10;
                        this->comboBox1->SelectedIndex =0;
                        this->comboBox1->SelectedIndexChanged += gcnew
System::EventHandler(this, &Form1::comboBox1_SelectedIndexChanged);
                        //
                        // groupBox2
                        //
                        this->groupBox2->Controls->Add(this->comboBox3);
                        this->groupBox2->Location = System::Drawing::Point(12, 158);
                        this->groupBox2->Name = L"groupBox2";
                        this->groupBox2->RightToLeft =
System::Windows::Forms::RightToLeft::No;
                        this->groupBox2->Size = System::Drawing::Size(166, 60);
                        this->groupBox2->TabIndex = 0;
                        this->groupBox2->TabStop = false;
                        this->groupBox2->Text = L"Control Coordinates\?";
                        //
                        // comboBox3
                        //
                        this->comboBox3->FormattingEnabled = true;
                        this->comboBox3->Items->AddRange(gcnew cli::array< System::Object^
>(3) {L"Ground Frame", L"Wheelchair Frame", L"Gripper Frame"});
                        this->comboBox3->Location = System::Drawing::Point(7, 25);
                        this->comboBox3->Name = L"comboBox3";
                        this->comboBox3->Size = System::Drawing::Size(151, 21);
                        this->comboBox3->TabIndex = 10;
                        this->comboBox3->SelectedIndex =0;
                        //
                        // groupBox3
                        //
                        this->groupBox3->Controls->Add(this->comboBox4);
                        this->groupBox3->Location = System::Drawing::Point(12, 224);
                        this->groupBox3->Name = L"groupBox3";
                        this->groupBox3->RightToLeft =
System::Windows::Forms::RightToLeft::No;
                        this->groupBox3->Size = System::Drawing::Size(166, 60);
                        this->groupBox3->TabIndex = 0;
                        this->groupBox3->TabStop = false;
                        this->groupBox3->Text = L"Simulation\?";
                        //
                        // comboBox4
                        //
                        this->comboBox4->FormattingEnabled = true;
                        this->comboBox4->Items->AddRange(gcnew cli::array< System::Object^
>(4) {L"Virtual Reality", L"Matlab Graphics", L"BOTH Animations",
                                L"NO Animation"});
                        this->comboBox4->Location = System::Drawing::Point(7, 25);
                        this->comboBox4->Name = L"comboBox4";
                        this->comboBox4->Size = System::Drawing::Size(151, 21);
                        this->comboBox4->TabIndex = 10;
                        this->comboBox4->SelectedIndex =0;
                        this->comboBox4->SelectedIndexChanged += gcnew
System::EventHandler(this, &Form1::comboBox4_SelectedIndexChanged);
                        //
                        // groupBox4
                        //
                        this->groupBox4->Controls->Add(this->comboBox5);
                        this->groupBox4->Location = System::Drawing::Point(12, 290);
                        this->groupBox4->Name = L"groupBox4";
```

```
                    this->groupBox4->RightToLeft =
System::Windows::Forms::RightToLeft::No;
                    this->groupBox4->Size = System::Drawing::Size(166, 60);
                    this->groupBox4->TabIndex = 0;
                    this->groupBox4->TabStop = false;
                    this->groupBox4->Text = L"Run WMRA\?";
                    //
                    // comboBox5
                    //
                    this->comboBox5->FormattingEnabled = true;
                    this->comboBox5->Items->AddRange(gcnew cli::array< System::Object^
>(2) {L"No", L"Yes"});
                    this->comboBox5->Location = System::Drawing::Point(7, 25);
                    this->comboBox5->Name = L"comboBox5";
                    this->comboBox5->Size = System::Drawing::Size(151, 21);
                    this->comboBox5->TabIndex = 10;
                    this->comboBox5->SelectedIndex =0;
                    this->comboBox5->SelectedIndexChanged += gcnew
System::EventHandler(this, &Form1::comboBox5_SelectedIndexChanged);
                    //
                    // groupBox5
                    //
                    this->groupBox5->Controls->Add(this->comboBox6);
                    this->groupBox5->Location = System::Drawing::Point(12, 356);
                    this->groupBox5->Name = L"groupBox5";
                    this->groupBox5->RightToLeft =
System::Windows::Forms::RightToLeft::No;
                    this->groupBox5->Size = System::Drawing::Size(166, 60);
                    this->groupBox5->TabIndex = 0;
                    this->groupBox5->TabStop = false;
                    this->groupBox5->Text = L"Plots\?";
                    //
                    // comboBox6
                    //
                    this->comboBox6->FormattingEnabled = true;
                    this->comboBox6->Items->AddRange(gcnew cli::array< System::Object^
>(2) {L"No", L"Yes"});
                    this->comboBox6->Location = System::Drawing::Point(7, 25);
                    this->comboBox6->Name = L"comboBox6";
                    this->comboBox6->Size = System::Drawing::Size(151, 21);
                    this->comboBox6->TabIndex = 10;
                    this->comboBox6->SelectedIndex =0;
                    this->comboBox6->SelectedIndexChanged += gcnew
System::EventHandler(this, &Form1::comboBox6_SelectedIndexChanged);
                    //
                    // groupBox6
                    //
                    this->groupBox6->Controls->Add(this->checkBox2);
                    this->groupBox6->Controls->Add(this->checkBox1);
                    this->groupBox6->Controls->Add(this->comboBox7);
                    this->groupBox6->Location = System::Drawing::Point(12, 422);
                    this->groupBox6->Name = L"groupBox6";
                    this->groupBox6->RightToLeft =
System::Windows::Forms::RightToLeft::No;
                    this->groupBox6->Size = System::Drawing::Size(166, 121);
                    this->groupBox6->TabIndex = 0;
                    this->groupBox6->TabStop = false;
                    this->groupBox6->Text = L"Optimization method\?";
                    //
                    // checkBox2
                    //
                    this->checkBox2->AutoSize = true;
                    this->checkBox2->Checked = true;
                    this->checkBox2->CheckState =
System::Windows::Forms::CheckState::Checked;
                    this->checkBox2->Location = System::Drawing::Point(6, 88);
```

```
                        this->checkBox2->Name = L"checkBox2";
                        this->checkBox2->Size = System::Drawing::Size(73, 17);
                        this->checkBox2->TabIndex = 11;
                        this->checkBox2->Text = L"JL/O stop";
                        this->checkBox2->UseVisualStyleBackColor = true;
                        //
                        // checkBox1
                        //
                        this->checkBox1->AutoSize = true;
                        this->checkBox1->Checked = true;
                        this->checkBox1->CheckState =
System::Windows::Forms::CheckState::Checked;
                        this->checkBox1->Location = System::Drawing::Point(6, 65);
                        this->checkBox1->Name = L"checkBox1";
                        this->checkBox1->Size = System::Drawing::Size(90, 17);
                        this->checkBox1->TabIndex = 11;
                        this->checkBox1->Text = L"JL avoidance";
                        this->checkBox1->UseVisualStyleBackColor = true;
                        //
                        // comboBox7
                        //
                        this->comboBox7->FormattingEnabled = true;
                        this->comboBox7->Items->AddRange(gcnew cli::array< System::Object^
>(4) {L"SRIWLN", L"PIWLN", L"SRIENE", L"PIENE"});
                        this->comboBox7->Location = System::Drawing::Point(7, 25);
                        this->comboBox7->Name = L"comboBox7";
                        this->comboBox7->Size = System::Drawing::Size(151, 21);
                        this->comboBox7->TabIndex = 10;
                        this->comboBox7->SelectedIndex =0;
                        this->comboBox7->SelectedIndexChanged += gcnew
System::EventHandler(this, &Form1::comboBox7_SelectedIndexChanged);
                        //
                        // groupBox7
                        //
                        this->groupBox7->Controls->Add(this->comboBox8);
                        this->groupBox7->Location = System::Drawing::Point(12, 549);
                        this->groupBox7->Name = L"groupBox7";
                        this->groupBox7->RightToLeft =
System::Windows::Forms::RightToLeft::No;
                        this->groupBox7->Size = System::Drawing::Size(166, 60);
                        this->groupBox7->TabIndex = 0;
                        this->groupBox7->TabStop = false;
                        this->groupBox7->Text = L"Close all when done\?";
                        //
                        // comboBox8
                        //
                        this->comboBox8->FormattingEnabled = true;
                        this->comboBox8->Items->AddRange(gcnew cli::array< System::Object^
>(2) {L"Yes", L"No"});
                        this->comboBox8->Location = System::Drawing::Point(7, 25);
                        this->comboBox8->Name = L"comboBox8";
                        this->comboBox8->Size = System::Drawing::Size(151, 21);
                        this->comboBox8->TabIndex = 10;
                        this->comboBox8->SelectedIndex =0;
                        //
                        // groupBox9
                        //
                        this->groupBox9->Controls->Add(this->groupBox19);
                        this->groupBox9->Controls->Add(this->groupBox18);
                        this->groupBox9->Controls->Add(this->groupBox8);
                        this->groupBox9->Controls->Add(this->comboBox9);
                        this->groupBox9->Location = System::Drawing::Point(196, 26);
                        this->groupBox9->Name = L"groupBox9";
                        this->groupBox9->Size = System::Drawing::Size(326, 583);
                        this->groupBox9->TabIndex = 8;
                        this->groupBox9->TabStop = false;
```

```
                    this->groupBox9->Text = L"User Interface\?";
                    //
                    // groupBox19
                    //
                    this->groupBox19->Controls->Add(this->button3);
                    this->groupBox19->Controls->Add(this->label12);
                    this->groupBox19->Controls->Add(this->textBox22);
                    this->groupBox19->Controls->Add(this->label13);
                    this->groupBox19->Controls->Add(this->textBox21);
                    this->groupBox19->Controls->Add(this->label11);
                    this->groupBox19->Enabled = false;
                    this->groupBox19->Location = System::Drawing::Point(6, 501);
                    this->groupBox19->Name = L"groupBox19";
                    this->groupBox19->Size = System::Drawing::Size(314, 73);
                    this->groupBox19->TabIndex = 3;
                    this->groupBox19->TabStop = false;
                    this->groupBox19->Text = L"Spaceball / phychology mask / screen
control";
                    //
                    // button3
                    //
                    this->button3->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 12, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
                           static_cast<System::Byte>(0)));
                    this->button3->Location = System::Drawing::Point(250, 38);
                    this->button3->Name = L"button3";
                    this->button3->Size = System::Drawing::Size(56, 30);
                    this->button3->TabIndex = 3;
                    this->button3->Text = L"Exit";
                    this->button3->UseVisualStyleBackColor = true;
                    this->button3->Click += gcnew System::EventHandler(this,
&Form1::button3_Click);
                    //
                    // label12
                    //
                    this->label12->AutoSize = true;
                    this->label12->Location = System::Drawing::Point(217, 22);
                    this->label12->Name = L"label12";
                    this->label12->Size = System::Drawing::Size(45, 13);
                    this->label12->TabIndex = 2;
                    this->label12->Text = L"mm/sec";
                    //
                    // textBox22
                    //
                    this->textBox22->Location = System::Drawing::Point(110, 45);
                    this->textBox22->Name = L"textBox22";
                    this->textBox22->Size = System::Drawing::Size(100, 20);
                    this->textBox22->TabIndex = 1;
                    this->textBox22->Text = L"19711";
                    this->textBox22->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // label13
                    //
                    this->label13->AutoSize = true;
                    this->label13->Location = System::Drawing::Point(27, 48);
                    this->label13->Name = L"label13";
                    this->label13->Size = System::Drawing::Size(67, 13);
                    this->label13->TabIndex = 0;
                    this->label13->Text = L"Port number:";
                    //
                    // textBox21
                    //
                    this->textBox21->Location = System::Drawing::Point(111, 19);
                    this->textBox21->Name = L"textBox21";
                    this->textBox21->Size = System::Drawing::Size(100, 20);
```

```
                    this->textBox21->TabIndex = 1;
                    this->textBox21->Text = L"50";
                    this->textBox21->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // label11
                    //
                    this->label11->AutoSize = true;
                    this->label11->Location = System::Drawing::Point(16, 22);
                    this->label11->Name = L"label11";
                    this->label11->Size = System::Drawing::Size(89, 13);
                    this->label11->TabIndex = 0;
                    this->label11->Text = L"Linear Velocity V:";
                    //
                    // groupBox18
                    //
                    this->groupBox18->Controls->Add(this->label10);
                    this->groupBox18->Controls->Add(this->groupBox20);
                    this->groupBox18->Controls->Add(this->textBox20);
                    this->groupBox18->Controls->Add(this->label9);
                    this->groupBox18->Enabled = false;
                    this->groupBox18->Location = System::Drawing::Point(6, 362);
                    this->groupBox18->Name = L"groupBox18";
                    this->groupBox18->Size = System::Drawing::Size(313, 133);
                    this->groupBox18->TabIndex = 2;
                    this->groupBox18->TabStop = false;
                    this->groupBox18->Text = L"Velocity Control";
                    //
                    // label10
                    //
                    this->label10->AutoSize = true;
                    this->label10->Location = System::Drawing::Point(211, 105);
                    this->label10->Name = L"label10";
                    this->label10->Size = System::Drawing::Size(24, 13);
                    this->label10->TabIndex = 2;
                    this->label10->Text = L"sec";
                    //
                    // groupBox20
                    //
                    this->groupBox20->Controls->Add(this->button2);
                    this->groupBox20->Controls->Add(this->label8);
                    this->groupBox20->Controls->Add(this->label7);
                    this->groupBox20->Controls->Add(this->textBox19);
                    this->groupBox20->Controls->Add(this->textBox16);
                    this->groupBox20->Controls->Add(this->textBox18);
                    this->groupBox20->Controls->Add(this->textBox14);
                    this->groupBox20->Controls->Add(this->textBox17);
                    this->groupBox20->Controls->Add(this->textBox15);
                    this->groupBox20->Location = System::Drawing::Point(6, 19);
                    this->groupBox20->Name = L"groupBox20";
                    this->groupBox20->Size = System::Drawing::Size(300, 77);
                    this->groupBox20->TabIndex = 0;
                    this->groupBox20->TabStop = false;
                    this->groupBox20->Text = L"Velocity Components Vd";
                    //
                    // button2
                    //
                    this->button2->Location = System::Drawing::Point(268, 9);
                    this->button2->Name = L"button2";
                    this->button2->Size = System::Drawing::Size(25, 19);
                    this->button2->TabIndex = 3;
                    this->button2->Text = L"...";
                    this->button2->TextAlign =
System::Drawing::ContentAlignment::TopCenter;
                    this->button2->UseVisualStyleBackColor = true;
```

