TeachMover Virtual Command and D-H Assist: MATLAB GUIs for Simulation and Offline Programming of the Microbot TeachMover, and Visualization of Denavit-Hartenberg Parameters

An Undergraduate Thesis presented to the

Faculty of Electrical / Electronics and

Communications Engineering Department

University of San Carlos

Cebu City, Philippines

In Partial Fulfillment

Of the Requirements for the Degree

In

Bachelor of Science

in Electrical Engineering

Proponents:

Aliño, Al-Wilson

Monisit, Kristofer

Ramo, Job Jacob

Rubin, Ian

March 2009

Engr. Rafael D. Seva, Jr., MEEE
Adviser

ABSTRACT

The D-H Assist and TeachMover Virtual Command programs are built using MATLAB® Handle Graphics® tools that allow users to create custom graphical user interfaces in the MATLAB environment.

The D-H Assist program that has been developed allows instructors and students to visually construct models of serial robotic manipulators using Denavit-Hartenberg parameters. It consists of three-dimensional viewports and sliders which can easily visualize the effects of changes of D-H parameters on the overall structure of the robot arm. Once the parameters have been finalized, the movement of the robot arm can be simulated.

The TeachMover Virtual Command that has been developed simplifies and extends the operation and programming of the Microbot TeachMover. Using the on-screen interface, an operator can simulate and record motion sequences for the TeachMover. The visual simulation is the result of forward and inverse kinematics solutions. These sequences can be implemented to the TeachMover online or offline. In online mode, the TeachMover receives successive instructions from Virtual Command for each movement it performs. In offline mode, Virtual Command will relay the motion sequence into TeachMover memory for independent operation at a later time. Offline mode allows the TeachMover to perform the sequence even without receiving further instructions from Virtual Command.

APPROVAL SHEET

This Design Project entitled **TeachMover Virtual Command and D-H Assist: MATLAB GUIs for Simulation and Offline Programming of the Microbot TeachMover, and Visualization of Denavit-Hartenberg Parameters**, prepared and submitted by **Al-Wilson Aliño**, **Kristofer Monisit**, **Job Jacob Ramo**, and **Ian Rubin** in partial fulfillment of the requirements for the degree of **Bachelor of Science in Electrical Engineering,** has been examined and is recommended for acceptance and approval for Final Defense.

DESIGN PROJECT COMMITTEE

Joseph Karl G. Salva, MEECE
Chairman

Carlos C. Tan, M. Eng'g.
Member

Ryan M. Alocilja, BSECE
Member

Rafael D. Seva, Jr., MEEE
Adviser

PANEL OF ORAL EXAMINERS

Joseph Karl G. Salva, MEECE
Chairman

Carlos C. Tan, M. Eng'g.
Member

Ryan M. Alocilja, BSECE
Member

Rafael D. Seva, Jr., MEEE
Adviser

Accepted in partial fulfillment of the requirements for the degree of **Bachelor of Science in Electrical Engineering**.

February 27, 2009
Date of Project Defense

Thamar M. Tan, ECE
Department Chairman

ACKNOWLEDGEMENTS

This Study will not be a success without the help of the people who have their active support and benevolence. The researchers would like to express their heartfelt gratitude to the following:

To Our Lord Jesus our Messiah and Savior, for guiding us all the way in making this project a success,

To our adviser Engr. Rafael D. Seva Jr., for allowing us to borrow his books, past thesis documents, for his advice and giving us a room, computer and the TeachMover® for us to work on our project,

To our panelists, Engr. Ryan M. Alocilja, Engr. Carlos C. Tan and our panel chairman Engr. Joseph Karl G. Salva for their questions, comments, suggestions and guidance in drawing out the best in us,

To the Special Applications Laboratory (SAL) staff: Knyl, John, Chris, Berlai, Jolly, and Jeric, for allowing us to use the lab office and for accompanying us during our thesis sessions,

To Engr. Warlito Nimis, for allowing us to use the Power Audit room as an alternative room for our thesis session,

To our classmates and friends, for encouraging and inspiring us to give the best of our potential,

To Nanay Luming, for preparing us delicious Lunch during our thesis sessions at home,

And to our parents for loving and supporting us morally and financially,

Thank you very much!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

THE PROBLEM AND ITS SETTING

1.1 Introduction

Majority of today’s industries use automation systems in some or all of their industrial processes. These automation systems may consist of sensors, conveyor belts, actuators, motors, robotic manipulators, programmable logic controllers and human-machine interfaces. One of the most versatile and adaptable of these components is the robotic manipulator.

A definition used by the Robot Institute of America in (Fu et al., 1987) states:

> A robot is a reprogrammable multi-functional manipulator designed to move materials, parts, tools, or specialized devices, through variable programmed motions for the performance of a variety of tasks.

Industrial manipulator arms easily satisfy this definition. Since it is reprogrammable, an industrial robot can be adapted to perform a variety of tasks, such as stacking items, pick-and-place, welding, drilling, and painting. These tasks may be unsuitable for humans because they are too repetitive, physically demanding, or are usually performed in hazardous environments. Robots are controlled with the help of sensors, image processing, and motion sequence programs. Major robot manufacturers even provide simulation software which can reduce industrial downtime and provide safety for the engineers who program the robots (Critchlow, 1985).

The increasing use of robots in the industry has created a demand for engineers with a background in this technology. In the academic setting, robotics education heavily draws on the concepts of kinematics of rigid body motion. The subject builds on the students’ prior knowledge of trigonometry, geometry, vectors, and matrix algebra. It integrates these math skills in order to describe the position and orientation of rigid bodies in two- and three-dimensional space. Thus the subject taps extensively into the visual-spatial learning and communication abilities of the students and the instructor.

One of the educational tools in this field is the Microbot TeachMover industrial robotic arm. It can be controlled and programmed using its built-in teach pendant, lead-by-the-nose, or through its serial interface. Since the TeachMover’s embedded software is only limited to moving the joints, grip sensing, and program storage, the serial interface provides a venue for some of the more creative and innovative uses of the TeachMover. Through the serial interface, the TeachMover can be connected to a more powerful computer. This computer may be tasked to do joint angle calculations and simulations before relaying the actual commands to the TeachMover.

Robotics education and simulation programs have been implemented in several programming languages, such as C++ and Java. MATLAB also has comprehensive visualization and animation capabilities which can assist in robotics education.

1.2 Statement of the problem

Presently, the EE/ECE Department, University of San Carlos, Philippines, offers a Robotics course as part of its graduate programs. To augment the theory and mathematics, course lecturers use improvised physical models representing coordinate frames and PowerPoint presentations as visual aids in teaching the subject. It will be very helpful for the class to have an interactive software package that can present visualizations of coordinate frames in two- and three-dimensional space in the process of deriving the Denavit-Hartenberg parameters for robotic manipulators.

An offline simulation and programming software will also be developed for the Microbot TeachMover. Its serial interface implementation requires the user to type in commands in the TeachMover’s programming language. To enable the students to control the TeachMover without having to learn its specialized language, this software will provide an interactive visual environment for controlling and programming the TeachMover through its serial interface.

Specifically, the study intends to do the following:

1. Understand the capabilities and serial interface commands of the Microbot Teachmover.
2. Using MATLAB’s graphical user interface (GUI) programming capabilities, develop a software program that will visualize the construction of robotic links according to Denavit-Hartenberg parameters.

3. Develop a GUI-based software program using MATLAB that will:
   a. Present an interactive 3D model of the Microbot TeachMover.
   b. Allow the user to simulate control of the TeachMover.
   c. Record motion sequence programs for the TeachMover.
   d. Translate these programs into the TeachMover’s programming language.
   e. Relay these programs to the TeachMover through the serial interface.

1.3 Significance of the study

The study will have two main outputs: a) the coordinate frame visualization software and b) the Microbot TeachMover simulator and programmer. Both software packages can serve as a laboratory platform as well as a lecture demonstration tool for instructors, which can assist senior and graduate level robotics students in their studies.

As lecture demonstration tools, the software can help instructors visually convey to students the basic mathematical concepts related to robotics. In the laboratory setting, experiment modules can be designed for the software to provide students with practical hands-on experience especially in deriving Denavit-Hartenberg parameters and in controlling the TeachMover.

The TeachMover simulator and programmer can possibly revive interest in the two robotic arm units already present in the

Special Applications Laboratory. In today’s computing environment, beginner to intermediate computer users usually prefer the more intuitive graphical user interfaces (GUI) over text-based command line interfaces (CLI). With the use of MATLAB’s GUI programming tools, mouse actions and minimal keyboard inputs can be translated to the programming language specified by the TeachMover, thus smoothing out the learning curve required to be able to control and program the TeachMover.

1.4 Scope and limitations of the study

The coordinate frame visualization software will focus on visualizing the derivation of Denavit-Hartenberg parameters and the effect of varying these parameters on the structure of the serial link robotic arm. Parallel link robotic arms will not be covered.

The TeachMover simulator and programmer will focus on point-to-point motion control of the robotic arm. Other types of motion control, such as trajectory planning and obstacle avoidance will be beyond the capabilities of the software. Aside from the TeachMover’s built-in grip sensor, there will be no other environment sensing techniques employed. The programmer will also be able to record motion sequences. The software will only be able to control the Microbot TeachMover. No other robotic arms can be controlled.

The software will serve as laboratory platforms upon which future experiment modules can be based. Laboratory experiment modules will not be a part of the output of this study.

1.5 Definition of terms

3D model – a computer graphical representation of an actual solid object.

BASIC programming language – a family of high-level programming languages. BASIC stands for Beginner’s All-purpose Symbolic Instruction Code.

Command line interface (CLI) – a method of communicating with a computer system with text commands using a keyboard input.

Coordinate frame – an infinite volume in 3D space which has a point of origin upon which locations and orientations are referred to. For example, coordinate frame B may be located at some point and rigidly attached (cannot move relative to) another coordinate frame A, and a point in space may be expressed relative to either coordinate frame B or A.

Denavit-Hartenberg parameters – the parameters specified by the Denavit-Hartenberg convention which uses four values that unambiguously describe the joint and link configuration of a robotic manipulator.

Graphical user interface (GUI) – a method of communicating with a computer system with the help of graphical and visual cues. It is generally more intuitive and easy to use than the CLI. A mouse or a touch screen are examples of input devices for GUIs.

Interactive software – a software program which accepts inputs and returns results immediately. This may add to the software program’s user-friendliness.

Kinematics of rigid body motion – in the context of robotics, the concepts in this subject are used to find the location and orientation of a coordinate frame with respect to the coordinate frame before it or at the origin.

Lead-by-the-nose – a method of training a robotic arm by means of manually taking its end effector tip to a desired location and orientation in space.

MATLAB® – a numerical computing environment and programming language created by The MathWorks, Inc. It is used for high-performance technical computing in the industry and academe.

Microbot TeachMover – an industrial robotic arm designed for educational use.

Offline robot programming – a method of robot programming in which a robot program can be simulated before downloading it to the robot. The robot needs to be connected to the computer only during downloading of the program.

Parallel link robotic manipulator – a robotic manipulator in which two or more series links connect the end-effector to the base, forming a closed chain.

Robotic manipulator – an electromechanical device consisting of joints and links. It can be programmed to do a variety of physical tasks such as moving items, welding, drilling, painting, etc.

Serial interface – the physical connection through which the TeachMover and the PC can communicate.

Serial link robotic manipulator – a robotic manipulator in which all of its links form an open chain.

Simulation – a representation of a certain action or system. It is useful in evaluating a decision before actually implementing it.

Teach pendant – a device connected to the TeachMover which functions similarly to a joystick or game console controller. Even in the absence of a computer, the TeachMover can be controlled and programmed with the use of its teach pendant.

Waypoint – one of a sequence of points along a planned route.

Chapter 2

REVIEW OF RELATED LITERATURE

2.1 Programming a robot

One of the most important facets of human-robot interaction (HRI) will be communication (Brooks, 2007). Industrial robot manipulators are generally used in automation and these robots must acquire a hardware and software mechanism that will enable them to execute certain tasks with high precision and accuracy (Ayco et al., 2007). However, robots do not understand human methods of communication. Different methods are now developed to provide an interface between robots and humans.

There are several ways to communicate with a robot and these are grouped into three major approaches. These are *1)* discrete word recognition; *2)* teach and playback; and *3)* high-level programming languages (Fu et al., 1987). In discrete word recognition, robots are made to follow verbal commands. They are speaker dependent and quite primitive. Speech recognition generally requires a large memory or secondary storage to store speech data, and it usually requires a training period to build up speech templates for recognition.

