

FMH606 Master's Thesis

Real time control of robotic arm manipulators

ROS



[1]

Course: FMH606 Master's Thesis, 2022

Title: Real time control of robotic arm manipulators.

This thesis report forms part of the basis for assessing the student's performance in the course.

Thesis code: FMH606 Master's Thesis

Participant: Mohammad Saifuddin Chowdhury

Supervisor: Roshan Sharma

Thesis partner: University of South-East Norway (USN)

Summary:

This thesis is based around creating an interface between a ROS-based robot arm and MATLAB, focusing mostly on robots in the industry. MATLAB will be used to implement a simulator of the robot and to create communication with it. In addition, Denavit Hartenberg representation and optimization were used to calculate forward and inverse kinematics. The results of this thesis are functional communication and robot arm movement, pick and place operations with objects, multi-robot interactivity along with trajectory tracking and position-based movements.

ACKNOWLEDGEMENT

As part of my thesis at the University of South-Eastern Norway, Porsgrunn, I wrote this dissertation. A lot of people helped the author with the thesis during this period, so they deserve special thanks. I'd like to send my thanks to all of them.

First of all, I want to thank the one above all, the omnipresent God, for giving me strength to write this thesis.

In addition, the author thanks his supervisor and guide Dr. Roshan Sharma, Associate Professor, Department of Electrical Engineering, IT, and Cybernetics, USN, Porsgrunn, for his valuable guidance, support, and encouragement. The work was completed successfully and on time. Thanks to his constant inspiration and constructive criticism. The author can't thank him enough.

Finally, the author has been away from home for long hours to complete this work. It's impossible to express his gratitude to his wife Saima Rashid for her understanding, patience, active cooperation throughout the master's dissertation. Thank you all for being supportive and caring. Special thanks go to his parents and relatives for their inseparable support and prayers.

Abstract

This thesis report has been written as a part of the 30-credit course FMH606 Thesis. Several ReactorX 150 Robot Arms from Trossen Robotics have been purchased by the University of South-East Norway. As a part of this purchase, a thesis has been created with the goal of creating, interfacing with the robot arm, forward and inverse kinematic problem solution using DH representation and optimization mechanism, pick and place operation by using gripper mechanism, multiple robot's interactive operation in MATLAB. This document will serve academic purposes in possible labs, or other applications, for robot arms.

It is a key objective for academic or industrial robot manipulators to achieve desired positions and orientations of their end effectors or tools to perform the specified task. It is necessary to possess solid knowledge of inverse kinematic problems to accomplish the above stated goal. Robot manipulators are used in many different fields for completing various tasks such as material handling, pick and place, interactive operation, collaborative operation, and hazardous field work, etc. Furthermore, medical robotics is applicable to therapy and surgery that require kinematic, and control functions. To build a career in robotics, academic robotic manipulators must be familiar to a student with their joint variables and kinematic parameters.

It is the joint actuators that control the motion of end effectors or manipulators, which causes the motion to occur in each joint. For this reason, it is imperative that the controller provide an accurate value for each joint variable to the end effector position. A robot manipulator's forward kinematics is a simple problem with a unique solution. By converting joint space to cartesian space of the manipulator, we can derive forward kinematics. The inverse kinematics, on the other hand, can be determined by making a transformation from Cartesian to joint space. The researchers have been primarily concerned with obtaining an exact solution of the joint variables. This paper presents an overview of academic robot manipulator including its evolution and classification. In this study, the difficulties of solving forward and inverse kinematics of robot manipulators are discussed, and an optimization method is introduced for solving inverse kinematics. A systematic study of the available tools and techniques has been undertaken in order to achieve the objective of the work and solve the robot arm manipulator interfacing and kinematic problem.

Porsgrunn, May 18th, 2022

Mohammad Saifuddin Chowdhury

Table of Contents

Chapter 1	
1.1 Overview	13
1.2 Evolution of robot manipulators	14
1.3 Structure of industrial robots	15
1.4 Classification by mechanism.....	16
1.5 Review analysis and outcomes.....	16
1.6 Problem statement	17
1.7 Scope of work	18
1.8 Organization of the thesis	18
1.9 Summary	19
Chapter 2.....	20
2.1 Overview	20
2.2 Materials	20
2.2.1 ReactorX 150 Robot Arm.....	20
2.2.2 DYNAMIXEL Servo Technology.....	21
2.2.3 Robot Operating System	21
2.2.4 MATLAB	21
2.2.5 Simulink	22
2.2.6 Simulation	22
2.3 Methods	22
2.3.1 Kinematics	23
2.3.2 Optimization to solve Inverse kinematic	27
2.4 Summary.....	28
Chapter 3.....	29
3.1 Overview	29
3.2 DYNAMIXEL Wizard.....	29
3.3 Mode of servo operation	30
3.4 Implementing the Library.....	30
3.4.1 Bulk_Read_Write.m	30
3.5 Communication Protocol and Run Mode	32
3.5.1 Communication Protocol Details	32
3.6 Software stack and libraries	35
3.7 Summary.....	36
Chapter 4.....	38
4.1 Xacro to URDF in ROS2	38
4.2 Robot Arm Calibration and Movement from Simulink.....	38
4.3 Importing URDF model in Simulink	39
4.4 Movement and Calibration of Simulation.....	42
4.5 Trajectory Visualization with Inverse Kinematics in MATLAB Simulink	45
4.6 Inverse kinematics output stored in MATLAB workspace.....	46
4.7 Summary.....	47
Chapter 5.....	48
5.1 Overview	48
5.2 Representation methods and kinematics	48
5.3 Kinematic variables and parameters.....	48

5.3.1 DH-Parameters	49
5.3.2 DH-algorithm for frame assignment.....	50
5.3.3 DH table representation of rx150 robot arm with offset.....	51
5.3.4 Code snippet to calculate end effector coordinate by using DH table.....	54
5.4 Forward kinematic result with DH representation	55
5.5 Inverse kinematic result using Simulink library function (IK engine).....	56
5.5.1 Real arm movement picture frame and video using Simulink (IK Library).....	57
5.6 Inverse kinematic result using DH model and optimization in simulation	58
5.6.1 Code snippet.....	60
5.6.2 Real arm movement picture frame and video	61
5.7 Summary	62
Chapter 6.....	63
6.1 Overview.....	63
6.2 Sleep position	63
6.3 Home position.....	64
6.4 Movement and Calibration of Physical Robot	65
6.5 Robot arm pick and place operation	69
6.6 Multiple robot interactive task.....	70
Chapter 7.....	72
7.1 Overview.....	72
7.2 ROS2 Installation	72
7.3 Dynamixel Servo ID	72
7.4 Selection Of Servo Operation Mode	73
7.5 Servo Motor Communication.....	73
7.6 Servo Motor Rebooting	74
7.7 Interfacing.....	74
7.8 Robot Arm Jerking Solution	75
7.9 Code Shifting	76
7.10 Extrinsic Function	76
7.11 DH Model Tuning	77
7.12 Summary	78
Chapter 8.....	79
8.1 Overview.....	79
8.2 Conclusions	79
8.3 Contributions	80
8.4 Future work	80
Chapter 9.....	82
9.1 References.....	82

List of Tables

Table 2.1: Detailed specifications of the ReactorX Robot Arm.....	18
Table 3. 1: Different mode of servo operation.....	27
Table 3. 2: Address code in "bulk_read_write.m"	28
Table 3. 3: Communication related code in "read_write.m"	28
Table 4. 1: ROS2 code line for file conversion.....	35
Table 5. 1: DH parameters.....	47
Table 5. 2: DH-parameters table for 4-dof revolute manipulator with offset.....	49

List Of Figures

Figure 1. 1: (a) Serial [11], (b) Parallel [11] and (c) Hybrid mechanisms [11].....	13
Figure 2. 1: A 2-DOF robot with coordinate transformations [2].....	23
Figure 2. 2: Animated image of a 2-DOF robot arm [3].....	23
Figure 3. 1: The interface for the DYNAMIXEL Wizard 2.....	26
Figure 3. 2: The build-up of communication packets	29
Figure 3. 3: The reading of a sent packet with error status.....	30
Figure 3. 4: Table with instructions, and descriptions, for the servos	30
Figure 3. 5: Instruction packet flow chart between software application and robot arm.....	31
Figure 3. 6: Start of the EEPROM Area data table.....	31
Figure 3. 7: Start of the RAM Area data table	32
Figure 3. 8: Software stack flow	33
Figure 3. 9: Dynamixel wizard operating Area.	33
Figure 3. 10: MATLAB script and command window with communication result.....	34
Figure 3. 11: MATLAB Hardware function in Simulink for multi robot interactive function.	34
Figure 4. 1: Simulink code for the geometric link and joints for the robot arm.....	36
Figure 4. 2: Visual block for the upper arm of rx150	37
Figure 4. 3: Visual block for selection of actuation for the waist joint	38
Figure 4. 4: Block diagram of the "rx150_robot_arm" with the slider inputs	38
Figure 4. 5: The "rx150_robot_arm" visualization with the slider inputs	39
Figure 4. 6: ZDP for the simulation with slider inputs	40
Figure 4. 7: Random position with slider inputs.....	40
Figure 4. 8: The angel plot for the joints at the random position.....	41
Figure 4. 9: The simulation of the RX 150 Robot Arm with trajectory based on inputs.....	41
Figure 4. 10: Inverse kinematic simulation block diagram of the RX150 robot arm	43
Figure 4. 11: RX150 robot arm simulation result.....	43
Figure 4. 12: RX150 robot arm joint angles result saved in workspace	44
Figure 5. 1: Position and direction of a cylindrical joint in a coordinate frame.....	47
Figure 5. 2: DH table coordinate frame of RX-150 robot arm configuration with an offset...50	
Figure 5. 3: Forward kinematic result by using DH model.	54

Figure 5. 4: Inverse kinematic simulation block.....	54
Figure 5. 5: Inverse kinematic simulation result.....	55
Figure 5. 6: Inverse kinematic result implemented on RX150 robot arm	55
Figure 5. 7: Inverse kinematic simulation result implemented on RX150 robot arm.....	56
Figure 5. 8: Video link for robot arm movement using inverse kinematic (Simulink library)	56
Figure 5. 9: Inverse kinematic simulation using optimization.....	57
Figure 5. 10: Optimization result comparison with real trajectory	57
Figure 5. 11: Optimization result RX150 robot arm simulation.....	58
Figure 5. 12: Optimization MATLAB code	59
Figure 5. 13: Optimization result implemented in RX150 robot arm real unit.....	59
Figure 5. 14: Video link for robot arm movement using inverse kinematic by optimization technique.....	60
Figure 6. 1: Image of the ReactorX 150 in the Sleep Position.....	61
Figure 6. 2: Simulink block to call ReactorX 150 in the Sleep Position	62
Figure 6. 3: Image of the ReactorX 150 in the Home Position.....	62
Figure 6. 4: Simulink block to call ReactorX 150 in the Home Position	63
Figure 6. 5: Image of the ReactorX 150 in the ZDP	64
Figure 6. 6: Slider inputs based on the ZDP	64
Figure 6. 7: Image of the ReactorX 150 with 90° change, from the ZDP, in joint 3.....	65
Figure 6. 8: Slider inputs based on the 90° change from ZDP	65
Figure 6. 9: Image of the ReactorX 150 with a random position from slider inputs.....	66
Figure 6. 10: Slider inputs based on the random position.....	66
Figure 6. 11: Simulink blocks for robot arm pick and place operation	67
Figure 6. 12: Video link for robot arm pick and place operation	67
Figure 6. 13: Simulink blocks for multiple robot arm	68
Figure 6. 14: MATLAB script write function code snippet.....	68
Figure 6. 15: Video link for multiple robot arm interactive operation	69
Figure 7. 1: Dynamixel wizard to change servo ID.....	72
Figure 7. 2: Dynamixel wizard to select servo operation mode	72
Figure 7. 3: Dynamixel wizard to check servo address with respect to ID	73
Figure 7. 4: Dynamixel wizard to reboot servo motor.....	73
Figure 7. 5: Dynamixel wizard default servo acceleration and velocity.....	74

Figure 7. 6: Dynamixel wizard changed servo acceleration and velocity	74
Figure 7. 7: MATLAB editor to get access to workspace from Simulink.....	75
Figure 7. 8: Extrinsic library function used in hardware interfacing MATLAB code	75
Figure 7. 9: RX150 each joint length measurement [9].....	76
Figure 7. 10: DH model verification model in Simulink.....	77

List of Symbols

a_i Link Length

α_i Twist angle

d_i Joint Distance

θ_i Joint angle

c_i $\cos\theta_i$, (i 1,2,3...n)

s_i $\sin\theta_i$, (i 1,2,3...n)

Nomenclature

Degrees of Freedom, DOF

Operating System, OS

Robot Operating System, ROS

Denavit Hartenberg, DH

Forward Kinematic, FK

Inverse Kinematic, IK

University of South-East Norway, USN

Unified Robot Description Format, URDF

Visual Studio, VS

Zero-Degree Position, ZDP

Chapter 1

Introduction

1.1 Overview

Today's world observes a rapid revolution in the field of automation. We live in an era where automation is sweeping the world. Automation is taking over the world these days. An example of such a development is the increasing desire for the implementation of robots, or robot arms, which is an ever-increasing demand. For example, one of the more noticeable trends in recent times appears to be the increasing desire to create robots or robot arms. It is particularly fascinating when we consider the growing desire to implement robotics into our everyday lives.

There have been reports that while robots in the industry are generally created and used for very specific purposes, this has led to a lack of diversity when it comes to the software and the utility, which can lead to a lack of diversity as well.

This kind of application does have a characteristic in common, it requires the robot to operate in an unstructured environment, as opposed to a structure industrial work cell. Due to the uncertainties in system modelling, sensor quality, and robot actuation, the control of robot movements and trajectory planning in unstructured environments presents significant challenges. Most robot applications at present can be classified into two categories: those that deal with both structured and unstructured environments, while the rest can be classified as a broader area of robot applications [10]. A rigorous treatment of the topics given in this text, which were intended to serve as a first introduction to robotics, should be included in any first introduction. A robot helps the human situation with additional convenient tasks as well as being of assistance in the workplace as well as providing an isolated environment that would prevent any harm, discomfort, repetition, etc. that would normally be present in such a situation. Almost all the tasks performed by machines are becoming more complex as technology progresses. Machines are becoming more capable of performing tasks which were previously performed by men, and which are now performed by robots in spite of the danger they pose. The robots of the future should be thought of as having human excellence in terms of their structure, intelligence, subtlety, as well as ability to react in a timely manner, to accomplish human tasks in human-like ways and to have an effective and safe co-operation between humans and robots. The aforementioned reasons make robots comprised of electromechanical systems that are capable of a high level of autonomy extremely complex electromechanical systems whose analytic description requires advanced mathematical methods. To develop such devices, represent many of the most interesting and challenging problems and issues in the field of Robotics. Reprogram ability is more important than any other feature of robots. Computer control is what gives the robot its utility and adaptability, thus giving the robot its utility and adaptability. A greater initiative has been undertaken by the government to encourage the use of robotics in the workplace. Robot manipulators are capable of performing a wide variety of tasks in a variety of fields. There are a number of fields where

this technology has been used, including cars, household goods, pick-and-place, radioactive field defusing, and defusing of explosives. It has found application in fields such as reconstruction, surgery and rehabilitation that involve activities such as kinematic, dynamic and control operations.

With the sole objective of creating a connection between the robot arm and MATLAB, Forward and Inverse kinematic calculation, Pick and Place operation and Optimization technique to build own inverse kinematic engine. This thesis examines a robot arm manipulator manufactured by Trossen Robotics, the ReactorX 150. As of today, Trossen Robotics utilizes robotics-focused software in Linux, but in order to create an approachable solution with an established general foundation, MATLAB was chosen as it is a program that many people, such as students, are already familiar with, and it has Microsoft support, which is the most popular operating system when it comes to personal computers.

As a first step towards achieving this interfacing goal, the robotic arm ReactorX, Linux-based robotics software, as well as the platform MATAB will be used together in order to create a fully functional communication with the robotic arm. It is recommended that communication between robots is built around the several servos so that the robot's movements can be controlled at the user's discretion, within the limits of safety and security. During robot manipulation, manipulators move along prespecified trajectories, which are sequences of points that display end effector positions as well as their orientations. Depending on their kinematics, trajectory can be either joint space or Cartesian space based on time. A robot designed for industrial applications can be explicitly understood as an open-chain mechanism consisting of a rigid body and a number of joints that connects the rigid bodies together [11]. The joints allow the connected bodies to follow specific motions with respect to one another. The rotational joint acts as a hinge between the connected bodies, and the rotational motion between them is limited to a small, relative rotation about the axis of the joint. It is known that a kinematic chain is made up of a group of rigid bodies linked to one another by joints [10]. It is made up of links which are individual rigid bodies that make up the chain. An open, closed, or branched kinematic chain can be serial, parallel, or a combination of both. This means that kinematic chains can be random, or open, or closed. The smooth operation requires the computation of all the points in Cartesian coordinates to be successful. Inverse kinematics refers to the process of converting trajectory locations from Cartesian coordinates to joint coordinates by converting the trajectory from Cartesian to joint coordinates [10].

1.2 Evolution of robot manipulators

During the twentieth century, the Czech playwright Karel Capek proved that the concept of the robot had been conceived by the playhouses of Rosassum's Universal Robots (R.U.R.) [10]. The word "robot" derives from "robota", which means subordinate labor in slave languages [11]. Asimov, the Russian science-fiction writer, in his novel 'Run-around' wrote about three fundamental laws that govern interactions between robots and human beings. This rule was formulated in 1940, shortly after his novel was published [10]. By the middle of the twentieth century, a new era of artificial intelligence (AI) research had begun as the first connections between human intelligence and machines were explored, triggering a long period of fertile research. During this time, the first robots were developed [10]. As the technologies of

mechanics, controls, computers, and electronic have advanced, the advancements in these sectors have become more and more important. With the advances that had been made over time, this virtuous circle began to produce that knowledge and understanding needed to give rise to what is known as robotics, the science and technology of robots. There was a confluence of two technologies that led to the construction of the first robots built in the 1960s: numerical control machines to make precision products and tele-operation to handle radioactive material remotely. In comparison with the human arm, these master slave arms reproduce one-to-one the mechanics of the human arm and had minimal control and little awareness of surroundings [11]. There was a development in the medium to late twentieth century when integrated circuits, digital computers and miniaturized parts led to the possibility of designing and programming computer-controlled robots. In the 1970's, industrial robots - also known as automated industrial robots - played an essential role in automating flexible manufacturing systems.[10]. The industrial robots were also used successfully in general industry in addition to their wide application in the automotive industry.

1.3 Structure of industrial robots

We are attempting to categorize industrial robots in this section by looking at the architectures of serial structures. The primary focus in this study is limited to robots that will be used primarily for manipulation tasks, optimization tasks and to numerical simulations of serial kinematic chains. Based on the number of degrees of freedom (DOF) or axes that a robot has, and their characteristics in terms of kinematics, robots can generally be classified. From the extent of freedom possessed by a robot manipulator, its working excellence can be judged. Typical 6-dof robot manipulators can generally accomplish a general task in 3-dimensional space, where an object can be anywhere in space at any time without regard to its position or orientation. However, one may need to design robot manipulators or other manipulators for specific applications, depending on parameters such as the degree of freedom (DOF) or kinematic characteristics. It is true that there are numerous criteria to classify robot manipulators, but typically, one can choose based on the degree of freedom or the number of axes [10]. In contrast, organizations such as the Robotics Institute of America (RIA), the Association Francaise de Robotique (AFR) and the Japanese Industrial Robot Association have grouped robotics into six distinct categories as follows:

- Manual handling devices
- Fixed sequence robot
- Variable sequence robot
- Playback robot
- Numerical control robot
- Intelligent robot

It is noteworthy that aside from the modules that have been mentioned above, industrial robot manipulators can also be divided into categories based on their mechanisms, degrees of freedom, actuation, workspace, control, motion, and application [10].

1.4 Classification by mechanism

Robot manipulators typically come in either a serial or a parallel form, depending on whether they contain an open or closed loop. There are two types of robot manipulators used in real world applications, the prismatic type (P) and the revolute type (R) joints, with one of the possible link types being rigid and flexible. It's simple that the axes of two adjacent joints can either be parallel or orthogonal depending on whether you choose to combine these joints and links in different ways. Due to the joint R and P, the axes of the two adjoining axes can be parallel or orthogonal. In orthogonal joints, one axis will rotate 90 degrees with respect to the other, and the two axes will intersect at 90 degrees with respect to their common normal [10], see Figure 1.1.

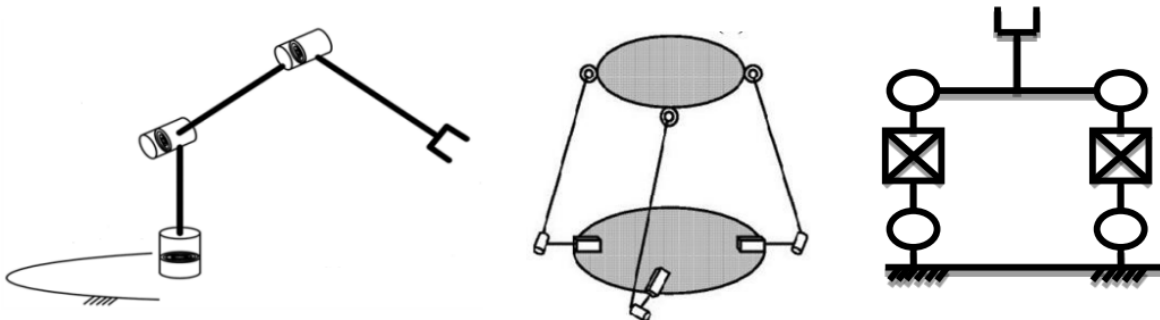


Figure 1. 1: (a) Serial [11], (b) Parallel [11] and (c) Hybrid mechanisms [11]

Examples of serial manipulators are PUMA, SCARA, Gough platform, Delta robot, are parallel manipulators and (a) (b) (c) 7 Fanuc S-9000W is an example of hybrid manipulator as shown in Figure 1.1(c).

1.5 Review analysis and outcomes

Taking on the structure and kinematics of manipulators and highlighting the importance of the configuration of the manipulators, the post demonstrates that kinematics has been regarded as a goldmine for robot designers. It is fair to say that there have been several researchers that have tried to develop inverse kinematic solutions since the late 80's, with different methods and for a variety of robot configurations [10]. There is increasing usage of robot manipulators in various industrial settings to perform services such as pick and place on a robot, so the major constraint was the finding of joint variables of the manipulator in order to achieve the desired position with certain object coordinates. Currently there is a growing need for the use of robot manipulators in other fields than the industrial world. They can now be used in a variety of fields, ranging from clinical rehabilitation, underwater applications, assembly tasks, agriculture, mining, space etc. to human interaction as well. Based on the literature review, it can be concluded that to achieve the desired position and orientation of the end-effector or tool, it is also crucial that manipulability and precision be preserved. There is a need for forward and inverse kinematics in trajectory planning as well as modeling. There is no question that the human arm is the key point in motivation and the driving force behind the development of

robot manipulators. In order to find out the optimum design from a design point of view, it is necessary to calculate the kinematic relationship between each joint variable in order to come up with the best possible design. The design of a mechanism or robot manipulator may have several configurations, structures, or functions, but the characteristics of a mechanism or robot manipulator that are most important for design are kinematic analysis, workspace analysis, trajectory generation and control. The significance of the explicit properties is always given to the robot manipulator applications about the applications of the robot manipulator. When the number of joint variables of the manipulator increases, the working space and manipulability of the robot manipulator increases as well, resulting in more complex mathematical formulations for inverse kinematic resolutions and difficulties in controlling the manipulator. In many human environments, as well as in industrial applications, robot manipulators are currently being used in a variety of designs and configurations. An important aim of the review analysis is to examine the various methods and techniques that can be used to solve the kinematics and optimization of any configuration of robot manipulator. A literature review reveals that various thesis proposals generally follow DH-algorithm, homogeneous transformation matrix, analytical approach, algebraic approach and geometric approach [10]. Algebraic solution of inverse kinematics has been the most commonly used method among all the developed methodologies. If the robot configuration is complex and has a greater degree of freedom, conventional algebra is difficult to model and obtain appropriate solutions. Quaternion and screw algebra reduce the complexity of higher mathematical formulations.