```
                    this->button2->Click += gcnew System::EventHandler(this,
&Form1::button2_Click);
                    //
                    // label8
                    //
                    this->label8->AutoSize = true;
                    this->label8->Location = System::Drawing::Point(202, 49);
                    this->label8->Name = L"label8";
                    this->label8->Size = System::Drawing::Size(89, 13);
                    this->label8->TabIndex = 2;
                    this->label8->Text = L"(tx, ty, tz) rad/sec";
                    //
                    // label7
                    //
                    this->label7->AutoSize = true;
                    this->label7->Location = System::Drawing::Point(203, 26);
                    this->label7->Name = L"label7";
                    this->label7->Size = System::Drawing::Size(81, 13);
                    this->label7->TabIndex = 2;
                    this->label7->Text = L"(x, y, z) mm/sec";
                    //
                    // textBox19
                    //
                    this->textBox19->Location = System::Drawing::Point(139, 46);
                    this->textBox19->Name = L"textBox19";
                    this->textBox19->Size = System::Drawing::Size(57, 20);
                    this->textBox19->TabIndex = 1;
                    this->textBox19->Text = L"0.001";
                    this->textBox19->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // textBox16
                    //
                    this->textBox16->Location = System::Drawing::Point(139, 20);
                    this->textBox16->Name = L"textBox16";
                    this->textBox16->Size = System::Drawing::Size(57, 20);
                    this->textBox16->TabIndex = 1;
                    this->textBox16->Text = L"-70";
                    this->textBox16->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // textBox18
                    //
                    this->textBox18->Location = System::Drawing::Point(13, 46);
                    this->textBox18->Name = L"textBox18";
                    this->textBox18->Size = System::Drawing::Size(57, 20);
                    this->textBox18->TabIndex = 1;
                    this->textBox18->Text = L"0.001";
                    this->textBox18->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // textBox14
                    //
                    this->textBox14->Location = System::Drawing::Point(13, 20);
                    this->textBox14->Name = L"textBox14";
                    this->textBox14->Size = System::Drawing::Size(57, 20);
                    this->textBox14->TabIndex = 1;
                    this->textBox14->Text = L"70";
                    this->textBox14->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // textBox17
                    //
                    this->textBox17->Location = System::Drawing::Point(76, 46);
                    this->textBox17->Name = L"textBox17";
                    this->textBox17->Size = System::Drawing::Size(57, 20);
```

```
                        this->textBox17->TabIndex = 1;
                        this->textBox17->Text = L"0.001";
                        this->textBox17->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                        //
                        // textBox15
                        //
                        this->textBox15->Location = System::Drawing::Point(76, 20);
                        this->textBox15->Name = L"textBox15";
                        this->textBox15->Size = System::Drawing::Size(57, 20);
                        this->textBox15->TabIndex = 1;
                        this->textBox15->Text = L"70";
                        this->textBox15->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                        //
                        // textBox20
                        //
                        this->textBox20->Location = System::Drawing::Point(105, 102);
                        this->textBox20->Name = L"textBox20";
                        this->textBox20->Size = System::Drawing::Size(100, 20);
                        this->textBox20->TabIndex = 1;
                        this->textBox20->Text = L"2";
                        this->textBox20->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                        //
                        // label9
                        //
                        this->label9->AutoSize = true;
                        this->label9->Location = System::Drawing::Point(51, 105);
                        this->label9->Name = L"label9";
                        this->label9->Size = System::Drawing::Size(48, 13);
                        this->label9->TabIndex = 0;
                        this->label9->Text = L"Time Ts:";
                        //
                        // groupBox8
                        //
                        this->groupBox8->Controls->Add(this->groupBox17);
                        this->groupBox8->Controls->Add(this->groupBox16);
                        this->groupBox8->Controls->Add(this->groupBox12);
                        this->groupBox8->Location = System::Drawing::Point(6, 46);
                        this->groupBox8->Name = L"groupBox8";
                        this->groupBox8->Size = System::Drawing::Size(314, 310);
                        this->groupBox8->TabIndex = 1;
                        this->groupBox8->TabStop = false;
                        this->groupBox8->Text = L"Position Control";
                        //
                        // groupBox17
                        //
                        this->groupBox17->Controls->Add(this->comboBox10);
                        this->groupBox17->Location = System::Drawing::Point(7, 256);
                        this->groupBox17->Name = L"groupBox17";
                        this->groupBox17->Size = System::Drawing::Size(301, 47);
                        this->groupBox17->TabIndex = 2;
                        this->groupBox17->TabStop = false;
                        this->groupBox17->Text = L"Trajectory function";
                        //
                        // comboBox10
                        //
                        this->comboBox10->FormattingEnabled = true;
                        this->comboBox10->Items->AddRange(gcnew cli::array< System::Object^
>(3) {L"Polynomial WB", L"Polynomial", L"Linear"});
                        this->comboBox10->Location = System::Drawing::Point(90, 18);
                        this->comboBox10->Name = L"comboBox10";
                        this->comboBox10->Size = System::Drawing::Size(121, 21);
                        this->comboBox10->TabIndex = 0;
                        this->comboBox10->SelectedIndex =0;
```

```
                        //
                        // groupBox16
                        //
                        this->groupBox16->Controls->Add(this->label6);
                        this->groupBox16->Controls->Add(this->textBox13);
                        this->groupBox16->Controls->Add(this->label5);
                        this->groupBox16->Location = System::Drawing::Point(7, 206);
                        this->groupBox16->Name = L"groupBox16";
                        this->groupBox16->Size = System::Drawing::Size(301, 44);
                        this->groupBox16->TabIndex = 1;
                        this->groupBox16->TabStop = false;
                        this->groupBox16->Text = L"Linear Velocity of gripper";
                        //
                        // label6
                        //
                        this->label6->AutoSize = true;
                        this->label6->Location = System::Drawing::Point(204, 21);
                        this->label6->Name = L"label6";
                        this->label6->Size = System::Drawing::Size(45, 13);
                        this->label6->TabIndex = 2;
                        this->label6->Text = L"mm/sec";
                        //
                        // textBox13
                        //
                        this->textBox13->Location = System::Drawing::Point(98, 18);
                        this->textBox13->Name = L"textBox13";
                        this->textBox13->Size = System::Drawing::Size(100, 20);
                        this->textBox13->TabIndex = 1;
                        this->textBox13->Text = L"100";
                        this->textBox13->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                        //
                        // label5
                        //
                        this->label5->AutoSize = true;
                        this->label5->Location = System::Drawing::Point(70, 21);
                        this->label5->Name = L"label5";
                        this->label5->Size = System::Drawing::Size(17, 13);
                        this->label5->TabIndex = 0;
                        this->label5->Text = L"V:";
                        //
                        // groupBox12
                        //
                        this->groupBox12->Controls->Add(this->button1);
                        this->groupBox12->Controls->Add(this->groupBox15);
                        this->groupBox12->Controls->Add(this->groupBox14);
                        this->groupBox12->Controls->Add(this->groupBox13);
                        this->groupBox12->Location = System::Drawing::Point(7, 19);
                        this->groupBox12->Name = L"groupBox12";
                        this->groupBox12->Size = System::Drawing::Size(301, 181);
                        this->groupBox12->TabIndex = 0;
                        this->groupBox12->TabStop = false;
                        this->groupBox12->Text = L"T desired Td: (4x4)";
                        //
                        // button1
                        //
                        this->button1->Location = System::Drawing::Point(268, 9);
                        this->button1->Name = L"button1";
                        this->button1->Size = System::Drawing::Size(25, 19);
                        this->button1->TabIndex = 3;
                        this->button1->Text = L"...";
                        this->button1->TextAlign =
System::Drawing::ContentAlignment::TopCenter;
                        this->button1->UseVisualStyleBackColor = true;
                        this->button1->Click += gcnew System::EventHandler(this,
&Form1::button1_Click);
```

```
//
// groupBox15
//
this->groupBox15->Controls->Add(this->label4);
this->groupBox15->Controls->Add(this->label3);
this->groupBox15->Controls->Add(this->label2);
this->groupBox15->Controls->Add(this->label1);
this->groupBox15->Location = System::Drawing::Point(12, 133);
this->groupBox15->Name = L"groupBox15";
this->groupBox15->Size = System::Drawing::Size(280, 39);
this->groupBox15->TabIndex = 2;
this->groupBox15->TabStop = false;
this->groupBox15->Text = L"(1x3)";
//
// label4
//
this->label4->AutoSize = true;
this->label4->Location = System::Drawing::Point(242, 17);
this->label4->Name = L"label4";
this->label4->Size = System::Drawing::Size(13, 13);
this->label4->TabIndex = 0;
this->label4->Text = L"1";
//
// label3
//
this->label3->AutoSize = true;
this->label3->Location = System::Drawing::Point(158, 16);
this->label3->Name = L"label3";
this->label3->Size = System::Drawing::Size(13, 13);
this->label3->TabIndex = 0;
this->label3->Text = L"0";
//
// label2
//
this->label2->AutoSize = true;
this->label2->Location = System::Drawing::Point(95, 17);
this->label2->Name = L"label2";
this->label2->Size = System::Drawing::Size(13, 13);
this->label2->TabIndex = 0;
this->label2->Text = L"0";
//
// label1
//
this->label1->AutoSize = true;
this->label1->Location = System::Drawing::Point(30, 16);
this->label1->Name = L"label1";
this->label1->Size = System::Drawing::Size(13, 13);
this->label1->TabIndex = 0;
this->label1->Text = L"0";
//
// groupBox14
//
this->groupBox14->Controls->Add(this->textBox12);
this->groupBox14->Controls->Add(this->textBox11);
this->groupBox14->Controls->Add(this->textBox3);
this->groupBox14->Location = System::Drawing::Point(223, 26);
this->groupBox14->Name = L"groupBox14";
this->groupBox14->Size = System::Drawing::Size(69, 101);
this->groupBox14->TabIndex = 1;
this->groupBox14->TabStop = false;
this->groupBox14->Text = L"(3x1) mm";
//
// textBox12
//
this->textBox12->Location = System::Drawing::Point(6, 71);
this->textBox12->Name = L"textBox12";
```

```
                    this->textBox12->Size = System::Drawing::Size(57, 20);
                    this->textBox12->TabIndex = 1;
                    this->textBox12->Text = L"999";
                    this->textBox12->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // textBox11
                    //
                    this->textBox11->Location = System::Drawing::Point(6, 45);
                    this->textBox11->Name = L"textBox11";
                    this->textBox11->Size = System::Drawing::Size(57, 20);
                    this->textBox11->TabIndex = 1;
                    this->textBox11->Text = L"369";
                    this->textBox11->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // textBox3
                    //
                    this->textBox3->Location = System::Drawing::Point(6, 19);
                    this->textBox3->Name = L"textBox3";
                    this->textBox3->Size = System::Drawing::Size(57, 20);
                    this->textBox3->TabIndex = 1;
                    this->textBox3->Text = L"1455";
                    this->textBox3->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // groupBox13
                    //
                    this->groupBox13->Controls->Add(this->textBox10);
                    this->groupBox13->Controls->Add(this->textBox7);
                    this->groupBox13->Controls->Add(this->textBox2);
                    this->groupBox13->Controls->Add(this->textBox9);
                    this->groupBox13->Controls->Add(this->textBox6);
                    this->groupBox13->Controls->Add(this->textBox1);
                    this->groupBox13->Controls->Add(this->textBox8);
                    this->groupBox13->Controls->Add(this->textBox4);
                    this->groupBox13->Controls->Add(this->textBox5);
                    this->groupBox13->Location = System::Drawing::Point(12, 26);
                    this->groupBox13->Name = L"groupBox13";
                    this->groupBox13->Size = System::Drawing::Size(200, 101);
                    this->groupBox13->TabIndex = 0;
                    this->groupBox13->TabStop = false;
                    this->groupBox13->Text = L"(3x3)";
                    //
                    // textBox10
                    //
                    this->textBox10->Location = System::Drawing::Point(135, 71);
                    this->textBox10->Name = L"textBox10";
                    this->textBox10->Size = System::Drawing::Size(57, 20);
                    this->textBox10->TabIndex = 1;
                    this->textBox10->Text = L"0";
                    this->textBox10->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // textBox7
                    //
                    this->textBox7->Location = System::Drawing::Point(135, 45);
                    this->textBox7->Name = L"textBox7";
                    this->textBox7->Size = System::Drawing::Size(57, 20);
                    this->textBox7->TabIndex = 1;
                    this->textBox7->Text = L"0";
                    this->textBox7->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // textBox2
                    //
```

```
                        this->textBox2->Location = System::Drawing::Point(134, 19);
                        this->textBox2->Name = L"textBox2";
                        this->textBox2->Size = System::Drawing::Size(57, 20);
                        this->textBox2->TabIndex = 1;
                        this->textBox2->Text = L"1";
                        this->textBox2->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                        //
                        // textBox9
                        //
                        this->textBox9->Location = System::Drawing::Point(72, 71);
                        this->textBox9->Name = L"textBox9";
                        this->textBox9->Size = System::Drawing::Size(57, 20);
                        this->textBox9->TabIndex = 1;
                        this->textBox9->Text = L"-1";
                        this->textBox9->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                        //
                        // textBox6
                        //
                        this->textBox6->Location = System::Drawing::Point(72, 45);
                        this->textBox6->Name = L"textBox6";
                        this->textBox6->Size = System::Drawing::Size(57, 20);
                        this->textBox6->TabIndex = 1;
                        this->textBox6->Text = L"0";
                        this->textBox6->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                        //
                        // textBox1
                        //
                        this->textBox1->Location = System::Drawing::Point(71, 19);
                        this->textBox1->Name = L"textBox1";
                        this->textBox1->Size = System::Drawing::Size(57, 20);
                        this->textBox1->TabIndex = 1;
                        this->textBox1->Text = L"0";
                        this->textBox1->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                        //
                        // textBox8
                        //
                        this->textBox8->Location = System::Drawing::Point(9, 71);
                        this->textBox8->Name = L"textBox8";
                        this->textBox8->Size = System::Drawing::Size(57, 20);
                        this->textBox8->TabIndex = 1;
                        this->textBox8->Text = L"0";
                        this->textBox8->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                        //
                        // textBox4
                        //
                        this->textBox4->Location = System::Drawing::Point(9, 45);
                        this->textBox4->Name = L"textBox4";
                        this->textBox4->Size = System::Drawing::Size(57, 20);
                        this->textBox4->TabIndex = 1;
                        this->textBox4->Text = L"-1";
                        this->textBox4->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                        //
                        // textBox5
                        //
                        this->textBox5->Location = System::Drawing::Point(8, 19);
                        this->textBox5->Name = L"textBox5";
                        this->textBox5->Size = System::Drawing::Size(57, 20);
                        this->textBox5->TabIndex = 1;
                        this->textBox5->Text = L"0";
```