Teach and playback method is also known as guiding. The method involves teaching the robot by leading it through the motions the user wishes the robot to perform. This is usually accomplished in three steps: *1)* leading the robot to the desired joint angles using manual control, joysticks or teach pendants; *2)* editing or playing back the taught motion; *3)* playing the taught motion in a repetitive mode once the taught motion is correct. Some robots have built in memory, allowing it to store or save certain sequence of movements. These robots are suitable for teach and playback method. The TYR Programmable Robot Arm (Almeda et al., 1992) uses this type of communication. Its three modes of operation, *a)* manual mode, *b)* record mode and *c)* play mode comprise the whole teach and playback process. In manual mode, the user inputs movement commands using a joystick. In record mode, the user's joystick commands will be stored in the TYR's program memory. Finally, in play mode, the TYR moves sequentially according to the recorded program. The TYR Programmable Robot Arm has a built in memory that allows it to store up to two separate sequences of movements which can be performed without manual manipulation. The Microbot TeachMover also uses teach and playback method in programming its motion sequences through its teach pendant.

High-level programming languages are advanced programming languages that simplify a set of commands for a robot. According to Critchlow (1985), high-level programming languages provide much stronger programming capabilities and simplify the task of programming. These languages all have the capability to express in 1 line of a program 2, 5 or even 25 assembly language steps. Examples of these languages is FORTRAN, BASIC, and C++.

Rasouliha, Sproule and Wong (2004) made use of Visual C++ to design a graphical user interface for an Armatron toy robot built by Radio Shack and Tandy in the 1980's. The Armatron, originally controlled by moving two built-in joysticks, was heavily modified so that it can be controlled by a computer through the parallel port.

Piskeric and Bockus (2005) redesigned the serial port control interface of the Microbot TeachMover. They observed that the TeachMover’s control implementation in BASIC was limited and difficult to use. In addressing this problem, a software package, TeachVAL II, was written in Java to extend the TeachMover’s capabilities. It runs under its own environment, where editing, compilation, and execution are all performed in an intuitive graphical user interface (GUI).

2.2 Motion control

Manufacturing industry robots are employed to do specific tasks such as arc-welding, spot welding, drilling, spray painting, and punching holes. These tasks require different methods of controlling the robot’s motion. Theses methods are: *1)* point-to-point, *2)* continuous-path, and *3)* controlled-path (Critchlow, 1989).

Point-to-point control moves the robot from one point in space to another without regard to the path taken between points. It is particularly useful in drilling, punching, pick and place. This type of control teaches the robot a series of discrete points with a joystick or a teach pendant. Balintag, et al. (2007) used the process of point-to-point control based on an image captured by a camera. Using MATLAB’s image processing toolbox, this project designs and implements an algorithm that obtains an image of the workspace and locates predetermined cylindrical objects. A Microbot TeachMover robotic arm, interfaced through the serial port, picks up the objects and places them in a designated area. As a result, the TeachMover is able to pick up objects at random locations within a defined workspace and place them at the assigned location.

Controlled-path control is basically point-to-point control done in small increments (Pessen, 1989). Since the robot is made to follow a roughly-defined path, it serves as an intermediate between point-to-point and continuous-path control.

Continuous-path control defines the entire path of the robot’s end-effector. It can generate straight lines, circles, curves, and other paths with accuracy. Only the path definition and the start and finish coordinates are needed for the control. These are used in arc welding, spray painting and the like. In addition, the velocity of the end-effector can be controlled. These robots require the most sophisticated controllers and software development (Spong et al., 2006).

2.3 Robot arm configurations

Robot arms are built in several types of joint configurations. These are the articulated robot arm, polar coordinate robot arm, cylindrical coordinate robot arm, and Cartesian coordinate robot arm. The joints and movements of each arm create a different work envelope. The number of joint axes is directly related to the maneuverability of a robotic arm (Rasouliha et al., 2004). The articulated robot arm is very similar to the human arm and it is capable of many of the same motions as a human arm. The shoulder of the revolute robot rotates by spinning the arm at its base. The polar coordinate robot arm is very flexible and can grasp different kinds of objects around the robot. The robot rotates by a turntable base and the elbow joint is the second degree of freedom and moves the forearm up and down. The cylindrical coordinate robot arm has the shape of a robotic forklift. The area which this arm works in is the shape of a thick cylinder. The rotation of the shoulder is done by revolving the base like the polar coordinate system. The Cartesian coordinate robot arm consists of a carrier belt like track that makes the arm go back and forth.

2.4 Mathematical modeling of robots

Robots can be represented as mathematical models. Equipped with mathematical models, we will develop methods of planning and controlling robot motions to perform specified tasks (Spong et al., 2006). Robot movement is done by moving its various joints anywhere in space. A robot manipulator is composed of a set of links connected together by various joints. The joints can either be a revolute joint or a prismatic joint (Fu et al., 2007). In moving these joints to obtain the desired end-effector position and orientation, kinematics and dynamics computations are needed. Kinematics is further classified into inverse kinematics and forward kinematics. Corke (1996) defines forward kinematics as the problem of solving the Cartesian position and orientation of the end-effector given knowledge of the kinematic structure and the joint coordinates. He further defined inverse kinematics as the problem of finding the robot joint coordinates, given a homogeneous transform representing the pose of the end-effector. It is very useful when the path is planned in Cartesian space. A commonly used convention for selecting frames of reference in robotic applications is the Denavit-Hartenberg or D-H convention (Spong et al., 2006). The Denavit-Hartenberg convention is used to model the kinematic relationship of the robot links and joints.

2.5 Robot simulations and offline programming

Major industrial robotics companies such as ABB, Fanuc, Panasonic, and Adept provide simulation and offline programming software to accompany their robots. Simulations are an important aspect of robotics such that they visualize abstract mathematical operations and reduce industrial downtime.

Rohrmeirer (1997), uses Virtual Reality Modeling Language (VRML) to visually simulate robot movements. The language enables the integration of interactive 3D graphics into the web. Specialized and expensive hardware or software is not needed. Any web browser with a VRML viewer is able to run the program which makes the application independent from any underlying hardware platform. The method is cost-effective; however, an in-depth knowledge in programming is greatly required.

At the MATLAB Central File Exchange, contributors Andrade and Faria (2007), Kontz (2001) and Riley (2001) have written MATLAB graphical applications which simulate a generic cylindrical manipulator, a generic articulated manipulator and a

PUMA 762 robot arm, respectively. These applications allow the user to input joint angles to invoke the forward kinematics solver. Inverse kinematics simulations are also possible when the user inputs the desired end-effector position and orientation.

2.6 Robotics education

Education plays a great part in enhancing our knowledge in robotics. Linnell (1993) writes, “When teaching robotics to students, a resource base of training programs will be helpful for providing guidance and direction.” Also, visual representations such as 3D models and GUIs can be used to enhance and further optimize the learning process in robotics education.

Corke (1996) introduces a Robotics Toolbox for MATLAB. He emphasized that “The toolbox is useful for simulation as well as analyzing results from experiments with real robots, and can be a powerful tool for education”. The tool was based on general methods of representing the serial-link manipulation of robots particularly by description matrices and Denavit-Hartenberg parameters. The aid of theory, mathematical concepts, and programming provides advantages in understanding of robots such as kinematics, dynamics, and trajectory generation. The toolbox also provides functions for manipulating data types such as vectors, homogeneous transformations and unit-quaternion which are necessary to represent 3-dimensional position and orientation.

Robinette and Manseur (2001) present a more graphical approach to robotics education, especially in D-H modeling. Their work, Robot-Draw, can generate 3D virtual models of robotic manipulators based on D-H parameter tables. Effects of parameter variations are also readily and visually observed. Furthermore, it is an Internet-based software which allows access for remote users.

Analysis and simulation allows interactive understanding between human and robot. This study arises in the context of effort to develop a software package comprised of a graphical user interface, D-H visualization and offline programming of robot arm. In order to achieve the goal of this study, MATLAB software will be used to simulate in 2D/3D space view, to calculate the parameters, to visualize the links and joints of the robot arm, and to interface the Microbot TeachMover to a computer. Mathematically, forward/inverse kinematics and Denavit-Hartenberg convention approaches will be applied to this study.

Chapter 3

METHODOLOGY

3.1 Systems Interfacing

Fundamental communication among humans involves speech, sounds, written words, pictures, and body language; among digital computers, it is a stream of physical signals (e.g. electrical, radio, light signals) representing, at any instant, one of only two words, 0 and 1. Therefore, in order for a computer to be useful to humans, there has to be some form of interfacing between them. An input interface translates the human language into a computer language and an output interface translates in the opposite direction.

An example of an input interface is the keyboard and mouse. A human can use his hands to operate these devices, which automatically translate his actions into electrical signals which the computer can understand. The monitor and printer are examples of output interfaces. The computer sends the appropriate electrical signals to these devices, which then translate these signals into graphics and written words which the human can understand.

One of this study’s main requirements is a simulation and programming interface between a human operator and the Microbot TeachMover. The TeachMover is actually already supplied with an interface called a teach pendant. However, it needs to be online if one wishes to use the teach pendant during operation. The output of this study will allow the user to specify programs and movements for the TeachMover while it is offline. It only needs to be online during program downloading and implementation.

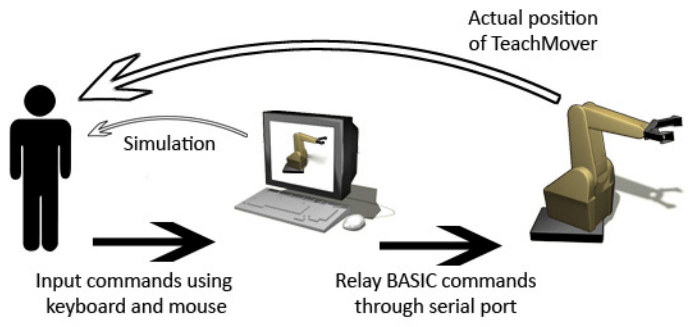

Actual position of TeachMover
Simulation
Input commands using keyboard and mouse
Relay BASIC commands through serial port

Figure 3.1. *An interface between the operator and the TeachMover.*

A symbolic diagram of the system is shown in Figure 3.1. The user enters simple commands into the computer using the keyboard or mouse. In the computer, a MATLAB-based graphical user interface interprets these commands and stores the data for later processing. The data can be used for on-screen three-dimensional simulations of the robotic arm. When the user decides to implement the sequence program, the stored data will then be translated to the TeachMover’s programming language. The translated commands will then be passed to the TeachMover through the serial port.

This method of training a robot is called offline programming. Since everything is simulated in a virtual environment prior to physical implementation, this method affords safety for the operator, and allow him to point out and correct errors which might injure him or damage the equipment (Critchlow, 1985).

Overall, the software translates mouse and keyboard inputs into commands for the TeachMover. It serves as an interface between human and machine.

3.2 MATLAB Graphical User Interface

Since robot arm kinematics involves several matrix manipulations, MATLAB is an optimal environment for this computational task. It can be tasked to perform calculations in response to keyboard text inputs, or a graphical user interface (GUI) can be built to respond to the user's inputs from both the keyboard and mouse.

This study will employ MATLAB's built-in GUI programming capabilities. Figure 3.2 shows how data is passed between main software modules.