Inverse kinematics formulations are further simplified by a few effective elimination methods. When it comes to geometric algebra, when the first three joints of any manipulator or mechanism do not form any joint angles between them, the inverse kinematic problem cannot be solved exactly [11]. Furthermore, if the Jacobian matrix is not in good condition or suffers from singularities, the problem becomes unstable. Thus, the conventional method is reliable, but the configurations of manipulators and DOF's always present a mathematical complexity problem. The research community has adopted a number of intelligent techniques to resolve these issues, including artificial neural networks, fuzzy logic, hybrid ANNs, and others [10]. It is cheaper and easier to use these methods than traditional algorithmic solutions. In addition to these approaches, optimization approaches such as heuristics, metaheuristics, numeric-based approaches, etc. have shown promising results for solving inverse kinematics problems of robot manipulators of any configuration [11]. There is often a trapping effect in local optimum points that prevents most optimization algorithms from providing a global optimal solution. In order to accomplish global optimum for fitness, it is necessary to develop an algorithm.

1.6 Problem statement

The prime objective of the present thesis work is to set up, configure and test the ReactorX 150 robotic arm, use the robotic arm for direct and inverse kinematics, use the robot arm for pick and place operation, use two or more similar robot arms to perform interactive task, optimization technique to create inverse kinematic engine to solve inverse kinematic problem. To make it useful for real-time applications, the developed technique should be able to yield faster results.

1.7 Scope of work

Robotics has progressed so rapidly and is being adopted by companies increasingly, that many design and operational challenges have emerged. Various macro and micro problems are being explored to make the robot control system more user-friendly. Each component of the robot technology has been developed in order to provide a wide range of thesis interests. Aspects of the present thesis work, which focuses on academic robot arm RX150 bought from Trossen Robotics.

A detailed plan of the thesis work is presented as follows which is based on appendix A, at the end of this report:

- Set up, configure, and test the ReactorX 150 robotic arm which will be used with Simulink/MATLAB for various functionalities. In this task, necessary communication interfaces between the physical robotic arm and Simulink/MATLAB would be created.
- Using the robotic arm for direct and inverse kinematics. For the inverse kinematic, the end-effector trajectory can either be (i) specified using polynomials, or (ii) it can also be created by simply moving the robot arm manually by hand along the desired trajectory.
- Using the robotic arm for pick and place operations. In order to do this the gripper mechanism will be used.
- Using at least two or more ReactorX 150 robot arms, perform collaborative or interactive tasks. In this task, additional hardware/software support can be taken if require.
- Optimization technique can be developed to create inverse kinematic engine to solve inverse kinematic problem. The developed technique should be able to yield faster results, in order to make it useful for real-time applications.

1.8 Organization of the thesis

Chapter 1 is the Introduction part of the dissertation that provides a brief historical overview of robot evolution along with information about the types of manipulators, classifications, and application of robots in various fields. Forthcoming chapters apart from introduction chapter are organized as follows.

Chapter 2 delivers the methods, several software and hardware used for this thesis.

Chapter 3 will go over the results in terms of the simulator and physical interfacing

Chapter 4 robot manipulator configurations are analyzed and mathematically modeled. A brief discussion of various conventional techniques is presented, including algebra, analytical method, iterative method, numerical method, geometric method, homogeneous matrix, DH algorithm, and quaternion algebra. A classification of 4-dof manipulators and the DH parameters as well as their mathematical modeling has been presented. A series of steps will be presented to implement the simulator.

Chapter 5 discusses about the adopted optimization algorithms for the solution of inverse kinematics of robot manipulators. In this chapter forward kinematics equations are derived from DH representation to find out the joint variables of robot manipulator. Kinematic results achieved through all adopted techniques and comparison has been made with other existing techniques. Forward and inverse kinematics along with the workspace analysis and joint angle behavior has been addressed and compared.

Chapter 6 presents the real hardware unit implementation for various task. Output in the form of pictures and Simulink blocks are depicted in this chapter in terms of the physical interfacing.

Chapter 7 will be a discussion of different problems about interfacing and coding in the thesis which the author has experienced.

Chapter 8 presents the conclusions of the dissertation and future research guidance with summary of contribution. That will conclude the report with an overall conclusion of the thesis.

1.9 Summary

In the current chapter, the general synopsis of the different types of robot manipulator, classifications, history of developments are presented. The chronological progresses of some techniques like kinematics and optimization are presented and status has been briefed. Basic applications of kinematics and objectives are also discussed in this chapter.

Chapter 2

MATERIALS AND METHODS

2.1 Overview

As a group of rigid joints connected by specific joints, robot manipulators can be considered. It is possible to manufacture joints that are revolving, prismatic, screw-mounted, universal, or cylindrical. A robot manipulator is considered to have the first link embedded at its base while the last link can move around within its working space. Revolute joints rotate about their axes, while prismatic joints slide along their axes without rotating. Later in the section are described the selected configurations of robot arm manipulators and few methods applied on it to solve interfacing with real hardware unit, kinematic solution and optimization.

2.2 Materials

There are two main categories of robot manipulators, which are classified according to the type of kinematic chain they are made from. It is not only necessary to determine the classification based on the type of robot, but it is also important to make it based on the joints and connections, as explained in the previous section. Ideally, from a research perspective, we are looking to develop university level robot manipulators whose simplest configuration is a 4 to 6-dof revolute robot. There are various types of robots used in industries for various tasks, and it is evident that robot manipulators with Trossen robotics configuration and revolute robots with 6-degrees of freedom are mostly used in academic research. As a result of this, it was decided to take only this kind of robot manipulators into consideration.

2.2.1 ReactorX 150 Robot Arm

The thesis involves the ReactorX 150 Robot Arm from Trossen Robotics and Interbotix, called the X-series Robot Arms. The ReactorX Robot Arm is a part of a new series of different models from Trossen Robotics including a total of 11 different versions with different degrees of freedom (DOF), weight, operating weights, and reach. [1]

The models span from 4-6 DOF, working payload of 100g, as well as repeatability of 5-1mm depending on the chosen model. The ReactorX model was chosen seeing as it has 6 DOF, a repeatability of 2.5mm, while at the same time being on the cheaper side of the spectrum. A table with the physical specifications for the ReactorX150 Robot Arm, given from the Trossen Robotics, is given below, in Table 2.1.

Table 2.1: Detailed specifications of the ReactorX150 Robot Arm [12]

Degrees of Freedom:	6
Reach:	450mm
Span:	900mm
Repeatability:	2.5mm
Working Payload:	100g
Weight:	4lbs \approx 1.8kg

2.2.2 DYNAMIXEL Servo Technology

The ReactorX Robot Arm uses two different types of servos, the DYNAMIXEL XM-430-W350T and the DYNAMIXEL XL430-W250-T, both of which offer monitoring of temperature, as well as positional feedback, voltage levels, and load, all given through communication with the DYNAMIXEL. The U2D2 is a communication protocol which gives the user access to communicate with DYNAMIXEL servo motors. The servos are connected through Daisy Chain. Daisy Chain refers to communication between multiple items through a singular connection. As for the Trossen Robotics robot arms, all servos of a robot arm are connected through Daisy Chain, thereby allowing the operator to communicate with each servo despite only being connected with a connection point at the base of the robot arm. This allows for a simplified communication as well as preventing the need for several ports or wires to cluster or inhibit the movement of the robot arm [12].

2.2.3 Robot Operating System

Robot Operating System (ROS) is a framework mainly created for robot support in Ubuntu Linux. While there are experimental versions available for other systems, such as Microsoft Windows and macOS, the main goal is to create a loosely based framework which supports packages and libraries from several different distributors in terms of robot knowledge. Through this philosophy, the framework relies on different sectors experience in creating an operating system (OS) which contains useful documentation, and code, for the different utilizations for different robots [12].

Interbotix offers packages and documentation for most of their robot arms, all created and tested in Ubuntu Linux, with the goal of creating easy to understand simulation software to run

in ROS. This thesis uses the newer ROS2 software, specifically the “ROS2 Debian package” version.

2.2.4 MATLAB

MATLAB is a coding language, or rather a programming platform/environment, owned by MathWorks and is designed for programming with a higher focus mathematics, simulation, and general manipulation of lot of data. The general purpose of MATLAB is to interface programs cross different languages based on matrix data, i.e., use matrices to manipulate raw data despite the original language of the data gathering. This matrix focus is also a part of the name MATLAB, which in full is “matrix laboratory”. The language itself bears similarities to other languages, such as Python, C++, and simpler versions of C#. MATLAB also supports add-ons created by both the company itself, MathWorks, or other third parties. These add-ons may add a lot of specific functions and code necessary for more specific purposes such as Vision Technology, Machine Learning, and state machines [12].

2.2.5 Simulink

Simulink is a graphical simulation-based add-on to MATLAB which focuses on block-based coding rather than script-coding, although user-defined blocks based on user-created scripts are allowed. Much like the base-version of MATLAB, Simulink has the possibility of adding on a lot of code and block created by MathWorks or third parties [12].

2.2.6 Simulation

MATLAB, and Simulink, will operate as the main software for the thesis seeing as the thesis goal is to create interfacing between the robot arm and MATLAB. Based on the xacro-URDF file conversions, described in chapter Xacro to URDF, the new URDF-files should be imported to MATLAB and ran in Simulink to create a manual simulation of the robot arm. This simulation will operate the same way as the original ROS2 simulation, documented through several videos at Trossen Robotic [12].

2.3 Methods

This thesis primarily concentrated on inverse kinematics when interacting with real robot arms, setting up DH tables, and analyzing robot manipulators based on inverse kinematics. One must compute the joint variability associated with each joint of a robot manipulator in order to determine all possible formations in which the end effector can be positioned in space. As a result, authors have faced the following problems over the past few decades:

- There is a relationship between the complexity of the robot manipulator geometry and the complexity of the inverse kinematic robotics problem. It is necessary that the first three joints are the same to reach a satisfactory geometric solution.
- There are a few calculations which cannot be performed in real-time to solve the inverse kinematic problem.

- The possibility of achieving a closed form or single solution is not always possible.
- In addition, when it comes to some robot manipulator configurations, it is difficult to get real solutions. A numerical solution is achieved from the algebraic solution of kinematic equations.
- This thesis seeks to find out the joint variables or inverse kinematic solutions for the RX150 manipulators but annotated with numerical solutions- when the Jacobian matrix is singular (ill-conditioned), or the initial approximation is not precise, the solution may be unstable. The main objective of this thesis is to find out the joint variables or inverse kinematic solutions for the rx150 manipulators.

To achieve the thesis goal, several different methods have been used. Some of these methods are the use of different software, such as MATLAB, with Simulink, and Robot Operating System, while a physical approach includes the robot arm itself, to have a concrete approach to testing and utilizing the different software programs. This chapter will explain more about each method used and why these methods were relevant to the thesis.

2.3.1 Kinematics

A part of the goal in operating the robot arm was to fully utilize the kinematics part of physics. Kinematics are based on motion of physical elements, which in this case is the robot arm, and how force is used to manipulate a physical element [12].

Kinematics may significantly differ in expression from element to element, as well as between different thesis or situations. This thesis has had a focus on inverse- and direct kinematics, also called forwards- and inverse kinematics. The kinematics for this thesis would encompass how different end effector positions and angles would affect the joints of the ReactorX arm, Six joints in total. However, joint number five has not been used since this servo motor uses in rotational application like screw fixing which is not a concern or interest for this thesis work.

2.3.1.1 Forward Kinematics

Direct/forward kinematics encompasses a more linear view on kinematics and could also be seen as a more cause-and-effect approach. Forward kinematics is based on kinematics where we can provide some known joint angles to the robot arm, which will calculate the end effector position resulting movement by the robot arm.

By using forward kinematics, we can select the joint positions ourselves, as well as find/see the position of the end-effector in 3D space based on the joint positions. The middle portion of the gripper arm is the end-effector for the RX150 robot arm. As part of the simulation of the forward kinematics of this robot arm, Simulink and Simscape Multibody are going to be used. The first step is, to get the URDF file that tells us what physical and inertial properties of the robot arms contains [12].

2.3.1.2 Inverse Kinematics

Inverse kinematics focuses on the end-product/goal of the movement and from there calculates the necessary angles required for the joints. This could, for example, be by having the

knowledge of the gripper position, of the robot arm, and a declared end effector position, and from this deviation calculate the resulting angles/torque needed to move the gripper to the goal.

In order to solve the inverse kinematic problem, we assume that the position of the robot arm's end-effector is known. A robot arm's end effector, as we mentioned earlier, is the center of the left and right gripper links. For inverse kinematics to work, the position of the end-effectors in space must first be defined. Trajectory generation is the process of defining this position.

2.3.1.3 Example of Kinematics

Transforming (either rotating or translating) a coordinate frame is also called as homogenous transformation. In order to rotate a coordinate frame by an angle q_1 , we can use the rotation matrix in a two-dimensional plane as shown below,

$$R_{(q_1)} = \begin{bmatrix} \cos q_1 & -\sin q_1 & 0 \\ \sin q_1 & \cos q_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad 2-1$$

In order to translate a coordinate frame by l_x and l_y length/distance, we can use the translation matrix in a two-dimensional plane as shown below,

$$T(l_x, l_y) = \begin{bmatrix} 1 & 0 & l_x \\ 0 & 1 & l_y \\ 0 & 0 & 1 \end{bmatrix} \quad 2-2$$

To find the position of end effector (given the joint angles) using forward kinematics, the following transformations should be taken in their order:

- (1) $R(q_1)$: Rotate the reference x – y coordinate frame by joint angle q_1 in counterclockwise direction (Figure 2.1(b)).
- (2) $T_x(l_1)$: Translate in the x direction by length l_1 (Figure 2.1(c)).
- (3) $R(q_2)$: Rotate by joint angle q_2 in counterclockwise direction (Figure 2.1(d)).
- (4) $T_x(l_2)$: Translate in the x direction by length l_2 (Figure 2.1(e)).

The homogeneous transformation matrix is then given by the product of rotation and translation matrices as,

$$\begin{aligned} E &= R_{q_1} \cdot T_{xl_1} \cdot R_{q_2} \cdot T_{xl_2} \\ &= \begin{bmatrix} \cos(q_1 + q_2) & -\sin(q_1 + q_2) & l_1 \cos(q_1) + l_2 \cos(q_1 + q_2) \\ \sin(q_1 + q_2) & \cos(q_1 + q_2) & l_1 \sin(q_1) + l_2 \sin(q_1 + q_2) \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \quad 2-3$$

As an example of the two aforementioned types of kinematics, the below equations, 2-4 and 2-5, may be given as example equations of a 2DOF. This example is based on a two-jointed

robot arm. From the homogeneous transformation matrix, we can find the position of the end effector (x_e, y_e) . It is given by the first two elements of the last column in equation 2-3 as,

$$x_e = l_1 \cos(q_1) + l_2 \cos(q_1 + q_2) \quad 2-4$$

$$y_e = l_1 \sin(q_1) + l_2 \sin(q_1 + q_2) \quad 2-5$$

The above equations calculate the resulting x and y coordinates based on the given length of each joint arm, l_1 and l_2 , and the angle of each joint, q_1 and q_2 . This is direct kinematics. Known angles and lengths result in a coordinate.

For inverse kinematics, the problem now is to find the joint angles q_1 and q_2 that would orient the robot arm in such a way that the end-effector will reach the given position (x_e, y_e) .

Equations 2-4, 2-5 can be solved for q_1 and q_2 using algebra. Known quantities in equation 2-4, 2-5 are x_e, y_e, l_1 and l_2 . The unknown variables to solve are q_1 and q_2 . Through a series of formulations, the corresponding equations for q_1 and q_2 are given in 2-6 and 2-7, below.

$$q_1 = \tan^{-1}\left(\frac{y_e}{x_e}\right) - \tan^{-1}\left(\frac{l_2 \sin(q_2)}{l_1^2 + l_1 l_2 \cos(q_2) + l_2 \cos(q_2)}\right) \quad 2-6$$

$$q_2 = \cos^{-1}\left(\frac{x_e^2 + y_e^2 - l_1^2 - l_2^2}{2l_1 l_2}\right) \quad 2-7$$

The above equations for q_1 and q_2 are based on the knowledge of the joint lengths, l_1 and l_2 , just as before, but instead of knowing the angles, the x and y coordinates are now needed instead. This is inverse kinematics. Seeing as a coordinate is given, with the goal of finding the corresponding angles, more complicated equations are needed. In addition to being more complicated, the equation for q_1 requires the resulting angle for q_2 as well, thereby showing the dependency between angles when using inverse kinematics. A vector-based schematic of the 2-DOF robot arm is displayed in Figure 2. 1, while a more realistic depiction of a 2-DOF robot arm is displayed in Figure 2. 2.

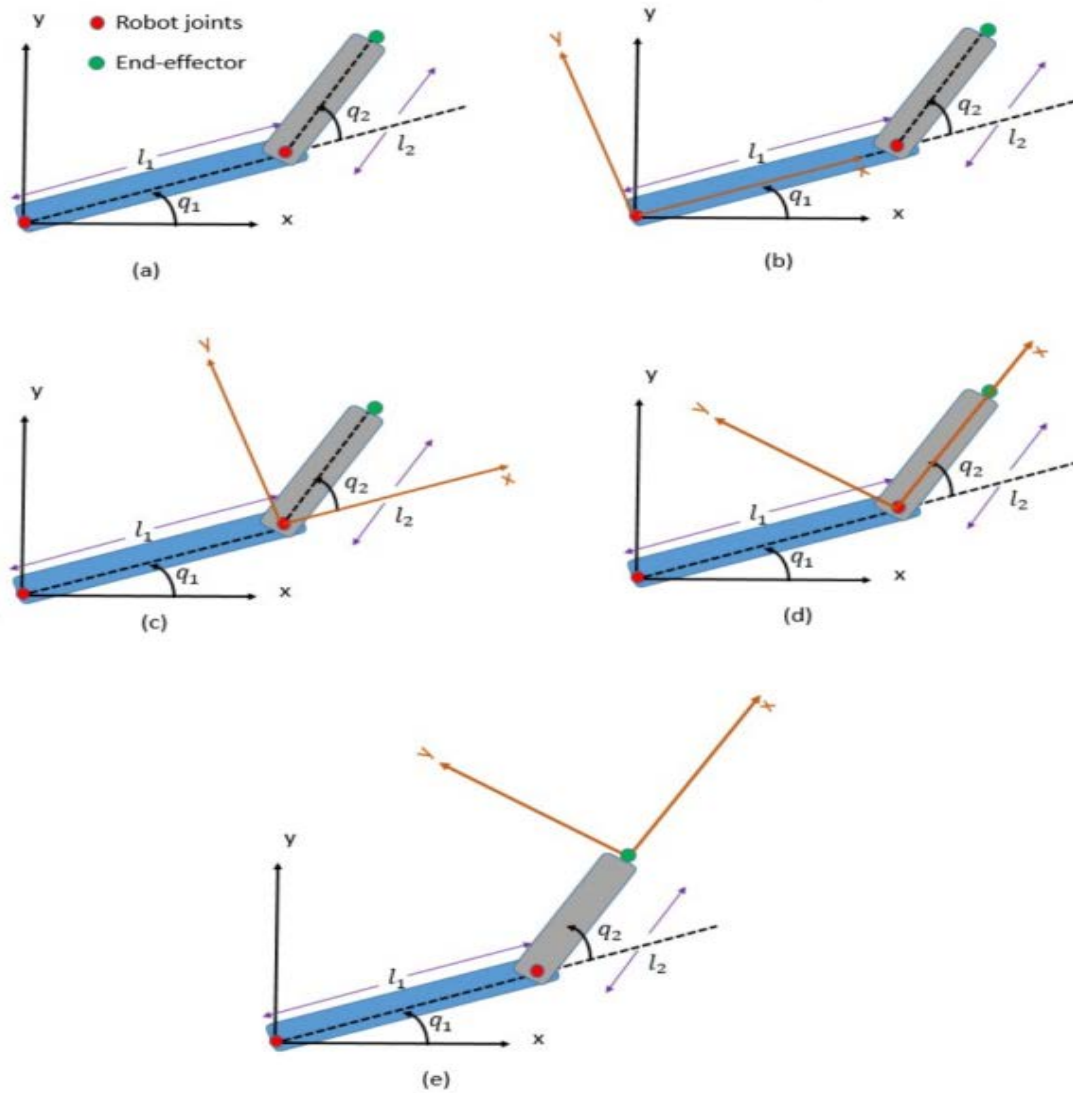


Figure 2. 1: A 2-DOF robot with coordinate transformations [2]



Figure 2. 2: Animated image of a 2-DOF robot arm [3]

2.3.2 Optimization to solve Inverse kinematic

Inverse kinematic solutions can be determined by using different optimization algorithms. To apply optimization algorithms, all that is required is to define the objective function for the respective manipulator. It is discussed in detail in chapter 5 how to formulate objective functions, which can then be applied to any manipulator configuration with minor modifications. A candidate solution is also produced for each individual joint variable by the objective function, which can be defined by the configuration vector of the manipulator. A forward kinematic equation and its associated constants or parameters are all that are needed for this method. By using this methodology, many tasks related to robot manipulators can be completed, such as design, kinematic analysis, and synthesis of kinematic structures. Alternatively, MATLAB's 'fmincon' optimization function and 'sqp' algorithm can be used to optimize robot manipulator tasks with higher degrees of freedom and more complex tasks. Nonlinearity should be handled by optimization algorithms. Inverse kinematics can be solved by implementing any optimization algorithm that is capable of solving various multimodal functions. An optimization-based approach can be implemented to overcome the limitations of conventional tools and intelligent methods [10]. It is generally assumed that the optimization-based methods are more stable, and they often lead to convergent global solutions. The current work investigates optimization methods based on error minimization objective function.

Here, the objective function, associated constraints, and the formulation of the objective function are all discussed in detail.

2.3.2.1 Position based function

The current position of the manipulator is described by (2.1):

$$P_{current} = [x_e, y_e, z_e] \quad (2.1)$$

Desired position of end effector can be denoted by (2.2):

$$P_{desired} = [x_{e_hat}, y_{e_hat}, z_{e_hat}] \quad (2.2)$$

A comparison will be made between the current position of the end effector and the desired position. This may result in the following equation being used to represent the fitness function. This was determined using the equation (2.3) which is based on a homogeneous Euclidian distance between the current positions and the desired positions of end effectors evaluated by the number of iterations.

$$e_{min} = P_{desired(i)} - P_{current(i)} \quad 2.3$$

Current position x_e, y_e, z_e can be evaluated from equations. (2.4), (2.5) and (2.6) as follows.

End effector position x_e equation as,

$$x_e = (\cos(q1)*\cos(q2))/20 + (3*\cos(q4)*(cos(q1)*\cos(q2)*\cos(q3) - \cos(q1)*\sin(q2)*\sin(q3)))/20 - (3*\sin(q4)*(cos(q1)*\cos(q2)*\sin(q3) + \cos(q1)*\cos(q3)*\sin(q2)))/20 + (3*\cos(q1)*\cos(q2)*\cos(q3))/20 - (3*\cos(q1)*\sin(q2)*\sin(q3))/20 \quad (2.4)$$

End effector position y_e equation as,

$$y_e = (\cos(q_2)\sin(q_1))/20 - (3\cos(q_4)(\sin(q_1)\sin(q_2)\sin(q_3) - \cos(q_2)\cos(q_3)\sin(q_1)))/20 - (3\sin(q_4)(\cos(q_2)\sin(q_1)\sin(q_3) + \cos(q_3)\sin(q_1)\sin(q_2)))/20 - (3\sin(q_1)\sin(q_2)\sin(q_3))/20 + (3\cos(q_2)\cos(q_3)\sin(q_1))/20 \quad (2.5)$$

End effector position z_e equation as,

$$z_e = \sin(q_2)/20 + (3\cos(q_2)\sin(q_3))/20 + (3\cos(q_3)\sin(q_2))/20 + (3\cos(q_4)(\cos(q_2)\sin(q_3) + \cos(q_3)\sin(q_2)))/20 + (3\sin(q_4)(\cos(q_2)\cos(q_3) - \sin(q_2)\sin(q_3)))/20 + 1/4 \quad (2-6) \quad (2.6)$$

The above three equations (2.4), (2.5) and (2.6) derived from DH representation and homogeneous transformation matrix which has been briefly discussed in chapter 5.