```
                        this->textBox5->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                        //
                        // comboBox9
                        //
                        this->comboBox9->FormattingEnabled = true;
                        this->comboBox9->Items->AddRange(gcnew cli::array< System::Object^
>(5) {L"Position Control", L"Velocity ", L"Spaceball ",
                                L"Psychology Mask ", L"Touch Screen "});
                        this->comboBox9->Location = System::Drawing::Point(29, 19);
                        this->comboBox9->Name = L"comboBox9";
                        this->comboBox9->Size = System::Drawing::Size(269, 21);
                        this->comboBox9->TabIndex = 0;
                        this->comboBox9->SelectedIndex =0;
                        this->comboBox9->SelectedIndexChanged += gcnew
System::EventHandler(this, &Form1::comboBox9_SelectedIndexChanged);
                        //
                        // groupBox10
                        //
                        this->groupBox10->Controls->Add(this->comboBox11);
                        this->groupBox10->Controls->Add(this->groupBox22);
                        this->groupBox10->Controls->Add(this->groupBox21);
                        this->groupBox10->Location = System::Drawing::Point(540, 27);
                        this->groupBox10->Name = L"groupBox10";
                        this->groupBox10->Size = System::Drawing::Size(200, 371);
                        this->groupBox10->TabIndex = 9;
                        this->groupBox10->TabStop = false;
                        this->groupBox10->Text = L"Where to start\?";
                        //
                        // comboBox11
                        //
                        this->comboBox11->FormattingEnabled = true;
                        this->comboBox11->Items->AddRange(gcnew cli::array< System::Object^
>(3) {L"Ready Position", L"Current Position", L"Elsewhere"});
                        this->comboBox11->Location = System::Drawing::Point(41, 18);
                        this->comboBox11->Name = L"comboBox11";
                        this->comboBox11->Size = System::Drawing::Size(121, 21);
                        this->comboBox11->TabIndex = 10;
                        this->comboBox11->SelectedIndex =0;
                        this->comboBox11->SelectedIndexChanged += gcnew
System::EventHandler(this, &Form1::comboBox11_SelectedIndexChanged);
                        //
                        // groupBox22
                        //
                        this->groupBox22->Controls->Add(this->groupBox24);
                        this->groupBox22->Controls->Add(this->groupBox25);
                        this->groupBox22->Enabled = false;
                        this->groupBox22->Location = System::Drawing::Point(6, 106);
                        this->groupBox22->Name = L"groupBox22";
                        this->groupBox22->RightToLeft =
System::Windows::Forms::RightToLeft::No;
                        this->groupBox22->Size = System::Drawing::Size(188, 257);
                        this->groupBox22->TabIndex = 0;
                        this->groupBox22->TabStop = false;
                        this->groupBox22->Text = L"Location\?";
                        //
                        // groupBox24
                        //
                        this->groupBox24->Controls->Add(this->button6);
                        this->groupBox24->Controls->Add(this->textBox23);
                        this->groupBox24->Controls->Add(this->textBox24);
                        this->groupBox24->Controls->Add(this->textBox25);
                        this->groupBox24->Location = System::Drawing::Point(97, 19);
                        this->groupBox24->Name = L"groupBox24";
                        this->groupBox24->Size = System::Drawing::Size(85, 230);
                        this->groupBox24->TabIndex = 1;
```

```
                    this->groupBox24->TabStop = false;
                    this->groupBox24->Text = L"wci (mm,rad)";
                    //
                    // button6
                    //
                    this->button6->Location = System::Drawing::Point(47, 14);
                    this->button6->Name = L"button6";
                    this->button6->Size = System::Drawing::Size(25, 19);
                    this->button6->TabIndex = 3;
                    this->button6->Text = L"...";
                    this->button6->TextAlign =
System::Drawing::ContentAlignment::TopCenter;
                    this->button6->UseVisualStyleBackColor = true;
                    this->button6->Click += gcnew System::EventHandler(this,
&Form1::button6_Click);
                    //
                    // textBox23
                    //
                    this->textBox23->Location = System::Drawing::Point(15, 168);
                    this->textBox23->Name = L"textBox23";
                    this->textBox23->Size = System::Drawing::Size(57, 20);
                    this->textBox23->TabIndex = 1;
                    this->textBox23->Text = L"0";
                    this->textBox23->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // textBox24
                    //
                    this->textBox24->Location = System::Drawing::Point(15, 118);
                    this->textBox24->Name = L"textBox24";
                    this->textBox24->Size = System::Drawing::Size(57, 20);
                    this->textBox24->TabIndex = 1;
                    this->textBox24->Text = L"0";
                    this->textBox24->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // textBox25
                    //
                    this->textBox25->Location = System::Drawing::Point(15, 65);
                    this->textBox25->Name = L"textBox25";
                    this->textBox25->Size = System::Drawing::Size(57, 20);
                    this->textBox25->TabIndex = 1;
                    this->textBox25->Text = L"0";
                    this->textBox25->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // groupBox25
                    //
                    this->groupBox25->Controls->Add(this->button5);
                    this->groupBox25->Controls->Add(this->textBox29);
                    this->groupBox25->Controls->Add(this->textBox28);
                    this->groupBox25->Controls->Add(this->textBox30);
                    this->groupBox25->Controls->Add(this->textBox27);
                    this->groupBox25->Controls->Add(this->textBox26);
                    this->groupBox25->Controls->Add(this->textBox31);
                    this->groupBox25->Controls->Add(this->textBox32);
                    this->groupBox25->Location = System::Drawing::Point(6, 19);
                    this->groupBox25->Name = L"groupBox25";
                    this->groupBox25->Size = System::Drawing::Size(85, 230);
                    this->groupBox25->TabIndex = 0;
                    this->groupBox25->TabStop = false;
                    this->groupBox25->Text = L"qi (rad)";
                    //
                    // button5
                    //
                    this->button5->Location = System::Drawing::Point(44, 14);
```

```
                    this->button5->Name = L"button5";
                    this->button5->Size = System::Drawing::Size(25, 19);
                    this->button5->TabIndex = 3;
                    this->button5->Text = L"...";
                    this->button5->TextAlign =
System::Drawing::ContentAlignment::TopCenter;
                    this->button5->UseVisualStyleBackColor = true;
                    this->button5->Click += gcnew System::EventHandler(this,
&Form1::button5_Click);
                    //
                    // textBox29
                    //
                    this->textBox29->Location = System::Drawing::Point(14, 195);
                    this->textBox29->Name = L"textBox29";
                    this->textBox29->Size = System::Drawing::Size(57, 20);
                    this->textBox29->TabIndex = 1;
                    this->textBox29->Text = L"0";
                    this->textBox29->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // textBox28
                    //
                    this->textBox28->Location = System::Drawing::Point(14, 169);
                    this->textBox28->Name = L"textBox28";
                    this->textBox28->Size = System::Drawing::Size(57, 20);
                    this->textBox28->TabIndex = 1;
                    this->textBox28->Text = L"1.5708";
                    this->textBox28->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // textBox30
                    //
                    this->textBox30->Location = System::Drawing::Point(14, 91);
                    this->textBox30->Name = L"textBox30";
                    this->textBox30->Size = System::Drawing::Size(57, 20);
                    this->textBox30->TabIndex = 1;
                    this->textBox30->Text = L"0";
                    this->textBox30->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // textBox27
                    //
                    this->textBox27->Location = System::Drawing::Point(14, 143);
                    this->textBox27->Name = L"textBox27";
                    this->textBox27->Size = System::Drawing::Size(57, 20);
                    this->textBox27->TabIndex = 1;
                    this->textBox27->Text = L"1.5708";
                    this->textBox27->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // textBox26
                    //
                    this->textBox26->Location = System::Drawing::Point(14, 117);
                    this->textBox26->Name = L"textBox26";
                    this->textBox26->Size = System::Drawing::Size(57, 20);
                    this->textBox26->TabIndex = 1;
                    this->textBox26->Text = L"1.5708";
                    this->textBox26->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                    //
                    // textBox31
                    //
                    this->textBox31->Location = System::Drawing::Point(14, 65);
                    this->textBox31->Name = L"textBox31";
                    this->textBox31->Size = System::Drawing::Size(57, 20);
                    this->textBox31->TabIndex = 1;
```

```
                        this->textBox31->Text = L"1.5708";
                        this->textBox31->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                        //
                        // textBox32
                        //
                        this->textBox32->Location = System::Drawing::Point(14, 39);
                        this->textBox32->Name = L"textBox32";
                        this->textBox32->Size = System::Drawing::Size(57, 20);
                        this->textBox32->TabIndex = 1;
                        this->textBox32->Text = L"1.5708";
                        this->textBox32->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
                        //
                        // groupBox21
                        //
                        this->groupBox21->Controls->Add(this->comboBox12);
                        this->groupBox21->Location = System::Drawing::Point(6, 45);
                        this->groupBox21->Name = L"groupBox21";
                        this->groupBox21->RightToLeft =
System::Windows::Forms::RightToLeft::No;
                        this->groupBox21->Size = System::Drawing::Size(188, 55);
                        this->groupBox21->TabIndex = 0;
                        this->groupBox21->TabStop = false;
                        this->groupBox21->Text = L"Include park to ready\?";
                        //
                        // comboBox12
                        //
                        this->comboBox12->FormattingEnabled = true;
                        this->comboBox12->Items->AddRange(gcnew cli::array< System::Object^
>(2) {L"Yes", L"No"});
                        this->comboBox12->Location = System::Drawing::Point(35, 19);
                        this->comboBox12->Name = L"comboBox12";
                        this->comboBox12->Size = System::Drawing::Size(121, 21);
                        this->comboBox12->TabIndex = 10;
                        this->comboBox12->SelectedIndex =0;
                        //
                        // groupBox23
                        //
                        this->groupBox23->Controls->Add(this->comboBox13);
                        this->groupBox23->Controls->Add(this->groupBox27);
                        this->groupBox23->Location = System::Drawing::Point(540, 404);
                        this->groupBox23->Name = L"groupBox23";
                        this->groupBox23->Size = System::Drawing::Size(200, 110);
                        this->groupBox23->TabIndex = 9;
                        this->groupBox23->TabStop = false;
                        this->groupBox23->Text = L"Go back to ready\?";
                        //
                        // comboBox13
                        //
                        this->comboBox13->FormattingEnabled = true;
                        this->comboBox13->Items->AddRange(gcnew cli::array< System::Object^
>(2) {L"Yes", L"No"});
                        this->comboBox13->Location = System::Drawing::Point(41, 18);
                        this->comboBox13->Name = L"comboBox13";
                        this->comboBox13->Size = System::Drawing::Size(121, 21);
                        this->comboBox13->TabIndex = 10;
                        this->comboBox13->SelectedIndex =0;
                        this->comboBox13->SelectedIndexChanged += gcnew
System::EventHandler(this, &Form1::comboBox13_SelectedIndexChanged);
                        //
                        // groupBox27
                        //
                        this->groupBox27->Controls->Add(this->comboBox14);
                        this->groupBox27->Location = System::Drawing::Point(6, 45);
                        this->groupBox27->Name = L"groupBox27";
```

```
                        this->groupBox27->RightToLeft =
System::Windows::Forms::RightToLeft::No;
                        this->groupBox27->Size = System::Drawing::Size(188, 55);
                        this->groupBox27->TabIndex = 0;
                        this->groupBox27->TabStop = false;
                        this->groupBox27->Text = L"Go back to park\?";
                        //
                        // comboBox14
                        //
                        this->comboBox14->FormattingEnabled = true;
                        this->comboBox14->Items->AddRange(gcnew cli::array< System::Object^
>(2) {L"Yes", L"No"});
                        this->comboBox14->Location = System::Drawing::Point(35, 19);
                        this->comboBox14->Name = L"comboBox14";
                        this->comboBox14->Size = System::Drawing::Size(121, 21);
                        this->comboBox14->TabIndex = 10;
                        this->comboBox14->SelectedIndex =0;
                        //
                        // groupBox26
                        //
                        this->groupBox26->Controls->Add(this->button10);
                        this->groupBox26->Controls->Add(this->button9);
                        this->groupBox26->Location = System::Drawing::Point(540, 521);
                        this->groupBox26->Name = L"groupBox26";
                        this->groupBox26->Size = System::Drawing::Size(97, 88);
                        this->groupBox26->TabIndex = 10;
                        this->groupBox26->TabStop = false;
                        this->groupBox26->Text = L"Gripper";
                        //
                        // button10
                        //
                        this->button10->Location = System::Drawing::Point(11, 58);
                        this->button10->Name = L"button10";
                        this->button10->Size = System::Drawing::Size(75, 23);
                        this->button10->TabIndex = 11;
                        this->button10->Text = L"Close";
                        this->button10->UseVisualStyleBackColor = true;
                        //
                        // button9
                        //
                        this->button9->Location = System::Drawing::Point(11, 24);
                        this->button9->Name = L"button9";
                        this->button9->Size = System::Drawing::Size(75, 23);
                        this->button9->TabIndex = 11;
                        this->button9->Text = L"Open";
                        this->button9->UseVisualStyleBackColor = true;
                        //
                        // button7
                        //
                        this->button7->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 11, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
                        this->button7->Location = System::Drawing::Point(659, 527);
                        this->button7->Name = L"button7";
                        this->button7->Size = System::Drawing::Size(75, 25);
                        this->button7->TabIndex = 11;
                        this->button7->Text = L"Start";
                        this->button7->UseVisualStyleBackColor = true;
                        this->button7->Click += gcnew System::EventHandler(this,
&Form1::button7_Click);
                        //
                        // button8
                        //
                        this->button8->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 11, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
```