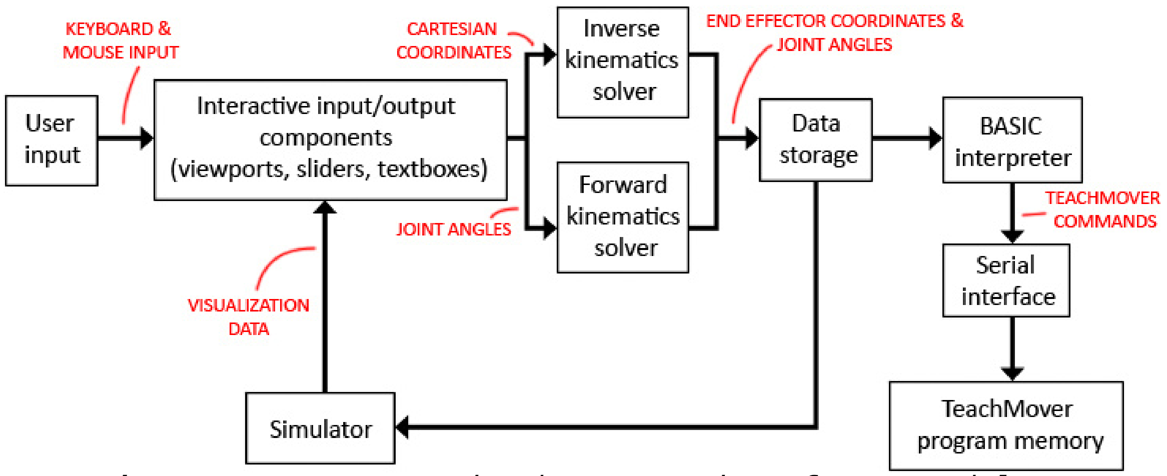

KEYBOARD & MOUSE INPUT
User input
Interactive input/output components (viewports, sliders, textboxes)
CARTESIAN COORDINATES
JOINT ANGLES
Inverse kinematics solver
Forward kinematics solver
END EFFECTOR COORDINATES & JOINT ANGLES
Data storage
BASIC interpreter
TEACHMOVER COMMANDS
Serial interface
TeachMover program memory
VISUALIZATION DATA
Simulator

Figure 3.2. *Data passing between main software modules.*

A GUI is composed of several components into which specific responses are programmed. These responses are called *callback functions* in MATLAB. When the user initiates an *event*, such as clicking inside a viewport, its callback function will trigger, and MATLAB will perform the tasks defined in the function. In the

GUI, the user can specify the desired x, y, and z coordinates, and pitch and roll angles by clicking in the corresponding viewports, manipulating the sliders, or typing in values in the text boxes. An initial sketch of the simulator is shown in Figure 3.3.

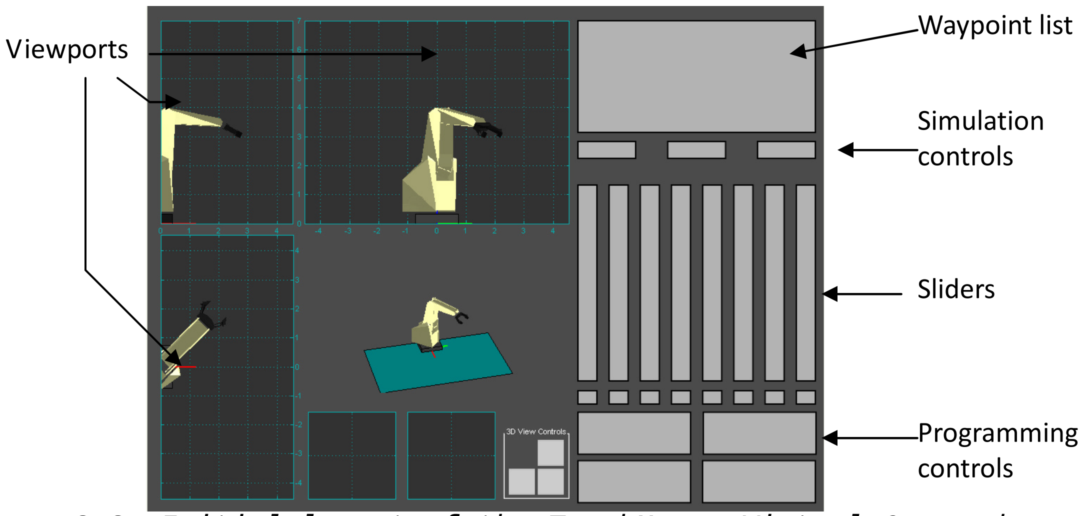


Figure 3.3. *Initial layout of the TeachMover Virtual Command.*

The callbacks associated with these components will automatically store the data, pass the data to the kinematics solvers, and update the 3D model to simulate the new position to be acquired by the robot. In addition, individual joint angles can be fine-tuned using the sliders and text boxes provided. Input components are programmed to be interactive. This means that an input from one component may affect the values of the other components. Thus, these components become a single channel for both input and output information.

The GUI will have components that allow the user to save, view, and simulate sequence programs. The user will then be able to finalize the sequence programs and download them to the TeachMover for implementation.

3.3 Denavit-Hartenberg Visualization

The visualization software for Denavit-Hartenberg (D-H) robotic link modeling reuses some of the program code of the TeachMover simulator. This software tool will have a three-dimensional viewport wherein coordinate frames are represented. The user will have the ability to specify the number of series-linked coordinate frames and configure these frames with parameters according to the D-H convention. These parameters are link offset d, joint angle θ, link length a, and link twist α(Spong et al., 2006). As the user inputs values into the text boxes and manipulates the sliders corresponding to the parameters, the 3D representation in the viewport changes accordingly to reflect the new D-H parameters. The user will also have the option to toggle the coordinate frame models with generic jointed links to help visualize a virtual customized robotic arm.

Chapter 4

RESULTS AND DISCUSSION

4.1 Denavit-Hartenberg Parameters of the Microbot TeachMover

The Denavit-Hartenberg (D-H) convention describes the spatial relationships between two coordinate frames. For purposes of this discussion the first coordinate frame is called the *parent* and the next coordinate frame is the called the *child*. It is important to note that the *child's* motion is based on the *parent's* z-axis.

Before deriving the TeachMover's D-H, parameters, a home position must be defined. This is the position where all joint angles θ are set to $\theta = 0°$. In this study, the home position is defined as when the TeachMover is extended straight out to its front. After defining the home position, a permanent inertial reference frame is defined at its base. This will be called O_0. Figure 4.1 shows the five coordinate frames in the process of deriving their D-H parameters.

Some constant values must first be established. These refer to permanent dimensions of the TeachMover. These are $H = 195.0\text{ mm}$, $L_1 = 177.8\text{ mm}$, and $L_2 = 96.5\text{ mm}$.

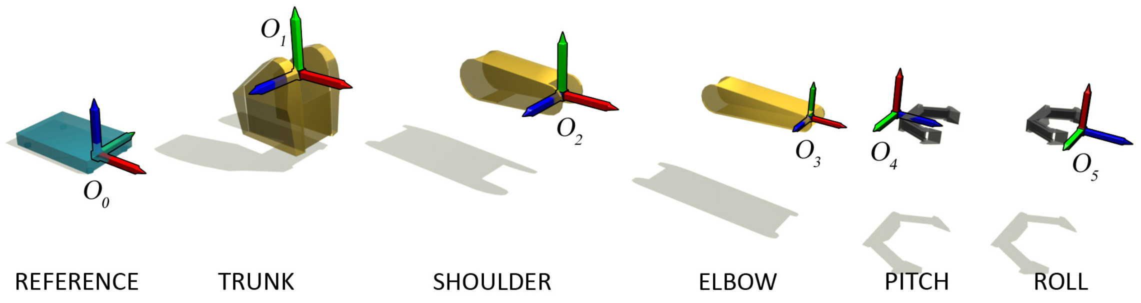


Figure 4.1. *Individual coordinate frames of the Microbot TeachMover.*

The next coordinate frame O_1 is located at distance H from the reference frame along z_0. It is rotated $90°$ about x_1 such that z_1 points in the direction of motion of its child coordinate frame O_2. The distance H along z_0 is called the link offset d.

The same is done with the next coordinate frames O_2 and O_3 but this time, they are each located at distance L_1 from O_1 and O_2 respectively. We have to remember that O_2 is moved along x_2 and O_3 is moved along x_3. The distance L_1 along x_2 and x_3 is called the link offset a.

The next coordinate frame O_4 coincides in position with O_3 but is rotated $90°$ joint angle about z_3 and $90°$ link twist about x_4 such that z_4 points in the direction of motion of the last coordinate frame O_5. Since we are setting all joint angles θ to $\theta = 0°$, θ_4 should be added with $90°$. The last coordinate frame O_5 is located L_2 distance from O_4 along z_4. The end-effector is located at O_5. Table 4.1 shows the final D-H parameters.

Table 4.1. *Denavit-Hartenberg parameters of the Microbot TeachMover.*

Joint Name	**Joint Number**	**Link Offset** d_i	**Joint Angle** θ_i	**Link Length** a_i	**Link Twist** α_i
Trunk	1	H	θ_1	0	$90°$
Shoulder	2	0	θ_2	L_1	$0°$
Elbow	3	0	θ_3	L_1	$0°$
Pitch	4	0	$\theta_4 + 90°$	0	$90°$
Roll	5	L_2	θ_5	0	$0°$

Figure 4.2 shows the TeachMover as it is fully assembled after derivation of D-H parameters. It also shows the axes of motion for each coordinate frame.

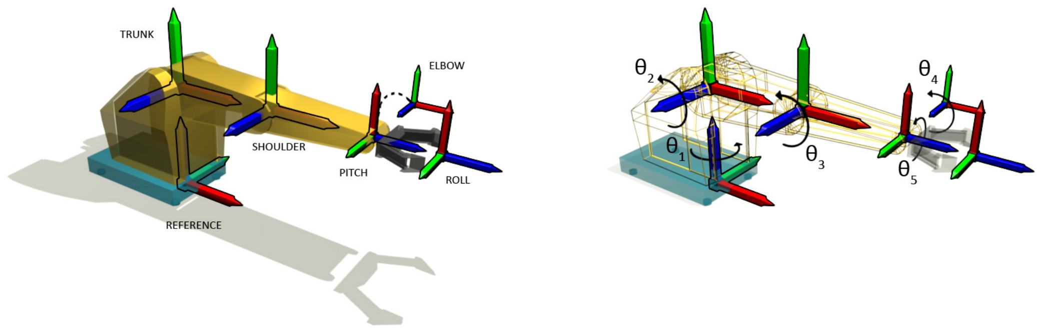


Figure 4.2. *The Microbot TeachMover with its joints and links fully assembled and axes of motion for each coordinate frame.*

4.2 Kinematics Solutions for the Microbot TeachMover

For the Microbot TeachMover, which consists entirely of revolute joints, the forward kinematics solution obtains the position and orientation of the end-effector given the individual joint angles. On the other hand, the inverse kinematics solution obtains the individual joint angles required in order to place the end-effector into the desired position and orientation.

i) Forward Kinematics

Using the Denavit-Hartenberg convention in (Spong et al, 2006) and (Fu et al., 1987), the forward kinematics problem is tackled using 4x4 homogeneous transform matrices. A transform matrix contains all the information pertaining to a coordinate frame’s position and orientation with respect to another coordinate frame. From this point on, the term “transform matrix” will be used to describe a coordinate frame’s position and orientation.

From the TeachMover's D-H parameters as presented in Table 4.1, the end-effector's transform matrix with respect to the reference frame, $\mathbf{T}_5^0$, can be obtained using matrix multiplication.

$$\mathbf{T}_5^0 = \mathbf{T}_1^0\mathbf{T}_2^1\mathbf{T}_3^2\mathbf{T}_4^3\mathbf{T}_5^4$$

$$\mathbf{T}_5^0 = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & H \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} c_2 & -s_2 & 0 & L_1c_2 \\ s_2 & c_2 & 0 & L_1s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} c_3 & -s_3 & 0 & L_1c_3 \\ s_3 & c_3 & 0 & L_1s_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\cdots \tag{4.1}$$

$$\begin{bmatrix} -s_4 & 0 & c_4 & 0 \\ c_4 & 0 & s_4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ s_5 & c_5 & 0 & 0 \\ 0 & 0 & 1 & L_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The trigonometric expressions $\sin\theta_m$ and $\cos\theta_m$ are abbreviated to s_m and c_m, respectively, where m indicates which coordinate frame is rotated by an angle θ. The resulting transform matrix $\mathbf{T}_5^0$ is interpreted in the following manner:

$$\mathbf{T}_5^0 = \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

where the upper-left 3x3 submatrix represents the orientation of the end-effector, and the upper-right 3x1 vector represents its position.

The upper-left 3x3 submatrix consists of three vectors, $\mathbf{n}$ (normal vector), $\mathbf{s}$ (sliding vector), and $\mathbf{a}$ (approach vector). These vectors respectively correspond to the x-, y-, and z-axes of the end-effector coordinate frame. Figure 4.3 shows how the transform matrix describes a coordinate frame in three-dimensional space.