Subjected to joint limits in radian

$$-1.57 \leq q_1 \leq 1.57$$

$$-1.57 \leq q_2 \leq 1.57$$

$$-1.57 \leq q_3 \leq 1.57$$

$$-1.57 \leq q_4 \leq 1.57$$

Now overall error minimization objective function can be given as follows,

$$J = e^T Q e \quad (2.7)$$

where Q is 3x3 weighting matrix for the minimization of the problem and calculation of the entire joint angles base on constraint can be achieved using objective function (2.8). The performance of considered algorithm is checked with the parameters: $l_1 = 0.117$ m, $l_2 = 0.133$ mm, $l_3 = 0.150$ mm, $l_4 = 0.147$ mm, $b_1 = 0.015$ mm. Upper and lower limit of five joint angles(in degree) are: $q_1 = [-90, 90]$; $q_2 = [-75, 90]$; $q_3 = [-90, 30]$, and $q_4 = [-90, 90]$.

2.4 Summary

This chapter presents the discussion of different materials and methods adopted for the kinematic analysis. The main purpose of this chapter is to avail the detail description of adopted material for interfacing with robot manipulator, kinematic analysis, and different methods to achieve the objective of the thesis. The detailed derivation of forward and inverse kinematic solution has been given in chapter 5 where the result of inverse kinematic solution from optimization technique also represented.

Chapter 3

Interfacing with ReactorX 150 Robot Arm

3.1 Overview

Interfacing with the robot arm is one of the major goals in this thesis since ‘Forward’ and ‘Inverse’ kinematic, ‘Pick and Place’ operation, and ‘Collaborative or Interactive’ task between two ReactorX 150 robot arms shall be done in real hardware. Dynamixel wizard and manual sliders are used which provide inputs or set goals to test the different servos. This chapter will focus basically on the interfacing, as well as how to communicate, with the five different servo motors of the robot arm.

3.2 DYNAMIXEL Wizard

Before connecting and running any code, it is very useful to connect to the robot arm using the DYNAMIXEL Wizard application, which is provided by DYNAMIXEL. Out of the many different features of the application, the most important feature is to view all the information of every register/servo, i.e., current position, temperature of the servo, PID values etc. Moreover, control of servos can also be done from this application. Safety boundary of different servos can also be set from this wizard. The most useful part of this application is the initial scan for different baud rate and DXL_ID, which is the ID for the different servos. The DYNAMIXEL Wizard is displayed in Figure 3-1 [12].

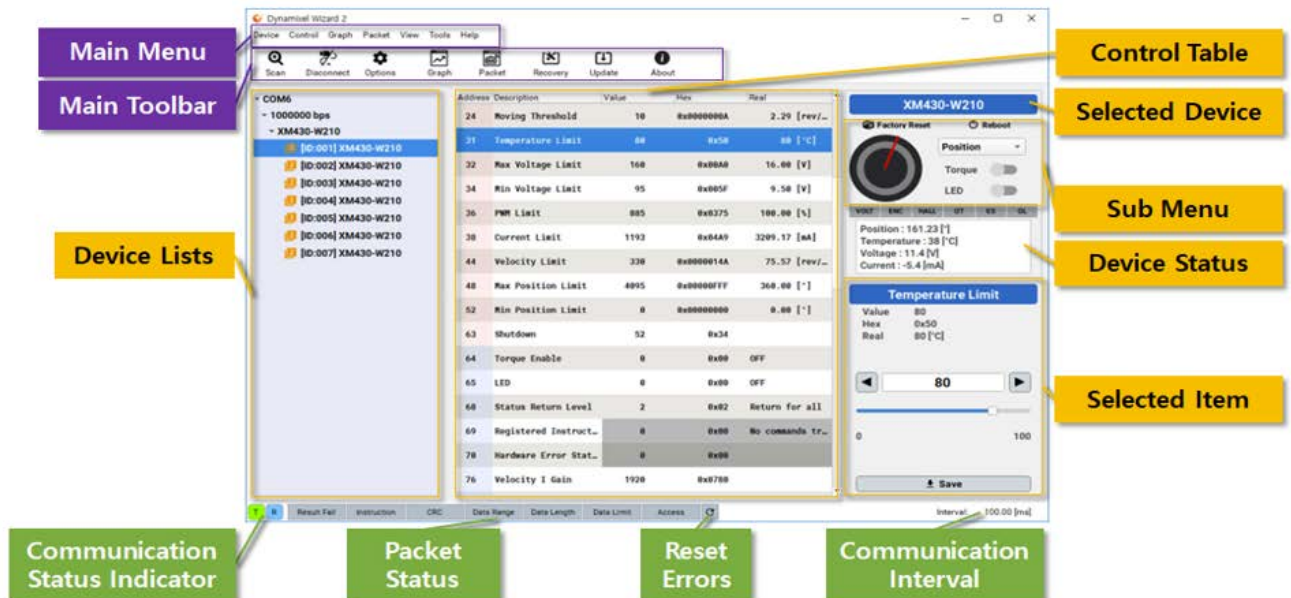


Figure 3. 1: The interface for the DYNAMIXEL Wizard 2

3.3 Mode of servo operation

RX150 robot arm uses Dynamixel servo motor in each joint. There are two types of servo motor (XL430 and XM430) are used in this manipulator arm. Dynamixel servo has different operation mode which has been presented in Table 3-1.

Table 3. 1: Different mode of servo operation

Mode	Description
(0) Current Control	This mode can also be referred and torque mode. You need to have own controller.(Built-in controller is disabled in this mode)
(1) Velocity Control	To control velocity of wheel
(3) Position Control	Controlled by internal PID. Only need to update goal position.
(5) Extended-Position Control	Same and Position Control but <u>upto</u> 512 full turn.
(5) Current base position control	Controls both position and current (torque)
(16) PWM control	Directly controls PWM for output

3.4 Implementing the Library

Another way to connect to the robot arm is the DYNAMIXEL SDK. It is a software library, mainly created in C and C++, which enables the user to communicate with the various DYNAMIXEL units, such as the servos. This library is integral for interfacing with the ReactorX 150 Robot Arm, or any of the other ROS-based robot arms. This library is available in several different programming languages, such as C, C#, C++, LabVIEW, JAVA, MATLAB, etc.[12].

In this thesis, the MATLAB library was used as an interface to communicate with ReactorX 150 Robot Arm. The library includes several methods which simplifies the process of communication. Some of the more relevant methods include a “read_write.m” file, which is used for simple communication with the arm, a “bulk_read_write.m”, which is used for multiple messages, and a “rebooting.m”, which is helpful to reboot the robot without having the possibility of doing something wrong [12]. For a full list of the different methods generated, please use this [link](#) [5].

3.4.1 Bulk_Read_Write.m

The “read_write.m” file should be the first MATLAB program to change seeing as other programs build of the communication method. The program uses a while loop in which the communication is executed, and in each loop, program waits for the next user input. Initially, this program sends command to first servo (DXL_ID_1) to rotate 360° rotation which is base servo. Later the program is modified to give command to all 5 servos in the same while loop. Different DXL_ID from 1 to 5 is used to address individual servos [12].

The file starts with the different addresses which is relevant for the communication. This may, e.g., be referring to torque, velocity, position, etc. A code snippet of the “read_write.m”, with the addresses, is displayed in Table 3-2.

Care should be taken to check the ADDRESS of the registers which may not be same with working robot. The baud rate is usually 1,000,000 by default but may be adjusted if needed.

Table 3. 2: Address code in "bulk_read_write.m" [12]

```
% Control table address
ADDR_PRO_TORQUE_ENABLE      = 64;% Control table address is different in
                             DYNAMIXEL model
ADDR_PRO_GOAL_POSITION      = 116;
ADDR_PRO_PRESENT_POSITION   = 132;
ADDR_PRO_GOAL_VELOCITY      = 104;
```

The communication with the robot arm is done through a computer port, and the port number can be found in the Device Manager, in Windows. After connecting computer with the robot arm, the port is then mentioned as the communication port in the program.

After the connection, the next goal for the program is to write instruction to the relevant addresses to the robot arm. After successful connection next part of code will be to enable torque of servo. The code which enables torque is displayed in Table 3-3. If the connection is unsuccessful, the program ends [12].

Table 3. 3: Communication related code in "read_write.m"[12]

```
% Enable Dynamixel Torque DXL_ID_2
writelByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID_2,
ADDR_PRO_TORQUE_ENABLE, TORQUE_ENABLE);
dxl_comm_result = getLastTxRxResult(port_num, PROTOCOL_VERSION);
dxl_error = getLastRxPacketError(port_num, PROTOCOL_VERSION);
if dxl_comm_result ~= COMM_SUCCESS
    fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));
elseif dxl_error ~= 0
    fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, dxl_error));
else
    fprintf('Dynamixel has been successfully connected \n');
end
```

Upon enabling torque, the program then starts a while loop which reads, and waits for, user input continuously. This keyboard input check code is showed in Table 3-3. The example code currently communicates with servo number 1, which is the rotation-based servo.

Expansion to all five servos is done by expanding the program to include checks for other keyboard inputs related to other servos, differentiated by their respective DXL_ID number.

3.5 Communication Protocol and Run Mode

ReactorX 150 Robot arm has 2 - XM430-W350-T servos and 4 - XL430-W250-T servos. Both types of servos are operated with an ARM Cortex-M3 32-bit controller. They are also attached with 12-bit contactless encoder to allow for 360° movement without having cables in the way. It supports baud rates from 9600 up to 4.5 Mbps, and this thesis uses a baud rate of 1 Mbps. Each servo has a resolution of 4096 pulse/rev, resulting in a high accuracy and has the PID control [12].

Each servo has four different modes: Velocity Control, Position Control, Extended Position Control, and PWM Control. PWM Control operates on a voltage spectrum, and Extended Position Control allows multiple turns instead of the 360° limit in the Position Control. This thesis utilizes the Position Control seeing as this was deemed as the more relevant control for positioning.

The physical connection between the computer and the ReactorX 150 Robot Arm is a TTL half-duplex asynchronous serial communication. Half-duplex implies that both units may read and write, but not at the same time. The serial communication packets are 8 bits, 1 start bit, and no parity.

3.5.1 Communication Protocol Details

The DYNAMIXEL servos allow two different protocols, being DYNAMIXEL Protocol 1.0 and DYNAMIXEL Protocol 2.0, where Protocol 2.0 is the recommended protocol. These protocols use instruction packets to execute tasks with the robot arm. The packets are built according to Figure 3-2. The “ID” references the servo ID, the “Length” references the length of the packets which is to ensure the entire packet is received, the “Inst” references the given instruction for the robot arm, while the “Param” is extra parameters to execute instruction which is not mandatory for all instruction [12].

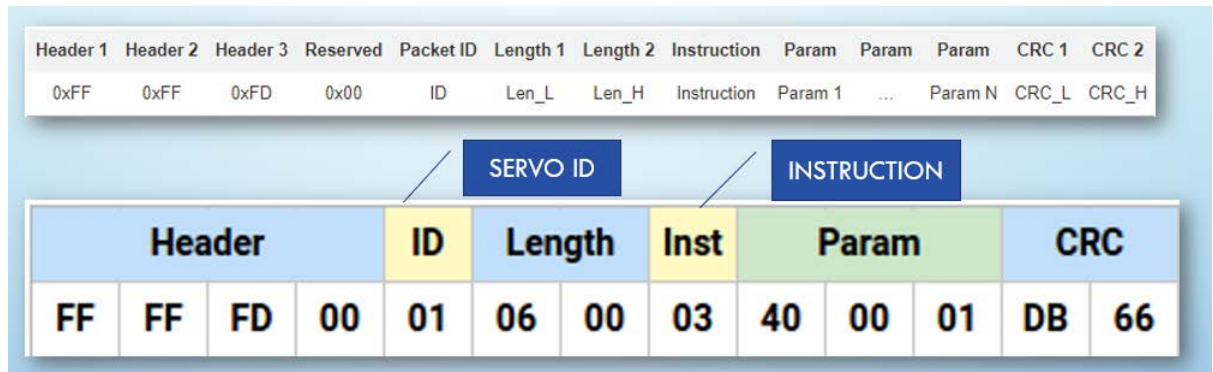


Figure 3. 2: The build-up of communication packets

In addition to sending packets, it is also possible to read the status of the sent packet to check for communication errors. An example of such a packet read is depicted in Figure 3-3.

Status packet

Header 1	Header 2	Header 3	Reserved	Packet ID	Length 1	Length 2	Instruction	ERR	PARAM	PARAM	PARAM	CRC 1	CRC 2
0xFF	0xFF	0xFD	0x00	ID	Len_L	Len_H	Instruction	Error	Param 1	...	Param N	CRC_L	CRC_H

Error Number	Error	Description
0x01	Result Fail	Failed to process the sent Instruction Packet
0x02	Instruction Error	Undefined Instruction has been used Action has been used without Reg Write
0x03	CRC Error	CRC of the sent Packet does not match
0x04	Data Range Error	Data to be written in the corresponding Address is outside the range of the minimum/maximum value
0x05	Data Length Error	Attempt to write Data that is shorter than the data length of the corresponding Address (ex: when you attempt to only use 2 bytes of a item that has been defined as 4 bytes)
0x06	Data Limit Error	Data to be written in the corresponding Address is outside of the Limit value
0x07	Access Error	Attempt to write a value in an Address that is Read Only or has not been defined Attempt to read a value in an Address that is Write Only or has not been defined Attempt to write a value in the ROM domain while in a state of Torque Enable(ROM Lock)

Figure 3. 3: The reading of a sent packet with error status [12]

There are several different instructions, as well as different instructions dependent on the control table, the servos are operating on. As previously mentioned, the servos receive their instructions as a packet. An example of such a table of instructions is given in Figure 3-4 and instruction packet flow chart between software application and robot arm illustrated in Figure 3-5.

Instruction packet

Value	Instructions	Description
0x01	Ping	Instruction that checks whether the Packet has arrived to a device with the same ID as Packet ID
0x02	Read	Instruction to read data from the Device
0x03	Write	Instruction to write data on the Device
0x04	Reg Write	Instruction that registers the Instruction Packet to a standby status; Packet is later executed through the Action command
0x05	Action	Instruction that executes the Packet that was registered beforehand using Reg Write
0x06	Factory Reset	Instruction that resets the Control Table to its initial factory default settings
0x08	Reboot	Instruction to reboot the Device
0x10	Clear	Instruction to reset certain information
0x20	Control Table Backup	Instruction to store current Control Table status data to a Backup area or to restore EEPROM data.
0x55	Status(Return)	Return packet for the Instruction Packet
0x82	Sync Read	For multiple devices, Instruction to read data from the same Address with the same length at once
0x83	Sync Write	For multiple devices, Instruction to write data on the same Address with the same length at once
0x8A	Fast Sync Read	For multiple devices, Instruction to read data from the same Address with the same length at once
0x92	Bulk Read	For multiple devices, Instruction to read data from different Addresses with different lengths at once
0x93	Bulk Write	For multiple devices, Instruction to write data on different Addresses with different lengths at once
0x9A	Fast Bulk Read	For multiple devices, Instruction to read data from different Addresses with different lengths at once

Figure 3. 4: Table with instructions, and descriptions, for the servos [12]

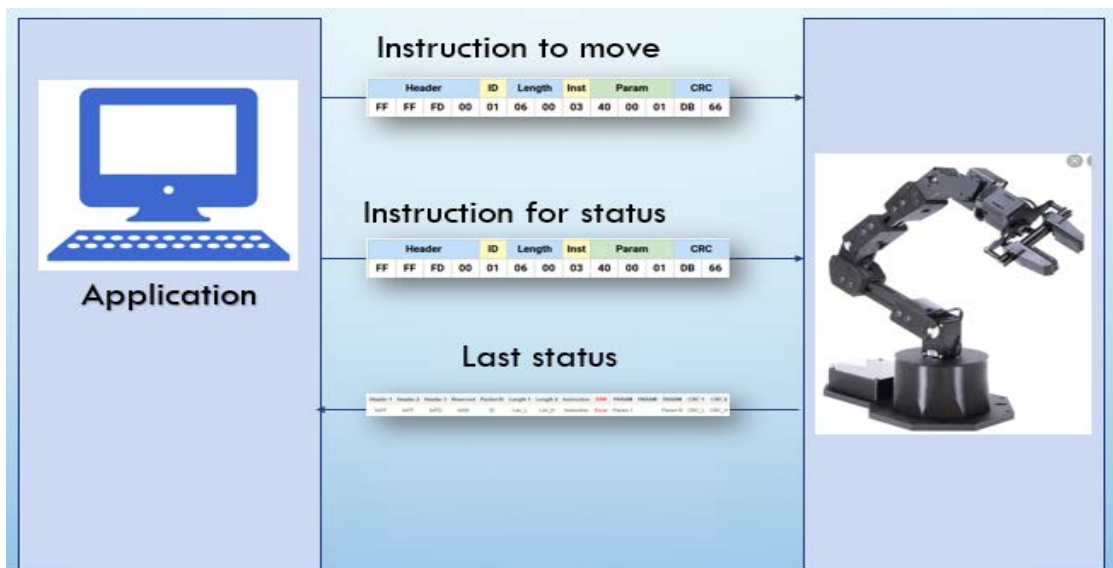


Figure 3. 5: Instruction packet flow chart between software application and robot arm

As a part of the several settings and register value available within the servos, it is also possible to set limitations or retrieve register values from the servos. These values are separated into two different tables, a EEPROM Area table and a RAM Area table. The RAM data is data which is deleted/removed on shutdown, while the EEPROM data is data which is stored with the servos. RAM data may for example, information regarding the current speed, position, if torque is enabled, and trajectory, while EEPROM is data regarding the model number, ID

number, baud rate, and limitations. Images of the start of each data table is depicted in Figure 3-6 and Figure 3-7.

2. 2. Control Table of EEPROM Area

Address	Size(Byte)	Data Name	Access	Initial Value	Range	Unit
0	2	Model Number	R	1,020	-	-
2	4	Model Information	R	-	-	-
6	1	Firmware Version	R	-	-	-
7	1	ID	RW	1	0 ~ 252	-
8	1	Baud Rate	RW	1	0 ~ 7	-
9	1	Return Delay Time	RW	250	0 ~ 254	2 [µsec]
10	1	Drive Mode	RW	0	0 ~ 5	-
11	1	Operating Mode	RW	3	0 ~ 16	-
12	1	Secondary(Shadow) ID	RW	255	0 ~ 252	-
13	1	Protocol Type	RW	2	1 ~ 2	-
20	4	Homing Offset	RW	0	-1,044,479 ~ 1,044,479	1 [pulse]

Figure 3. 6: Start of the EEPROM Area data table [12]

2. 3. Control Table of RAM Area

Address	Size(Byte)	Data Name	Access	Initial Value	Range	Unit
64	1	Torque Enable	RW	0	0 ~ 1	-
65	1	LED	RW	0	0 ~ 1	-
68	1	Status Return Level	RW	2	0 ~ 2	-
69	1	Registered Instruction	R	0	0 ~ 1	-
70	1	Hardware Error Status	R	0	-	-
76	2	Velocity I Gain	RW	1,920	0 ~ 16,383	-
78	2	Velocity P Gain	RW	100	0 ~ 16,383	-
80	2	Position D Gain	RW	0	0 ~ 16,383	-
82	2	Position I Gain	RW	0	0 ~ 16,383	-
84	2	Position P Gain	RW	800	0 ~ 16,383	-
88	2	Feedforward 2nd Gain	RW	0	0 ~ 16,383	-
90	2	Feedforward 1st Gain	RW	0	0 ~ 16,383	-
98	1	Bus Watchdog	RW	0	1 ~ 127	20 [msec]
100	2	Goal PWM	RW	-	-PWM Limit(36) ~ PWM Limit(36)	0.113 [%]

Figure 3. 7: Start of the RAM Area data table [12]

3.6 Software stack and libraries

Software components that comprise a software stack may include virtualization or abstracted hardware resources, as well as other components necessary to run an application. Stacking is the process which combines individual components to support the application to execute. In a hierarchical structure, each component consists of an architectural layer, protocols, operating system, runtime environments, databases, and function calls. Higher-level components usually perform specific tasks and services for end users in our thesis its MATLAB application while the lower-level components typically interact with hardware which is RX150 robot arm communication board. Through complex instructions traversing the stack, components communicate directly with applications. “MATLAB C” library and U2D2 software protocol been used by Dynamixel servo at the lowest level which is drawn in Figure 3-8.

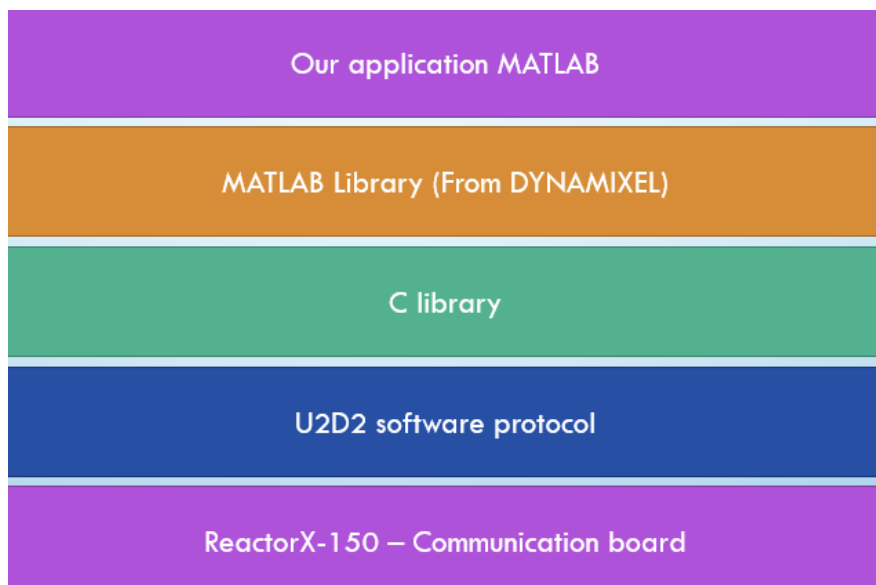


Figure 3. 8: Software stack flow

3.7 Summary

Several important steps are given as follows which are required during interfacing with RX150 manipulator arm (real hardware unit).

- Dynamixel wizard 2.0 need to be installed.
- Real unit setup, calibration, servo motor testing shall be done by using Dynamixel wizard shown in Figure 3-9, before interfacing through MATLAB.

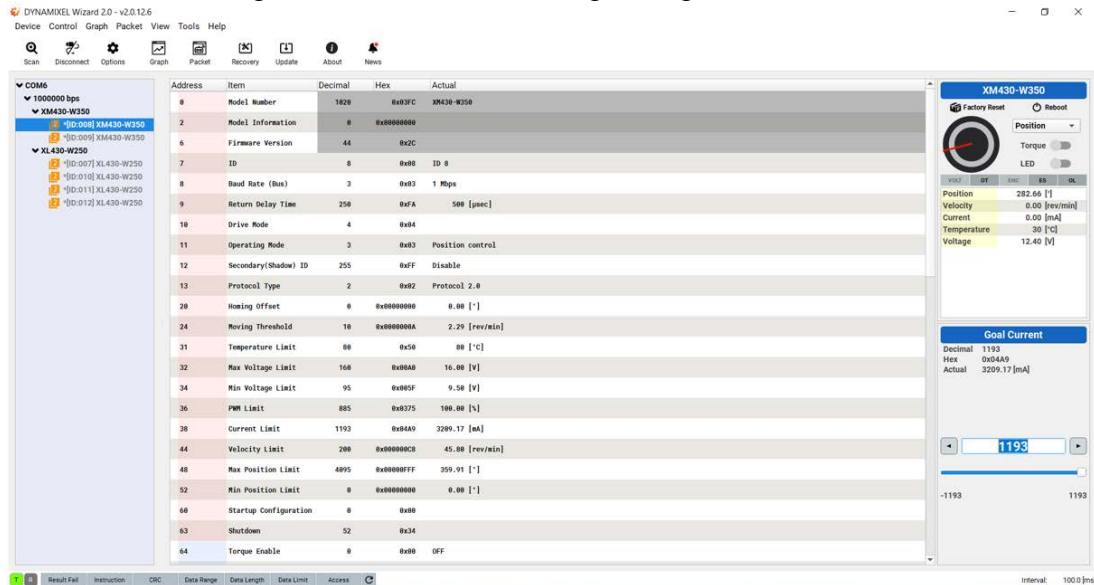


Figure 3. 9: Dynamixel wizard operating Area.