```
                        this->button8->Location = System::Drawing::Point(659, 584);
                        this->button8->Name = L"button8";
                        this->button8->Size = System::Drawing::Size(75, 25);
                        this->button8->TabIndex = 12;
                        this->button8->Text = L"Stop";
                        this->button8->UseVisualStyleBackColor = true;
                        this->button8->Click += gcnew System::EventHandler(this,
&Form1::button8_Click);
                        //
                        // menuStrip1
                        //
                        this->menuStrip1->Items->AddRange(gcnew cli::array<
System::Windows::Forms::ToolStripItem^  >(2) {this->fileToolStripMenuItem,
                                this->helpToolStripMenuItem});
                        this->menuStrip1->Location = System::Drawing::Point(0, 0);
                        this->menuStrip1->Name = L"menuStrip1";
                        this->menuStrip1->Size = System::Drawing::Size(752, 24);
                        this->menuStrip1->TabIndex = 13;
                        this->menuStrip1->Text = L"menuStrip1";
                        //
                        // fileToolStripMenuItem
                        //
                        this->fileToolStripMenuItem->DropDownItems->AddRange(gcnew
cli::array< System::Windows::Forms::ToolStripItem^  >(4) {this->openToolStripMenuItem,
                                this->saveToolStripMenuItem, this->saveAsToolStripMenuItem,
this->exitToolStripMenuItem});
                        this->fileToolStripMenuItem->Name = L"fileToolStripMenuItem";
                        this->fileToolStripMenuItem->Size = System::Drawing::Size(35, 20);
                        this->fileToolStripMenuItem->Text = L"File";
                        //
                        // openToolStripMenuItem
                        //
                        this->openToolStripMenuItem->Name = L"openToolStripMenuItem";
                        this->openToolStripMenuItem->Size = System::Drawing::Size(171, 22);
                        this->openToolStripMenuItem->Text = L"Open       Ctrl + O";
                        //
                        // saveToolStripMenuItem
                        //
                        this->saveToolStripMenuItem->Name = L"saveToolStripMenuItem";
                        this->saveToolStripMenuItem->Size = System::Drawing::Size(171, 22);
                        this->saveToolStripMenuItem->Text = L"Save        Ctrl + S";
                        //
                        // saveAsToolStripMenuItem
                        //
                        this->saveAsToolStripMenuItem->Name = L"saveAsToolStripMenuItem";
                        this->saveAsToolStripMenuItem->Size = System::Drawing::Size(171,
22);
                        this->saveAsToolStripMenuItem->Text = L"Save As   Ctrl + A";
                        //
                        // exitToolStripMenuItem
                        //
                        this->exitToolStripMenuItem->Name = L"exitToolStripMenuItem";
                        this->exitToolStripMenuItem->Size = System::Drawing::Size(171, 22);
                        this->exitToolStripMenuItem->Text = L"Exit          Ctrl + X";
                        //
                        // helpToolStripMenuItem
                        //
                        this->helpToolStripMenuItem->Name = L"helpToolStripMenuItem";
                        this->helpToolStripMenuItem->Size = System::Drawing::Size(40, 20);
                        this->helpToolStripMenuItem->Text = L"Help";
                        //
                        // button11
                        //
                        this->button11->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->button11->Location = System::Drawing::Point(659, 555);
                    this->button11->Name = L"button11";
                    this->button11->Size = System::Drawing::Size(75, 25);
                    this->button11->TabIndex = 11;
                    this->button11->Text = L"Restore";
                    this->button11->UseVisualStyleBackColor = true;
                    this->button11->Click += gcnew System::EventHandler(this,
&Form1::button11_Click);
                    //
                    // Form1
                    //
                    this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                    this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                    this->ClientSize = System::Drawing::Size(752, 619);
                    this->Controls->Add(this->button8);
                    this->Controls->Add(this->groupBox11);
                    this->Controls->Add(this->button11);
                    this->Controls->Add(this->button7);
                    this->Controls->Add(this->groupBox26);
                    this->Controls->Add(this->groupBox23);
                    this->Controls->Add(this->groupBox10);
                    this->Controls->Add(this->groupBox9);
                    this->Controls->Add(this->groupBox7);
                    this->Controls->Add(this->groupBox5);
                    this->Controls->Add(this->groupBox2);
                    this->Controls->Add(this->groupBox6);
                    this->Controls->Add(this->groupBox4);
                    this->Controls->Add(this->groupBox3);
                    this->Controls->Add(this->groupBox1);
                    this->Controls->Add(this->menuStrip1);
                    this->MaximizeBox = false;
                    this->Name = L"Form1";
                    this->Text = L"WMRA Main GUI";
                    this->Load += gcnew System::EventHandler(this, &Form1::Form1_Load);
                    this->groupBox11->ResumeLayout(false);
                    this->groupBox1->ResumeLayout(false);
                    this->groupBox2->ResumeLayout(false);
                    this->groupBox3->ResumeLayout(false);
                    this->groupBox4->ResumeLayout(false);
                    this->groupBox5->ResumeLayout(false);
                    this->groupBox6->ResumeLayout(false);
                    this->groupBox6->PerformLayout();
                    this->groupBox7->ResumeLayout(false);
                    this->groupBox9->ResumeLayout(false);
                    this->groupBox19->ResumeLayout(false);
                    this->groupBox19->PerformLayout();
                    this->groupBox18->ResumeLayout(false);
                    this->groupBox18->PerformLayout();
                    this->groupBox20->ResumeLayout(false);
                    this->groupBox20->PerformLayout();
                    this->groupBox8->ResumeLayout(false);
                    this->groupBox17->ResumeLayout(false);
                    this->groupBox16->ResumeLayout(false);
                    this->groupBox16->PerformLayout();
                    this->groupBox12->ResumeLayout(false);
                    this->groupBox15->ResumeLayout(false);
                    this->groupBox15->PerformLayout();
                    this->groupBox14->ResumeLayout(false);
                    this->groupBox14->PerformLayout();
                    this->groupBox13->ResumeLayout(false);
                    this->groupBox13->PerformLayout();
                    this->groupBox10->ResumeLayout(false);
                    this->groupBox22->ResumeLayout(false);
                    this->groupBox24->ResumeLayout(false);
                    this->groupBox24->PerformLayout();
                    this->groupBox25->ResumeLayout(false);
```

Appendix B (Continued)

```
                    this->groupBox25->PerformLayout();
                    this->groupBox21->ResumeLayout(false);
                    this->groupBox23->ResumeLayout(false);
                    this->groupBox27->ResumeLayout(false);
                    this->groupBox26->ResumeLayout(false);
                    this->menuStrip1->ResumeLayout(false);
                    this->menuStrip1->PerformLayout();
                    this->ResumeLayout(false);
                    this->PerformLayout();

                }
#pragma endregion
                private: System::Void Form1_Load(System::Object^  sender,
System::EventArgs^  e) {
                     Form1::CenterToScreen();
                 }
private: System::Void comboBox2_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e) {
        comboBox3->Items->Clear();
        this->comboBox3->Items->AddRange(gcnew cli::array< System::Object^  >(3) {L"Ground
Frame", L"Wheelchair Frame", L"Gripper Frame"});
        int Index2 = comboBox2->SelectedIndex;
        if (Index2==2){
                comboBox3->Items->RemoveAt(2);
                comboBox7->Enabled = false;
                checkBox1->Enabled = false;
                checkBox1->Checked = false;
                comboBox1->Enabled = false;
                comboBox1->SelectedIndex = 1;
        }
        else {
                comboBox7->Enabled = true;
                checkBox1->Enabled = true;
                comboBox1->Enabled = true;
                checkBox1->Checked = true;
        }
        }
private: System::Void comboBox1_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e) {
                    int Indexcombo1 = comboBox1->SelectedIndex;
                    int Indexcombo9 = comboBox9->SelectedIndex;
                    if (Indexcombo1==1){
                            if (Indexcombo9==0){
                                    groupBox13->Enabled = false;
                                    groupBox15->Enabled = false;
                            }
                            else {
                                    textBox18->Enabled = false;
                                    textBox17->Enabled = false;
                                    textBox19->Enabled = false;
                                    label8->Enabled = false;
                            }
                    }
                    else {
                            if (Indexcombo9==0){
                                    groupBox13->Enabled = true;
                                    groupBox15->Enabled = true;
                            }
                            else {
                                    textBox18->Enabled = true;
                                    textBox17->Enabled = true;
                                    textBox19->Enabled = true;
                                    label8->Enabled = true;
                            }
                    }
                    int selectedIndex = comboBox1->SelectedIndex;
```

```
		}
private: System::Void comboBox4_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e) {
			int Indexcombo4 = comboBox4->SelectedIndex;
			int Indexcombo5 = comboBox5->SelectedIndex;
			int Indexcombo6 = comboBox6->SelectedIndex;
			int Indexcombo11 = comboBox11->SelectedIndex;
			if (Indexcombo4==3){
				if (Indexcombo6==1){
					comboBox8->Enabled = true;
				}
				else{
					comboBox8->Enabled = false;
				}
				if (Indexcombo5==0){
					groupBox10->Enabled = true;
					comboBox12->Enabled = false;
					groupBox23->Enabled = false;

				}
				else{
					groupBox10->Enabled = true;
					comboBox12->Enabled = true;
					groupBox23->Enabled = true;
					comboBox8->Enabled = true;

				}
			}
			else {
				groupBox10->Enabled = true;
				groupBox23->Enabled = true;
				comboBox8->Enabled = true;
			}
		}
private: System::Void comboBox5_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e) {
			int Indexcombo4 = comboBox4->SelectedIndex;
			int Indexcombo5 = comboBox5->SelectedIndex;
			int Indexcombo6 = comboBox6->SelectedIndex;
			if (Indexcombo5==0){
				if (Indexcombo4==3){
					groupBox10->Enabled = true;
					comboBox12->Enabled = false;
					groupBox23->Enabled = false;
					if (Indexcombo6==1){
						comboBox8->Enabled = true;
					}
					else{
						comboBox8->Enabled = false;
					}
				}
			}
			else{
				groupBox10->Enabled = true;
				comboBox12->Enabled = true;
				groupBox23->Enabled = true;
				comboBox8->Enabled = true;
			}
		}
private: System::Void comboBox6_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e) {
			int Indexcombo6 = comboBox6->SelectedIndex;
			int Indexcombo4 = comboBox4->SelectedIndex;
			int Indexcombo5 = comboBox5->SelectedIndex;
			if (Indexcombo6==0){
				if (Indexcombo4==3 && Indexcombo5==0){
```

Appendix B (Continued)

```
                                        comboBox8->Enabled = false;
                                }
                        }
                        else {
                                comboBox8->Enabled = true;
                        }
                }
private: System::Void comboBox9_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e) {
                        int Indexcombo9 = comboBox9->SelectedIndex;
                        int Indexcombo1 = comboBox1->SelectedIndex;
                        if (Indexcombo9==1){
                                groupBox18->Enabled = true;
                                groupBox8->Enabled = false;
                                groupBox19->Enabled = false;
                                if (Indexcombo1==1){
                                        textBox18->Enabled = false;
                                        textBox17->Enabled = false;
                                        textBox19->Enabled = false;
                                        label8->Enabled = false;
                                }
                                else {
                                        textBox18->Enabled = true;
                                        textBox17->Enabled = true;
                                        textBox19->Enabled = true;
                                        label8->Enabled = true;
                                }
                        }
                        else if (Indexcombo9==2){
                                groupBox19->Enabled = true;
                                button3->Enabled = false;
                                groupBox8->Enabled = false;
                                groupBox18->Enabled = false;
                                textBox22->Enabled = false;
                        }
                        else if (Indexcombo9==3){
                                groupBox19->Enabled = true;
                                button3->Enabled = false;
                                groupBox8->Enabled = false;
                                groupBox18->Enabled = false;
                                textBox22->Enabled = true;
                        }
                        else if (Indexcombo9==4){
                                groupBox19->Enabled = true;
                                button3->Enabled = false;
                                groupBox8->Enabled = false;
                                groupBox18->Enabled = false;
                                textBox22->Enabled = false;

                                if (varscreenopn!=1){
                                        Form2 myForm;
                                        myForm.ShowDialog() ==
System::Windows::Forms::DialogResult::OK;
                                }
                        }
                        else {
                                groupBox8->Enabled = true;
                                groupBox19->Enabled = false;
                                groupBox18->Enabled = false;
                                if (Indexcombo1==1){
                                        groupBox13->Enabled = false;
                                        groupBox15->Enabled = false;
                                }
                                else {
                                        groupBox13->Enabled = true;
                                        groupBox15->Enabled = true;
```

Appendix B (Continued)

```
                        }
                    }
                }
private: System::Void comboBox11_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e) {
                    int Indexcombo11 = comboBox11->SelectedIndex;
                    int Indexcombo4 = comboBox4->SelectedIndex;
                    int Indexcombo5 = comboBox5->SelectedIndex;
                    if (Indexcombo11==0){
                        if (Indexcombo4!=3 || Indexcombo5==1){
                            comboBox12->Enabled = true;
                        }
                        groupBox22->Enabled = false;
                        this->textBox32->Text = L"1.5708";
                        this->textBox31->Text = L"1.5708";
                        this->textBox30->Text = L"0";
                        this->textBox26->Text = L"1.5708";
                        this->textBox27->Text = L"1.5708";
                        this->textBox28->Text = L"1.5708";
                        this->textBox29->Text = L"0";
                        this->textBox25->Text = L"0";
                        this->textBox24->Text = L"0";
                        this->textBox23->Text = L"0";
                    }
                    else if (Indexcombo11==1){
                        comboBox12->Enabled = false;
                        groupBox22->Enabled = false;
                        this->textBox32->Text = varQi(0,0).ToString();
                        this->textBox31->Text = varQi(1,0).ToString();
                        this->textBox30->Text = varQi(2,0).ToString();
                        this->textBox26->Text = varQi(3,0).ToString();
                        this->textBox27->Text = varQi(4,0).ToString();
                        this->textBox28->Text = varQi(5,0).ToString();
                        this->textBox29->Text = varQi(6,0).ToString();
                        this->textBox25->Text = varWCi(0,0).ToString();
                        this->textBox24->Text = varWCi(1,0).ToString();
                        this->textBox23->Text = varWCi(2,0).ToString();

                    }
                    else {
                        comboBox12->Enabled = false;
                        groupBox22->Enabled = true;
                    }
                }
private: System::Void comboBox13_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e) {
                    int Indexcombo13 = comboBox13->SelectedIndex;
                    if (Indexcombo13==1){
                        comboBox14->Enabled = false;
                    }
                    else {
                        comboBox14->Enabled = true;
                    }
                }
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
                    if (comboBox1->SelectedIndex == 0){
                        varmatrix=1;
                    }
                    else {
                        varmatrix=2;
                    }
                    Form3 myForm3;
                    myForm3.ShowDialog() == System::Windows::Forms::DialogResult::OK;
                    if (closevar==1){
                    if (varmatrix==1){
                        this->textBox5->Text = matrixentry(0,0).ToString();
```

```
                        this->textBox1->Text = matrixentry(0,1).ToString();
                        this->textBox2->Text = matrixentry(0,2).ToString();
                        this->textBox3->Text = matrixentry(0,3).ToString();
                        this->textBox4->Text = matrixentry(1,0).ToString();
                        this->textBox6->Text = matrixentry(1,1).ToString();
                        this->textBox7->Text = matrixentry(1,2).ToString();
                        this->textBox11->Text = matrixentry(1,3).ToString();
                        this->textBox8->Text = matrixentry(2,0).ToString();
                        this->textBox9->Text = matrixentry(2,1).ToString();
                        this->textBox10->Text = matrixentry(2,2).ToString();
                        this->textBox12->Text = matrixentry(2,3).ToString();
                    }
                    else {
                        this->textBox3->Text = matrixentry(0,1).ToString();
                        this->textBox11->Text = matrixentry(0,2).ToString();
                        this->textBox12->Text = matrixentry(0,3).ToString();
                    }
                    }
                }
private: System::Void button2_Click(System::Object^  sender, System::EventArgs^  e) {
                    if (comboBox1->SelectedIndex == 0){
                        varmatrix=1;
                    }
                    else {
                        varmatrix=2;
                    }
                    Form4 myForm4;
                    myForm4.ShowDialog() == System::Windows::Forms::DialogResult::OK;
                    if (closevar2==1){
                    if (varmatrix==1){
                        this->textBox14->Text = matrixentry2(0,0).ToString();
                        this->textBox15->Text = matrixentry2(0,1).ToString();
                        this->textBox16->Text = matrixentry2(0,2).ToString();