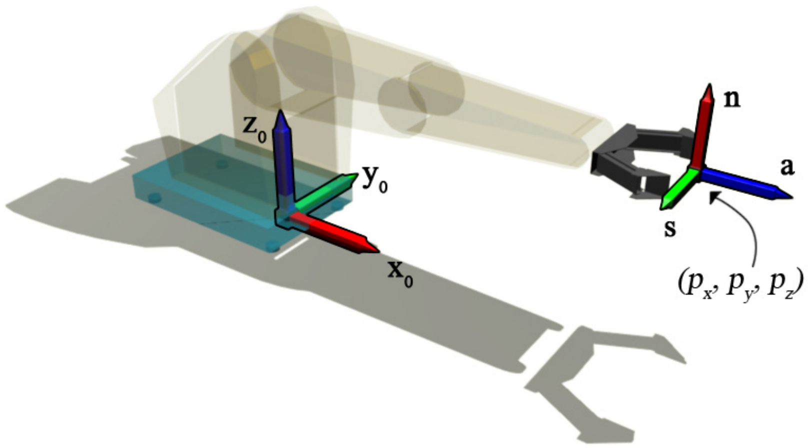


Figure 4.3. *The end-effector as described by Eq. 4.2.*

After matrix multiplication, $\mathbf{T}_5^0$ is found to be

$$\mathbf{T}_5^0 = \begin{bmatrix} -c_1 s_{234} c_5 & c_1 s_{234} s_5 + s_1 c_5 & c_1 c_{234} & L_2 c_1 c_{234} + L_1 c_1 \left(c_{23} + c_2 \right) \\ -s_1 s_{234} c_5 - c_1 s_5 & s_1 s_{234} s_5 - c_1 c_5 & s_1 c_{234} & L_2 s_1 c_{234} + L_1 s_1 \left(c_{23} + c_2 \right) \\ c_{234} c_5 & -c_{234} s_5 & s_{234} & L_2 s_{234} + L_1 \left(s_{23} + s_2 \right) + H \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.3)$$

where s_{234} and c_{234} are abbreviated expressions of $\sin\left(\theta_2 + \theta_3 + \theta_4\right)$ and $\cos\left(\theta_2 + \theta_3 + \theta_4\right)$, respectively.

The MATLAB code that implements this solution in Virtual Command is shown in lines 12-67 of FUNCTIONS_solvers.m.

ii) Inverse Kinematics

In actual robot applications, it is more common to specify the position and orientation of the end-effector. To accomplish this, a robot controller will need to know the necessary joint angles. In an effort to simplify the solution, the geometric approach is used. The position is defined by (x, y, z) and the orientation is determined by pitch angle P with respect to ground and roll angle R. At the end of the solution, roll angle

θ_5 is automatically equal to R because it doesn't affect the position of the end-effector.

The first step of the inverse kinematics solution is to determine the trunk angle θ_1 and radius vector RR from the base to the end-effector as shown in Figure 4.4.

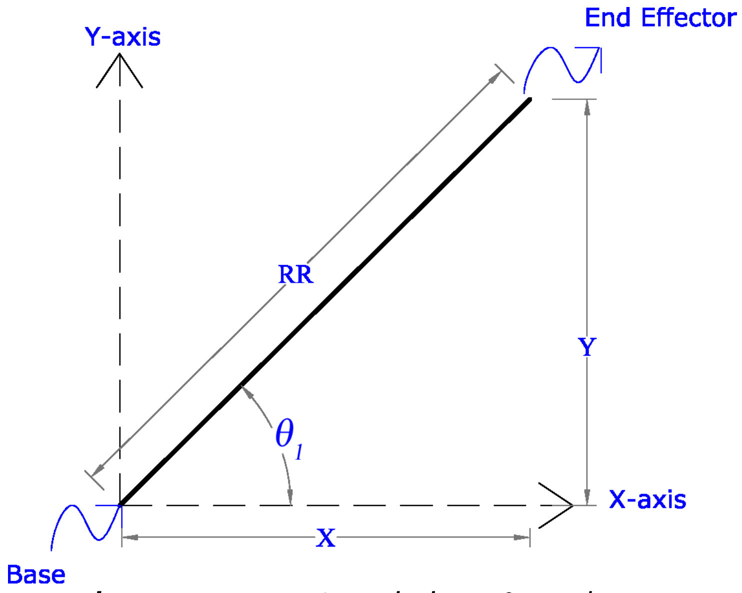


Figure 4.4. *Determining θ_1 and RR.*

$$RR = \sqrt{x^2 + y^2} \tag{4.4}$$

$$\theta_1 = \operatorname{arctan2}(y, x) \tag{4.5}$$

The function arctan2 is the two-argument arctangent function. It accounts for the signs of its two arguments to be able to specify angles in all four quadrants of the Cartesian plane.

The second step is to determine the distance R_o from the shoulder to the wrist and the height Z_o of the wrist above the shoulder. To do this, first determine the Cartesian coordinates from wrist to end-effector given the pitch angle P with respect to ground,

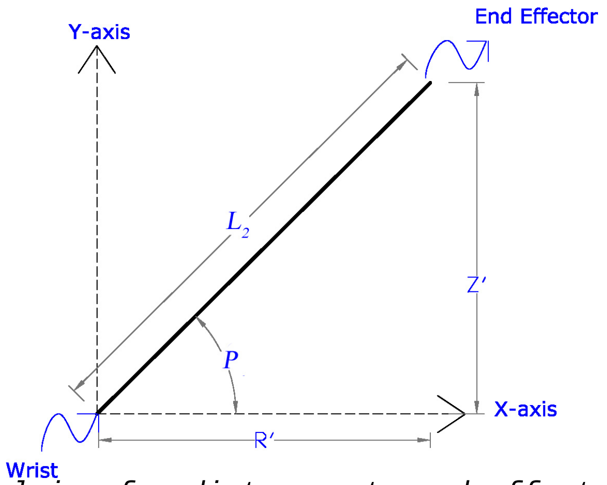


Figure 4.5. *Solving for distances to end-effector from wrist.*

$$Z' = L_2 \sin P \tag{4.6}$$

$$R' = L_2 \cos P \tag{4.7}$$

Then determine the Cartesian coordinates from base to the end-effector.

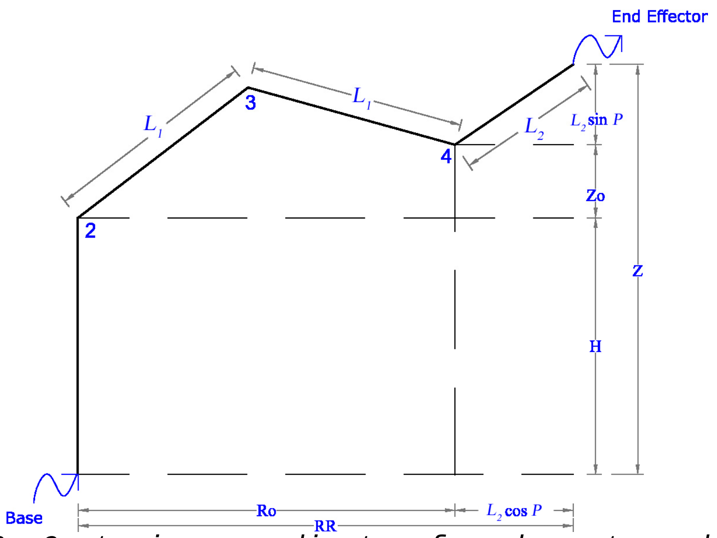


Figure 4.6. *Cartesian coordinates from base to end-effector.*

$$Z_o = z - L_2 \sin P - H \tag{4.8}$$

$$R_o = RR - L_2 \cos P \tag{4.9}$$

The third step in the inverse kinematics solution is to solve the shoulder-elbow-wrist triangle for θ_2 and θ_3. Two new angles α and β and also distances B and b are introduced to facilitate the solution.

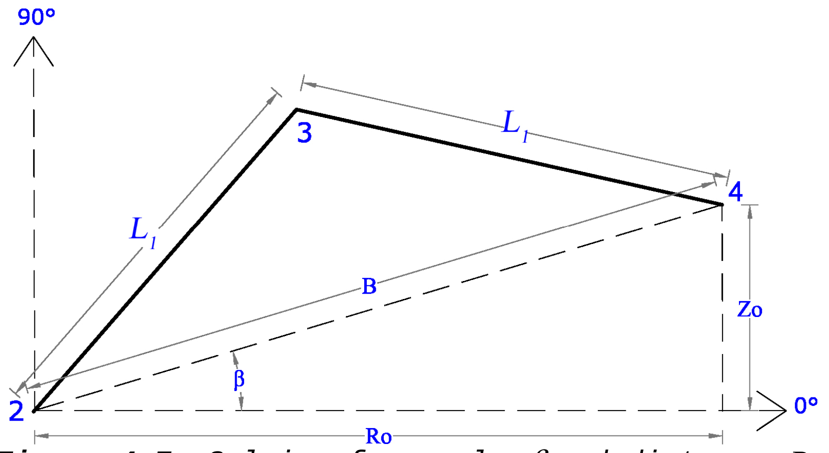


Figure 4.7. *Solving for angle β and distance B.*

$$B = \sqrt{R_o^2 + Z_o^2} \tag{4.10}$$

$$\beta = \arctan2(Z_o, R_o) \tag{4.11}$$

The simplified triangle can be redrawn into an isosceles triangle since it has two sides of length L_1.

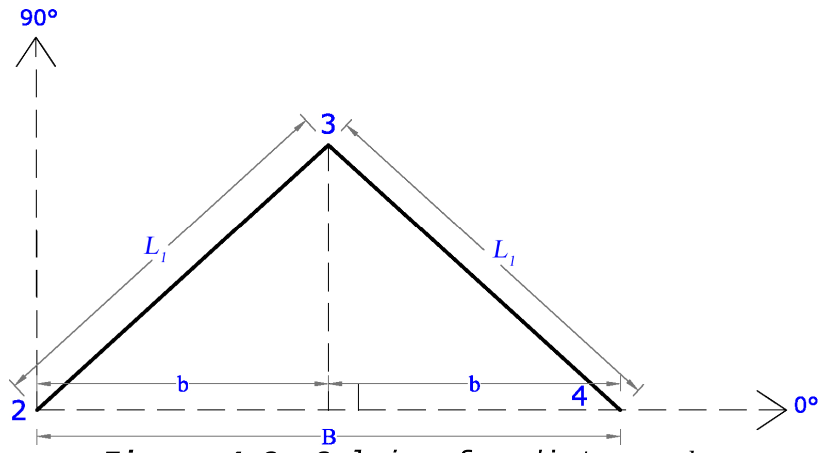


Figure 4.8. *Solving for distance b.*

$$b = \frac{B}{2} = \frac{\sqrt{R_o^2 + Z_o^2}}{2} \tag{4.12}$$

Angle α can then be solved as shown in Figure 4.9.

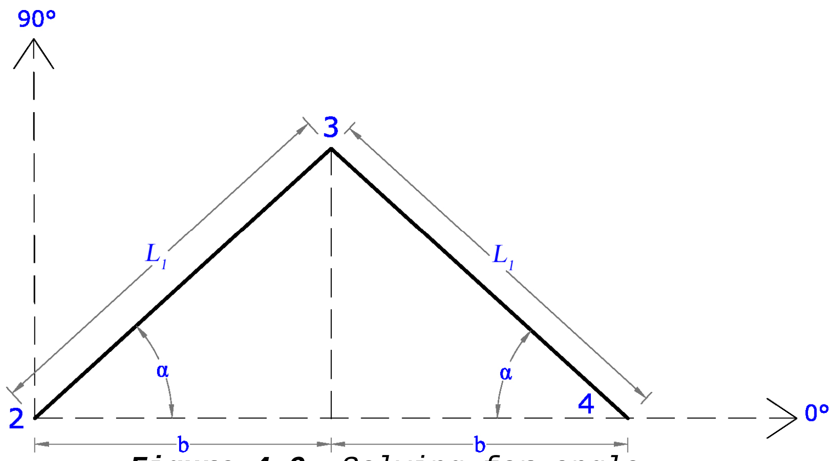


Figure 4.9. *Solving for angle* α.

$$\alpha = \cos^{-1}\left(\frac{b}{L_1}\right) \tag{4.13}$$

The shoulder joint angle θ_2 is the sum of α and β.

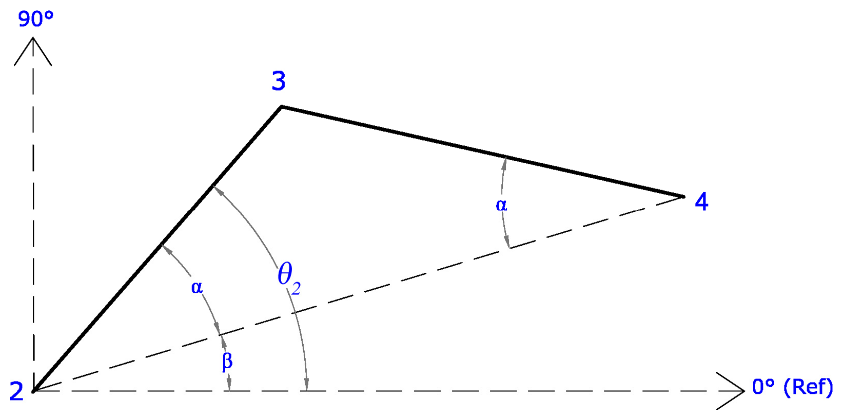


Figure 4.10. *Solving for* θ_2.

$$\theta_2 = \alpha + \beta \tag{4.14}$$

The elbow angle θ_3 can be solved using equivalent angles when two parallel lines are intersected by a line.