- Dynamixel servo Id, mode of control, comport for communication, profile acceleration and velocity, goal and present position address, rebooting/resetting servo motor in case of overload can be monitored and change by using this application illustrated in Figure 3-9.
- During interfacing through MATLAB, first thing needs to do is run the Hardware initialization script and check comport and all the servos are communicating with MATLAB. Figure 3-10 shows the servo motor communication result in MATLAB command window. If there is any problem in communication with any of the servo motor, then error message “There is no status packet” will pop up in command window. In that case, user have to check in Dynamixel wizard either servo ID is wrongly addressed in MATLAB script or servo motor got tripped.

Chapter 4

ReactorX 150 Robot Arm Simulation

As a start to the goal of successfully interface MATLAB with the ReactorX 150 Robot Arm, a functioning simulator must be created first. This simulator should be able to replicate the functions of the physical arm, as well as operating as a safer step seeing as it is possible avoiding physical damage. This chapter will go through the several steps needed to be able to implement the ReactorX 150 simulator with MATLAB.

4.1 Xacro to URDF in ROS2

Interbotix offers many simulations, and packages, for ROS2 and seeing as the simulations are all stored in the xacro file format, the main goal is to convert these files, and simulations, into a format usable for MATLAB.

After installing, and running, the ROS2 in Linux platform, the xacro files were converted to a Unified Robot Description Format (URDF) format. These files were given from an Interbotix [GitHub](#) repository [4]. This conversion is supported in ROS2 and is simply done with the below code line, showed in Table 4.1. The “rx150” refers to the specific robot arm which the files are based on. There are simulators for all the aforementioned robot arms, mentioned in chapter ReactorX 150 Robot Arm, and dependent on the robot arm, the name of the file must be changed [12].

Table 4. 1: ROS2 code line for file conversion[12]

```
ros2 run xacro xacro -o rx150.urdf rx150.urdf.xacro
```

Both the URDF and xacro file formats are based on the standard XML file format which makes the conversion possible. The URDF file is then ready to be imported, and used, in MATLAB with Simulink, in order to simulate the robot.

4.2 Robot Arm Calibration and Movement from Simulink

For the purpose of simulating the robot arm in a MATLAB environment, the URDF and rigid body tree models are needed. A rigidbodytreemodel is one way to represent the mechanical properties of a robot arm in MATLAB and to visualize and control robot arms in Simulink. This is the first step for utilizing the robotics toolbox in MATLAB and for visualizing and controlling the robot arms in Simulink. As shown in Figure 4-1, a rigid body tree is made up from rigid bodies that are attached to each other via joints, similar to the joints on robot arms. There are several joints found in rigid bodies that determine how these bodies move in relation to their parent bodies. According to Figure 4-1, the top-left corner shows the world coordinate frame, which is created by linking to the base link. All other links of this system are positioned

according to the world coordinate frame. Joints are the means by which two links/bodies are connected to each other. The links in joint2 are formed by both the parent link Body1 and the child link Body2. There are three main types of joints that are supported by the rigid body tree, namely fixed, revolute and prismatic joints. It is also important to note that a fixed joint is not able to move the rigid body/link. The rigid link in this case is rigidly attached to the parent body. As the name implies, the revolute joint enables a body/link to rotate around one of its axes (either the x, y, or z axis) with respect to its parent. It should be noted that when using the prismatic joint, there is only the possibility of translation (and no rotation). According to the axis of motion, the body moves linearly in relation to its parent link. There are several details that should be provided in order to describe the robot, such as the type of joints, the length of the links, the mass of the link, as well as the inertial properties of the link/body and ideally the origin position(coordinate) of the body [12].

There is a file called URDF (Universal Robot Description Format) that contains a bundle of information describing the physical properties of a robot. A 3D-model of a robot can also be included in the URDF file which allows one to describe the robot. This type of 3-dimensional model of a robot can be created by the use of computer-aided design (CAD) programs such as AutoCAD, Solidworks, TinkerCAD, etc. For 3D rendering, URDF supports the export of .stl files. Importing URDF model in Simulink

The simulation was created by running the newly created URDF file, or more specifically, using the MATLAB command “smimport ‘RX150_robot’” in the command window. Then the geometry file should thereby be created within the same path as the URDF file. Upon starting the simulation, a file for the ReactorX 150 geometry opens and executes customizations for clear visualization which then prepares the simulation [12].

The first step after importing the command is to connect each joint to their respective geometry files and settings. The block diagram of this file is shown in Figure 4-1. The block diagram is mainly based on “link” blocks, the white squares, and “revolute joint” blocks, which are a part of the simulation system. In addition, the five main joints are given their respective ID number, ranging from 1-5 where id number 5 is not used because it is not relevant for this thesis, for example, this servo is used for tightening the screw.

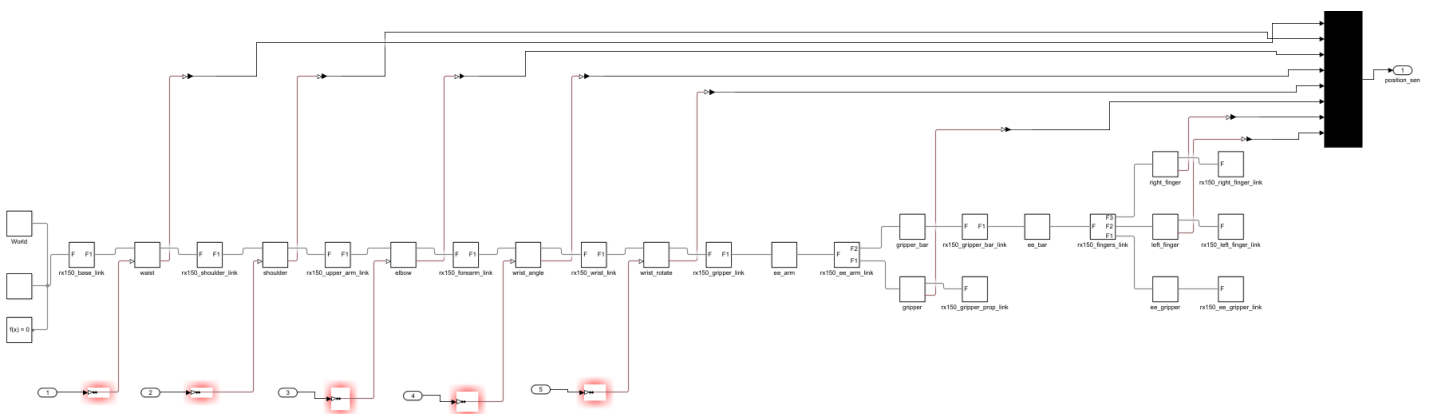


Figure 4. 1: Simulink code for the geometric link and joints for the robot arm

The file also changes the settings for the joints when the files are linked and changes the unit, which is changed from meters to millimeters, to get the clear view of entire robot arm. As an example, a visual block, for the upper arm, is displayed in Figure 4-2, as well as the change of unit for the revolute joint. This change is done for all joints.

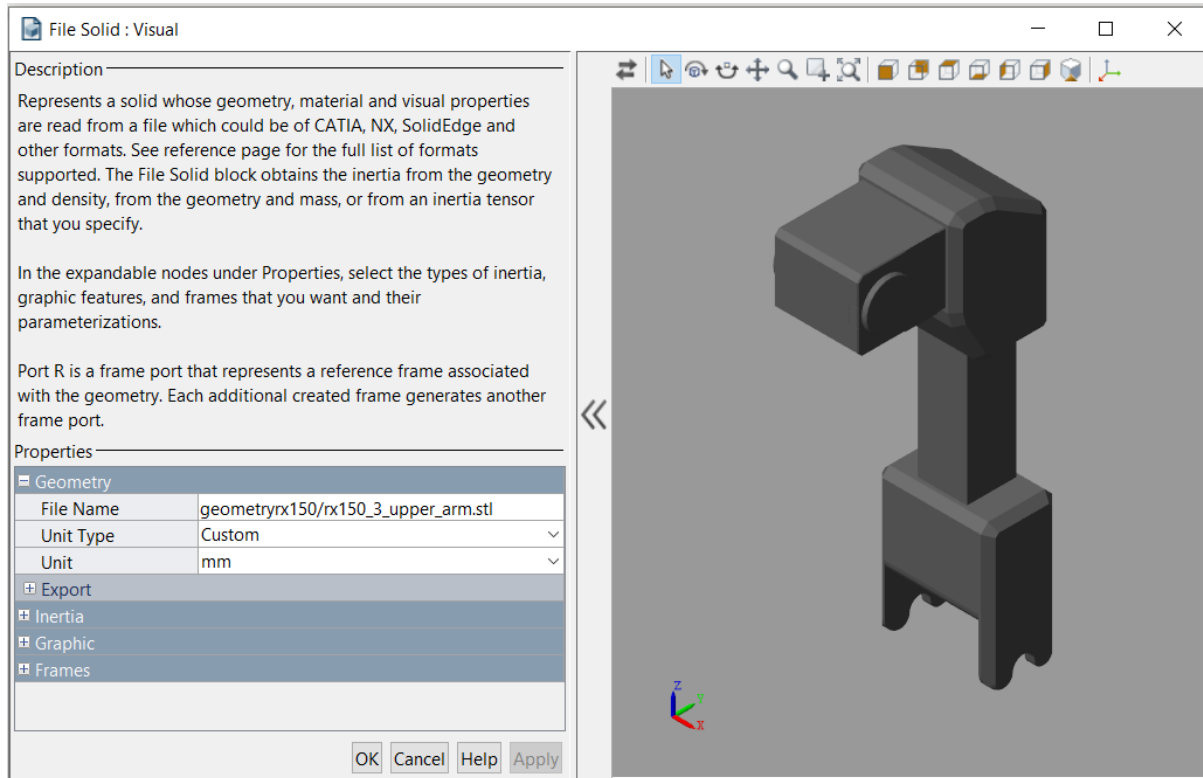


Figure 4. 2: Visual block for the upper arm of rx150

The next step is to change the actuation of all five movable joints, namely the base, waist, shoulder, elbow, and wrist joints. The actuations of these joints are set to “Automatically Computed” for torque, and “Provided by Input” for motion. This actuation setting results in inverse kinematics seeing as the resulting motion is the input, and the necessary torque is computed. The inputs for motion are given through several sliders. An example of a visual block for waist, in which this setting is changed, is displayed in Figure 4-3.

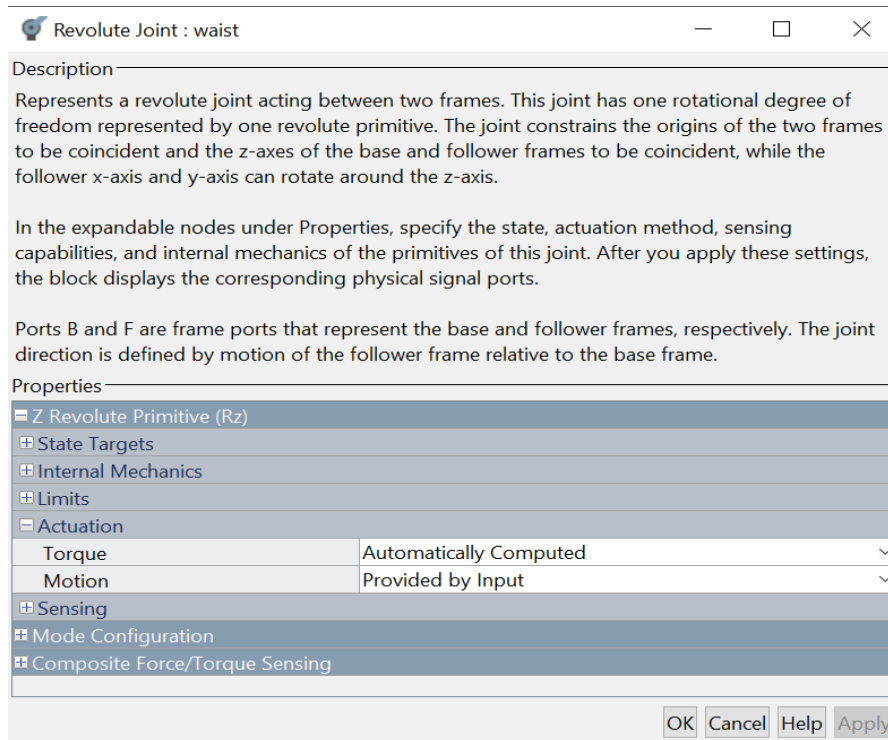


Figure 4. 3: Visual block for selection of actuation for the waist joint[12]

The third, and final, step of this customization file is to connect the aforementioned slider inputs with the simulation block named "rx150_robot_arm". This is the main block for the simulation. To connect these sliders, five inputs are provided. These inputs are based on the input selection from the previous step. Each joint angle input is connected to a slider based on degrees, and the degree signal is converted into radians before being sent to the simulator or Simulink model. In addition to this input, a MATLAB library function is used to convert the MATLAB Simulink signal into a physical signal. This converter is named "Simulink PS" shown in figure 4-1. The system is currently running as a simulator, but the program itself is created as a physical movement, thereby requiring the converter regardless. The block diagram for the simulator, with the inputs, is displayed in Figure 4-4 and robot arm simulation shown in Figure 4-5.

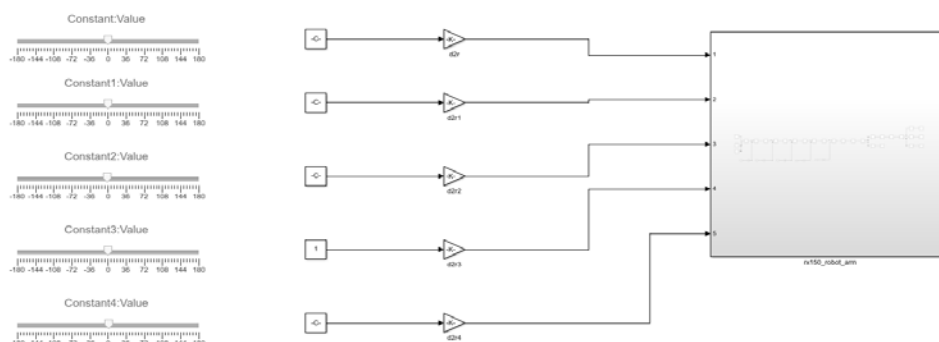


Figure 4. 4: Block diagram of the "rx150_robot_arm" with the slider inputs [12]

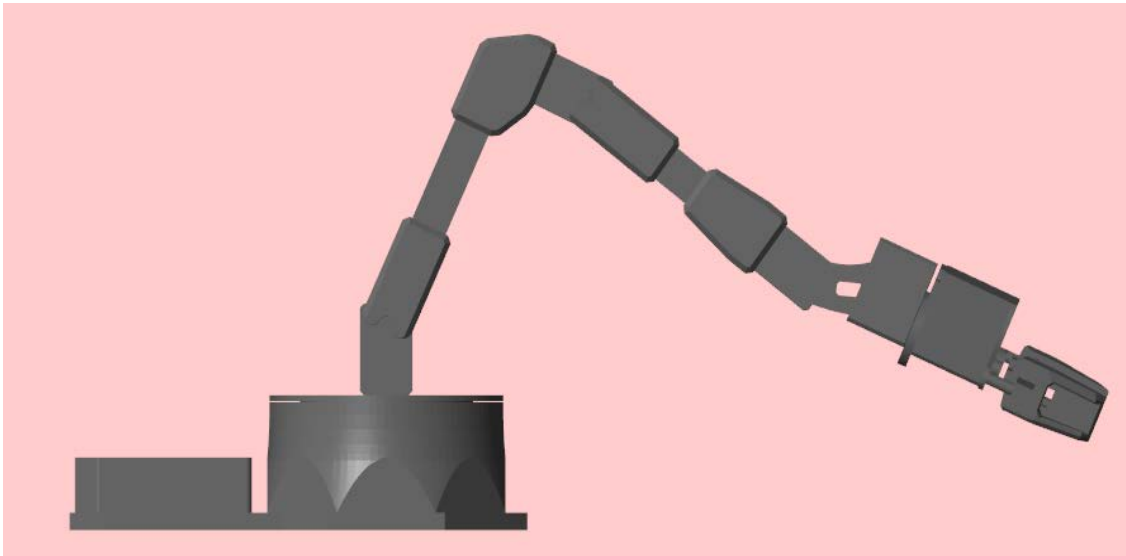


Figure 4. 5: The "rx150_robot_arm" visualization with the slider inputs

4.3 Movement and Calibration of Simulation

The implementation of the simulator was necessary both in terms of testing the movement and communication but was also necessary in order to test other functions such as trajectory tracking and inverse kinematics. The simulator offers a graphical system in which the communication and movement of the robot arm may be tested.

Figure 4. 6, shows the zero-degree position (ZDP) for the simulation arm. The ZDP is defined at 0° for joint 2, 3, 4, and 5, while joint 1, the rotation, is defined at 180° . This ZDP is being used as starting point, i.e., all movement is based on its difference from the ZDP, which is then used to calculate the current position. Seeing as the declared ZDP differs from some of the joints' 0° values, equations has been used to track the declared ZDP. These equations are given below as equation 4-1 and 4-2. Equations are made based on servo characteristic.

$$q_1 = \frac{-(2047 - \text{digitalbit})}{11.375} \quad (4.1)$$

$$q_2 = q_3 = q_4 = \frac{(2047 - \text{digitalbit}_{2,3,4})}{11.375} \quad (4.2)$$

The 180° limitation for rotation is based on physical construction of robotic arm, which is considered in the simulation. The limited ranges of joint 2 and 3 is also based on physical limitation.

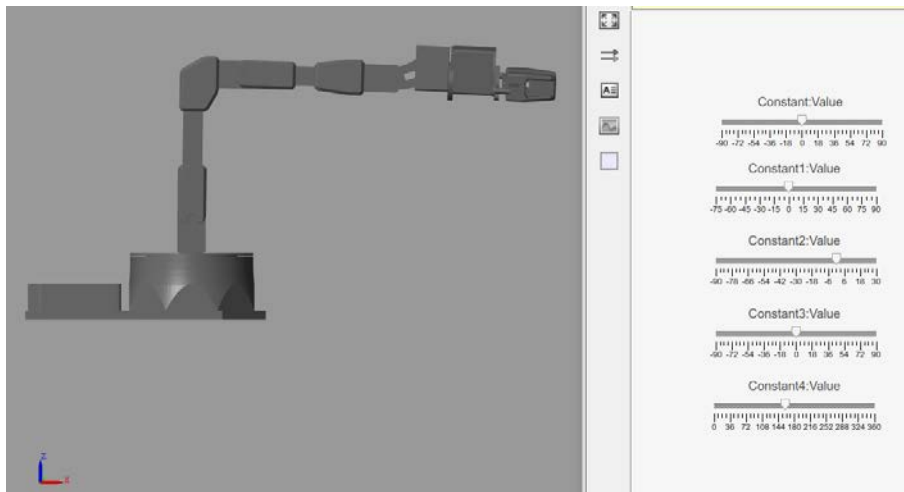


Figure 4. 6: ZDP for the simulation with slider inputs [12]

The simulator movement may be both position-based, using inverse kinematics, or slider-based, sending inputs to each joint. As an example of the slider-based movement, the simulation arm was given several angles to set, and one of these inputs are given in Figure 4.6 which is zero-degree position of robot arm means slider inputs are zero for all joints. The Figure 4-6 shows both the position of the simulation arm as well as the input angles from the sliders. The simulation is based on thumb rule of robotics which is negative degree of angle for clockwise and positive degree of angle for anti-clockwise. This is a setting given in the original URDF file. From the image, Figure 4. 7, the 2nd slider is given 90° as an input, resulting an anticlockwise movement 90° in joint 2. In joint 3 and 4, negative degree of movement is provided, resulting in clockwise movement. The plots for each of the joint values, at the random position, is given in Figure 4. 8.

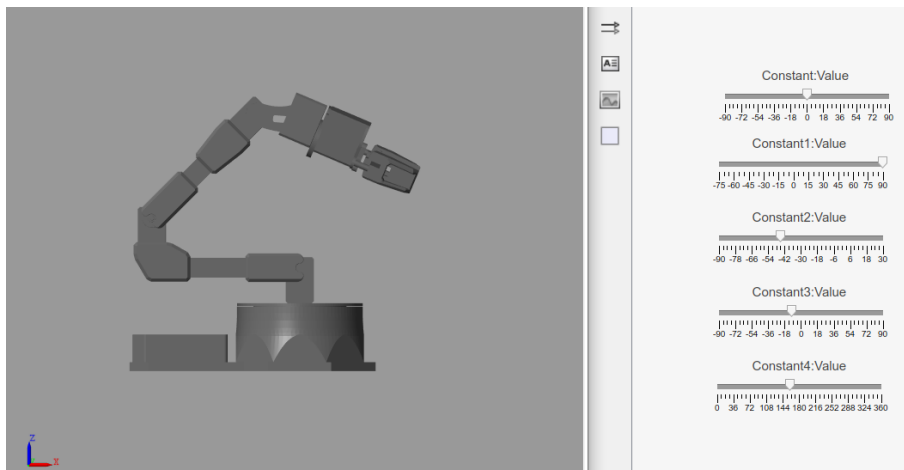


Figure 4. 7: Random position with slider inputs [12]

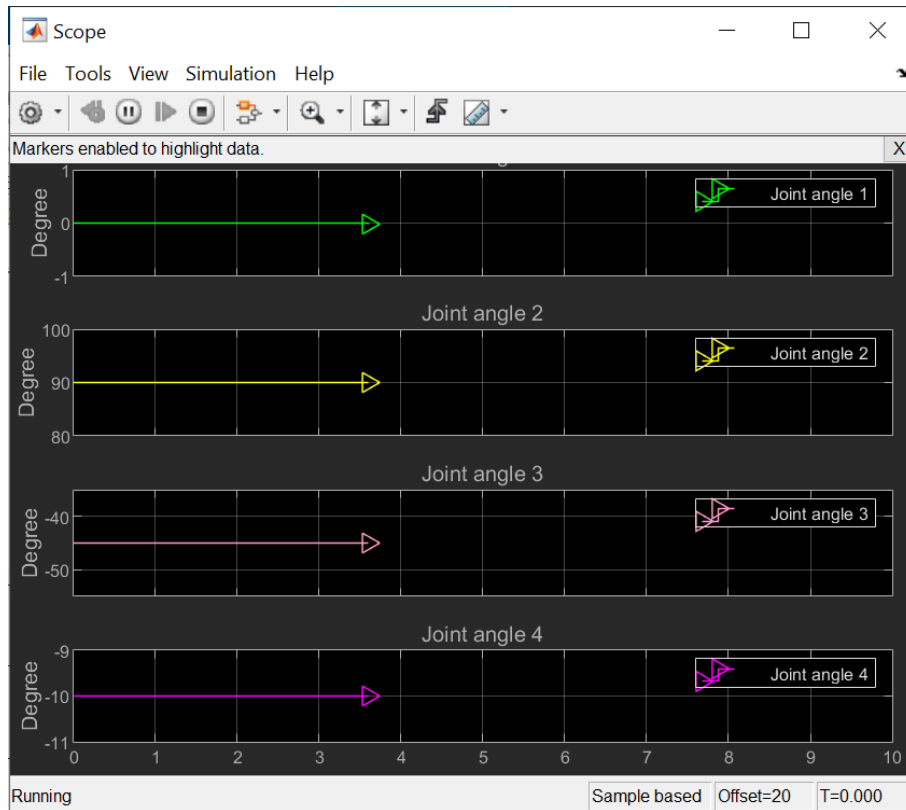


Figure 4. 8: The angel plot for the joints at the random position [12]

An example of using position-based movement in the simulation is given in Figure 4.9. By using inverse kinematics, as documented in chapter Inverse Kinematics, the chosen position is used to calculate the necessary joint angles. As a part of these calculations, the trajectory may be plotted together with the simulation viewer. This plotting is displayed in Figure 4.9. The movement is documented with red points, while the trajectory coordinates are displayed as blue circles. It is therefore also possible to see the most efficient line of movement by watching the red line between two coordinates.