                        this->textBox18->Text = matrixentry2(1,0).ToString();
                        this->textBox17->Text = matrixentry2(1,1).ToString();
                        this->textBox19->Text = matrixentry2(1,2).ToString();
                    }
                    else {
                        this->textBox14->Text = matrixentry2(0,1).ToString();
                        this->textBox15->Text = matrixentry2(0,2).ToString();
                        this->textBox16->Text = matrixentry2(0,3).ToString();
                    }
                    }
                }
private: System::Void button3_Click(System::Object^  sender, System::EventArgs^  e) {
                    varloop = 0;
                    button11->Enabled = true;
                    button7->Enabled = true;
                    button8->Enabled = false;
                    button3->Enabled = false;
                }
private: System::Void button5_Click(System::Object^  sender, System::EventArgs^  e) {
                    Form5 myForm5;
                    myForm5.ShowDialog() == System::Windows::Forms::DialogResult::OK;
                    if (closevar3==1){
                        this->textBox32->Text = matrixentry3(0,0).ToString();
                        this->textBox31->Text = matrixentry3(0,1).ToString();
                        this->textBox30->Text = matrixentry3(0,2).ToString();

                        this->textBox26->Text = matrixentry3(0,3).ToString();
                        this->textBox27->Text = matrixentry3(0,4).ToString();
                        this->textBox28->Text = matrixentry3(0,5).ToString();
                        this->textBox29->Text = matrixentry3(0,6).ToString();
                    }
                }
```

```
private: System::Void button6_Click(System::Object^  sender, System::EventArgs^  e) {
                         Form6 myForm6;
                         myForm6.ShowDialog() == System::Windows::Forms::DialogResult::OK;
                         if (closevar4==1){
                                 this->textBox25->Text = matrixentry4(0,0).ToString();
                                 this->textBox24->Text = matrixentry4(0,1).ToString();
                                 this->textBox23->Text = matrixentry4(0,2).ToString();

                         }
                 }
private: System::Void button11_Click(System::Object^  sender, System::EventArgs^  e) {
                         //     1st Column
                         this->comboBox2->SelectedIndex = 0;
                         this->comboBox1->SelectedIndex = 0;
                         this->comboBox3->SelectedIndex = 0;
                         this->comboBox4->SelectedIndex = 0;
                         this->comboBox5->SelectedIndex = 0;
                         this->comboBox6->SelectedIndex = 0;
                         this->comboBox7->SelectedIndex = 0;
                         this->checkBox1->Checked = true;
                         this->checkBox2->Checked = true;
                         this->comboBox8->SelectedIndex = 0;
                         //     2nd Column
                         this->comboBox9->SelectedIndex = 0;
                         this->groupBox8->Enabled = true;
                         this->textBox5->Text = L"0";
                         this->textBox1->Text = L"0";
                         this->textBox2->Text = L"1";
                         this->textBox4->Text = L"-1";
                         this->textBox6->Text = L"0";
                         this->textBox7->Text = L"0";
                         this->textBox8->Text = L"0";
                         this->textBox9->Text = L"-1";
                         this->textBox10->Text = L"0";
                         this->textBox3->Text = L"1455";
                         this->textBox11->Text = L"369";
                         this->textBox12->Text = L"999";
                         this->textBox13->Text = L"100";
                         this->comboBox10->SelectedIndex = 0;
                         //     3rd Column
                         this->comboBox11->SelectedIndex = 0;
                         this->comboBox12->SelectedIndex = 0;
                         this->comboBox13->SelectedIndex = 0;
                         this->comboBox14->SelectedIndex = 0;
                 }
private: System::Void button8_Click(System::Object^  sender, System::EventArgs^  e) {
                         varstop = 1;
                         button11->Enabled = true;
                         button7->Enabled = true;
                         button3->Enabled = false;
                         button8->Enabled = false;
                 }
private: System::Void button7_Click(System::Object^  sender, System::EventArgs^  e) {
        varstop = 0;
        varDX.Null(7,1);
        button11->Enabled = false;
        button7->Enabled = false;
        button8->Enabled = true;

        // User input prompts:
        Matrix Td(4,4), Vd(3,1), qi(7,1), WCi(3,1);
        int error, td_err, notfilled, ts_err;
        error=0; td_err=0; notfilled=0; ts_err=0;
```

Appendix B (Continued)

```
       if ((comboBox9->SelectedIndex == 2) || (comboBox9->SelectedIndex == 3) ||
(comboBox9->SelectedIndex == 4)){
                button3->Enabled = true;
                varloop = 1;
        }

        if (checkBox1->Checked){choice50 = 1;}
        else {choice50 = 0;}
        if (checkBox2->Checked){choice500 = 1;}
        else {choice500 = 0;}
        choice5 = comboBox7->SelectedIndex;

        cart = comboBox1->SelectedIndex + 1;
        WCA = comboBox2->SelectedIndex + 1;
        if (WCA == 3){
                optim = 0;
                JLA = 0;
                JLO = choice500;
        }
        else {
                optim = choice5+1;
                JLA = choice50;
                JLO = choice500;
        }

        coord = comboBox3->SelectedIndex + 1;

        choice1 = comboBox4->SelectedIndex;
       if (choice1==1) {
                vr=0; ml=1;
        }
        else if (choice1==2){
                vr=1; ml=1;
        }
        else if (choice1==3){
                vr=0; ml=0;
        }
        else if (choice1==0){
                vr=1; ml=0;
        }

        arm = comboBox5->SelectedIndex;
        choice4 = comboBox6->SelectedIndex;
        plt = choice4+1;
        choice8 = comboBox8->SelectedIndex;

        if (varstop == 1){stop1 = 1;}
        else {stop1 = 0;}

        if (stop1 == 0){
                choice0 = comboBox9->SelectedIndex;
                cont = choice0+1;
                if (cont == 1){
                if (cart == 1){
                try {Td00 = Double::Parse( textBox5->Text );}
                catch ( Exception^ ) {error=1; notfilled=1;}
                try {Td01 = Double::Parse( textBox1->Text );}
                                catch ( Exception^ ){error=1; notfilled=1;}
                                try {Td02 = Double::Parse( textBox2->Text );}
                                catch ( Exception^ ){error=1; notfilled=1;}
                                try {Td10 = Double::Parse( textBox4->Text );}
                                catch ( Exception^ ){error=1; notfilled=1;}
                                try {Td11 = Double::Parse( textBox6->Text );}
                                catch ( Exception^ ){error=1; notfilled=1;}
                                try {Td12 = Double::Parse( textBox7->Text );}
                                catch ( Exception^ ){error=1; notfilled=1;}
```

Appendix B (Continued)

```
                                try {Td20 = Double::Parse( textBox8->Text );}
                                catch ( Exception^ ){error=1; notfilled=1;}
                                try {Td21 = Double::Parse( textBox9->Text );}
                                catch ( Exception^ ){error=1; notfilled=1;}
                                try {Td22 = Double::Parse( textBox10->Text );}
                                catch ( Exception^ ){error=1; notfilled=1;}

                                if ((Td00 > 1) || (Td00 < -1) || (Td01 > 1) || (Td01 < -1)
|| (Td02 > 1) || (Td02 < -1) || (Td10 > 1) || (Td10 < -1) || (Td11 > 1) || (Td11 < -1) ||
(Td12 > 1) || (Td12 < -1) || (Td20 > 1) || (Td20 < -1) || (Td21 > 1) || (Td21 < -1) ||
(Td22 > 1) || (Td22 < -1)) {
                                        error=1;
                                        td_err=1;
                                }
                        }
                        else {
                                Td00=1;
                                Td01=0;
                                Td02=0;
                                Td10=0;
                                Td11=1;
                                Td12=0;
                                Td20=0;
                                Td21=0;
                                Td22=1;
                        }
                        try {Td03 = Double::Parse( textBox3->Text );}
                        catch ( Exception^ ){error=1; notfilled=1;}
                        try {Td13 = Double::Parse( textBox11->Text );}
                        catch ( Exception^ ){error=1; notfilled=1;}
                        try {Td23 = Double::Parse( textBox12->Text );}
                        catch ( Exception^ ){error=1; notfilled=1;}

                        Td(0,0)=Td00;
                        Td(0,1)=Td01;
                        Td(0,2)=Td02;
                        Td(0,3)=Td03;
                        Td(1,0)=Td10;
                        Td(1,1)=Td11;
                        Td(1,2)=Td12;
                        Td(1,3)=Td13;
                        Td(2,0)=Td20;
                        Td(2,1)=Td21;
                        Td(2,2)=Td22;
                        Td(2,3)=Td23;
                        Td(3,0)=0;
                        Td(3,1)=0;
                        Td(3,2)=0;
                        Td(3,3)=1;

                        try { v = Double::Parse( textBox13->Text );}
                        catch ( Exception^ ){error=1; notfilled=1;}
                        trajf = comboBox10->SelectedIndex + 1;
                }
                else if (cont == 2){
                        try {Vd00 = Double::Parse( textBox14->Text );}
                        catch ( Exception^ ){error=1; notfilled=1;}
                        try {Vd01 = Double::Parse( textBox15->Text );}
                        catch ( Exception^ ){error=1; notfilled=1;}
                        try {Vd02 = Double::Parse( textBox16->Text );}
                        catch ( Exception^ ){error=1; notfilled=1;}

                        if (cart == 1){
                                try {Vd10 = Double::Parse( textBox18->Text );}
                                catch ( Exception^ ){error=1; notfilled=1;}
                                try {Vd11 = Double::Parse( textBox17->Text );}
```

Appendix B (Continued)

```
                        catch ( Exception^ ){error=1; notfilled=1;}
                        try {Vd12 = Double::Parse( textBox19->Text );}
                        catch ( Exception^ ){error=1; notfilled=1;}
                }

                if (cart == 2){
                        Vd.Null(3,0);
                        Vd(0,0) = Vd00;
                        Vd(1,0) = Vd01;
                        Vd(2,0) = Vd02;
                }
                else {
                        Vd.Null(6,1);
                        Vd(0,0) = Vd00;
                        Vd(1,0) = Vd01;
                        Vd(2,0) = Vd02;
                        Vd(3,0) = Vd10;
                        Vd(4,0) = Vd11;
                        Vd(5,0) = Vd12;
                }

                try {ts = Double::Parse( textBox20->Text );}
                catch ( Exception^ ){error=1; notfilled=1;}
                if (ts<0) {error=1; ts_err=1;}
        }
        else {
                try {v = Double::Parse( textBox21->Text );}
                catch ( Exception^ ){error=1; notfilled=1;}
                if (cont == 4){
                        try {port1 = int::Parse( textBox22->Text );}
                        catch ( Exception^ ){error=1; notfilled=1;}
                }
        }
}

if (stop1 == 0){
        if (varstop == 1){stop1 = 1;}

        else {stop1 = 0;}

        if (stop1 == 0){
                choice2 = comboBox11->SelectedIndex;
                start = choice2 + 1;
                if (start == 1){
                        if (vr == 1 || ml == 1 || arm == 1){
                                if (comboBox12->SelectedIndex == 0){ini = 1;}
                                else {ini = 0;}
                        }
                        else {ini = 0;}
                        float qi2 [7]= {90, 90, 0, 90, 90, 90, 0};
                        for ( j = 0 ; j < 7 ; j++ ) {
                                qi(j,0)=qi2[j]*d2r;
                        }
                        varQi = qi;
                        WCi.Null(3,1);
                        varWCi = WCi;
                }
                else {
                        try {qi00 = Double::Parse( textBox32->Text );}
                        catch ( Exception^ ) { error=1; notfilled=1;}
                        try {qi01 = Double::Parse( textBox31->Text );}
                        catch ( Exception^ ) { error=1; notfilled=1;}
                        try {qi02 = Double::Parse( textBox30->Text );}
                        catch ( Exception^ ) { error=1; notfilled=1;}
                        try {qi03 = Double::Parse( textBox26->Text );}
```

Appendix B (Continued)

```
                                catch ( Exception^ ) { error=1; notfilled=1;}

                                try {qi04 = Double::Parse( textBox27->Text );}
                                catch ( Exception^ ) { error=1; notfilled=1;}

                                try {qi05 = Double::Parse( textBox28->Text );}
                                catch ( Exception^ ) { error=1; notfilled=1;}

                                try {qi06 = Double::Parse( textBox29->Text );}
                                catch ( Exception^ ) { error=1; notfilled=1;}
                                qi(0,0)=qi00;
                                qi(1,0)=qi01;
                                qi(2,0)=qi02;
                                qi(3,0)=qi03;
                                qi(4,0)=qi04;
                                qi(5,0)=qi05;
                                qi(6,0)=qi06;
                                varQi = qi;

                                try {WCi00 = Double::Parse( textBox32->Text );}
                                catch ( Exception^ ) { error=1; notfilled=1;}

                                try {WCi01 = Double::Parse( textBox31->Text );}
                                catch ( Exception^ ) { error=1; notfilled=1;}

                                try {WCi02 = Double::Parse( textBox30->Text );}
                                catch ( Exception^ ) { error=1; notfilled=1;}
                                WCi(0,0)=WCi00;
                                WCi(1,0)=WCi01;
                                WCi(2,0)=WCi02;
                                varWCi = WCi;

                                ini = 0;
                        }

                        choice6 = comboBox13->SelectedIndex;
                        if (choice6 == 1) {choice7 = comboBox14->SelectedIndex;}
                }
        }

        if (error == 1){
                if (notfilled == 1){
                        MessageBox::Show("One or more required inputs are not filled or
filled wrongly","WMRA GUI Error", MessageBoxButtons::OK,MessageBoxIcon::Error);

                }
                if (td_err == 1){
                        MessageBox::Show("Elements of Rd (Rotation Matrix) should be in
between -1 to +1","WMRA GUI Error", MessageBoxButtons::OK,MessageBoxIcon::Error);

                }
                if (ts_err == 1){
                        MessageBox::Show("Ts should be greater than zero","WMRA GUI
Error", MessageBoxButtons::OK,MessageBoxIcon::Error);
}
        }
        Matrix qn(1,1), Tnwc(4,4);
        float phi;
        if ((stop1 == 0) && (error == 0)){   //Code Entry
                if (varstop == 1){
                        stop1 = 1;
                }
                else {
                        stop1 = 0;
                }
```

Appendix B (Continued)