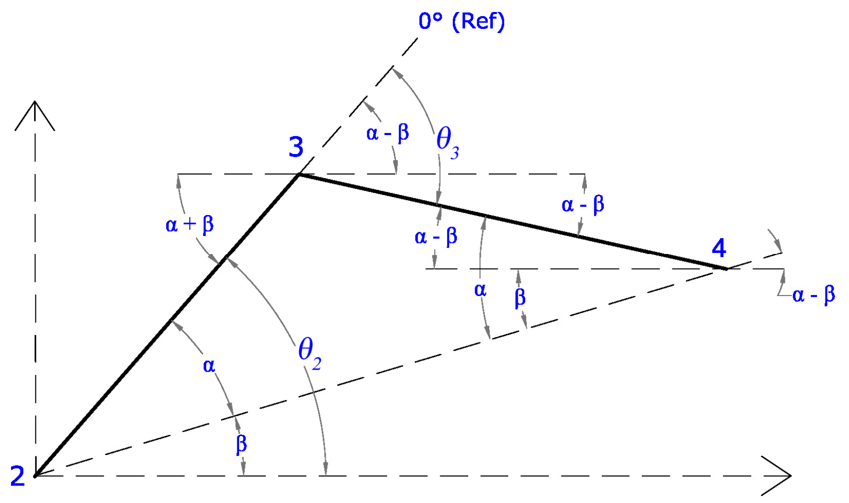


Figure 4.11. *Solving for* θ_3.

$$-\theta_3 = \alpha + \beta + \alpha - \beta$$
$$\theta_3 = -2\alpha \tag{4.15}$$

The value of θ_3 is negative because it is below the ground reference plane.

The last step is to solve for the pitch angle θ_4 with respect to the elbow coordinate frame.

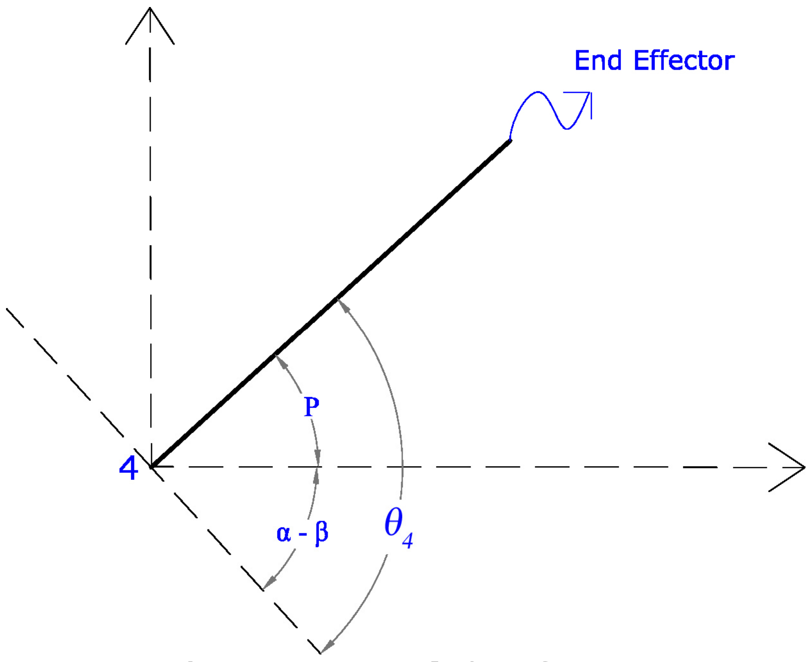


Figure 4.12. *Solving for* θ_4.

$$\theta_4 = P + \alpha - \beta \tag{4.16}$$

The inverse kinematics solution is summarized below, where the joint angles to be fed to the TeachMover and Virtual Command are emphasized. Given end-effector position (x, y, z), desired pitch angle P with respect to ground and roll angle R,

$$\begin{aligned}
\theta_1 &= \operatorname{arctan2}(y, x) \\
RR &= \sqrt{x^2 + y^2} \\
Z_o &= z - L_2 \sin P - H \\
R_o &= RR - L_2 \cos P \\
B &= \sqrt{R_o{}^2 + Z_o{}^2} \\
\beta &= \operatorname{arctan2}(Z_o, R_o) \\
b &= \frac{B}{2} \\
\alpha &= \cos^{-1}\left(\frac{b}{L}\right) \\
\theta_2 &= \alpha + \beta \\
\theta_3 &= -2\alpha \\
\theta_4 &= P + \alpha - \beta \\
\theta_5 &= R
\end{aligned}$$

The MATLAB code that implements the inverse kinematics solution is found in lines 114 to 143 of FUNCTIONS_solvers.m.

4.3 Building the Graphical User Interfaces using MATLAB

i) Program flow

Figure 4.13 shows the basic operational flow of TeachMover Virtual Command and D-H Assist. Before the user can manipulate data, the user interface must first be drawn on-screen. This user interface will contain MATLAB graphics objects and user interface controls such as buttons and text boxes. These GUI components are associated with *callback functions* which execute when they are manipulated. After building the GUI, the program is ready to be used.

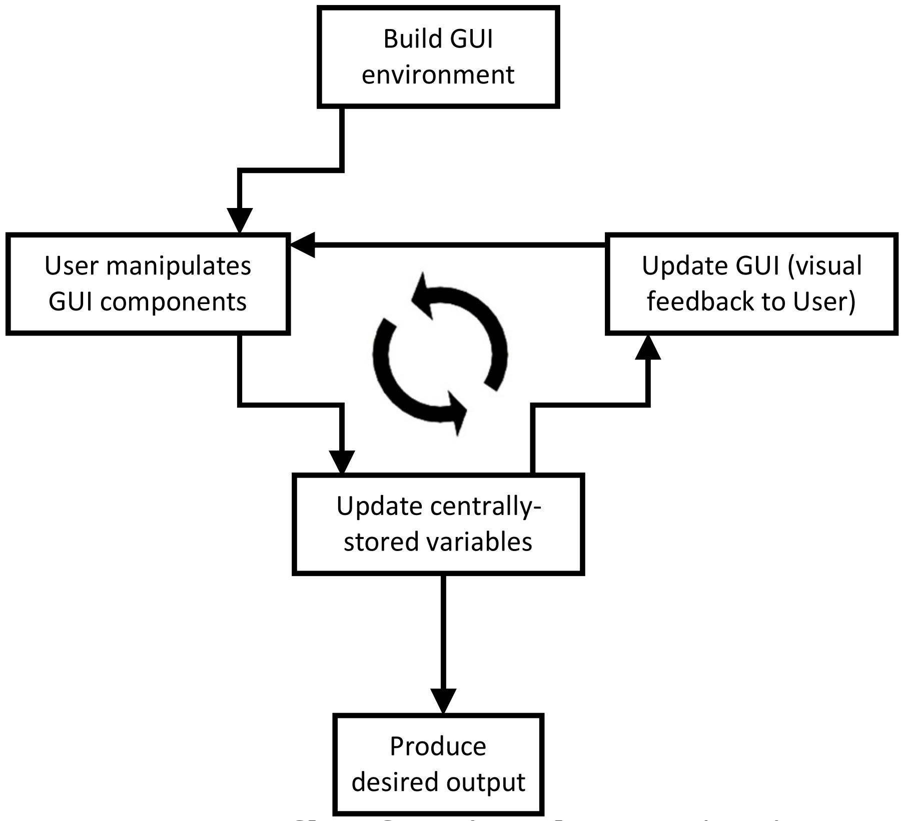

Build GUI environment
User manipulates GUI components
Update GUI (visual feedback to User)
Update centrally-stored variables
Produce desired output

Figure 4.13. *Program flow for Virtual Command and D-H Assist.*

The user then manipulates the GUI components, and depending on the callbacks that execute, centralized variables associated with the GUI are changed either directly or indirectly. Centrally-stored data is accessed using the built-in setappdata and getappdata functions.

For example, in Virtual Command, the user clicks in the viewports to specify the desired end-effector position. This directly changes the variable associated with that data. However, the program will still have to calculate the individual joint angles to simulate the inverse kinematics solution. This indirectly changes the variable associated with joint angles.

When the centralized data are properly updated, the program then updates the GUI components to provide visual feedback for the user. Continuing from the previous example, the 3D model of the TeachMover shown on screen will then move according to the joint angle data taken from centralized storage.

When the user is satisfied with the simulations, he may now output motion commands to the TeachMover. The procedures used by Virtual Command and D-H Assist to perform these actions will be explained in detail in their respective sections.

ii) MATLAB® Handle Graphics®

MATLAB provides a versatile set of graphical tools that augment learning of mathematical concepts. It is called Handle Graphics®. The Handle Graphics objects used in this study are figure, axes, hgtransform, line, patch, uipanel, and uicontrol. They are organized in the hierarchy shown in Figure 4.14.

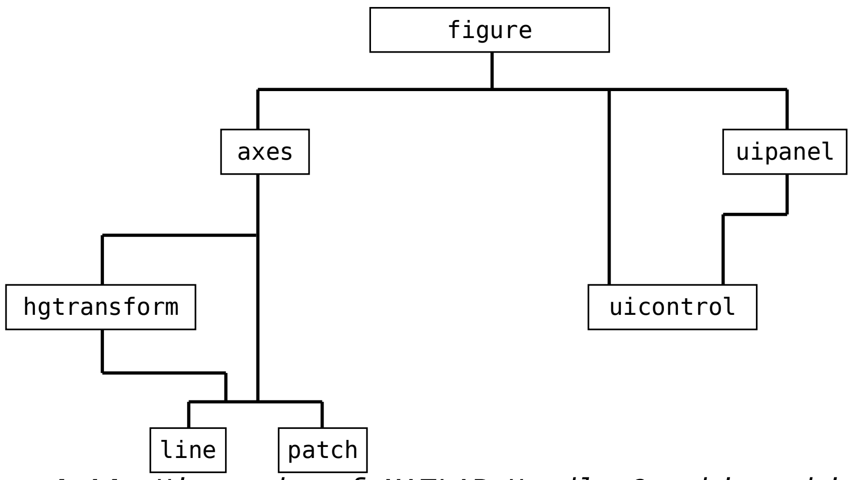


Figure 4.14. *Hierarchy of MATLAB Handle Graphics objects.*

The hierarchy shows parent-child relationships between objects of different levels. A direct connection between objects at different levels means that the upper object can be a parent of the bottom object.

Handle Graphics objects have special properties that define their position, size, color, coordinates, callback functions, and other appropriate attributes. These properties can be retrieved and modified using MATLAB's built-in get and set functions. Only the properties which are particularly useful in achieving the objectives of this study are explained in detail. The official MATLAB documentation provides a thorough treatment of this topic.

The figure object is the heart of all MATLAB graphical user interfaces. It creates a new window which can contain all other Handle Graphics objects, except other figures. It has a Position property which defines its location and size on screen. It also has a Name property which can set the text string shown on its window title bar.

The axes object creates a three-dimensional Cartesian-coordinate space where lines, patches, and hgtransforms can be placed. An important property of the axes object is DataAspectRatio. Setting this property to [1 1 1] ensures that lines and patches drawn inside the axes are proportional in all three dimensions. An axes object's Projection mode can be set to perspective or orthographic. Its View property is a two-element vector defining the azimuth and elevation of the camera displaying the axes. For example, setting Projection to orthographic and View to [0 90] displays the axes object's x- and y-axes as if it were just a two-dimensional Cartesian plane.

A line object draws invisible vertices in the axes. By default, it automatically draws solid line segments between these vertices. However, it can be set not to draw line segments at all by setting its LineStyle property to none. Its Marker property can be set to draw markers at each vertex. Vertex coordinate data is stored in the XData, YData, and ZData properties. For example, a three-vertex line has [x1 x2 x3] for XData, [y1 y2 y3] for YData, and [z1 z2 z3] for ZData, where the first vertex is located at (x_1, y_1, z_1) and so on and so forth. To designate a single point in space, use a line object with only one vertex, then set a Marker to make it visible.