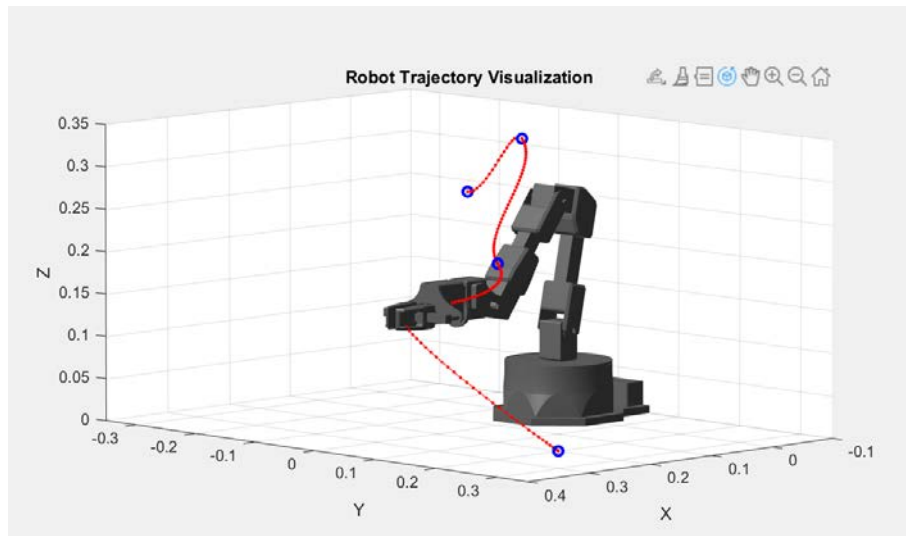


Figure 4. 9: The simulation of the RX 150 Robot Arm with trajectory based on inputs [12]

4.4 Trajectory Visualization with Inverse Kinematics in MATLAB Simulink

This sub chapter will focus of the block diagram of the inverse kinematic simulation, describing the general flow and purpose of the several blocks used for the simulation.

As a result of this, we have defined the trajectory (collection of the end-effector positions in 3D space) as well as the velocity of the waypoints. The robot arm has to be made to follow this trajectory using inverse kinematics, which in its simplest form asks what angle of joint must be applied to reach the position at the end-effector given the end-effector position. The inverse kinematic problem for the RX150 robot arm will be formulated and solved using Simulink in order to determine and solve the problem.

The first three “constant block” from the left side in Figure 4-10, represents the three inputs: waypoint, velocity, and time. Waypoint contains the coordinates of trajectory for the robot arm, velocity being the speed of travelling from one coordinate to next coordinate of trajectory, and time being the timeframe of the entire motion.

These inputs are sent to a ‘Polynomial Trajectory’ block which gives a matrix output of the given inputs, which is then transformed before being used as the basis for the necessary kinematic calculations, which in this case the inverse kinematics. The calculated joint angles are then given to robot simulation function block based on the initial position feedback taken from robot model and also stored in MATLAB workspace as variable name V as q1, X as q2, Y as q3 and Z as q4 shown in Figure 4-12, in order to use later to test forward and inverse kinematic problem with real hardware unit.

Some minor actions are taken to avoid algebraic looping, and to remove delay, before the final values are sent to the simulator, which then executes the movements. After the movement in

the simulator, the current position is returned to calculate error/deviation, thereby creating a feedback loop to account for any errors.

The final block is the visualization in Simulink, i.e., the function which makes the movements, as well as the simulation, graphically visible. This process is displayed in the block diagram shown in Figure 4-10. The inverse kinematics also makes it possible to track the movement of the simulation arm. The “visualizationRobot.m” file is also needed to visualize the robot arm movement, which follows the trajectory. According to the given waypoints and cubic polynomial trajectory formulation inverse kinematic solution has been displayed in 3-dimensional plot as in Figure 4-11.

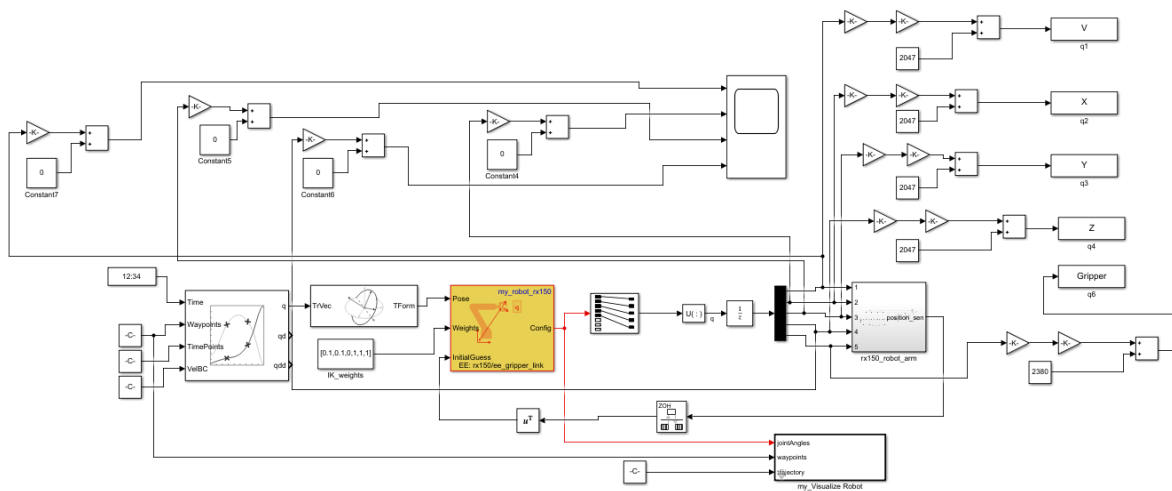


Figure 4. 10: Inverse kinematic simulation block diagram of the RX150 robot arm

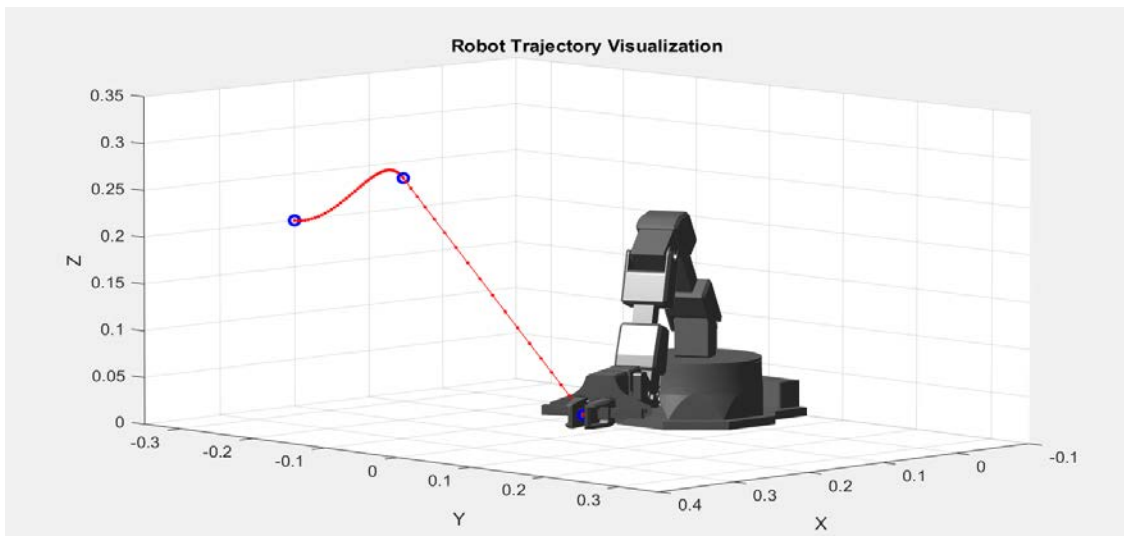


Figure 4. 11: RX150 robot arm simulation result

4.5 Inverse kinematics output stored in MATLAB workspace

Inverse kinematic output for defined trajectory stored in MATLAB workspace by using a library block called 'To Workspace' shown in Figure 4-12. The purpose to use this library block is to use the joint angles during real hardware interfacing.

In addition, these joint angles can be used in forward kinematic equation as timeseries data to move the robot in desired trajectory.

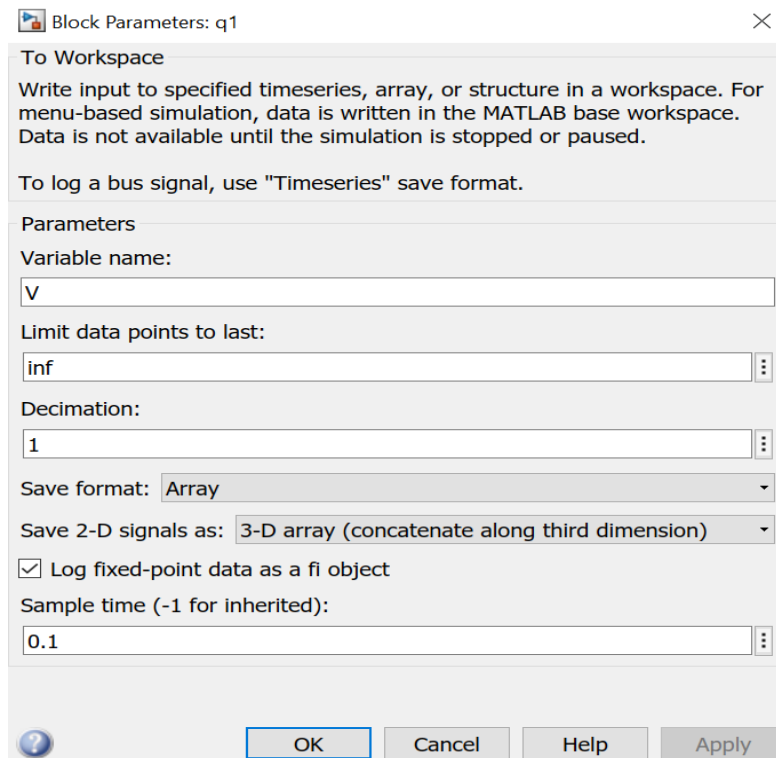


Figure 4. 12: RX150 robot arm joint angles result saved in workspace

4.6 Summary

This chapter represented the calibration and setup of Simscape model to make the RX150 robot arm ready for simulation environment. Robot arm manipulator movement by using slider was also a part of simulation. In addition, most important part of the simulation was trajectory following which has been done by using inverse kinematic engine from Simulink library. The main purpose of this chapter is to avail the simulation environment ready and tested the robot model before interfacing with real hardware unit.

Chapter 5

Mathematical Modelling , Kinematic And Optimization Analysis Using Denavit Hartenberg representation

5.1 Overview

In various fields of the recent technology and trend, the conventional solution approach of kinematics is pivotal, this section extends varying from computer graphics (e.g. analysis of animation characters) to further expansion of space manipulation and simulation. As a result, all these applications are fundamentally based on evaluating both the position and orientation of the Cartesian coordinates of the end effectors of robot manipulators as well as the joint variables of the robot. A homogeneous transformation matrix method can be used for evaluating the position and orientation of end effectors and their joint variables. There are two main types of methods for describing kinematic relations between joints and for describing the motion of links.

5.2 Representation methods and kinematics

It is possible to understand kinematics with the use of a chain or links connected to joints for the creation of relative motion, without studying the torques, forces, or mechanisms by which the motion occurs. The kinematic system of a robot can be understood as the motion of the robot link with respect to a single fixed coordinate system or base coordinate system with time as it is perceived [10]. It is also possible to study the higher derivatives of the kinematics of robot link, such as velocity, jerk, acceleration, etc.

5.3 Kinematic variables and parameters

Usually, a kinematic chain is composed of a pair of links, perhaps linked by revolute or prismatic joints with rotating or translating degrees of freedom. As described in the literature, there are numerous approaches for representing kinematic chains mathematically. There are major differences in how coordinate frames are attached to these approaches. For this reason, Denavit-Hartenberg parameters are commonly employed. For the placement of coordinate frames to links and joint variables, homogeneous transformation matrices are more appropriate [11]. There are four parameters in the method which are known as DH parameters of a kinematic chain. A link's geometry and the joint's relative displacement are defined by these scalars. As a result of this representational method, the kinematic description is simplified, and mathematical/arithmetical operations are reduced. Using the Figure 5.1 as an example, it's possible to determine the position and orientation of the joint axis relative to the base coordinates X, Y, Z with a minimum of four parameters. As a result, the magnitude of the length (a) that is located from the origin of the frame to the point of another joint at an

offset distance (d). θ represents the angle formed by OA , which is parallel to x -axis, and OP . α represents the angle formed by PQ , as shown in the diagram, this angle represents the rotation about the z axis, which is measured in terms of the x -axis. A joint axis can rotate by a certain angle if the axis is parallel to the z -axis and measured in the z -direction. The four scalars a , α , θ , and d represent Denavit Hartenberg parameters which are used to represent the position of the axis of joints in Cartesian coordinate systems. There is a more detailed discussion of these four parameters in the following section.

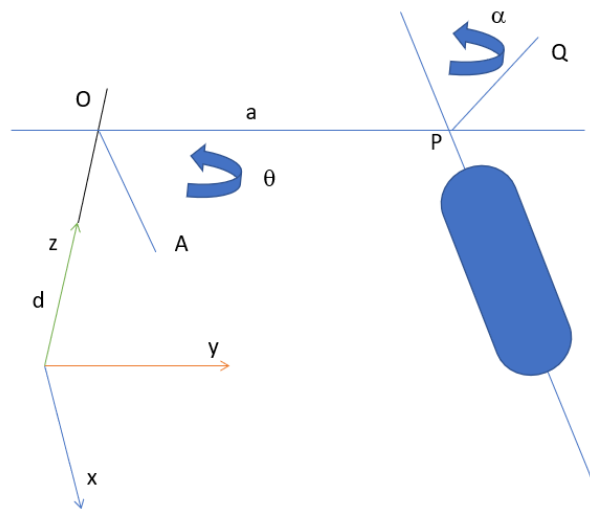


Figure 5. 1: Position and direction of a cylindrical joint in a coordinate frame

The Denavit-Hartenberg (DH) representation is a better choice when dealing with complex robotic systems, including robot arms with offsets. In industrial robot arm configurations, the DH representation is widely used and is regarded as a universal method of identifying robotic arm manipulators. For robot arm configurations with offsets, the DH representation is very useful. Using the DH representation, it is possible to calculate the kinematics of complex robotic systems with only four basic parameters. There are two displacement parameters and two rotation parameters in this list. As a result of the convention of DH representation, counterclockwise (CCW) rotation is positive, and clockwise (CW) rotation is negative.

5.3.1 DH-Parameters

From Table 5.1, DH- parameters can be defined as follows with the geometry and orientation of associated links considered:

Table 5. 1: DH parameters

Parameter Name	Description
θ_i	Rotation of the joint, which is measured as the angle at which links x_{i-1} and x_i rotate about the z_{i-1} .
a_i	It can be defined as the length of the common normal between links $i+1$ and i , measured in the direction of x_i , i.e. from axis i to $i+1$.
d_i	This parameter describes the difference between the common normal and the coordinate x_{i-1} or the distance from the origin of coordinate frame to the positions of i a in direction of z_{i-1} .
α_i	This parameter, also known as the twist angle parameter, is defined as the angle of inclination between the axes of the links measured in x_i direction and the angle from z_{i-1} to z_i .

These parameters describe the complete geometry of kinematic pair, if the joint is revolute then θ_i , d_i will be only variables and rest of the parameters will be constant while in case of prismatic joint d_i will be the variable and similarly other parameters will be constant.

5.3.2 DH-algorithm for frame assignment

According to the DH algorithm, the references of x_0, y_0, z_0 coordinate are attached to non-moving links of the base coordinate frame and the local coordinate frames are attached to the joints of the moving links. In this case, $i-1$ to i are the connecting links where $i=1,2,3\dots n$. Hence, the basic steps required in order to assign frames using the DH-algorithm are as follows.

When it comes to the coordinate frames, there are three rules that have to be followed. For each of the three coordinate frames, the following rules apply:

1. To determine the joint axis, the z-axis must be chosen in the same direction as the joint axis. When a joint is revolving, the axis of rotation is the joint axis, and this is the z-axis as well. In a prismatic joint, the joint axis is the axis along which the movement takes place.
2. It is also recommended that, the right-hand rule for the y-axis. The thumb is responsible for the x-axis, the index finger is responsible for the y-axis, and the middle finger is responsible for the z-axis.
3. In order to satisfy rule number 3 the x_i axis must be interested in the z_{i-1} axis.

Step 1 Draw the kinematic diagram of desired robot arm manipulator. The base frame is typically x_0, y_0, z_0 attached to the fixed body at the origin in such a way that the axis of rotation aligns with the z_0 axis, while x_0 will also be placed arbitrarily depending on the

direction in which the manipulator is moving towards the perpendicular rotation axis or with the direction in which the manipulator is moving forward. The last axis y_0 can be placed by using the right-hand coordinate rule which is $y_0 = z_0 * x_0$.

Step 2 The x_i axis shall be chosen such that it is perpendicular to both the z_i and z_{i-1} axes.

Step 3 Establish the coordinate system for the end-effector. Coordinate frames for the end-effector should be identical to those for the previous frame, i.e. the immediately preceding coordinate frame should be duplicated on the end-effector.

Step 4 Creating a table with DH parameter a_i, α_i, θ_i and d_i where

a_i = Displacement between two coordinate frames along x_i axis.

d_i = Displacement between two coordinate frames along z_{i-1} axis.

α_i = Rotation around x_i axis that is required to rotate z_{i-1} axis to get it match z_i axis.

θ_i = rotation around z_{i-1} axis. θ_i is a joint variable when the joint is revolute.

Step 5 Substituting the values for $i = 1, 2, \dots, n$ into A_i , a homogeneous transformation matrix shall be created.

$$A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5-1)$$

Step 6 In order to evaluate the homogeneous transformation matrix for the position and orientation of the end-effector, simply multiply all the individual A_i .

The homogeneous transformation matrix (H) as,

$$H = A_1 \cdot A_2 \cdot A_3 \cdot A_4 \quad (5-2)$$

For the forward kinematics, the x-coordinate, y-coordinate, and the z-coordinate of the end-effector position can be found from the first three rows of the last column of H respectively.

5.3.3 DH table representation of rx150 robot arm with offset

This paper discusses a mathematical modeling of forward and inverse kinematics of a robot manipulator using the homogeneous transformation matrix method with DH parameters.

With respect to the base coordinates X, Y, and Z, the position and orientation of the joint axis can be determined with minimum four parameters. This can be accomplished by drawing the joint axis and the Z axis of the base frame. As a result, the magnitude of the common normal represents the length l , which is represented as the offset distance d from the origin of Z axis. Here, θ_i represents a parallel angle to the x-axis. Angles represent rotations about z-axis and are measured on the x-axis. In the diagram below, angle θ_i represents the rotation of the axis of the joint which is parallel to z-axis and measured in direction of z-axis.

The first thing that should be done is to establish the coordinate frame.

In Figure 5-2, the coordinate frames are overlaid with the frame of the i joint, allowing the distance or offset distance d_i to be described, and the Joint Rotation parameter q_i , which corresponds to the rotation angle between links i and $i-1$ as measured from x_{i-1} to x_i about the z_{i-1} . In addition to the definition of DH parameters, the mathematical expression of the coordinate frames and the imposed frame can also be given by the homogeneous transformation matrix that is an iterative product of the homogeneous transformation matrices describing the DH parameters.

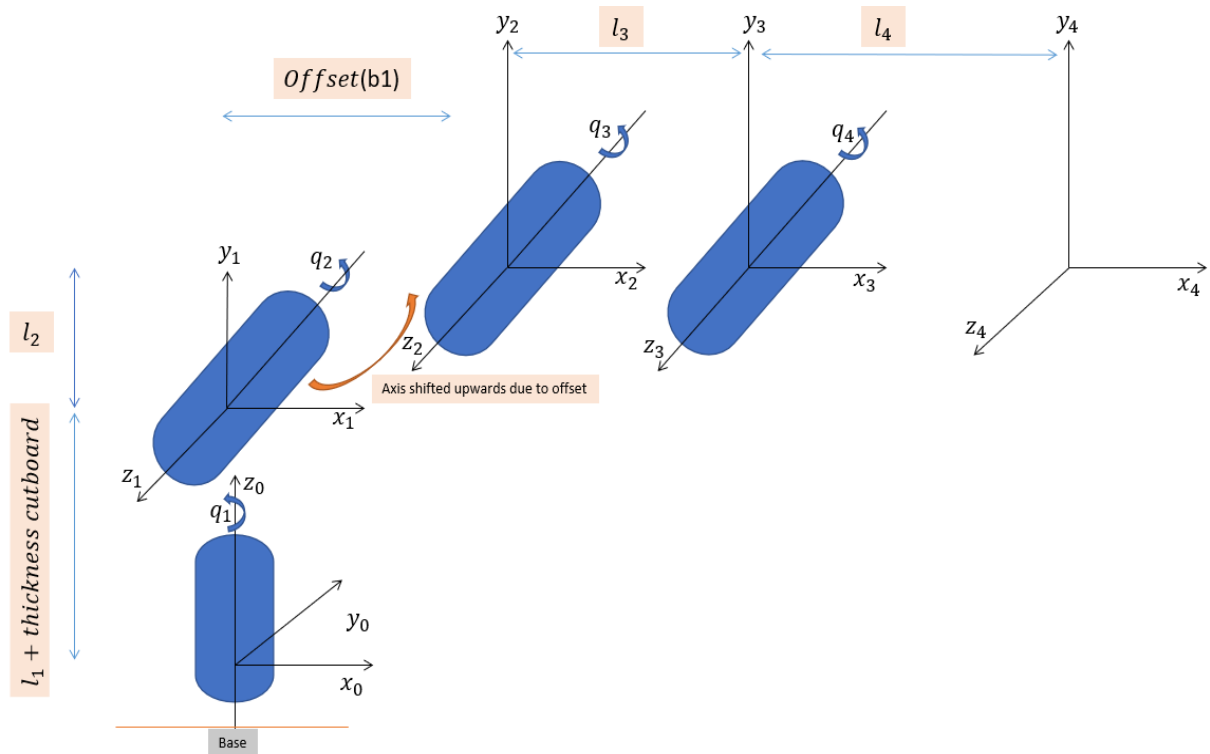


Figure 5. 2: DH table coordinate frame of RX-150 robot arm configuration with an offset.

DH parameters and Homogeneous transformation matrix can then be evaluated, and then written in a table form shown in Table 5-2, in this section, once all coordinate frames for all links $i = 1, 2, 3, \dots, n$ have been assigned.