```
		if (stop1 == 0) {// 1st point
			// Reading the Wheelchair's constant dimentions, all dimentions
are converted in millimeters:
			Matrix L(1,1);
			L=WMRA_WCD();

			// 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));

			//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);

			//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);

			Tiwc=Tiwco;

			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));
				}
				else if (coord==3){
					D=sqrt(pow(Td(0,3),2) + pow(Td(1,3),2) +
pow(Td(2,3),2));
				}
				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));
				}
				//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.
				// Calculating the Trajectory of the end effector, and
once the trajectory is calculated, we should redefine "Td" based on the ground frame:
				if (coord==2){
					Tt=WMRA_traj(trajf,Tia,Td,n+1);
					Td=Tiwc*Td;
				}
				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);
				}
			}
			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;
                                dt=0.05;
                        }
                        else if (cont==3){
                                FILE * fus;
                                fus = fopen("output.txt","w");
                                fprintf(fus,"0\t0\t0\t0\t0\t\n");
                                fclose(fus);
                                n=1;
                        }
                        else if (cont==4){
                        }
                        else {
                                dt=0.05;
                                n=1;
                        }

        if (varstop == 1){stop1 = 1;}
        else {stop1 = 0;}

        if (stop1 == 0) {     // 2nd point

                // 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);

                // Initializing the WMRA:
                if (ini==0){   // When no "park" to "ready" motion required.
                        // Initializing Virtual Reality Animation:
                        if (vr==1){
                                //WMRA_VR_Animation(1, Towc, qo);
                        }
                        // Initializing Robot Animation in Matlab Graphics:
                        if (ml==1){
                                //WMRA_ML_Animation(1, To, Td, Towc, T01, T12, T23, T34,
T45, T56, T67);
                        }
                        // Initializing the Physical Arm:
                        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)=dg;
                                WMRA_ARM_Motion(1, 2, qotemp, 0);
                                int check;
                                check=init(4, 19200, 2);
                                if (check == 0){
                                        MessageBox::Show("WMRA initialization has failed.
Please check your communications.","WMRA GUI Error",
MessageBoxButtons::OK,MessageBoxIcon::Error);

                                }
                                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);
                        if (arm==1){
                                ddt=0;
                        }
                }
                // Re-Drawing the Animation:
                Matrix Joa(6,7), Jowc(1,1), Jowctemp(1,1), Joatemp(1,1);
                float detJoa, detJo; //phi;

                if (varstop == 1){stop1 = 1;}
                else {stop1 = 0;}

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

                // Starting the Iteration Loop:
                while ((i<n) && (stop1 == 0)){ //3rd point
                        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);
                        // 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));

                        // 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){
```

Appendix B (Continued)

```
                        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);
                        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;
                }
                // 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);
                                                }
                                        }
                                        Joatemp=~Jo;
                                        Joatemp=Jo*Joatemp;
                                        detJo=sqrt(Joatemp.Det());
                                }

                                // 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;
```

Appendix B (Continued)

```
                                        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);
                                                }
                                                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');
                                        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);
                                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;
                        }
                        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);
                                dx(1,0) = atoi (dx2);
                                dx(2,0) = atoi (dx3);
                                dx(3,0) = atof (dx4);
                                dx(4,0) = atof (dx5);
                                dx(5,0) = atof (dx6);
                                dg = atoi (dgx);
                                varscreenopn = atoi (varscreenop);
                                dx*=(v*dt);                             }


                        // 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);
                        }
                        if (cart==2) {
                                dx.SetSize(3,1);
                        }

                        // Calculating the resolved rate with optimization:
                        // Index input values for "optim": 1= SR-I & WLN, 2= P-I & WLN, 3=
SR-I & ENE, 4= P-I & ENE:
                        if (WCA==2) {
                                dq.SetSize(7,1);
                                dq1.Null(7,1);
                                dq1=WMRA_Opt(optim, JLA, JLO, Joa, detJoa, dq, dx, dt,
q1);
                                dq1.SetSize(9,1);
                        }
                        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);
                                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);
                        }

                        if (varstop == 1){stop1 = 1;}
                        else {stop1 = 0;}

                        if (stop1 == 0) {     // 4th point
```

Appendix B (Continued)

```
                        // Calculating the new Joint Angles:
                qn=q1+dq1;

                        // 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);
                        // 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);
                                }
                                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);
                        }

                        // 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;
                                                WMRA_ARM_Motion(2, 1,
dqtemp, ddt);
                                                ddt=0;
                                        }
                                }

                                // Updating the old values with the new
values for the next iteration:
                                tt=tt+dt;

                                // Stopping the simulation when the exit
button is pressed:
                                if (cont==3 || cont==4){
                                        if (varloop == 1){
                                                n=n+1;
                                        }
```

Appendix B (Continued)

```
                                                        else {
                                                                break;
                                                        }
                                                }
                                        }

                                        //drawnow 5th point
                                        if (varstop == 1){stop1 = 1;}
                                        else {stop1 = 0;}

                                        i++;
                                }

                                //drawnow 6th point
                                if (varstop == 1){stop1 = 1;}
                                else {stop1 = 0;}

                                if (stop1 == 0) {       // 7th point
                                        // Reading the elapsed time and printing it with the
simulation time:

                                        // 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:
                                                if (choice6==1){
                                                        WMRA_any2ready(2, vr, ml, arm, Tnwc,
qn);

                                                        // Going back to the parking
position:

                                                        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); }
                                                }

                                                // Closing the Arm library and Matlab
Graphics Animation and Virtual Reality Animation and Plots windows:
                                                if (choice8==1){
                                                        Matrix temp(1,1);
                                                        temp.Null(1,1);
                                                        if (arm==1){
                                                                WMRA_ARM_Motion(3, 0, temp,
0);

                                                        }
                                                        if (vr==1){}
                                                        if (ml==1){}
                                                        if (plt==2){


                        }
                                                }
                                        }
                                }
                        }
                }
        }
        for (j=0;j<7;j++){
                varQi(j,0)=qn(j,0);
        }
        varWCi(0,0)=Tnwc(0,3);
```

Appendix B (Continued)

```
        varWCi(1,0)=Tnwc(1,3);
        varWCi(2,0)=phi;

        int selectedIndexpos = comboBox11->SelectedIndex;
         if (selectedIndexpos==1){
                textBox32->Text = varQi(0,0).ToString();
                textBox31->Text = varQi(1,0).ToString();
                textBox30->Text = varQi(2,0).ToString();
                textBox26->Text = varQi(3,0).ToString();
                textBox27->Text = varQi(4,0).ToString();
                textBox28->Text = varQi(5,0).ToString();
                textBox29->Text = varQi(6,0).ToString();

                textBox25->Text = varWCi(0,0).ToString();
                textBox24->Text = varWCi(1,0).ToString();
                textBox23->Text = varWCi(2,0).ToString();
         }


         button11->Enabled = true;
         button7->Enabled = true;
         button8->Enabled = false;
         button3->Enabled = false;

                MessageBox::Show("Done",
                "WMRA GUI", MessageBoxButtons::OK,
                MessageBoxIcon::Information);
}

private: System::Void comboBox7_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e) {
                }
};
}
```

Appendix B (Continued)

B.2 WMRA Touch-Screen

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

#pragma once

//double VAR_DX[7];
double VAR_DX[7]={0.0};

//int VAR_DX[7]={0};
//int VAR_DXnadis;
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;
```

Appendix B (Continued)

```
        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;
        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;
                        //
                        // checkBox14
                        //
                        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")));
                        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;
                        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
                    //
                    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
```

Appendix B (Continued)

```
                        //
                        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)));
                        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.
                        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;
        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;
                        this->checkBox6->FlatAppearance->BorderColor=Color::Red;
                        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;
                        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;
         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;
                        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;
            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;
```

Appendix B (Continued)

```
                        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);
                    }
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);
                    }
};
}
```

Appendix B (Continued)

B.3 Exit-Pause Window

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% COPY RIGHTS RESERVED %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Developed By: Ana Catalina Torres %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% April 2010 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#pragma once

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

double varscreenopn;
double varexit;

namespace Exit {

	/// <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;
			}
		}
	private: System::Windows::Forms::Button^  button1;
	private: System::Windows::Forms::Button^  button2;
	protected:

	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)
		{
			this->button1 = (gcnew System::Windows::Forms::Button());
			this->button2 = (gcnew System::Windows::Forms::Button());
			this->SuspendLayout();
			//
			// button1
			//
			this->button1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 12, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
				static_cast<System::Byte>(0)));
			this->button1->Location = System::Drawing::Point(46, 25);
			this->button1->Name = L"button1";
			this->button1->Size = System::Drawing::Size(80, 30);
			this->button1->TabIndex = 0;
			this->button1->Text = L"EXIT";
			this->button1->UseVisualStyleBackColor = true;
			this->button1->Click += gcnew System::EventHandler(this,
&Form1::button1_Click);
			//
			// button2
			//
			this->button2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 9.75F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
				static_cast<System::Byte>(0)));
			this->button2->Location = System::Drawing::Point(46, 70);
			this->button2->Name = L"button2";
			this->button2->Size = System::Drawing::Size(80, 30);
			this->button2->TabIndex = 1;
			this->button2->Text = L"Stop";
			this->button2->UseVisualStyleBackColor = true;
			this->button2->Click += gcnew System::EventHandler(this,
&Form1::button2_Click);
			//
			// Form1
			//
			this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
			this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
			this->ClientSize = System::Drawing::Size(172, 121);
			this->Controls->Add(this->button2);
			this->Controls->Add(this->button1);
			this->Name = L"Form1";
			this->Text = L"Exit";
			this->Load += gcnew System::EventHandler(this, &Form1::Form1_Load);
			this->ResumeLayout(false);

		}
#pragma endregion
	private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e) {
				Form1::CenterToScreen();
				varscreenopn = 1;
				varexit=1;
				FILE * ex;
				ex = fopen("outputexit.txt","w");
				fprintf(ex,"%1.f\t%1.f\t\n",varscreenopn, varexit);
				fclose(ex);
			 }
	private: System::Void button1_Click(System::Object^  sender, System::EventArgs^
e) {
				varscreenopn = 0;
				varexit=1;
				Form1::Close();
				FILE * ex;
				ex = fopen("outputexit.txt","w");
```

Appendix B (Continued)

```
                                fprintf(ex,"%1.f\t%1.f\t\n",varscreenopn, varexit);
                                fclose(ex);
                        }
        private: System::Void button2_Click(System::Object^  sender, System::EventArgs^
e) {
                                varscreenopn = 1;
                                varexit=0;
                                FILE * ex;
                                ex = fopen("outputexit.txt","w");
                                fprintf(ex,"%1.f\t%1.f\t\n",varscreenopn, varexit);
                                fclose(ex);
                        }
        };
}
```

Appendix C: Spaceball Project

C.1 SpaceBall File

```
// prjSpaceBall.cpp : Defines the entry point for the application.
//

#include "stdafx.h"


HDC         hdc;          //Handle to Device Context used to draw on screen
HWND        hWndMain;     //Handle to Main Window
SiHdl       devHdl;       //Handle to Spaceball Device

#define MAX_LOADSTRING 100

//Global Variables:
HINSTANCE hInst;                                                        // current
instance
TCHAR szTitle[MAX_LOADSTRING];                                      // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];                // The title bar text
int data[6]; //DATA[] IS WHERE SPACEBALL OUTPUT IS ****
FILE *results;

//foward declarations of functions included in this code module:
ATOM                        MyRegisterClass(HINSTANCE hInstance);
BOOL                        InitInstance(HINSTANCE, int);
LRESULT CALLBACK      WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK      About(HWND, UINT, WPARAM, LPARAM);

using namespace std;

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{

        MSG msg;
        HACCEL hAccelTable;

        int  res;  //SbInits result..if>0 it worked, if=0 it didnt work
        int            num;      //number of button pressed
        BOOL           handled;  //is message handled yet
        SiSpwEvent     Event;    //SpaceWare Event
        SiGetEventData EData;    //SpaceWare Event Data

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_PRJSPACEBALL, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }

        handled = SPW_FALSE;     //init handled

        res = SbInit();                      //intitialize spaceball
        //if SpaceBall was not detected then print error, close win., exit program
```

Appendix C (Continued)

```
        if (res < 1) {
    MessageBox(hWndMain,
                "Sorry - No supported Spacetec IMC device available.\n",
                        NULL, MB_OK);

    if (hWndMain != NULL) {
       DestroyWindow(hWndMain);    //destroy window
    }
    ExitProcess(1);                //exit program
    }

        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_PRJSPACEBALL);

        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
        {
                handled = SPW_FALSE;
                //init Window platform specific data for a call to SiGetEvent
                SiGetEventWinInit(&EData, msg.message, msg.wParam, msg.lParam);
                //check whether msg was a Spaceball event and process it
                if (SiGetEvent (devHdl, 0, &EData, &Event) == SI_IS_EVENT)
        {
                        if (Event.type == SI_MOTION_EVENT) {
                                SbMotionEvent(&Event);        //process Spaceball motion
event
                        }
                        if (Event.type == SI_ZERO_EVENT){
                                SbZeroEvent();                //process Spaceball zero
event
                        }
                        if (Event.type == SI_BUTTON_EVENT){
                                if ((num = SiButtonPressed (&Event)) != SI_NO_BUTTON) {
                                        SbButtonPressEvent(num);        //process Spaceball
button event
                                }
                                if ((num = SiButtonReleased (&Event)) != SI_NO_BUTTON) {
                                        SbButtonReleaseEvent(num);        //process
Spaceball button event
                                }
                    }
                   handled = SPW_TRUE;              //spaceball event handled
        }

                //not a Spaceball event, let windows handle it
                if (handled == SPW_FALSE) {
                        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                        {
                                TranslateMessage(&msg);
                                DispatchMessage(&msg);
                        }
                }
        }
        return msg.wParam;
}

//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
//  COMMENTS:
//
//    This function and its usage is only necessary if you want this code
//    to be compatible with Win32 systems prior to the 'RegisterClassEx'
//    function that was added to Windows 95. It is important to call this function
```

```
//    so that the application will get 'well formed' small icons associated
//    with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProc;
        wcex.cbClsExtra                 = 0;
        wcex.cbWndExtra                 = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_PRJSPACEBALL);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = (LPCSTR)IDC_PRJSPACEBALL;
        wcex.lpszClassName      = szWindowClass;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

        return RegisterClassEx(&wcex);
}

//
//   FUNCTION: InitInstance(HANDLE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   HWND hWnd;

   hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);


   if (!hWnd)
   {
      return FALSE;
   }

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);
   hWndMain = hWnd;
   //get handle of our window to draw on
   hdc = GetDC(hWndMain);