A patch object enables 3D solids to be drawn in the axes. Its Vertices property is an mx3 matrix where m is the number of vertices in the patch object. Each column corresponds to the x-, y-, and z-coordinate of each vertex. Its Faces property is an nx3 matrix where n is the number of faces making up the patch object. In each row are three numbers that specify which vertices connect to form a triangular face. Putting all these triangular faces together produces a 3D solid patch object.

An hgtransform is an invisible object that can group together lines, patches and other hgtransforms. This enables the grouped objects to be moved together in unison just by translating or rotating the hgtransform. In formal mathematical theory, an hgtransform is a coordinate frame in space. An important property of this object is the Matrix property, a 4x4 homogeneous transform matrix. Together with the built-in makehgtform function, this property allows translation and rotation of the hgtransform object. Figure 4.15 illustrates usage of the hgtransform object. For the purpose of this discussion, the invisible hgtransform object is represented by a coordinate frame.

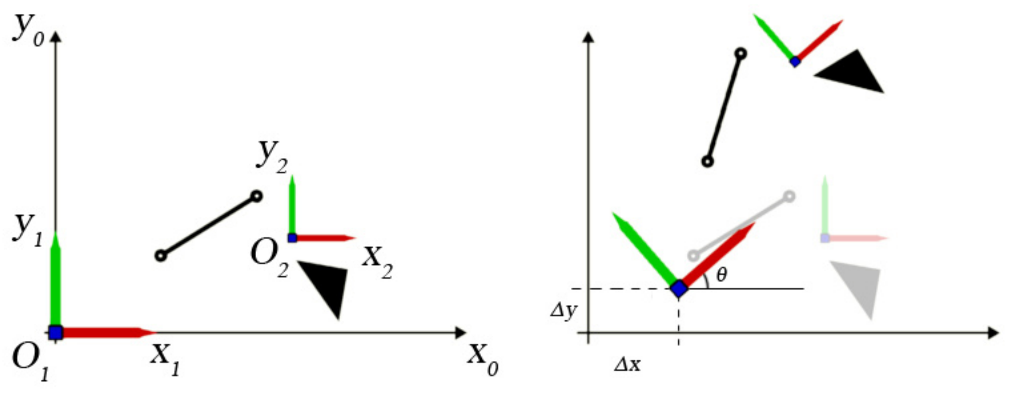


Figure 4.15. *Usage of the hgtransform object.*

In Figure 4.15(a), hgtransform O_1 is coincident with the origin. Therefore, its transform matrix is

$$\mathbf{T}_1^0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.17}$$

which is an identity homogeneous transform matrix, signifying that it is coincident with its Parent, in this case the axes object, both in position and orientation. Performing the following get operation on O_1's Matrix property will store the same 4x4 matrix into T:

```
T = get(O1,'Matrix');
```

Three other objects are *parented* to O_1: a line object, a triangular patch object, and another hgtransform O_2. This means that the Parent properties of these *child* objects are pointed to hgtransform O_1.

In Figure 4.15(b), O_1 has been translated by $(\Delta x, \Delta y)$ from the origin and rotated by θ about its own z-axis, z_1, which is pointing out from the page. In order to perform this operation,

O_1's transform matrix will have to be manipulated accordingly and stored into matrix $\mathbf{A}$:

$$\mathbf{A} = \mathbf{T}_{x,\Delta x}\mathbf{T}_{y,\Delta y}\mathbf{T}_1^0\mathbf{R}_{z,\theta}$$

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ \sin\theta & -\cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.18)$$

$$\mathbf{A} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & \Delta x \\ \sin\theta & -\cos\theta & 0 & \Delta y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To create the same transformation matrix $\mathbf{A}$ in MATLAB, the following code is executed:

```
A = makehgtform('translate',[delta_x, delta_y, 0])*...
    makehgtform('zrotate',theta);
```

where theta must be expressed in radians. Finally, to move O_1 as desired, a set operation is performed:

```
set(O1,'Matrix',A);
```

It is important to note that O_1's three child objects have moved with it, as though they are anchored to it. These child objects will treat the parent hgtransform as their point of origin $(0,0,0)$.

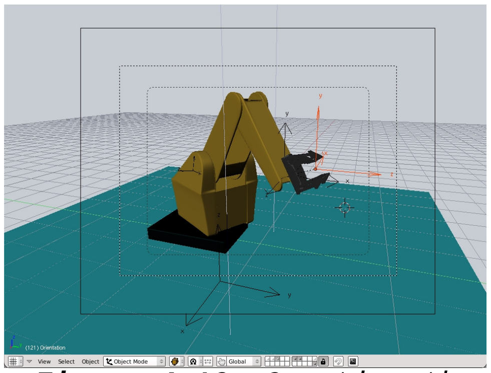
To draw built-in user interface controls such as pushbuttons and textboxes, the uicontrol object is used. Its Style property can be set to pushbutton, radiobutton, text (for static text), edit (for editable textbox), checkbox, or listbox. Furthermore, a uipanel object can be used to visually group together other uicontrol objects.

Since MATLAB doesn't have the built-in capability to create columnar listboxes, a MATLAB program was obtained from the File Exchange to perform this task. The columnar listbox was particularly useful in creating the waypoint list in Virtual Command.

iii) Creating and importing 3D models of the TeachMover and coordinate frames

As discussed previously in this section, a patch object is created by defining its Vertices and Faces. Doing this manually is a tedious task. Using freely downloadable tools from the Internet, creating and importing the 3D models became an easier task. These tools are Blender, IVCON, and cad2matdemo.m.

Blender, an open-source 3D modeling tool, was used in creating the 3D models. Figure 4.16 shows screenshots of this process.



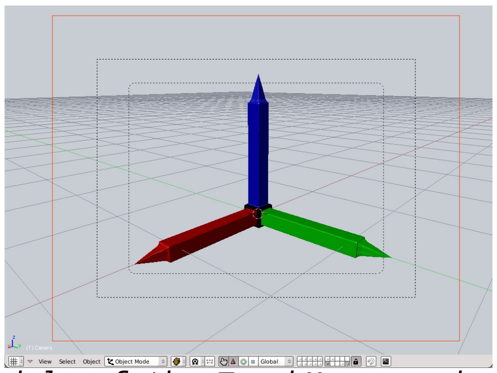


Figure 4.16. *Creating the 3D models of the TeachMover and coordinate frame.*

Blender is capable of exporting 3D model data as binary Stereolithography (STL) files. This is an industry-standard 3D graphics file format which specifies vertices and faces in a way that is similar to that of a patch object. In the case of the TeachMover, each joint is exported in its own STL file.

Since cad2matdemo.m requires ASCII STL files to work, Blender's output must first be converted to ASCII format using IVCON, a free command-line software for graphics format conversion. Available in the MATLAB File Exchange, cad2matdemo.m can now read the ASCII STL file and create a patch object that is usable in the MATLAB environment.

4.4 D-H Assist

i) Operation

D-H Assist software is developed separately for the visualization of the D-H parameters of a robot manipulator. It will aid on the derivation of the D-H parameters using a Graphical User Interface. The software interface as shown in Figure 4.17 consists of GUI objects such as axes, buttons and sliders.

To start deriving D-H parameters using D-H Assist, the user needs to go into Build Mode. Before specifying the D-H parameters, the user must first define the desired number of total frames. The Total Frames button generates a number of frames for the robot manipulator. The software is capable of up to seven total frames.

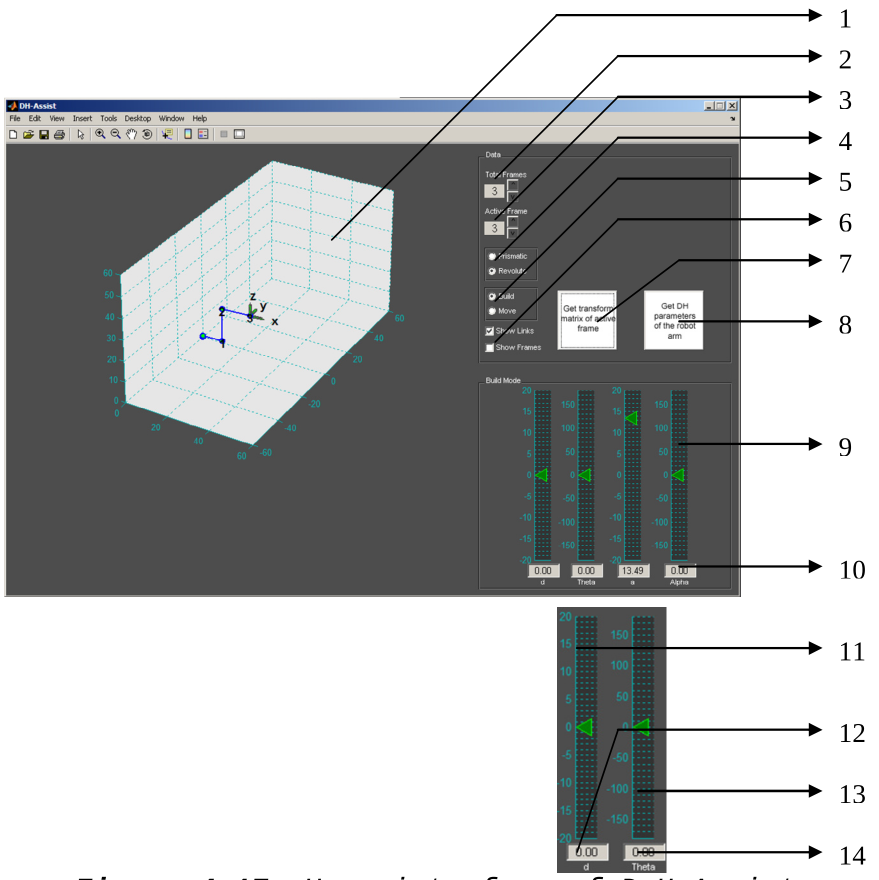


Figure 4.17. User interface of D-H Assist.

Table 4.2. User interface components of D-H Assist.

Number	Name of UI Component	Remarks
1	Viewport	
2	Total Frames	
3	Active Frames	
4	Prismatic or Revolute Selection	Only available in Build Mode
5	Build or Move Mode Selection	
6	Show Links and Show Frames Selection	
7	Get Transform Matrix of Active Frame	
8	Get D-H Parameters of Robot Arm	
9	Sliders for Build Mode	Only available in Build Mode
10	Textboxes for Build Mode	
11	Slider for Move Mode	Only available in Move Mode and when frame is Prismatic
12	Textbox for Move Mode	
13	Slider for Move Mode	Only available in Move Mode and when frame is Revolute
14	Textbox for Move Mode	

After setting the number of total frames, the user can now modify each frame by selecting the desired frame number from the Active Frame button. In the Viewport, green colored frame designates the selected active frame. The active frame can now be set into either Prismatic or Revolute joint. After setting the joint into either Prismatic or Revolute, the user can now modify its D-H parameter using the d, θ, a, and α sliders.

The d (link offset) slider allows the movement of the selected active frame along the z-axis of its parent frame. The θ (joint angle) slider allows the rotation of the selected active frame about the z-axis of the parent frame. The a (link length) slider allows the movement of the selected frame along its own x-axis. The α (link twist) slider allows the rotation of the selected active frame about its own x-axis. With all parameters set in the current active frame, the user may proceed in modifying the other frames by following the same procedures stated above.

After specifying the D-H parameters for each coordinate frame, the user can now test whether the newly-modelled robotic arm can mover appropriately. The user may now switch to Move Mode. In this mode, the Total Frames and the Prismatic/Revolute buttons are disabled. The user is only allowed to select the active frame. If the current active frame was set to Prismatic while in Build Mode, the only D-H parameter that can be modified is the d (link offset) slider. On the other hand, if the active frame selected was set to Revolute in Build mode, the only D-H parameter that can be modified is the θ (joint angle) slider.

When the user is satisfied in the way the robot arm was modelled, he may now access the D-H parameters using the “Get D-H parameters of the robot arm” button. It will show the final D-H parameters of the robot arm as an mx4 matrix, where each row corresponds to each coordinate frame and each column corresponds to d, θ, a, and α, respectively. To help validate manual matrix computations, the “Get transform matrix of active frame” button will show the 4x4 homogeneous transform matrix of the active frame with respect to its parent frame.

ii) Graphics issues

A problem was encountered when the researchers set the limit on the total number of frames. D-H Assist is drawn on screen using MATLAB’s Painters renderer. This is found in the Renderer property of the figure object. This renderer runs slow when there are too many 3D patch objects drawn on screen. A better alternative might have been to use the OpenGL renderer, which can draw 3D objects faster and more efficiently. However, the OpenGL renderer has problems with drawing uipanels. The components inside the uipanels disappear, making them unusable. Therefore, the researchers were compelled to use the Painters renderer and limited the total number of frames to seven, where screen redraw rates were still at a reasonable level.