Table 5. 2: DH-parameters table for 4-dof revolute manipulator with offset

Link of joint	a_i	d_i	α_i	θ_i
$i = 1$	0	$l_1 + l_2$	90°	q_1
$i = 2$	$b_1(Offset)$	0	0	q_2
$i = 3$	l_3	0	0	q_3
$i = 4$	l_4	0	0	q_4

$$A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} \cos q_1 & -\sin q_1 \cos 90^\circ & \sin q_1 \sin 90^\circ & 0 \\ \sin q_1 & \cos q_1 \cos 90^\circ & -\cos q_1 \sin 90^\circ & 0 \\ 0 & \sin 90^\circ & \cos 90^\circ & l_1 + l_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos q_1 & 0 & \sin q_1 & 0 \\ \sin q_1 & 0 & -\cos q_1 & 0 \\ 0 & 1 & 0 & l_1 + l_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5-3)$$

$$A_2 = \begin{bmatrix} \cos q_2 & -\sin q_2 \cos 0^\circ & \sin q_2 \sin 0^\circ & b_1 \cos q_2 \\ \sin q_2 & \cos q_2 \cos 0^\circ & -\cos q_2 \sin 0^\circ & b_1 \sin q_2 \\ 0 & \sin 0^\circ & \cos 0^\circ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos q_2 & -\sin q_2 & 0 & b_1 \cos q_2 \\ \sin q_2 & \cos q_2 & 0 & b_1 \sin q_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5-4)$$

$$A_3 = \begin{bmatrix} \cos q_3 & -\sin q_3 \cos 0^\circ & \sin q_3 \sin 0^\circ & l_3 \cos q_3 \\ \sin q_3 & \cos q_3 \cos 0^\circ & -\cos q_3 \sin 0^\circ & l_3 \sin q_3 \\ 0 & \sin 0^\circ & \cos 0^\circ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos q_3 & -\sin q_3 & 0 & l_3 \cos q_3 \\ \sin q_3 & \cos q_3 & 0 & l_3 \sin q_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5-5)$$

$$A_4 = \begin{bmatrix} \cos q_4 & -\sin q_4 \cos 0^\circ & \sin q_4 \sin 0^\circ & l_4 \cos q_4 \\ \sin q_4 & \cos q_4 \cos 0^\circ & -\cos q_4 \sin 0^\circ & l_4 \sin q_4 \\ 0 & \sin 0^\circ & \cos 0^\circ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos q_4 & -\sin q_4 & 0 & l_4 \cos q_4 \\ \sin q_4 & \cos q_4 & 0 & l_4 \sin q_4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5-6)$$

The homogeneous transformation matrix (H) as,

$$H = A_1 \cdot A_2 \cdot A_3 \cdot A_4$$

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5-7)$$

End effector position X_e equation as,

$$h_{14} = x_e = (\cos(q1)*\cos(q2))/20 + (3*\cos(q4)*(cos(q1)*\cos(q2)*\cos(q3) - \cos(q1)*\sin(q2)*\sin(q3)))/20 - (3*\sin(q4)*(cos(q1)*\cos(q2)*\sin(q3) + \cos(q1)*\cos(q3)*\sin(q2)))/20 + (3*\cos(q1)*\cos(q2)*\cos(q3))/20 - (3*\cos(q1)*\sin(q2)*\sin(q3))/20 \quad (5-8)$$

End effector position Y_e equation as,

$$h_{24} = y_e = (\cos(q2)*\sin(q1))/20 - (3*\cos(q4)*(sin(q1)*\sin(q2)*\sin(q3) - \cos(q2)*\cos(q3)*\sin(q1)))/20 - (3*\sin(q4)*(cos(q2)*\sin(q1)*\sin(q3) + \cos(q3)*\sin(q1)*\sin(q2)))/20 - (3*\sin(q1)*\sin(q2)*\sin(q3))/20 + (3*\cos(q2)*\cos(q3)*\sin(q1))/20 \quad (5-9)$$

End effector position Z_e equation as,

$$h_{34} = z_e = \sin(q2)/20 + (3*\cos(q2)*\sin(q3))/20 + (3*\cos(q3)*\sin(q2))/20 + (3*\cos(q4)*(cos(q2)*\sin(q3) + \cos(q3)*\sin(q2)))/20 + (3*\sin(q4)*(cos(q2)*\cos(q3) - \sin(q2)*\sin(q3)))/20 + 1/4 \quad (5-10)$$

Using the homogeneous transformation matrix method with DH parameters, a mathematical model for end effector position of robot manipulator is presented. The objective of this application is to introduce users to kinematics, including both open and closed kinematic chains. A solution to the Inverse Kinematics problem is the opposite to that of the Forward Kinematics, in that the forward kinematics provides a single solution, whereas the inverse kinematics provides multiple solutions [10]. As a result of the addition of joint variables, the end effector or tool piece can assume a particular pose. The figure 5-2 illustrates the basic joint configuration of a revolute planar manipulator with 4 DOF and Table 5-2 represents the control parameters of the RX150 robot kinematic model. Coordinates of a joint can be used to specify the position and orientation of an end effector.

5.3.4 Code snippet to calculate end effector coordinate by using DH table

```
%Length of arms in meter
l1 = 0.140; %0.11791;%0.13891; % distance between joint 1 and 2
l2 = 0.110; %0.133; % distance between joint 2 and 3 in y direction
b1 = 0.05; % offset
l3 = 0.150;%0.150; %0.153; % distance between joint 3 and 4
l4 = 0.150;%0.147575; distance between joint 4 and end effector
%Chosen position in radian
q1 = 0;
q2 = 0;
q3 = 0;
q4 = 0;
syms q1 q2 q3 q4
%DH representation for 4 joint with offset
A1 = [cos(q1) 0 sin(q1) 0; sin(q1) 0 -cos(q1) 0; 0 1 0 l1+l2; 0 0 0 1];
A2 = [cos(q2) -sin(q2) 0 (b1)*cos(q2); sin(q2) cos(q2) 0 (b1)*sin(q2); 0 0 1 0; 0 0 0 1];
A3 = [cos(q3) -sin(q3) 0 l3*cos(q3); sin(q3) cos(q3) 0 l3*sin(q3); 0 0 1 0; 0 0 0 1];
A4 = [cos(q4) -sin(q4) 0 l4*cos(q4); sin(q4) cos(q4) 0 l4*sin(q4); 0 0 1 0; 0 0 0 1];
%Homogeneous transfer function
H_symb = A1 * A2 * A3 * A4;
%End effector coordinate point
xe = H_symb(1,end);
ye = H_symb(2,end);
ze = H_symb(3,end);

% In order to get symbolic expression
xe_subs = subs(xe,[q1 q2 q3 q4]);
ye_subs = subs(ye,[q1 q2 q3 q4]);
ze_subs = subs(ze,[q1 q2 q3 q4]);
```


5.4 Forward kinematic result with DH representation

For the known values of q_1, q_2, q_3, q_4 , we can use DH model equation to derive end effector position in 3D coordinate frame. To accomplish that, we need three mathematical model for x_e, y_e, z_e end effector position where we will supply joint angles. After providing the known joint angles in the three equations 5-8, 5-9, 5-10 it will give us the end effector point according to our forward kinematic rules which shows in Figure 5-3.

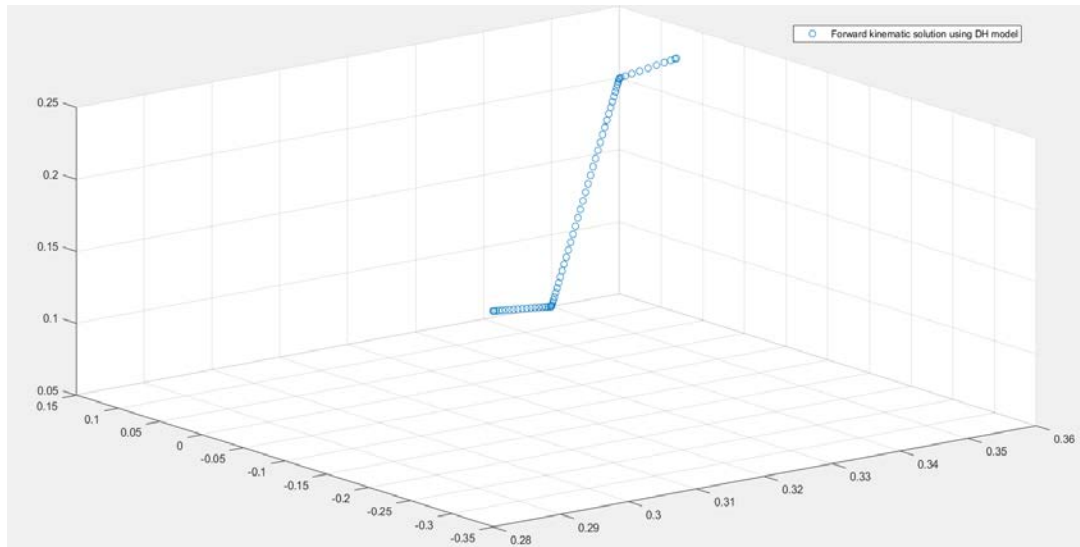


Figure 5. 3: Forward kinematic result by using DH model.

5.5 Inverse kinematic result using Simulink library function (IK engine)

Figure 5-4 illustrated the simulation block solving inverse kinematic by using inverse kinematic library function to make the robot following the trajectory. The purpose of this simulation is to implement it on real hardware unit and to compare the robot arm movement with inverse kinematic result by optimization technique. Figure 5-5 shows the robot arm, tracking the trajectory in 3D planar. Inverse kinematic solution stored in blocks called “To workspace” by the name of ‘V’, ‘X’, ‘Y’, ‘Z’ and ‘Gripper’ as ‘q1’, ‘q2’, ‘q3’ and ‘q4’ joint angles respectively which has also been discussed in detail in chapter 4.

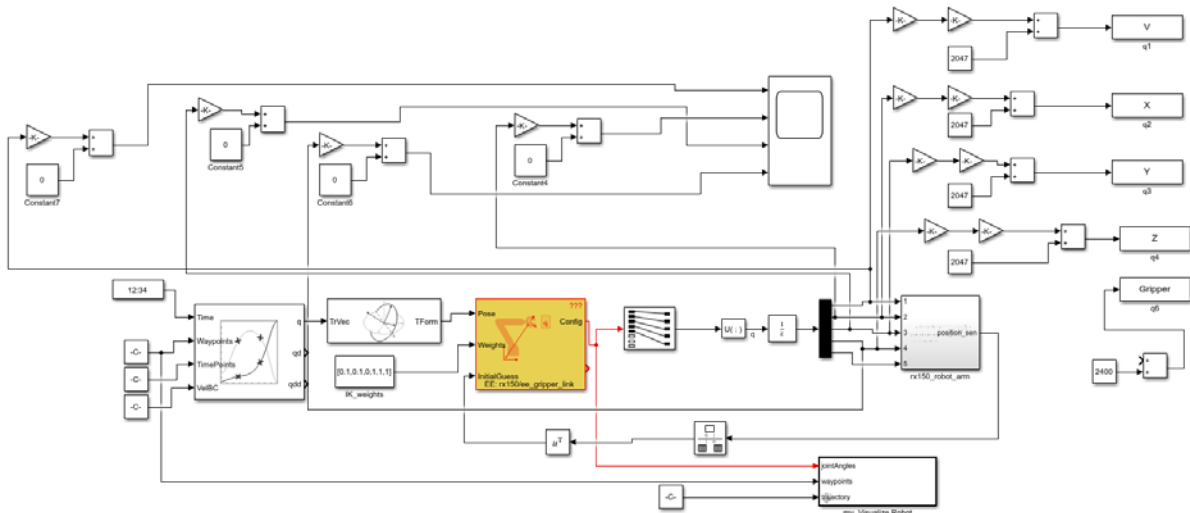


Figure 5. 4: Inverse kinematic simulation block

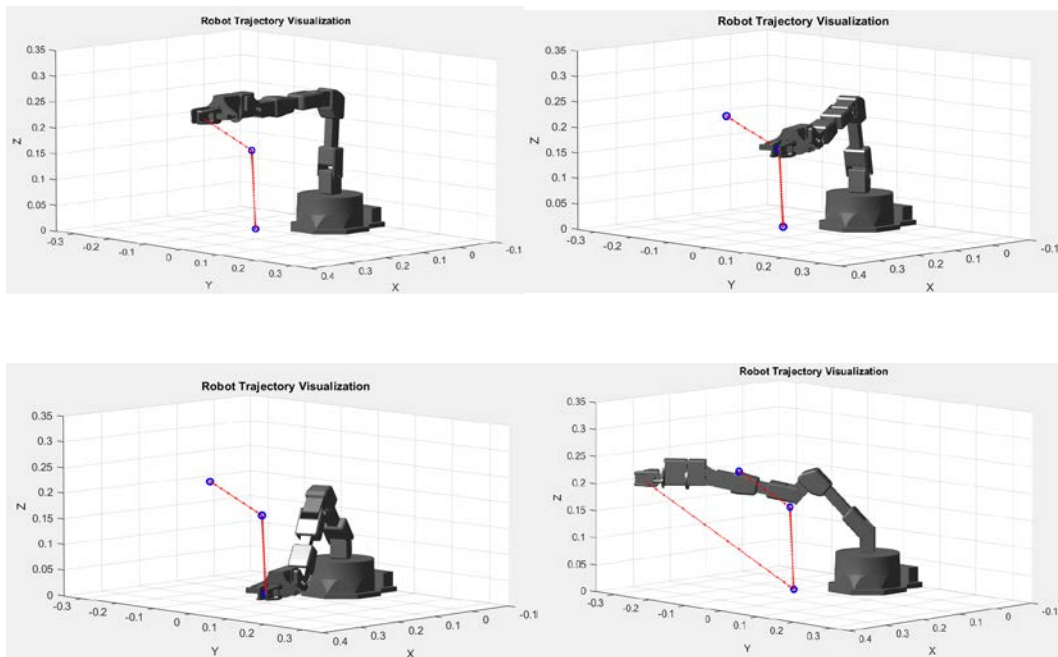


Figure 5. 5: Inverse kinematic simulation result

5.5.1 Real arm movement picture frame and video using Simulink (IK Library)

Joint angles value is picked from workspace variable to implement in real hardware unit. Joint angles implementation in real hardware has been done in two options. One is from Simulink as shown in Figure 5-6 and by writing codes in MATLAB editor script.

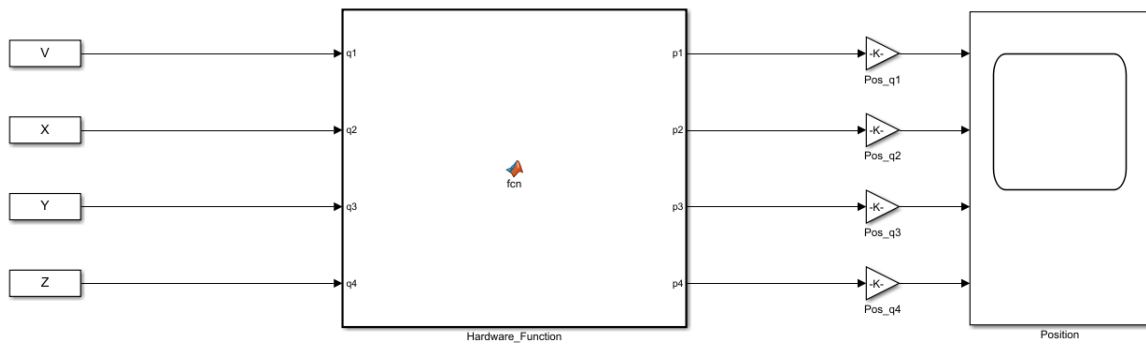


Figure 5. 6: Inverse kinematic result implemented on RX150 robot arm

Real hardware unit movement following the trajectory shown in Figure 5-7. Here, top left position is number 1, top right position is number 2, bottom left is number 3 and bottom right is number 4, same sequence followed in Figure 5-5.

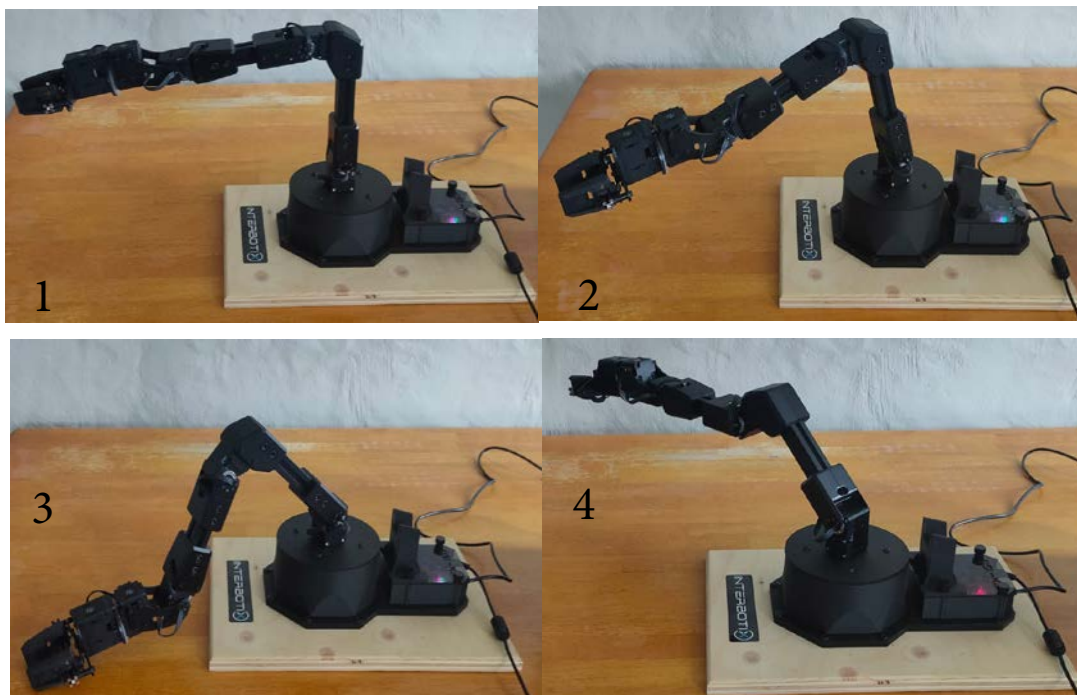


Figure 5. 7: Inverse kinematic simulation result implemented on RX150 robot arm

YouTube video link provided as follows and Figure 5-8 can be clicked to see robot arm movement using inverse kinematic (Simulink library) in YouTube.

<https://youtu.be/YrbDmKQGE MY>



Figure 5. 8: Video link for robot arm movement using inverse kinematic (Simulink library)

5.6 Inverse kinematic result using DH model and optimization in simulation

Inverse kinematic problem can also be solved by using optimization technique where we do not need to depend on IK library from Simulink. Detailed optimization technique and formula applied in this thesis has been demonstrated in method and material chapter. In this optimization method we tried to minimize the current and desired position error by applying the DH mathematical model as equations 5-8, 5-9, and 5-10 and optimization objective function as equation 2-8. At first, inverse kinematic solution from optimization formula implemented in Simscape model (simulation environment) shown in Figure 5-9, to make sure that real hardware unit movement is safe and also to avoid hardware damage.

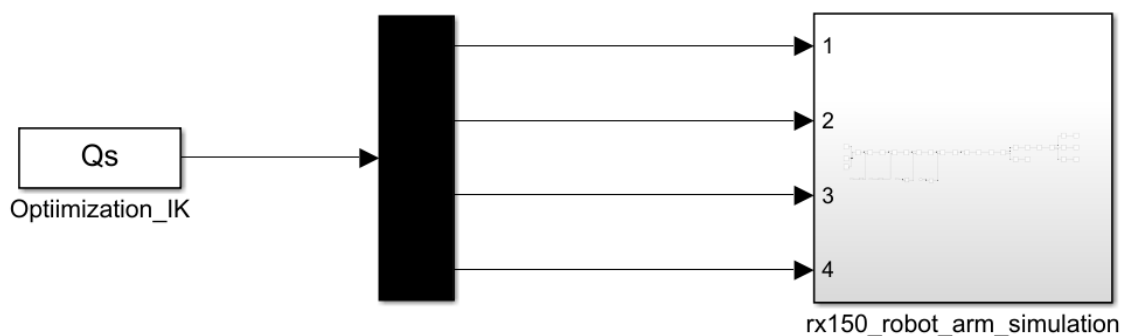


Figure 5. 9: Inverse kinematic simulation using optimization

After running the simulation model, we compare the end effector position with desired or original trajectory shown in Figure 5-10 where it shows some mismatch in the end effector

position which is not pinpoint same as our inverse kinematic solution derived by using library block from Simulink. However, the movement behavior for joint number 1 and 2 is same. That means there is still some possibility to improve optimization model furthermore to get it match. Figure 5-11 shows the robot arm movement in simulation.

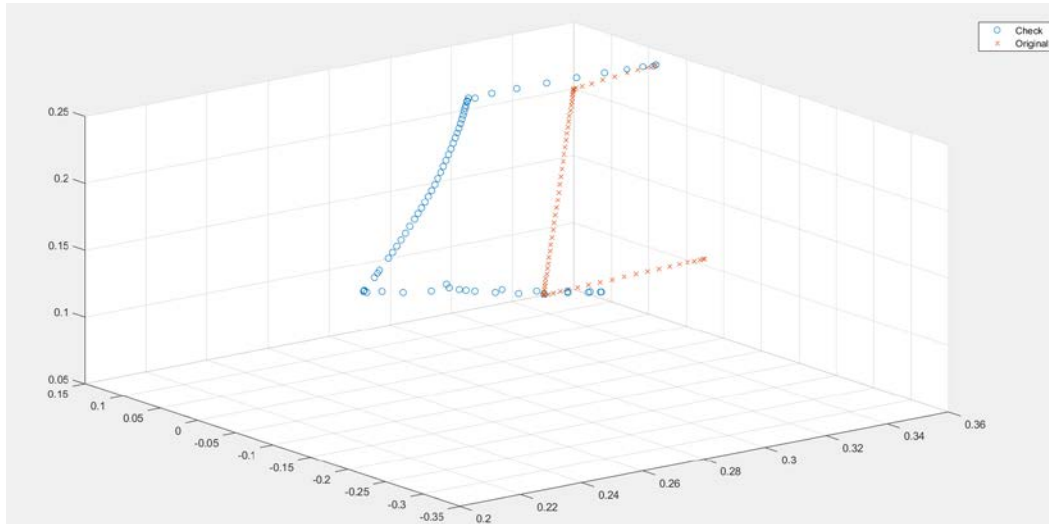


Figure 5. 10: Optimization result comparison with real trajectory

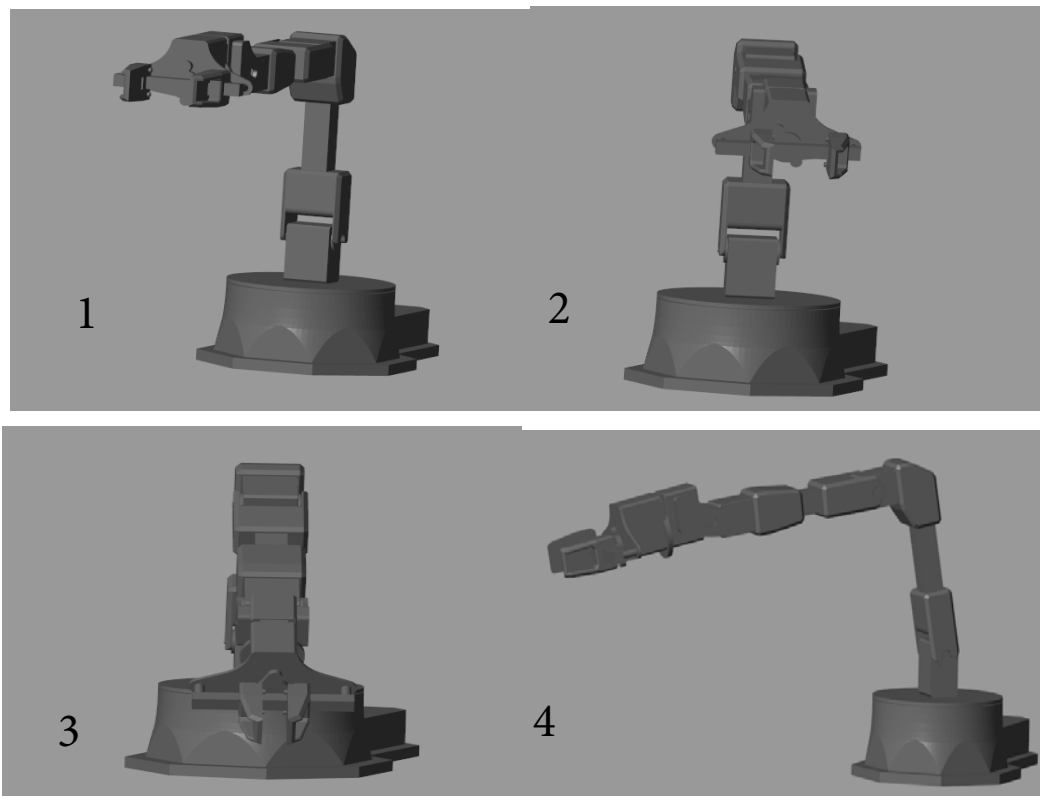


Figure 5. 11: Optimization result RX150 robot arm simulation

5.6.1 Code snippet

Figure 5-12 represents code snippet shows the error minimization optimization technique implemented in MATLAB script by using 'sqp' algorithm.

```
ops = optimset('Algorithm','sqp','Display','off','MaxIter',1500);
lb = [-1.57; -1.57; -1.57; -1.57]; % Lower bound in radian
ub = [1.57; 1.57; 1.57; 1.57]; % Upper bound in radian
q_ik = zeros(61,4);
elapsed_time = zeros(61,1);
for i= 1:61
    tic
    obj_func = @(q)objfun_ik_dh(q,p_eff(:,i));

    %use the fmincon solver
    [q,fval,exitflag,output,solutions] = fmincon(obj_func,q_ini,[],[],[],[],lb,ub,[],ops);
    q_ik(i,:) = q;
    q_ini = q;
    elapsed_time(i,1)= toc;
end

function J = objfun_ik_dh(q,p_eff)
% Joint angles
q1 = q(1);
q2 = -q(2);
q3 = -q(3);
q4 = q(4);
% Equation for xyz coordinate frame
xe = (cos(q1)*cos(q2))/20 + (3*cos(q4)*(cos(q1)*cos(q2)*cos(q3) - cos(q1)*sin(q2)*sin(q3)))/20

ye = (cos(q2)*sin(q1))/20 - (3*cos(q4)*(sin(q1)*sin(q2)*sin(q3) - cos(q2)*cos(q3)*sin(q1)))/20

ze = sin(q2)/20 + (3*cos(q2)*sin(q3))/20 + (3*cos(q3)*sin(q2))/20 + (3*cos(q4)*(cos(q2)*sin(q3)

xe_hat = p_eff(1);
ye_hat = p_eff(2);
ze_hat = p_eff(3);

ex = (xe_hat) - (xe);
ey = (ye_hat) - (ye);
ez = (ze_hat) - (ze);
e = [ex;ey;ez];
Q = diag([0.000999,0.000999,0.000999]); % Weighting matrix (Diagonal)
%Objective function
J = e'*Q*e;
```

Figure 5. 12: Optimization MATLAB code

5.6.2 Real arm movement picture frame and video

Real hardware unit movement following the trajectory solved by optimization shown in Figure 5-13. Here, top left position is number 1, top right position is number 2, bottom left is number 3 and bottom right is number 4 same sequence followed in Figure 5-11. We can consider the two videos containing real hardware movement solved by IK engine (one is from Simulink library and other is by optimization) for comparison which shown in Figure 5-8 and 5-14. After analyzing both, it can be noticed that the robot movement by optimization technique is not far away from the original one or the difference is not massive. That means the solution can be bit better if we improve our optimization formula by adding other relevant constraint. One improvement can be by adding joint angle error minimization along with position error.