   //print buffers
   TextOut(hdc, 0 , 0, "Robotics Lab -> Spaceball Ready", 15);
   TextOut(hdc, 15 , 100, "TX: 0", 5);
   TextOut(hdc, 15 , 120, "TY: 0", 5);
   TextOut(hdc, 15 , 140, "TZ: 0", 5);
   TextOut(hdc, 15 , 160, "RX: 0", 5);
   TextOut(hdc, 15 , 180, "RY: 0", 5);
   TextOut(hdc, 15 , 200, "RZ: 0", 5);

   //release our window handle
   ReleaseDC(hWndMain,hdc);

   UpdateWindow ( hWndMain );
```

Appendix C (Continued)

```
   return TRUE;
}

//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
//  PURPOSE:  Processes messages for the main window.
//
//  WM_COMMAND - process the application menu
//  WM_PAINT   - Paint the main window
//  WM_DESTROY - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
        TCHAR szHello[MAX_LOADSTRING];
        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch (message)
        {
                case WM_COMMAND:
                        wmId    = LOWORD(wParam);
                        wmEvent = HIWORD(wParam);
                        // Parse the menu selections:
                        switch (wmId)
                        {
                                case IDM_ABOUT:
                                   DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
(DLGPROC)About);
                                   break;
                                case IDM_EXIT:
                                   DestroyWindow(hWnd);
                                   break;
                                default:
                                   return DefWindowProc(hWnd, message, wParam, lParam);
                        }
                        break;
                case WM_ACTIVATEAPP:
                        hdc = GetDC(hWnd);
                        //print buffers
                        TextOut(hdc, 0 , 0, "Spaceball Ready", 15);
                        TextOut(hdc, 15 , 100, "TX: 0", 5);
                        TextOut(hdc, 15 , 120, "TY: 0", 5);
                        TextOut(hdc, 15 , 140, "TZ: 0", 5);
                        TextOut(hdc, 15 , 160, "RX: 0", 5);
                        TextOut(hdc, 15 , 180, "RY: 0", 5);
                        TextOut(hdc, 15 , 200, "RZ: 0", 5);

                        //release our window handle */
                        ReleaseDC(hWnd,hdc);
                case WM_KEYDOWN:
                case WM_KEYUP:
                        //user hit a key to close program */
                        if (wParam == VK_ESCAPE){
                                SendMessage(hWndMain, WM_CLOSE, 0, 0l );
                        }
                break;
                case WM_PAINT:
                        hdc = BeginPaint(hWnd, &ps);
                        RECT rt;
                        GetClientRect(hWnd, &rt);
                        DrawText(hdc, szHello, strlen(szHello), &rt, DT_CENTER);
                        EndPaint(hWnd, &ps);
```

```
                        break;
                case WM_DESTROY:
                        PostQuitMessage(0);
                        break;
                default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
   }
   return 0;
}

// Mesage handler for about box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
        switch (message)
        {
                case WM_INITDIALOG:
                                return TRUE;

                case WM_COMMAND:
                        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                        break;
        }
    return FALSE;
}

/*--------------------------------------------------------------------------
 * Function: SbInit()
 *
 * Description:
 *    This function initializes the Spaceball and opens ball for use.
 * Args: None
 * Return Value:
 *    int  res         result of SiOpen, =0 if Fail =1 if it Works
 *--------------------------------------------------------------------------*/
int SbInit(void) {
  int res;                              //result of SiOpen, to be returned
  SiOpenData oData;                     //OS Independent data to open ball

  //init the SpaceWare input library
  if (SiInitialize() == SPW_DLL_LOAD_ERROR)
     {
         MessageBox(hWndMain,"Error: Could not load SiAppDll dll files",
                        NULL, MB_ICONEXCLAMATION);
       }

  SiOpenWinInit (&oData, hWndMain);                 //init Win. platform specific data
  SiSetUiMode(devHdl, SI_UI_ALL_CONTROLS);   //Config SoftButton Win Display

  //open data, which will check for device type and return the device handle
  //to be used by this function

  if ( (devHdl = SiOpen("3DxTest32", SI_ANY_DEVICE, SI_NO_MASK,
                         SI_EVENT, &oData)) == NULL)
     {
     SiTerminate();  //called to shut down the SpaceWare input library
     res = 0;        //could not open device
     return res;
     }
  else
     {
     res = 1;        //opened device succesfully
     return res;
```

```
    }
}

/*-----------------------------------------------------------------------
 * Function: SbMotionEvent()
 * Description:
 *    This function recieves motion information and prints out the info
 *    on screen.
 * Args:
 *    SiSpwEvent *pEvent   Containts Data from a Spaceball Event
 * Return Value:
 *    NONE
 *-----------------------------------------------------------------------*/
void SbMotionEvent(SiSpwEvent *pEvent)
{
       int i = 0;
       char buff0[20];                          //text buffer for TX
       char buff1[20];                          //text buffer for TY
       char buff2[20];                          //text buffer for TZ
       char buff3[20];                          //text buffer for RX
       char buff4[20];                          //text buffer for RY
       char buff5[20];                          //text buffer for RZ

       int len0,len1,len2,len3,len4,len5;                  //length of each buffer

       data[0] = pEvent->u.spwData.mData[SI_TX];
       data[1] = pEvent->u.spwData.mData[SI_TY];
       data[2] = pEvent->u.spwData.mData[SI_TZ];
       data[3] = pEvent->u.spwData.mData[SI_RX];
       data[4] = pEvent->u.spwData.mData[SI_RY];
       data[5] = pEvent->u.spwData.mData[SI_RZ];

       //put the actual ball data into the buffers */
       len0= sprintf( buff0, "TX: %d         ", data[0] );
       len1= sprintf( buff1, "TY: %d         ", data[1] );
       len2= sprintf( buff2, "TZ: %d         ", data[2] );
       len3= sprintf( buff3, "RX: %d         ", data[3] );
       len4= sprintf( buff4, "RY: %d         ", data[4] );
       len5= sprintf( buff5, "RZ: %d         ", data[5] );

       //PUT CODE HERE TO RUN WMRA
       if ((results = fopen("output.txt", "w")) == NULL)
              fprintf(stderr, "Cannot open file");
       else
       {
              //fputs("TX\tTY\tTZ\tRX\tRY\tRZ\n",results);
              fprintf(results,"%i\t%i\t%i\t%i\t%i\t%i\n", data[0], data[1], data[2],
data[3], data[4], data[5]);
              fclose(results);
       }

       //get handle of our window to draw on
       hdc = GetDC(hWndMain);

   //print buffers
       TextOut(hdc, 0 , 0, "Motion Event               ", 28);
       TextOut(hdc, 15 , 100, buff0, len0);
       TextOut(hdc, 15 , 120, buff1, len1);
       TextOut(hdc, 15 , 140, buff2, len2);
       TextOut(hdc, 15 , 160, buff3, len3);
       TextOut(hdc, 15 , 180, buff4, len4);
       TextOut(hdc, 15 , 200, buff5, len5);

   //release our window handle
       ReleaseDC(hWndMain,hdc);
}
```

Appendix C (Continued)

```
/*--------------------------------------------------------------------
 * Function: SbZeroEvent()
 * Description:
 *    This function clears the previous data, no motion data was recieved
 * Args:
 *    NONE
 * Return Value:
 *    NONE
 *-------------------------------------------------------------------*/
void SbZeroEvent(void)
{
        //get handle of our window to draw on
        hdc = GetDC(hWndMain);

        //print null data
        TextOut(hdc, 0 , 0, "Zero Event                  ", 28);
        TextOut(hdc, 15 , 100, "TX: 0          ", 15);
        TextOut(hdc, 15 , 120, "TY: 0          ", 15);
        TextOut(hdc, 15 , 140, "TZ: 0          ", 15);
        TextOut(hdc, 15 , 160, "RX: 0          ", 15);
        TextOut(hdc, 15 , 180, "RY: 0          ", 15);
        TextOut(hdc, 15 , 200, "RZ: 0          ", 15);

        //release our window handle
        ReleaseDC(hWndMain,hdc);
}

/*--------------------------------------------------------------------
 * Function: SbButtonPressEvent()
 * Description:
 *    This function recieves Spaceball button information and prints out the
 *    info on screen.
 * Args:
 *    int     buttonnumber   //Containts number of button pressed
 * Return Value:
 *    NONE
 *-------------------------------------------------------------------*/
void SbButtonPressEvent(int buttonnumber)
{
   //get handle of our window to draw on
   hdc = GetDC(hWndMain);

   //print button pressed(does not include rezero button)
   switch (buttonnumber)
      {
      case SI_APP_FIT_BUTTON:           // #31 defined in si.h
         TextOut(hdc, 0 , 0, "Fit Button Pressed ", 20);
         break;
      case 1:
         TextOut(hdc, 0 , 0, "Button 1 Pressed ", 17);
         break;
      case 2:
         TextOut(hdc, 0 , 0, "Button 2 Pressed ", 17);
         break;
      case 3:
         TextOut(hdc, 0 , 0, "Button 3 Pressed ", 17);
         break;
      case 4:
         TextOut(hdc, 0 , 0, "Button 4 Pressed ", 17);
         break;
      case 5:
         TextOut(hdc, 0 , 0, "Button 5 Pressed ", 17);
         break;
      case 6:
         TextOut(hdc, 0 , 0, "Button 6 Pressed ", 17);
         break;
```

```
      case 7:
         TextOut(hdc, 0 , 0, "Button 7 Pressed ", 17);
         break;
      case 8:
         TextOut(hdc, 0 , 0, "Button 8 Pressed ", 17);
         break;
      default:
         TextOut(hdc, 0 , 0, "Button ? Pressed ", 17);
         break;
      }
    //release our window handle
    ReleaseDC(hWndMain,hdc);
}
/*-------------------------------------------------------------------------
 * Function: SbButtonReleaseEvent()
 * Description:
 *    This function recieves Spaceball button information and prints out the
 *    info on screen.
 * Args:
 *    int     buttonnumber   //Containts number of button pressed
 * Return Value:
 *    NONE
 *-------------------------------------------------------------------------*/
void SbButtonReleaseEvent(int buttonnumber)
{
   //get handle of our window to draw on
   hdc = GetDC(hWndMain);

   //print button pressed(does not include rezero button)
   switch (buttonnumber)
      {
      case SI_APP_FIT_BUTTON:           /* #31 defined in si.h*/
         TextOut(hdc, 0 , 0, "Fit Button Released", 20);
         break;
      case 1:
         TextOut(hdc, 0 , 0, "Button 1 Released", 17);
         break;
      case 2:
         TextOut(hdc, 0 , 0, "Button 2 Released", 17);
         break;
      case 3:
         TextOut(hdc, 0 , 0, "Button 3 Released", 17);
         break;
      case 4:
         TextOut(hdc, 0 , 0, "Button 4 Released", 17);
         break;
      case 5:
         TextOut(hdc, 0 , 0, "Button 5 Released", 17);
         break;
      case 6:
         TextOut(hdc, 0 , 0, "Button 6 Released", 17);
         break;
      case 7:
         TextOut(hdc, 0 , 0, "Button 7 Released", 17);
         break;
      case 8:
         TextOut(hdc, 0 , 0, "Button 8 Released", 17);
         break;
      default:
         TextOut(hdc, 0 , 0, "Button ? Released", 17);
         break;
      }

    //release our window handle
    ReleaseDC(hWndMain,hdc);
}
```

Appendix C (Continued)

C.2 3DxTest32 File

```
/*-----------------------------------------------------------------------
 *  3DxTest32.c  -- Basic Win32 Program to initinalize a spaceball, read
 *                  its data and print it out.
 *
 *  $Id: 3DxTest32.c,v 1.8.2.1.2.1 1998/05/26 18:13:04 equerze Exp $
 *
 *  Written By Elio Querze
 *
 *  NOTE: MUST LINK WITH SIAPP.LIB
 *
 *-----------------------------------------------------------------------
 *
 * (C) 1998-2005 3Dconnexion. All rights reserved.
 * Permission to use, copy, modify, and distribute this software for all
 * purposes and without fees is hereby grated provided that this copyright
 * notice appears in all copies.  Permission to modify this software is granted
 * and 3Dconnexion will support such modifications only is said modifications are
 * approved by 3Dconnexion.
 *
 */

static char cvsId[]="(C) 1997-2005 Spacetec IMC Corporation: $Id: 3DxTest32.c,v
1.8.2.1.2.1 1998/05/26 18:13:04 equerze Exp $";


/* Standard Win32 Includes */
#include <stdio.h>
#include <math.h>
#include <windows.h>
#include <float.h>
#include <stdlib.h>


/* SpaceWare Specific Includes */

#include "spwmacro.h"  /* Common macros used by SpaceWare functions. */
#include "si.h"        /* Required for any SpaceWare support within an app.*/
#include "siapp.h"     /* Required for siapp.lib symbols */


/* Program Specific Includes */

#include "3DxTest32.h"

//January 18/2006
#pragma comment( lib, "opengl32.lib" )       // Search For OpenGL32.lib While Linking
#pragma comment( lib, "glu32.lib" )          // Search For GLu32.lib While Linking
#pragma comment( lib, "glaux.lib" )
#pragma comment( lib, "siapp.lib")
#pragma comment( lib, "spwmath.lib")
//end

/* Function Definitions */

/*-----------------------------------------------------------------------
 * Function: WinMain()
 *
 * Description:
 *    This is the main window function and we use it to initialize data
 *    and then call our loop function. This is a std. Win32 function.
 *
 * Args:
```

```
 *    HINSTANCE hInstance         // handle to current instance
 *    HINSTANCE hPrevInstance     // handle to previous instance
 *    LPSTR     lpszCmdLine       // pointer to command line
 *    int       nCmdShow          // show state of window
 *
 * Return Value:
 *    int       Returns the return value of DispatchLoopNT
 *
 *-----------------------------------------------------------------------*/
int
WINAPI WinMain ( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                     LPSTR lpszCmdLine, int nCmdShow )
{
  int  res;            /* SbInits result..if>0 it worked, if=0 it didnt work   */
  int  hsize, vsize;   /* size of window to be created,for each Dimension */

  /* Set Window Size */
  hsize = 280;
  vsize = 215;

  /* create our apps window */
  CreateSPWindow(0, 0, hsize, vsize, "3DxTest32");

  /* update screen */
  InvalidateRect(hWndMain, NULL, FALSE);

  /* intitialize spaceball */
  res = SbInit();

  /* if SpaceBall was not detected then print error, close win., exit prog. */
  if (res < 1)
    {
    MessageBox(hWndMain,
               "Sorry - No supported Spacetec IMC device available.\n",
                          NULL, MB_OK);
    if (hWndMain != NULL)
       {
       DestroyWindow(hWndMain);    /* destroy window */
       }

    ExitProcess(1);               /* exit program */
    }

  /* Function To be Repeated */
  return(DispatchLoopNT());