4.5 Microbot TeachMover Virtual Command

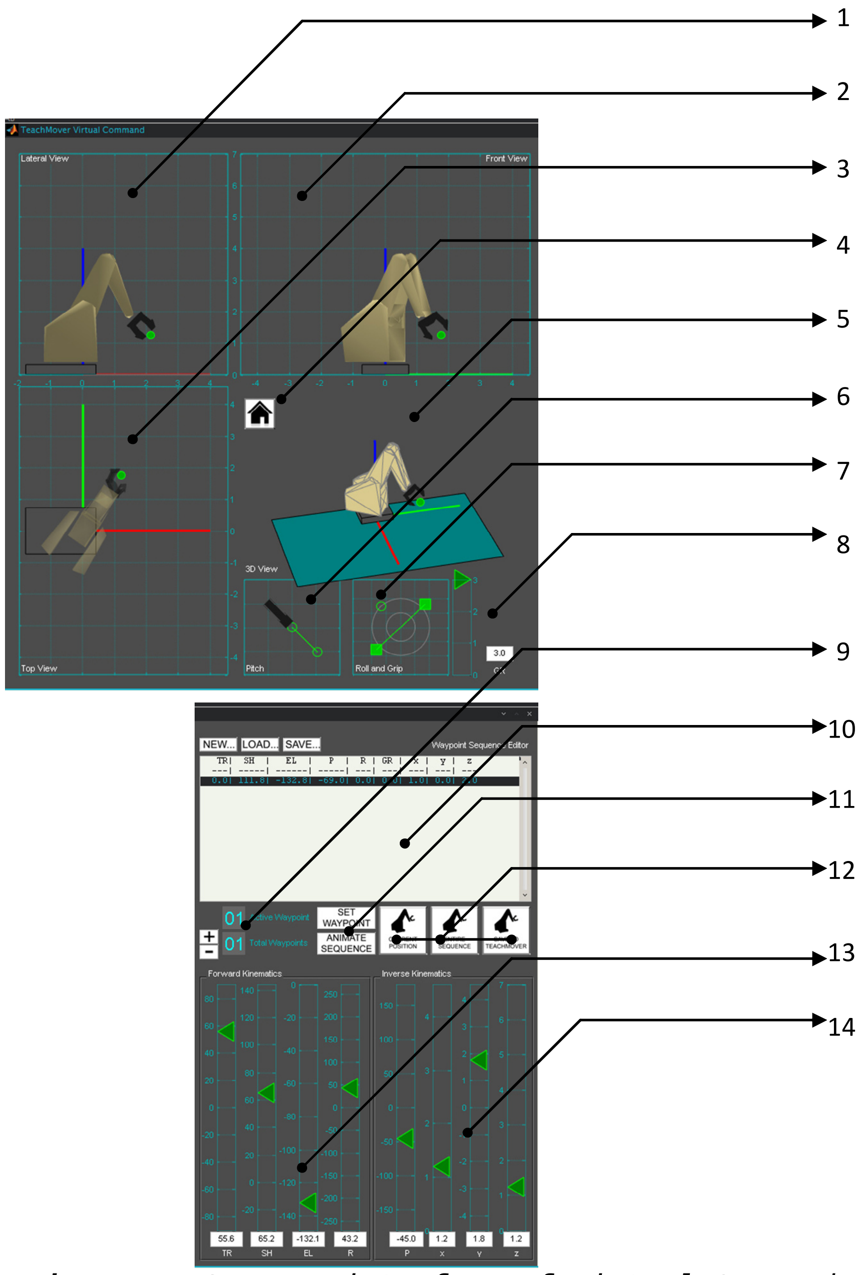
i) Operation

TeachMover Virtual Command was developed to simplify operation and programming of the Microbot TeachMover. To ensure that Virtual Command will run properly, the folder '\Listbox Column' must be added to the MATLAB path. Before using Virtual Command with the TeachMover, the robot arm must first be calibrated. Using the teach pendant, all joint angles must be configured such that the TeachMover is pointing straight out to its front as shown in Figure 4.3, except with its gripper closed. On the teach pendant, MODE+CLEAR and MODE+ZERO must be pressed to empty the TeachMover's program memory and reset step counters to zero.

Figure 4.18 shows the user interface of Virtual Command. To simulate control of the TeachMover, the user can click and drag inside the Lateral, Front, and Top viewports to specify end-effector position. Clicking and dragging the gripper's green handle in the Pitch viewport will specify the end-effector's orientation relative to ground. Doing these tasks will cause the program to automatically perform inverse kinematics and adjusts the joint angles so that the end-effector reaches the target.

The joint angle limits are also simulated. If the end-effector cannot reach the target, it changes to a yellow X mark. If a collision is detected, the target changes to a large red X mark.

Roll and Grip Opening can be specified using the Roll and Grip viewport. Grip Opening can also be controlled using the Grip slider and textbox. Clicking on the Home button will reset the simulated TeachMover to its home position.



```
 TR|   SH  |   EL   |   P   |  R  | GR  |  x  |  y  |  z
---| -----| ------| -----| ---| ---| ---| ---| ---
0.0| 111.8| -132.8| -69.0| 0.0| 0.0| 1.0| 0.0| 2.0
```

GR: 3.0

01 Active Waypoint

01 Total Waypoints

TR	SH	EL	R	P	x	y	z
55.6	65.2	-132.1	43.2	-45.0	1.2	1.8	1.2

Figure 4.18. *User interface of Virtual Command.*

Table 4.3. *User interface components of Virtual Command.*

Number	**Name of UI Component**	**Remarks**
1	Lateral viewport	Receives r, z input for end-effector
2	Front viewport	Receives x, z input for end-effector
3	Top Viewport	Receives x, y input for end-effector
4	Home button	Resets TM to home position
5	3D viewport	Visual feedback function only
6	Pitch viewport	Receives Pitch input for end-effector
7	Roll and Grip viewport	Receives Roll and Grip Opening input for end-effector
8	Grip Opening slider and textbox	Controls Grip Opening
9	Waypoint controls	Specifies number of waypoints
10	Waypoint list editor	Shows waypoint list in joint angles and end-effector positions; Loads and saves waypoint lists
11	Set waypoint and Animate buttons	Sets waypoint at current position and Animates motion sequence
12	Implementation controls	Implements current position, entire motion sequence, or records entire motion sequence to the TeachMover
13	Forward kinematics sliders and textboxes	Controls joint angles using FK
14	Inverse kinematics sliders and textboxes	Controls joint angles using IK

The user now has a choice of whether to implement the current simulated position or to set a waypoint. For now, setting waypoints will be discussed first. Since Waypoint 01 must be at home position, the user will need to click the “+” button to add a new waypoint. The Set Waypoint button is clicked to set the waypoint. The active row in the listbox will now update to reflect the change. If the active waypoint is the last waypoint in the list, clicking Set Waypoint will automatically add a new waypoint to the bottom of the list. To decrease the number of waypoints, the “-” button is clicked and the last waypoint is deleted.

When a motion sequence has been recorded into the listbox, it can be saved to a MAT-file. Clicking Save... will bring up a dialog box. Waypoint MAT-files must be saved in the same folder where Virtual Command is stored. Clicking Load... will bring up another dialog box that asks which Waypoint MAT-file to load. Because MATLAB uses MAT-files for several purposes, it is important to keep track of which MAT-files have Virtual Command Waypoints stored inside.

To implement the current position shown on screen to the TeachMover, the Implement Current Position button is clicked. A series of tasks is then performed to ensure correct operation. First, Virtual Command reads the TeachMover’s current actual joint angle configuration. Second, it will simulate the motion from the actual configuration to desired configuration. This allows collisions to be detected. If a collision is detected, the operation is cancelled. If no collisions are detected, Virtual Command goes ahead in relaying the appropriate commands to the TeachMover.

To implement an entire motion sequence, the Implement Entire Sequence button is clicked. First, Virtual Command will instruct the TeachMover to go back to home position. Second, the entire motion sequence is simulated to look for collisions. If no collisions are detected, Virtual Command will then relay instructions to the TeachMover in sequence until all waypoints in the list are accomplished.

So far, discussion has been limited to online implementation of TeachMover movements. Before recording a motion sequence to TeachMover memory, it is important to press MODE+CLEAR on the teach pendant to ensure that its memory is empty.

To record a motion sequence to the TeachMover, the Save to TeachMover button is clicked. First, Virtual Command will bring the TeachMover back to home position. Second, the entire motion sequence is simulated to look for collisions. If no collisions are detected, Virtual Command will then relay instructions to the TeachMover in sequence until all waypoints in the list have been recorded. Note that this doesn’t move the TeachMover. To use the recorded sequence, the buttons MODE+RUN on the teach pendant must be pressed. This will cause the TeachMover to endlessly loop through the sequence until the STOP button is pressed.

Since the motion sequence is now in the TeachMover’s memory, the computer can be turned off and the TeachMover will still remember the sequence. However, if the TeachMover is turned off, it will “forget” everything and reset its step counters to zero, requiring recalibration every time it is turned on.

ii) Graphics issues

TeachMover Virtual Command was drawn on screen using the OpenGL renderer, as opposed to D-H Assist being drawn using

Painters. Because of the choice of renderer, Virtual Command exhibited improved performance in drawing 3D objects. However, as mentioned in the discussion of D-H Assist graphics issues, uipanels were difficult to use with OpenGL. Without uipanels, the GUI components in Virtual Command were more difficult to organize.

iii) Relaying commands to the TeachMover

To relay commands to the TeachMover, Virtual Command needs access to the serial port. MATLAB's procedure in accessing the serial port involves creating a serial object which references the actual hardware serial port,

```
serialPort = serial('COM1',...
    'Terminator','CR',...
    'DataTerminalReady','off',...
    'RequestToSend','off',...
    'Timeout',60);
```

performing an fopen operation,

```
fopen(serialPort);
```

sending string messages through the serial object using fprintf, for example

```
command = sprintf('@READ');
fprintf(serialPort,command);
```

reading out the TeachMover's response if appropriate

```
response1 = fgetl(serialPort);
```

then performing an fclose operation and delete the serial object from memory:

```
fclose(serialPort);
delete(serialPort);
```

In Virtual Command, MATLAB displays simulations using homogeneous transform matrices based on joint angles expressed in radians or degrees and grip opening expressed in inches. In order to relay appropriate commands to the TeachMover, joint angles and grip opening must be expressed in motor steps as specified in the TeachMover User Manual.

For online use, the TeachMover has a serial command called @STEP. It has the following syntax:

```
@STEP SP,J1,J2,J3,J4,J5,J6
```

The argument SP specifies the speed at which the stepper motors turn. It is arbitrarily set to 240 because it is a fast enough speed. The arguments J1 up to J6 are described in Table 4.4.

Table 4.4. *Conversion factors for @STEP arguments J1 to J6.*

@STEP Argument	Stepper Motor	Conversion Factor
J1	Trunk	19.64 steps/degree
J2	Shoulder	19.64 steps/degree
J3	Elbow	11.55 steps/degree
J4	Left Wrist Gear	4.27 steps/degree
J5	Right Wrist Gear	4.27 steps/degree
J6	Gripper Opening	371 steps/inch

Due to the mechanical setup of the TeachMover's cable drives, moving the shoulder joint will cause the elbow joint to adjust, maintaining its orientation relative to ground. However, this is not the case for the simulation in Virtual Command, where motion of just the shoulder joint will not cause the elbow joint to adjust. Furthermore, the TeachMover interprets positive steps as downward rotation while Virtual Command interprets positive angles as upward rotation. The gripper cables are also linked to the shoulder and elbow cables, which would cause it to open or close when the shoulder or elbow is moved. These differences are compensated for with the following calculations as implemented in

MATLAB code. This code snippet is the same as the one found in lines 15-20 of convertToSTEPS.m. The variables TR (trunk), SH (shoulder), EL (elbow), P (pitch), and Ro (roll) are expressed in degrees while GR (grip opening) is expressed in inches.

```
J(:,1) = TR*19.64;
J(:,2) = -SH*19.64;
J(:,3) = -(SH+EL)*11.55;
J(:,4) = -(SH+EL+P+Ro)*4.27;
J(:,5) = -(SH+EL+P-Ro)*4.27;
J(:,6) = -(SH+EL)*11.55 + GR*371;
```

It must be noted that the @STEP command requires relative values for motor steps. Therefore, the actual configuration of the TeachMover must first be obtained using the @READ command. It is implemented in the following MATLAB code found in lines 952-958 in main.m.

```
command = sprintf('@READ');
fprintf(serialPort,command);
response1 = fgetl(serialPort);
TM_config_steps = fgetl(serialPort);
TM_config_steps = str2num(TM_config_steps);
TM_config_steps = TM_config_steps(1:6);
```

The resulting array of motor step values in J and TM_config_steps is then passed to the functions defined in TM_ONLINE_Single.m or TM_ONLINE_All.m. These M-files will process both arrays to find the relative step values to relay to the TeachMover. In line 7 of TM_ONLINE_Single.m,

```
stepsRelative = stepsAbsolute - TM_config_steps;
```

Implementing an entire motion sequence to the TeachMover using TM_ONLINE_All.m will not require reading the actual configuration, as relative step values will have been calculated beforehand.