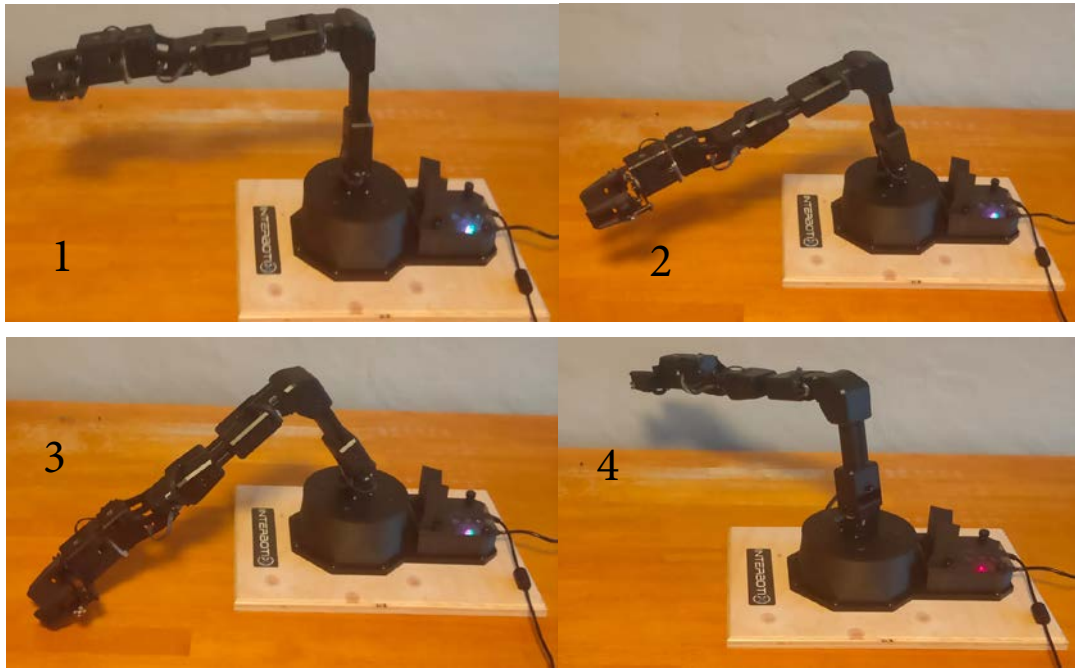


Figure 5. 13: Optimization result implemented in RX150 robot arm real unit

YouTube video link provided as follows and figure 5-14 can be clicked to see the robot arm movement using inverse kinematic by optimization technique in YouTube.

https://youtu.be/yWK_8rtbj90



Figure 5. 14: Video link for robot arm movement using inverse kinematic by optimization technique

5.7 Summary

It is presented in this chapter that a mathematical model of the forward and inverse kinematics of a robot manipulator is developed using the homogeneous transformation matrix method and DH parameters. This application describes methods for introducing robot kinematics, including open and closed kinematic chains. Unlike the forward kinematics, Inverse Kinematics gives multiple solutions rather than a single solution.

Chapter 6

Hardware Implementation

6.1 Overview

Chapter 3 and chapter 4 and 5 describes the process of creating, and implementing, the simulator, as well as the interfacing, for the ReactorX150 Robot Arm. This chapter will focus on the results from the interfacing and simulation process, seeing as the different position, robot arm movement and calibration by using slider, multiple robot interactive operation, and object pick and place are highly relevant for the conclusion of the thesis.

6.2 Sleep position

RX150 robot arm has different position. In Figure 6-1, shows the manipulator sleep position. After unpacking the robot arm user will get the robot arm in same position as Figure 6-1 shown.



Figure 6. 1: Image of the ReactorX 150 in the Sleep Position

There is one Simulink block created to call the real robot arm unit in sleep position shown in Figure 6-2. In the most right, there is a block called joint angles where appropriate joint

angles are provided by using slider. In addition, Slider represents the degrees and then degree converted to digital bit by using Simulink gain function to manipulate the robot arm. Since real robot arm unit does not familiar with degrees or radian, whereas robot arm simulation or URDF model required radian. Details can be seen from Dynamixel wizard about digital bit read and write addresses which has been discussed in Chapter 3 and 7. In the middle we have sleep position function which is a hardware interfacing function where joint angles are written in the specific address and one button named “Call_Sleep_Position” provided for the user, by pressing this button user can call the real robot arm unit into sleep position. Oscilloscope provided in the left to monitor the present joint angles in degree.

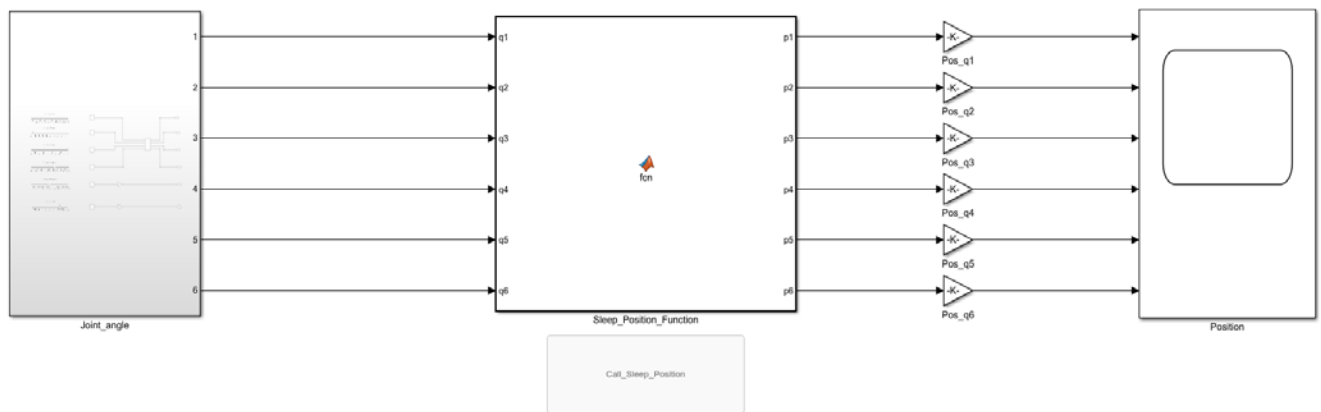


Figure 6. 2: Simulink block to call ReactorX 150 in the Sleep Position

6.3 Home position

The robot arm manipulator has the home position which is also known as zero-degree position shown in figure 6-3. Zero-degree position means, each joint angles has been provided zero degree or radian by using the sliders. All mechanism and blocks are same except joint angles are the only main difference between sleep and home position which has been provided by the slider shown in figure 6-4.

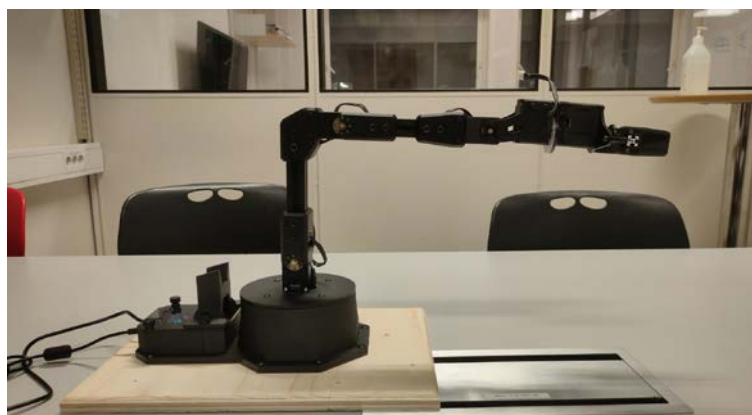


Figure 6. 3: Image of the ReactorX 150 in the Home Position

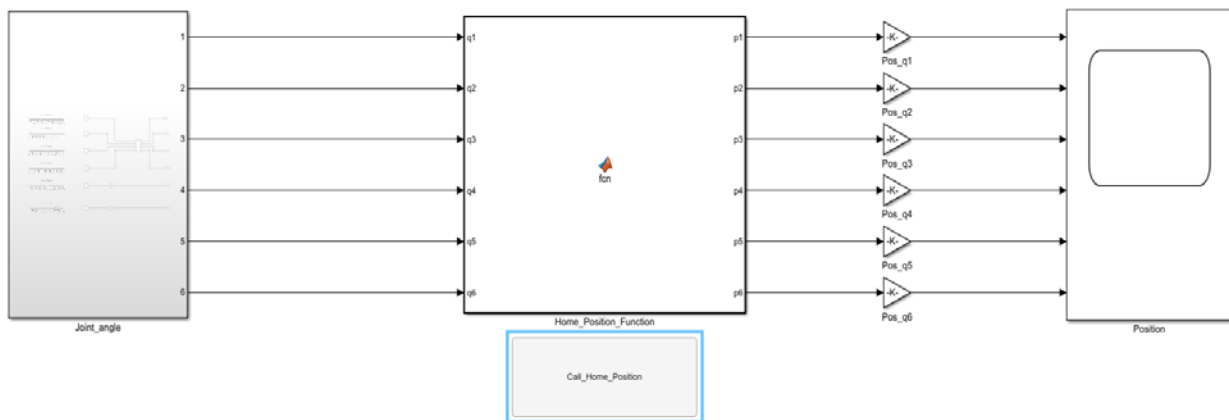


Figure 6. 4: Simulink block to call ReactorX 150 in the Home Position

6.4 Movement and Calibration of Physical Robot

Upon successfully creating communication with the robot arm, the next goal was to calibrate and check the response based on given signals. The first check was to check the ZDP seeing as this is important in terms of startup, as well as having a standardized starting point for trajectory tracking. The ZDP is displayed in Figure 6. 5, with the slider inputs displayed in Figure 6. 6.

The first thing to mention is the slight decline in the level arm, and this is due to the load of the arm itself. Due to the position tracking, it was deemed more important to have standard, meaningful, angles of the ZDP rather than compensating for the weight of the arm and thereby having uneven degrees of the joints.

The rotational servo is declared as 180° at ZDP. Due to safety limitations for the joint number 1, the joint cannot rotate to lower than 60° , the base servo is based on 2.5 turns, each turn 360° . However, in our case we have limited the robot arm turn between -90 to 90 degrees by considering the safety.

Both the 2nd and 3rd joint has limited range. The 2nd joint angles provided by the slider is higher than -75° , while the 3rd joint above -90° . Both ranges are limited due to minimizing collision risk with the surface. The 6th joint is not a revolute joint but is the prismatic joint of the robot arm which is unused in this thesis but can be used by giving some positive and negative values based on digital bit of servo motor [12].

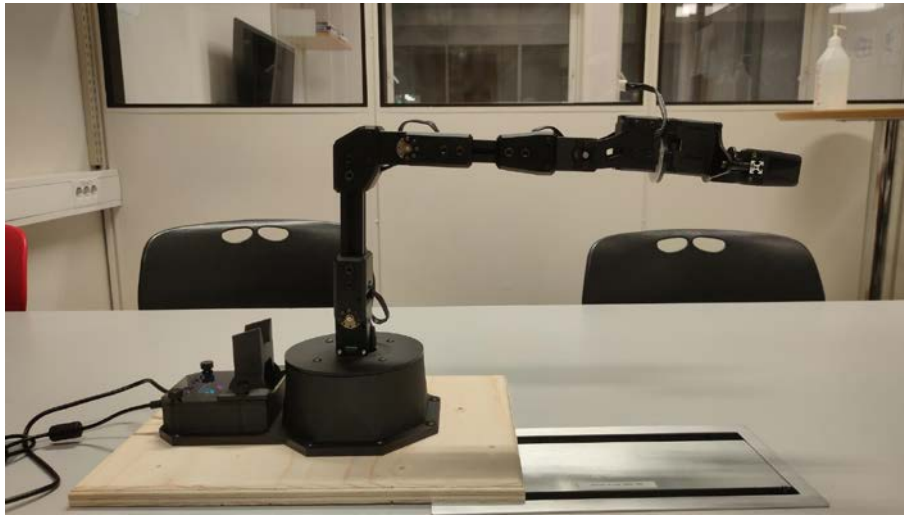


Figure 6. 5: Image of the ReactorX 150 in the ZDP[12]

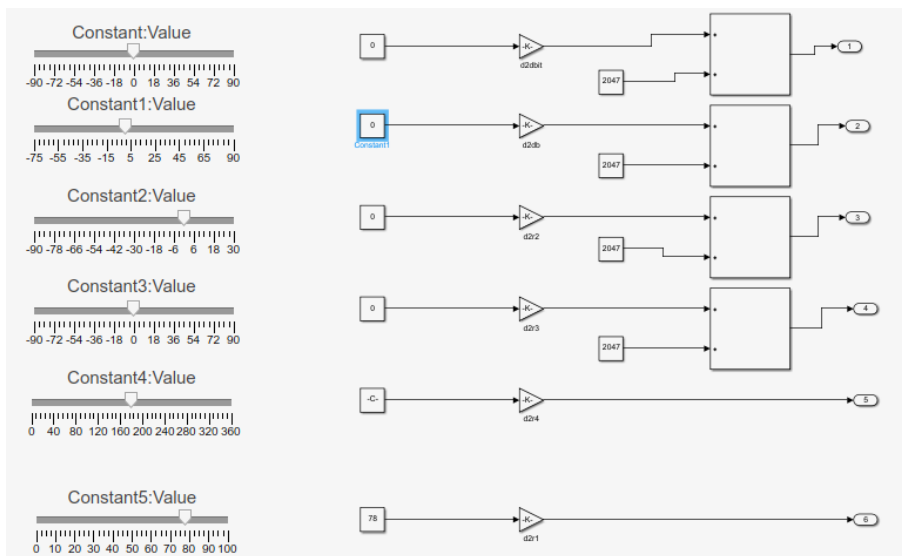


Figure 6. 6: Slider inputs based on the ZDP[12]

As a part of testing and calibrating the robot arm, several positions are tested and compared to the resulting movements of the robot arm. An example of this is moving a joint -90° , and test the amount of movement, as well as the direction of the movement. The movement of the robot arm, by changing the angle of joint 3 with -90° , is shown in Figure 6. 7 with the corresponding slider inputs in Figure 6. 8. The resulting position matches the predicted movement seeing as the “front” of the robot is towards the left and the 0° angle is declared as orthogonal with joint 2 [12].



Figure 6. 7: Image of the ReactorX 150 with 90° change, from the ZDP, in joint 3 [12]

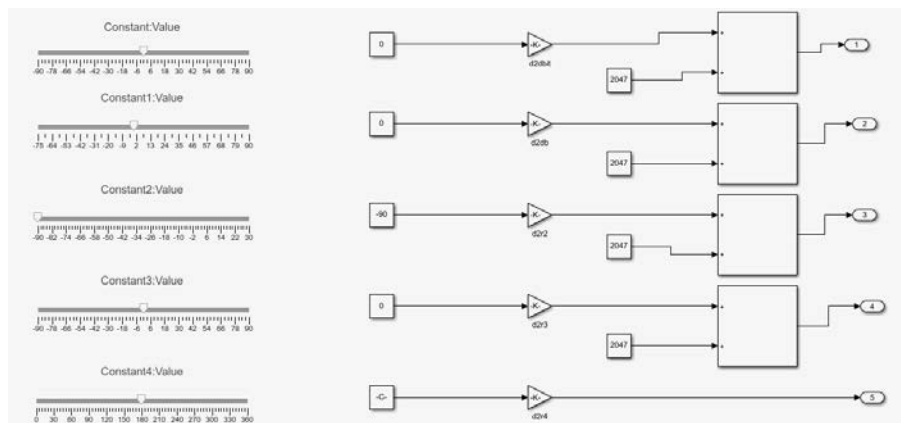


Figure 6. 8: Slider inputs based on the 90° change from ZDP [12]

As a third check, the robot arm was moved based on random inputs from the sliders. This check included several joints, with varying angles. The resulting position is displayed in Figure 6. 9 with the corresponding slider inputs in Figure 6. 10. From the slider inputs, the angle of joint 2 is set to 24°, while the angle of joint 4 is set to 54°. Joint 3 is kept at -90°. The position of the robot arm reflects these inputs seeing as joint 2 is at a slight angle from the upwards position, declared as 0°, with positive movement towards the left side, as well as a significant movement in joint 4.



Figure 6. 9: Image of the ReactorX 150 with a random position from slider inputs [12]



Figure 6. 10: Slider inputs based on the random position [12]

6.5 Robot arm pick and place operation

One of the requirements of the thesis is to do the pick and place operation with an object by using the gripper mechanism. Simulink block diagram has been created shown in Figure 6-11, to accomplish the pick and place task. Variable V, X, Y, Z, and Gripper are variable called “To workspace” which contains joint angle in a timeseries data. In order to get the timeseries data, trajectory has been created with desired waypoint, velocities, and way point time in MATLAB script then built in inverse kinematic engine used from Simulink library which shown in Figure 4-6, chapter 4. In the middle there is a MATLAB function block called Hardware_Function contains all the hardware information to write the joint angles value to the real hardware unit. There is a scope in the left side to monitor the present position in graph.

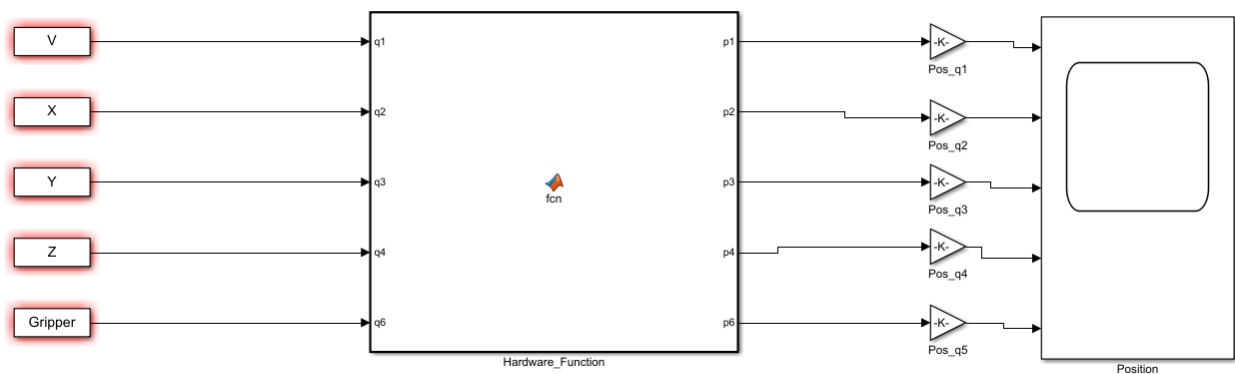


Figure 6. 11: Simulink blocks for robot arm pick and place operation

YouTube video link provided as follows and also Figure 6-12 can be clicked to see the pick and place operation in YouTube.

<https://youtu.be/QD5qRvh6MXs>

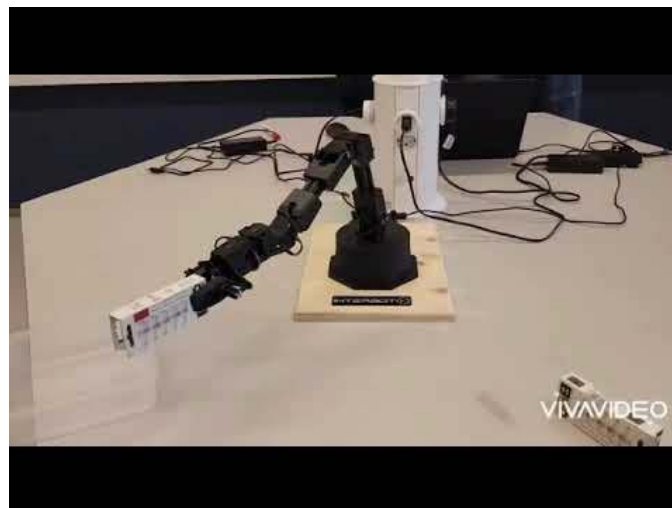


Figure 6. 12: Video link for robot arm pick and place operation

6.6 Multiple robot interactive task

One more exciting task of the thesis is to do the interactive operation by using multiple robot arm manipulator. Simulink block diagram has been created shown in Figure 6-13, to accomplish the interactive task. Figure 6-13, shown two MATLAB function block called “Robot_1_Write” and “Robot_2_Read”. “Robot_1_Write” function block will write the present position of Robot_1 which will activate with zero torque so that user can move the robot arm freely. On the other hand, “Robot_2_Read” function block activates the Robot_2 with torque and read the Robot_1 position and execute the task by writing the present position data of Robot_1 in the specific write address which shown in Figure 6-14. Hence, Robo_2 will follow the Robot_1 as user move it by hand. There is also a scope in the left side to monitor the present position of Robot_2 in graph.

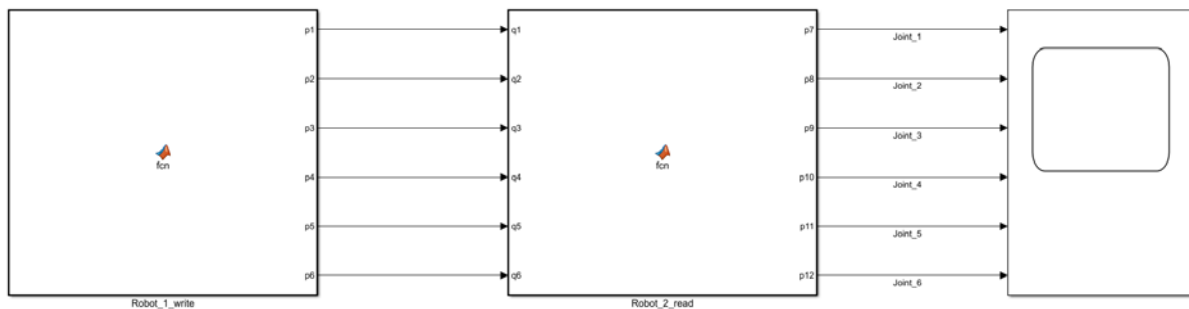


Figure 6. 13: Simulink blocks for multiple robot arm

```
function [p7,p8,p9,p10,p11,p12]= myrobot_2_read(q1,q2,q3,q4,q5,q6,port_num2)
PROTOCOL_VERSION = 2;
COMM_SUCCESS = 0;
ADDR_PRO_GOAL_POSITION = 116;
ADDR_PRO_PRESENT_POSITION = 132;
ADDR_PRO_GOAL_VELOCITY = 104;
ADDR_PRO_PRESENT_VELOCITY = 128;
ADDR_PRO_GOAL_CURRENT = 102;
ADDR_PRO_PRESENT_CURRENT = 126;
ADDR_PRO_GOAL_PWM = 100;
ADDR_PRO_PRESENT_PWM = 124;
PROFILE_ACCELERATION = 108;
PROFILE_VELOCITY = 112;
```



```
write4ByteTxRx(port_num2, PROTOCOL_VERSION, DXL_ID_7, ADDR_PRO_GOAL_POSITION, typecast(int32(q1), 'uint32'));  
%write4ByteTxRx(port_num2, PROTOCOL_VERSION, DXL_ID_7, ADDR_PRO_GOAL_VELOCITY, typecast(int32(q1), 'uint32'));  
%write4ByteTxRx(port_num2, PROTOCOL_VERSION, DXL_ID_7, ADDR_PRO_GOAL_PWM, typecast(int32(q1), 'uint32'));  
dxl_comm_result = getLastTxRxResult(port_num2, PROTOCOL_VERSION);  
dxl_error = getLastRxPacketError(port_num2, PROTOCOL_VERSION);  
if dxl_comm_result ~= COMM_SUCCESS  
    fprintf('%s\n', getTxRxResult(PROTOCOL_VERSION, dxl_comm_result));  
elseif dxl_error ~= 0  
    fprintf('%s\n', getRxPacketError(PROTOCOL_VERSION, dxl_error));  
end
```

Figure 6. 14: MATLAB script write function code snippet

YouTube video link provided as follows and also Figure 6-15 can be clicked to see the multiple robots interactive operation in YouTube.