}


/*-----------------------------------------------------------------------
 * Function: SbInit()
 *
 * Description:
 *    This function initializes the Spaceball and opens ball for use.
 *
 *
 * Args: None
 *
 *
 * Return Value:
 *    int  res         result of SiOpen, =0 if Fail =1 if it Works
 *
 *-----------------------------------------------------------------------*/
int
SbInit()
```

Appendix C (Continued)

```
{
  int res;                              /* result of SiOpen, to be returned  */
  SiOpenData oData;                     /* OS Independent data to open ball  */

  /*init the SpaceWare input library */
  if (SiInitialize() == SPW_DLL_LOAD_ERROR)
     {
          MessageBox(hWndMain,"Error: Could not load SiAppDll dll files",
                         NULL, MB_ICONEXCLAMATION);
        }

  SiOpenWinInit (&oData, hWndMain);    /* init Win. platform specific data  */
  SiSetUiMode(devHdl, SI_UI_ALL_CONTROLS); /* Config SoftButton Win Display */

  /* open data, which will check for device type and return the device handle
     to be used by this function */
  if ( (devHdl = SiOpen ("3DxTest32", SI_ANY_DEVICE, SI_NO_MASK,
                          SI_EVENT, &oData)) == NULL)
     {
     SiTerminate();  /* called to shut down the SpaceWare input library */
     res = 0;        /* could not open device */
     return res;
     }
  else
     {
     res = 1;        /* opened device succesfully */
     return res;
     }
}


/*-----------------------------------------------------------------------------
 * Function: DispatchLoopNT()
 *
 * Description:
 *    This function contains the main message loop which constantly checks for
 *    SpaceBall Events and handles them apropriately.
 *
 * Args: None
 *
 *
 * Return Value:
 *    int  msg.wparam                  // event passed to window
 *
 *---------------------------------------------------------------------------*/
int
DispatchLoopNT()
{
   int            num;      /* number of button pressed */
   MSG            msg;      /* incoming message to be evaluated */
   BOOL           handled;  /* is message handled yet */
   SiSpwEvent     Event;    /* SpaceWare Event */
   SiGetEventData EData;    /* SpaceWare Event Data */

   handled = SPW_FALSE;     /* init handled */

   /* start message loop */
   while ( GetMessage( &msg, NULL, 0, 0 ) )
      {
          handled = SPW_FALSE;

      /* init Window platform specific data for a call to SiGetEvent */
      SiGetEventWinInit(&EData, msg.message, msg.wParam, msg.lParam);

      /* check whether msg was a Spaceball event and process it */
      if (SiGetEvent (devHdl, 0, &EData, &Event) == SI_IS_EVENT)
```

Appendix C (Continued)

```
        {
        if (Event.type == SI_MOTION_EVENT)
           {
           SbMotionEvent(&Event);        /* process Spaceball motion event */
                       }
        if (Event.type == SI_ZERO_EVENT)
           {
           SbZeroEvent();                /* process Spaceball zero event */
                       }
        if (Event.type == SI_BUTTON_EVENT)
           {
           if ((num = SiButtonPressed (&Event)) != SI_NO_BUTTON)
              {
              SbButtonPressEvent(num);        /* process Spaceball button event */
              }
           if ((num = SiButtonReleased (&Event)) != SI_NO_BUTTON)
              {
              SbButtonReleaseEvent(num);        /* process Spaceball button event */
              }
                  }

                  handled = SPW_TRUE;              /* spaceball event handled */
        }

      /* not a Spaceball event, let windows handle it */
      if (handled == SPW_FALSE)
             {
             TranslateMessage( &msg );
           DispatchMessage( &msg );
           }
      }

   return( (int) msg.wParam );
}


/*--------------------------------------------------------------------------
 * Function: HandleNTEvent
 *
 * Description:  This is a std. Win32 function to handle various window events
 *
 *
 *
 * Args: HWND hWnd                    // handle to window
 *       unsigned msg                 // message to process
 *       WPARAM wParam                // 32 bit msg parameter
 *       LPARAM lParam                // 32 bit msg parameter
 *
 * Return Value:
 *    int  msg.wparam                 // program done
 *
 *--------------------------------------------------------------------------*/
LRESULT
WINAPI HandleNTEvent ( HWND hWnd, unsigned msg, WPARAM wParam,
                           LPARAM lParam )

{
  PAINTSTRUCT ps;           /* used to paint the client area of a window */
  LONG addr;                /* address of our window */

  addr = GetClassLong(hWnd, 0);  /* get address */

  switch ( msg )
     {
     case WM_ACTIVATEAPP:
       hdc = GetDC(hWnd);
```

Appendix C (Continued)

```
        /* print buffers */
        TextOut(hdc, 0 , 0, "Spaceball Ready", 15);
        TextOut(hdc, 15 , 100, "TX: 0", 5);
        TextOut(hdc, 15 , 120, "TY: 0", 5);
        TextOut(hdc, 15 , 140, "TZ: 0", 5);
        TextOut(hdc, 15 , 160, "RX: 0", 5);
        TextOut(hdc, 15 , 180, "RY: 0", 5);
        TextOut(hdc, 15 , 200, "RZ: 0", 5);

    /*release our window handle */
    ReleaseDC(hWnd,hdc);
      case WM_KEYDOWN:
      case WM_KEYUP:
        /* user hit a key to close program */
        if (wParam == VK_ESCAPE)
          {
          SendMessage ( hWndMain, WM_CLOSE, 0, 0l );
          }
        break;

      case WM_PAINT:
        /* time to paint the window */
        if (addr)
           {
           hdc = BeginPaint ( hWndMain, &ps );
           EndPaint ( hWndMain, &ps );
           }

        break;

      case WM_CLOSE:
        /* cleanup the object info created */

        break;

      case WM_DESTROY :
        PostQuitMessage (0);
        return (0);
      }
   return DefWindowProc ( hWnd, msg, wParam, lParam );

}


/*-------------------------------------------------------------------------
 * Function: CreateSPWindow
 *
 * Description:  This creates the window for our app
 *
 *
 *
 * Args:  int  atx        // horiz. start point to put window
 *        int  aty        // vert.  start point to put window
 *        int  hi         // hight of window
 *        int  wid        // width of window
 *        char *string    // window caption
 *
 * Return Value:
 *    NONE
 *
 *-------------------------------------------------------------------------*/
void
CreateSPWindow (int atx, int aty, int hi, int wid, char *string)

{
   WNDCLASS wndclass;        /* our own instance of the window class */
```

```
   HANDLE      hInst;        /* handle to our instance */

   hInst = NULL;             /* init handle */

   /* Register display window class */
   wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
   wndclass.lpfnWndProc   = (WNDPROC)HandleNTEvent ;
   wndclass.cbClsExtra    = 8 ;
   wndclass.cbWndExtra    = 0 ;
   wndclass.hInstance     = hInst;
   wndclass.hIcon         = NULL;
   wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
   wndclass.hbrBackground = GetStockObject(WHITE_BRUSH);
   wndclass.lpszMenuName  = NULL ;
   wndclass.lpszClassName = "3DxTest32";

   RegisterClass (&wndclass) ;

   /* create the window */
   hWndMain = CreateWindow ( "3DxTest32",           /*Window class name*/
                             string,               /*Window caption*/
                                          WS_OVERLAPPEDWINDOW, /*Window Style*/
                             atx,aty,wid,hi,
                             NULL,                  /*parent window handle*/
                                          NULL,                /*window menu handle*/
                                          hInst,                   /*program instance
handle*/
                                          NULL);               /*creation parameters*/


   /* display the window */
   ShowWindow ( hWndMain, SW_SHOW );

   /* get handle of our window to draw on */
   hdc = GetDC(hWndMain);

   /* print buffers */
   TextOut(hdc, 0 , 0, "Spaceball Ready", 15);
   TextOut(hdc, 15 , 100, "TX: 0", 5);
   TextOut(hdc, 15 , 120, "TY: 0", 5);
   TextOut(hdc, 15 , 140, "TZ: 0", 5);
   TextOut(hdc, 15 , 160, "RX: 0", 5);
   TextOut(hdc, 15 , 180, "RY: 0", 5);
   TextOut(hdc, 15 , 200, "RZ: 0", 5);

   /*release our window handle */
   ReleaseDC(hWndMain,hdc);

   UpdateWindow ( hWndMain );

} /* end of CreateWindow */


/*---------------------------------------------------------------------
 * Function: SbMotionEvent()
 *
 * Description:
 *    This function recieves motion information and prints out the info
 *    on screen.
 *
 *
 * Args:
 *    SiSpwEvent *pEvent   Containts Data from a Spaceball Event
 *
 * Return Value:
 *    NONE
```

```
 *
 *-----------------------------------------------------------------------*/
void
SbMotionEvent(SiSpwEvent *pEvent)
{


   char buff0[20];                              /* text buffer for TX */
   char buff1[20];                              /* text buffer for TY */
        char buff2[20];                              /* text buffer for TZ */
        char buff3[20];                              /* text buffer for RX */
        char buff4[20];                              /* text buffer for RY */
        char buff5[20];                              /* text buffer for RZ */

   int len0,len1,len2,len3,len4,len5;       /* length of each buffer */


   /* put the actual ball data into the buffers */
        len0= sprintf( buff0, "TX: %d          ", pEvent->u.spwData.mData[SI_TX] );
        len1= sprintf( buff1, "TY: %d          ", pEvent->u.spwData.mData[SI_TY] );
        len2= sprintf( buff2, "TZ: %d          ", pEvent->u.spwData.mData[SI_TZ] );
        len3= sprintf( buff3, "RX: %d          ", pEvent->u.spwData.mData[SI_RX] );
        len4= sprintf( buff4, "RY: %d          ", pEvent->u.spwData.mData[SI_RY] );
        len5= sprintf( buff5, "RZ: %d          ", pEvent->u.spwData.mData[SI_RZ] );


   /* get handle of our window to draw on */
        hdc = GetDC(hWndMain);

   /* print buffers */
   TextOut(hdc, 0 , 0, "Motion Event                ", 28);
   TextOut(hdc, 15 , 100, buff0, len0);
        TextOut(hdc, 15 , 120, buff1, len1);
        TextOut(hdc, 15 , 140, buff2, len2);
        TextOut(hdc, 15 , 160, buff3, len3);
        TextOut(hdc, 15 , 180, buff4, len4);
        TextOut(hdc, 15 , 200, buff5, len5);

   /*release our window handle */
        ReleaseDC(hWndMain,hdc);
}


/*-----------------------------------------------------------------------
 * Function: SbZeroEvent()
 *
 * Description:
 *    This function clears the previous data, no motion data was recieved
 *
 *
 *
 * Args:
 *    NONE
 *
 * Return Value:
 *    NONE
 *
 *-----------------------------------------------------------------------*/
void
SbZeroEvent()
{
   /* get handle of our window to draw on */
   hdc = GetDC(hWndMain);

   /* print null data */
   TextOut(hdc, 0 , 0, "Zero Event                ", 28);
```

```
   TextOut(hdc, 15 , 100, "TX: 0          ", 15);
        TextOut(hdc, 15 , 120, "TY: 0          ", 15);
   TextOut(hdc, 15 , 140, "TZ: 0          ", 15);
   TextOut(hdc, 15 , 160, "RX: 0          ", 15);
   TextOut(hdc, 15 , 180, "RY: 0          ", 15);
   TextOut(hdc, 15 , 200, "RZ: 0          ", 15);

   /*release our window handle */
   ReleaseDC(hWndMain,hdc);
}



/*-----------------------------------------------------------------------
 * Function: SbButtonPressEvent()
 *
 * Description:
 *    This function recieves Spaceball button information and prints out the
 *    info on screen.
 *
 *
 * Args:
 *    int     buttonnumber   //Containts number of button pressed
 *
 * Return Value:
 *    NONE
 *
 *-----------------------------------------------------------------------*/
void
SbButtonPressEvent(int buttonnumber)
{
   /* get handle of our window to draw on */
   hdc = GetDC(hWndMain);

   /* print button pressed(does not include rezero button) */
   switch (buttonnumber)
      {
      case SI_APP_FIT_BUTTON:           /* #31 defined in si.h*/
         TextOut(hdc, 0 , 0, "Fit Button Pressed ", 20);
         break;
      case 1:
         TextOut(hdc, 0 , 0, "Button 1 Pressed ", 17);
         break;
      case 2:
         TextOut(hdc, 0 , 0, "Button 2 Pressed ", 17);
         break;
      case 3:
         TextOut(hdc, 0 , 0, "Button 3 Pressed ", 17);
         break;
      case 4:
         TextOut(hdc, 0 , 0, "Button 4 Pressed ", 17);
         break;
      case 5:
         TextOut(hdc, 0 , 0, "Button 5 Pressed ", 17);
         break;
      case 6:
         TextOut(hdc, 0 , 0, "Button 6 Pressed ", 17);
         break;
      case 7:
         TextOut(hdc, 0 , 0, "Button 7 Pressed ", 17);
         break;
      case 8:
         TextOut(hdc, 0 , 0, "Button 8 Pressed ", 17);
         break;

      default:
```

```
         TextOut(hdc, 0 , 0, "Button ? Pressed ", 17);
         break;
      }
    /*release our window handle */
    ReleaseDC(hWndMain,hdc);
}

/*--------------------------------------------------------------------------
 * Function: SbButtonReleaseEvent()
 *
 * Description:
 *    This function recieves Spaceball button information and prints out the
 *    info on screen.
 *
 *
 * Args:
 *    int     buttonnumber   //Containts number of button pressed
 *
 * Return Value:
 *    NONE
 *
 *--------------------------------------------------------------------------*/
void
SbButtonReleaseEvent(int buttonnumber)
{
   /* get handle of our window to draw on */
   hdc = GetDC(hWndMain);
   /* print button pressed(does not include rezero button) */
   switch (buttonnumber)
      {
      case SI_APP_FIT_BUTTON:           /* #31 defined in si.h*/
         TextOut(hdc, 0 , 0, "Fit Button Released", 20);
         break;
      case 1:
         TextOut(hdc, 0 , 0, "Button 1 Released", 17);
         break;
      case 2:
         TextOut(hdc, 0 , 0, "Button 2 Released", 17);
         break;
      case 3:
         TextOut(hdc, 0 , 0, "Button 3 Released", 17);
         break;
      case 4:
         TextOut(hdc, 0 , 0, "Button 4 Released", 17);
         break;
      case 5:
         TextOut(hdc, 0 , 0, "Button 5 Released", 17);
         break;
      case 6:
         TextOut(hdc, 0 , 0, "Button 6 Released", 17);
         break;
      case 7:
         TextOut(hdc, 0 , 0, "Button 7 Released", 17);
         break;
      case 8:
         TextOut(hdc, 0 , 0, "Button 8 Released", 17);
         break;

      default:
         TextOut(hdc, 0 , 0, "Button ? Released", 17);
         break;

      }
    /*release our window handle */
    ReleaseDC(hWndMain,hdc);
}
```