Found in lines 14 and 16 and lines 26 and 30, respectively for both M-files, are the MATLAB statements that formulate the command string and relay it to the TeachMover. This is the procedure for using the TeachMover in online mode.

```
command = sprintf('@STEP
   240,%0.0f,%0.0f,%0.0f,%0.0f,%0.0f,%0.0f',stepsRelative)
fprintf(serialPort,command);
```

The arguments for @STEP must be integer values. If real values are used, it will result in misinterpretation by the TeachMover, causing erratic behavior.

For offline use, motion sequences must be programmed into the TeachMover's built-in memory using the @QWRITE serial command. It has the following syntax:

```
@QWRITE L1,L2,L3,L4,L5,L6,L7,L8
```

The argument values for @QWRITE rely heavily on the output of convertToSTEPS.m, which are motor step values. These values are then represented as unsigned 16-bit integers, each having a high-order byte and a low-order byte. In the same way, arguments L1 to L8 are also unsigned 16-bit integers. This kind of representation is useful in formulating each argument as shown in Table 4.5.

Table 4.5. *Formulation of @QWRITE arguments.*

@QWRITE Argument	High-order byte	Low-order byte
L1	Program step number	
L2	255 – SP	1
L3	J2 low	J1 low
L4	J4 low	J3 low
L5	J6 low	J5 low
L6	J2 high	J1 high
L7	J4 high	J3 high
L8	J6 high	J5 high

Table 4.5 is interpreted in the following manner. L1 is the program step number, starting from 1. The high-order byte of L2 consists of 255 minus the SP value. Its low-order byte is set to 1, which is the TeachMover opcode for the MOVE type of program step. This is the only opcode used by this study. For L3, the low-order byte of J2 is taken to become L3's high-order byte, and the low-order byte of J1 is taken to become L3's low-order byte. The same process is applied to the other arguments. This process is implemented in lines 23-44 of convertToQWRITE.m.

```
%   Convert all elements of X into unsigned 16-bit integers
    X = uint16(X);
    for m = 1:6
        % Find low-order byte
        L(m) = rem(X(m),256);
        % Find high-order byte
        H(m) = (X(m) - L(m))/256;
    end; clear m

%   Formulate the QWRITE argument list
    QWRITE_prelim      = zeros(2,8);
    QWRITE_prelim(:,1) = [0; stepNumber];
    QWRITE_prelim(:,2) = [255-speed; 1];
    QWRITE_prelim(:,3) = [L(2); L(1)];
    QWRITE_prelim(:,4) = [L(4); L(3)];
    QWRITE_prelim(:,5) = [L(6); L(5)];
    QWRITE_prelim(:,6) = [H(2); H(1)];
    QWRITE_prelim(:,7) = [H(4); H(3)];
    QWRITE_prelim(:,8) = [H(6); H(5)];
    QWRITE_prelim(1,:) = QWRITE_prelim(1,:)*256;
    QWRITE_final       = sum(QWRITE_prelim);
```

Special consideration is to be taken for negative motor step values since @QWRITE requires unsigned arguments. Before negative values are processed for @QWRITE argument formulation, they are first converted to their unsigned 16-bit representation using two's complement. This process is implemented in lines 19-21 of convertToQWRITE.m.

```
%   Account for negative numbers and express them in their
%   negative binary form using two's complement
    if X(m) < 0
        X(m) = bitcmp(uint16(-X(m)),16) + 1;
    end
```

So far, the discussion has been about representing values in binary. However, there is no need to explicitly express arguments in binary since MATLAB has built-in functions such as bitcmp (bit complement) that directly manipulate decimal numbers in their bit level. Finally, to output the program to the TeachMover, line 47 of convertToQWRITE.m formulates the command string

```
commandString = sprintf('@QWRITE
    %0.0f,%0.0f,%0.0f,%0.0f,%0.0f,%0.0f,%0.0f,%0.0f',QWRITE
    _final);
```

Take note that all arguments are represented in decimal values with no fractional parts. Line 943 of main.m then sends the command to the serial port,

```
fprintf(serialPort,commandString);
```

The entire process is looped until all motion sequence waypoints are programmed into the TeachMover. After which, pressing MODE+RUN on the teach pendant will run the program on the TeachMover, even without assistance from Virtual Command.

Chapter 5

CONCLUSIONS

MATLAB’s Handle Graphics capabilities provide a robust environment for robotics simulation. This study has made extensive use of these tools, especially the axes, patch, and hgtransform objects.

In D-H Assist, it can help students easily grasp the visual-spatial interpretations of the Denavit-Hartenberg parameters. It allows them to immediately see the effects of changes to any parameter and how these changes affect the structure of the entire serial robotic arm.

In the TeachMover Virtual Command, forward and inverse kinematics concepts are visually demonstrated, while the tedious mathematical operations are hidden from view. It presents a motivation for the learning of robotics concepts.

Additionally, Virtual Command greatly simplifies the operation and programming of the Microbot TeachMover. Previously, the user had to control the individual joint angles using the teach pendant. On the other hand, using its serial operations capabilities required the formulation of long command strings. Simulations were also not possible, which may result in collisions and damage to the TeachMover. Virtual Command still uses the long command strings required in serial operations but can effectively automate the task as well as simulate the desired motions.

Chapter 6

RECOMMENDATIONS

The Microbot TeachMover doesn’t have an effective method of detecting its individual joint angle configuration. Since it relies heavily on the starting position where step counters are reset to zero, certain conventions need to be raised before implementing applications of the TeachMover. The tendency of the cable drives to slip near limits or under heavy load also contributes to the weakness of step counter based joint angle detection. A visually-based joint angle detection system can be developed to enhance the TeachMover’s capabilities.

In D-H Assist, graphics problems were encountered when using the Painters renderer. An implementation of D-H Assist using the OpenGL renderer can be developed, keeping in mind the changes needed in organizing the GUI components.

Virtual Command can be further improved by adding capabilities to control and program two TeachMovers simultaneously and synchronously. One of the TeachMover’s capabilities is to change the ‘@’ sign in serial commands to another character. This allows daisy-chained TeachMovers to discern whether a serial command was intended for one or the other.

MATLAB was designed and intended to be a rapid prototyping programming environment. This allows new ideas and algorithms to be quickly implemented and tested at the cost of operational speed. This is the case for D-H Assist and Virtual Command. These programs can be optimized for faster operation when ported to a compiled language, such as C++. It will also allow the usage of these programs even without installing MATLAB. A more structured planning approach is also needed for optimization and easier addition of new features.

REFERENCES

A. Books

K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, *Robotics: Control, sensing, vision, and intelligence*, New York: McGraw-Hill, 1987.

M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*, Hoboken, NJ: John Wiley & Sons, 2006.

A. J. Critchlow, *Introduction to Robotics*, New York: Macmillan; London: Collier Macmillan, 1985.

D. W. Pessen, *Industrial Automation: Circuit design and components*, New York: John Wiley & Sons, 1989.

B. Theses and Dissertations

A. Ayco, E. C. Balmes, M. J. D. Camilon, A. Cañete, and J. Yu, *Visual feedback for a two-link planar manipulator*, Department of Computer Engineering, University of San Carlos, 2007.

A. Almeda, R. Barangan, E. Ong and R. Tolibas, Jr., *The TYR programmable robot arm*, Department of Computer Engineering, University of San Carlos, 1992.

Q. M. Balintag, J. D. Cruse, A. G. Dulosa, D. F. Lim, and R. M. Lopeña, *Image-based pick and place program for the TeachMover robotic manipulator*, Department of Electrical, Electronics and Communications Engineering, University of San Carlos, 2007.

C. Manuals

The MathWorks, Inc., *MATLAB 7: Creating graphical user interfaces*, Natick, MA: The MathWorks, Inc., 2008.

The MathWorks, Inc., *MATLAB 7: Graphics*, Natick, MA: The MathWorks, Inc., 2008.

Questech, Inc. *MICROBOT® TeachMover™ User Manual,* Farmington Hills, MI: Questech, Inc., 1989.

D. Online sources - Journal articles

A. G. Brooks, *Coordinating human-robot communication*, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2007. http://proquest.umi.com/pqdweb?did=1372026431&Fmt=2&clientId=63427&RQT=309&VName=PQD. Accessed July 12, 2008.

S. Küçük, Zafer Bingül, *The inverse kinematics solutions of industrial robot manipulators,* Mechatronics, 2004. ICM '04. Proceedings of the IEEE International Conference on , vol., no., pp. 274-279, 3-5 June 2004
URL: http://ieeexplore.ieee.org/iel5/9394/29807/01364451.pdf?isnumber=29807∏=STD&arnumber=1364451&arnumber=1364451&arSt=+274&ared=+279&arAuthor=Kucuk%2C+S.%3B+Bingul%2C+Z.Accessed August 15, 2008.

M. Rohrmeier, “Web-based robot simulation using VRML”, in Proceedings of the 2000 Winter Simulation Conference. Orlando, FL, USA, 2000. http://www.informs-sim.org/wsc00papers/210.PDF. Accessed August 27, 2008.

E. Online sources - Software

V. M. de Andrade and S. P. Faria, “Cylindrical robot simulator” [Online], MATLAB Central File Exchange. 2007. http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=15803. Accessed August 7, 2008.

D. Riley, “3D PUMA robot graphical demo” [Online], *MATLAB Central File Exchange*. 2007. http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=14932&objectType=file. Accessed April 4, 2008.

D. Riley, “CAD2MATDEMO.M” [Online], *MATLAB Central File Exchange.* 2003. http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=3642&objectType=File. Accessed April 4, 2008.

M. Kontz, “Three-link planar robot” [Online], *MATLAB Central File Exchange*. 2001. http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=157&objectType=file. Accessed April 4, 2008.

J. Burkardt, “IVCON – 3D Graphics File Conversion” [Online]. 2002. Source code available: http://orion.math.iastate.edu/burkardt/g_src/ivcon/ivcon.html. Binaries available: http://sourceforge.net/project/showfiles.php?group_id=76306. Accessed May 22, 2008.

M. Wood, “HGTable - functions for displaying tabular data in a MATLAB list control” [Online], *MATLAB Central File Exchange.* 2006. http://www.mathworks.com/matlabcentral/fileexchange/10782. Accessed December 10, 2008.

Blender Foundation, “Blender 2.48” [Online]. 2008. http://www.blender.org. Accessed April 2008.

F. Other online sources

The ROVer Ranch, “Types of robots” [Online]. Available: http://prime.jsc.nasa.gov/ROV/types.html. Accessed September 5, 2008.

M. H. Rasouliha, D. Sproule, and J. Wong, “Computer controlled robot arm”, University of Victoria, 2004. http://www.ece.uvic.ca/499/2004a/group08/documents/final_report.pdf. Accessed August 27, 2008.

P. Corke, *A computer tool for simulation and analysis: the robotics toolbox for MATLAB*, 1996. http://www.petercorke.com/robot/ARA95.pdf. Accessed August 27, 2008.

M. Piskeric and D. Bockus, “TeachVAL II” [Online], Faculty of Mathematics and Science, Brock University, 2005. http://www.cosc.brocku.ca/~bockusd/TeachVal/manual.doc. Accessed August 1, 2008.

C. C. Linnell, “Robotics education and employment”, *The Technology Teacher*, v53 n2 p7-11. November 1993. Available: http://eric.ed.gov/ERICWebPortal/custom/portlets/recordDetails/detailmini.jsp?_nfpb=true&_&ERICExtSearch_SearchValue_0=EJ472054&ERICExtSearch_SearchType_0=no&accno=EJ472054. Accessed August 1, 2008.