<https://youtu.be/-LV71MMBAX4>



Figure 6. 15: Video link for multiple robot arm interactive operation

Chapter 7

DISCUSSION

7.1 Overview

Interfacing issues, Forward and Inverse kinematic solutions problem, Real unit critical calibration issue of RX150, academic robot arm manipulator is presented in this chapter. In chapter 2, we have discussed about how we're going to use the materials and methods for this thesis. To determine the forward and inverse kinematics of the adopted manipulators, conventional tools such as DH representation and homogeneous transformation methods are used. In homogeneous transformations, every orientation vector or transformation matrix needs to be stored with respect to the previous one from the beginning. The modern algebras, like quaternion algebras, require eight memory locations, whereas the homogeneous matrix method takes 12 spots. Space requirements affect the overall computational cost because bringing an operand from memory is more costly than performing a simple mathematical operation.

7.2 ROS2 Installation

As a part of creating the URDF file, both the Linux operating system and ROS2 was needed. The Linux OS was installed on a memory stick and used as the bootup for the computer used. ROS2 was then installed on the memory stick as well after running Linux. The problem which occurred in this installation process was the necessary files for ROS2 and the many dependencies needed. The code line, shown in Table 4. 1, is only possible if all dependencies are met, resulting in a lot of testing just to make sure the necessary software was downloaded for this file conversion. To make sure all the environmental set up done properly, installation sequence with proper feedback message is required which can be followed from ROS2 official website.

7.3 Dynamixel Servo ID

During the multiple robot interactive or collaborative operation, we need to connect multiple robots with our PC. Each RX150 robot arm manipulator having default servo id from 1 to 6. These id's are contradictory when we use multiple robots (more than 6 servos) at the same time. As a result, user will get error called "There is no status packet". To overcome this issue, we need Dynamixel wizard where servo id can be changed. As shown in Figure 7-1, we can change the decimal point of address 7. Now it is written as '8' which we can change based on our requirement for interfacing.

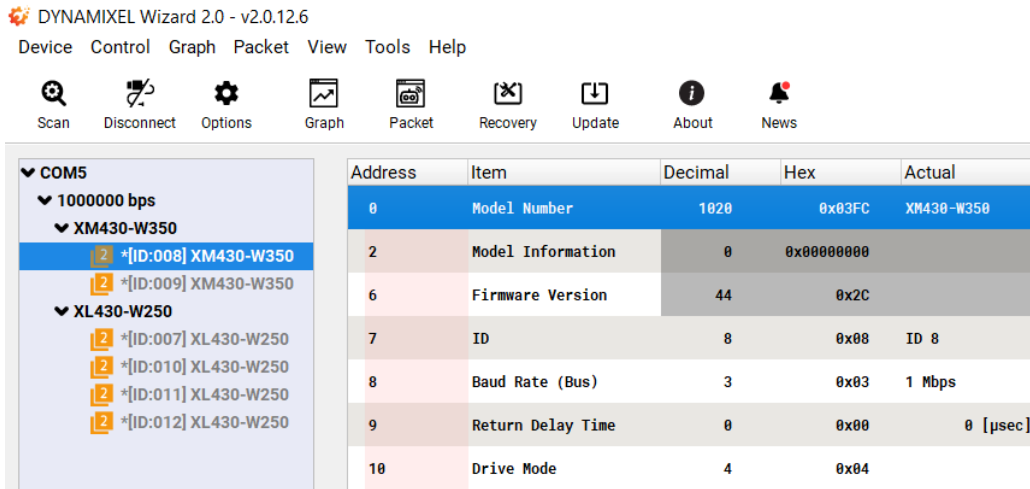


Figure 7. 1: Dynamixel wizard to change servo ID

7.4 Selection Of Servo Operation Mode

Dynamixel servo motor has many features, one of the special features is user can select different operation mode like Position, Current, PWM. This has been discussed in detail in chapter 3. It has shown top right corner in Figure 7-2, where we can select desired mode of operation. Moreover, the most important thing is, read and write address shall be changed based on the operation mode of selection.



Figure 7. 2: Dynamixel wizard to select servo operation mode

7.5 Servo Motor Communication

The last communication problem that occurred, based on the example code given in chapter 3 Table 3-2, was the communication protocol used in the given GitHub repository files. These communication protocols/memory addresses were given for a different type of servo motor. The solution was simply to find the manuals for the relevant servos for this thesis, namely the DYNAMIXEL XL430- and XM 430 servos, and use the correct memory addresses. In addition, we can do the address checking by simply connecting the real hardware unit with Dynamixel

wizard shown in Figure 7-3. For example, since in this thesis we have used position control, so address number 116 and 132 are required to write and read data respectively.

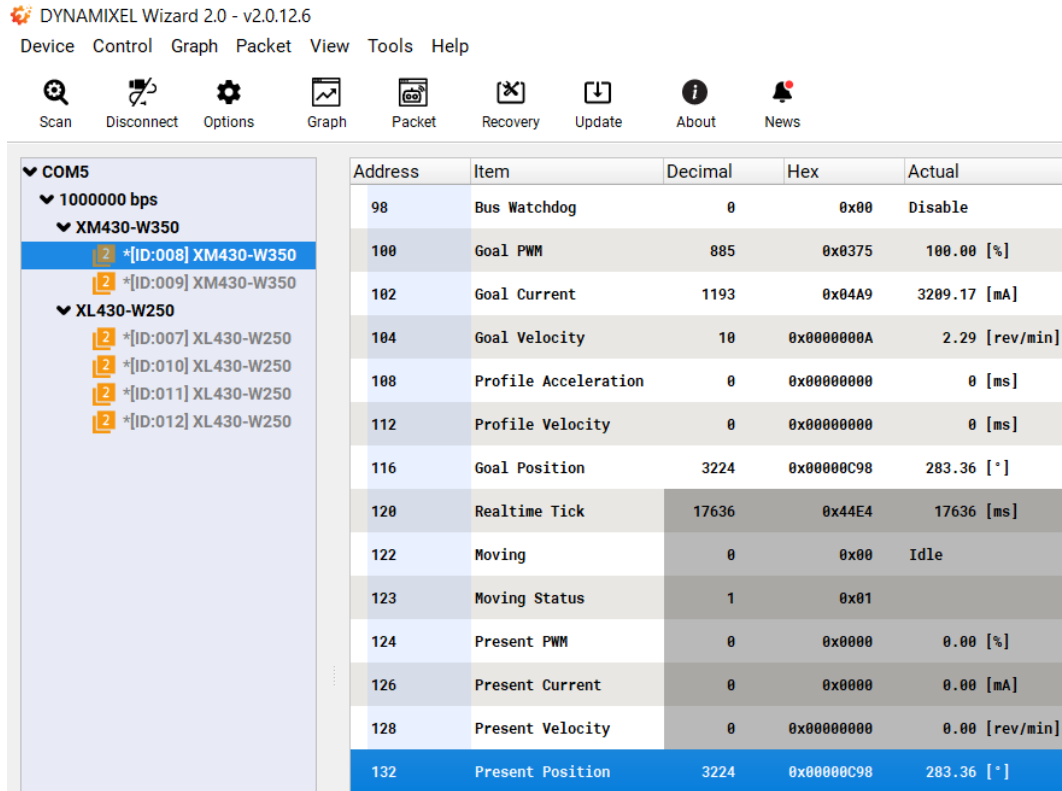


Figure 7. 3: Dynamixel wizard to check servo address with respect to ID

7.6 Servo Motor Rebooting

One of the most important issues faced during working with the real hardware unit is, motor overloading. Servo motor can be tripped during the operation for many reasons like overcurrent fault, overtemperature fault, overvoltage etc. To overcome this problem user should use the Dynamixel wizard to reboot the servo motor from Dynamixel wizard [shown in Figure 7-4.



Figure 7. 4: Dynamixel wizard to reboot servo motor

7.7 Interfacing

During development of the interfacing, another solution was created based on keyboard inputs. This solution was based on key binding the several movements of the robot arm, such as clockwise- and counter-clockwise rotation and moving each angle both ways, with different keys on the keyboard, resulting in a more operator focused interaction rather than having sliders in the code. This solution was changed to simplify the input from the operator, both in terms of the number of inputs and the simplicity of a single named slider shown in chapter 4, rather than two chosen buttons, for each joint [12].

7.8 Robot Arm Jerking Solution

Jerking problem face during trajectory following with real hardware unit which has been overcome by changing the address 108 ‘Profile Acceleration (700 ms)’ and address 112 ‘Profile Velocity (700 ms)’ in Dynamixel wizard. It has been shown in Figure 7-5, with default value of ‘Profile Acceleration’ and ‘Profile Velocity’ and Figure 7-6, with change value where profile defines the velocity on the programmed path. Default value (0 ms) provided following,

- The specified permissible velocities
- The accelerations and
- The set jerk is maintained.

One thing needs to be taken care is that, every time after shut down the power address 108 and 112 will contain the default value, so user should write the value again from Dynamixel wizard or from inside the read and write hardware code from MATLAB.

DYNAMIXEL Wizard 2.0 - v2.0.12.6

Device Control Graph Packet View Tools Help

Scan Disconnect Options Graph Packet Recovery Update About News

Address	Item	Decimal	Hex	Actual
98	Bus Watchdog	0	0x00	Disable
100	Goal PWM	885	0x0375	100.00 [%]
102	Goal Current	1193	0x04A9	3209.17 [mA]
104	Goal Velocity	10	0x0000000A	2.29 [rev/min]
108	Profile Acceleration	0	0x00000000	0 [ms]
112	Profile Velocity	0	0x00000000	0 [ms]

Figure 7. 5: Dynamixel wizard default servo acceleration and velocity

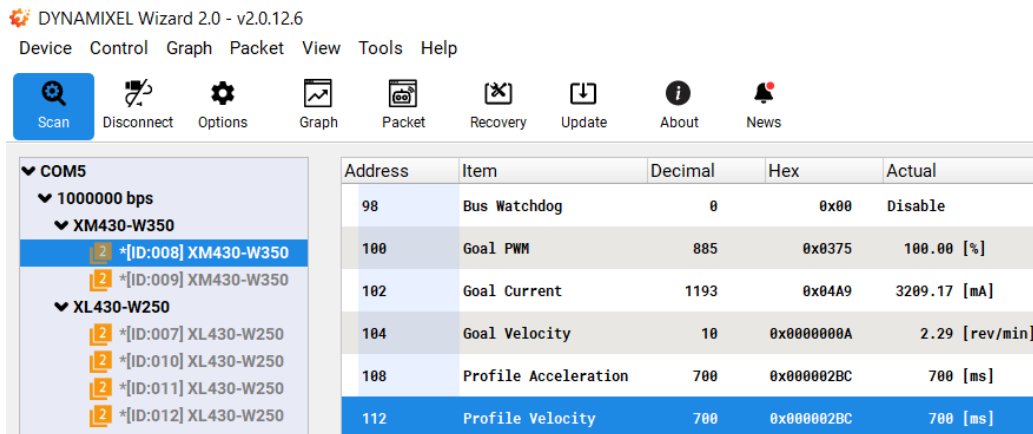


Figure 7. 6: Dynamixel wizard changed servo acceleration and velocity

7.9 Code Shifting

To shift the code from MATLAB script to Simulink, there was a problem to access MATLAB workspace variable from Simulink. This problem solved by Edit data option from Simulink where variables were added according to the requirement (i.e., Input, Output, Parameter etc.). In Figure 7-7, shown the tab to select or add additional input, output, and parameter.

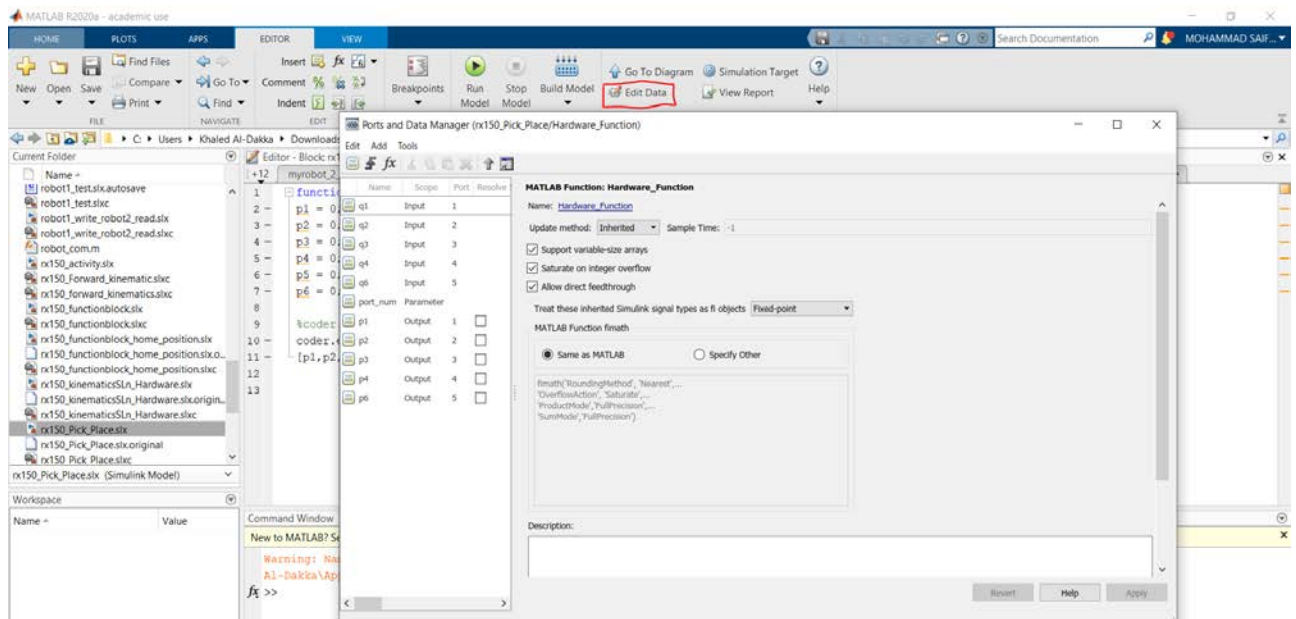


Figure 7. 7: MATLAB editor to get access to workspace from Simulink

7.10 Extrinsic Function

During communication with the physical robot arm, Simulink was not able to use the “calllib” function seeing as it was already compiled as external files. This problem was fixed by using the “extrinsic” function inside the code and call the MATLAB script where read, write and

different compiled function are used. “Extrinsic” function is a standard library function in MATLAB. Figure 7-8 shows how to use the function to make externally compiled file compatible with Simulink.

```
function [p1,p2,p3,p4,p6] = fcn(q1,q2,q3,q4,q6,port_num)
    p1 = 0;
    p2 = 0;
    p3 = 0;
    p4 = 0;
    p5 = 0;
    p6 = 0;%zeros(size(t));

    |
    coder.extrinsic('myrobot_PickPlace');
    [p1,p2,p3,p4,p6] = myrobot_PickPlace(q1,q2,q3,q4,q6,port_num);
```

Figure 7. 8: Extrinsic library function used in hardware interfacing MATLAB code

7.11 DH Model Tuning

Accuracy of Forward and inverse kinematic solution depends on good DH model. Therefore, model should be as accurate as possible. A good model can be made by following the steps mentioned in chapter 5. In addition, each joint length of robot arm manipulator should be correct which shown in Figure 7-9. The DH model shall be verified by plotting the simulation trajectory against DH model trajectory or implementing the DH model trajectory result in real hardware. In Figure 7-10, there is one Simulink model created to test the DH model and inverse kinematic solution created from optimization technique against the URDF model. Variable V, X, Y, and Z shown in Figure 7-10 are the q_1 , q_2 , q_3 , q_4 joint angles respectively saved from built in inverse kinematic engine solution of Simulink library detailed discussed in Figure 4-6 in chapter 4. These variables contain joint angles in time series data which provided to the URDF model and Qs variable contains joint angles which derived from inverse kinematic solved by optimization. The goal of this Simulink model to test as well as compare both type of joint angles derived from two different techniques, one is from inverse kinematic engine from Simulink library while other is inverse kinematic from optimization solution.

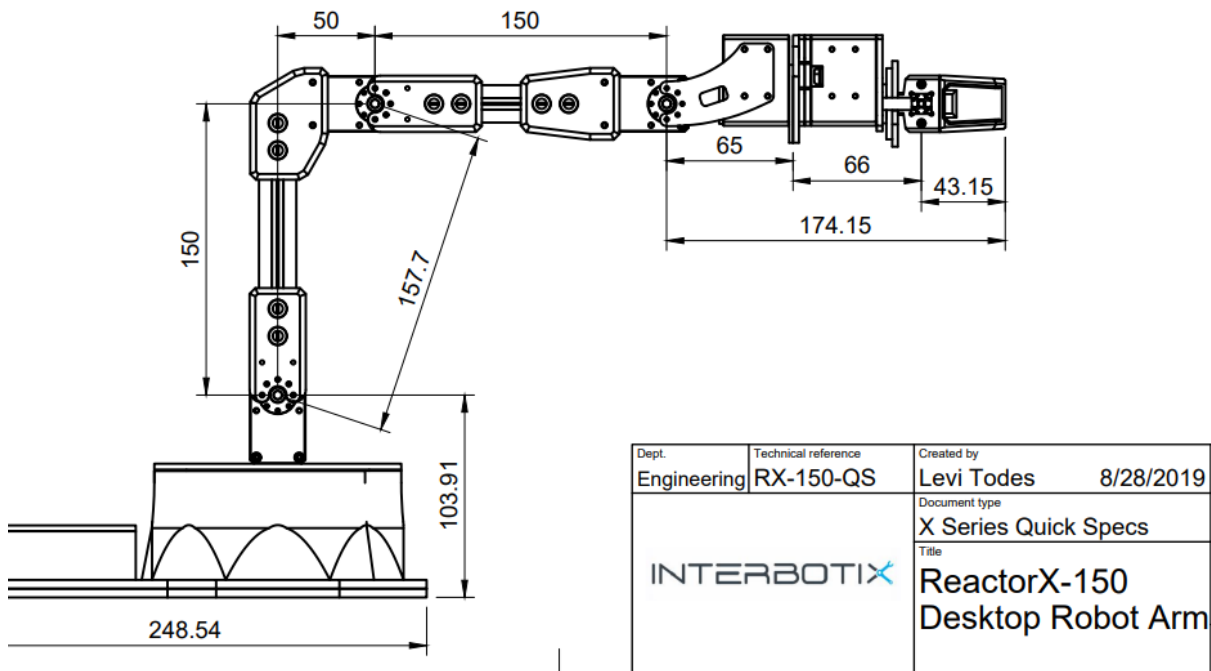


Figure 7. 9: RX150 each joint length measurement [9].

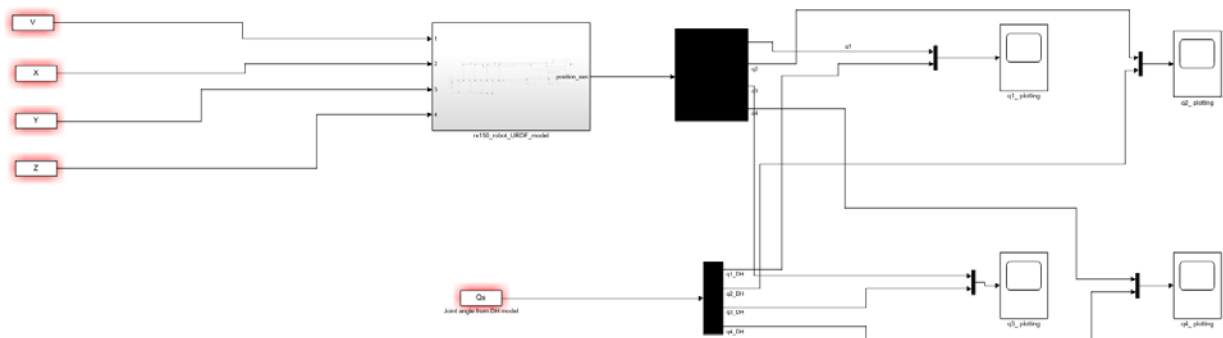


Figure 7. 10: DH model verification model in Simulink.

7.12 Summary

With the help of conventional and reactive approaches, a detailed analysis and discussion done of communicating with real robots, solving forward and inverse kinematic problems of selected manipulators and their simulations. The inverse kinematic solution for the selected benchmark manipulator was determined by DH model analysis in this chapter. However, optimization algorithms have been adopted, and comparisons have been made. To solve the inverse and forward kinematics problems for selected manipulators, MATLAB programs are used. Chapter 5 presents optimization algorithms and their comparison. In that chapter, the quality and efficiency of the proposed optimization technique are also presented.

Chapter 8

CONCLUSIONS AND FUTURE WORK

8.1 Overview

Even though industrial robots have advanced to a point where they can replace most of the various manual labor tasks, some of the fundamental problems in kinematics remain unsolved and act as a focus of thesis research. This thesis covers a substantial number of unsolved problems, namely the interfacing with robot arm manipulator through MATLAB, direct kinematics as well as inverse kinematics problem for serial chains. It is the main goal of this thesis work to discuss the interface with robot arms, the kinematics of the RX150 manipulator, and the optimization of it. An important aspect of this problem is that it can provide a starting point for other applications including molecular modeling and computer animations, in addition to manipulator design.

Today's world experiences a rapid expansion in terms of automation and this thesis has had the focus on how to create interfacing, with a robot arm, on a different and relevant platform. The ReactorX 150 Robot Arm, used in this thesis, already had some useful infrastructure, from Trossen Robotics, in terms of simulation files and documentation, but was designed for the Robot Operating System.

As a solution to this, interfacing with MATLAB was created, both in terms of the simulation and the physical communication with the ReactorX 150. The simulation, in MATLAB, includes a graphical display of the simulation, forward kinematic with DH model, inverse kinematics, and trajectory tracking based on the inverse kinematics and optimization to solve inverse kinematic problem. The interfacing with the physical robot arm includes movement based on slider input, as well as having the possibility to change between different control settings in the servos.

8.2 Conclusions

Robot manipulator configurations with inverse kinematic analysis are playing a major role in robotic systems, especially the control of the robot's movements. A key role that the kinematics of the robot plays when it comes to completing a given task plays a crucial role for the successful completion of different configurations of the robot for simulation and real-time control. Using conventional approaches to derive inverse kinematic formulations is costly and time-consuming because of the mathematical complexity. However, if one sets aside the mathematical expenses, the procedure provides a closed form solution.

8.3 Contributions

RX150 robot arm solutions include the following contributions based on Appendix A:

- i. Set up, configure, and test the ReactorX 150 robotic arm used with various functionalities of MATLAB. In this task, necessary communication interfaces between the physical robotic arm and Simulink/MATLAB should be created.
- ii. The robotic arm used for direct and inverse kinematics. For the inverse kinematic, the end-effector trajectory specified using polynomials.
- iii. DH model developed for the selected manipulators which has been used to find the forward and inverse kinematic solution.
- iv. The robot arm manipulator used for pick and place operations.
- v. Two ReactorX 150 robot arms performed interactive tasks.
- vi. Optimization technique developed to create inverse kinematic engine to solve inverse kinematic problem. The developed technique can be developed more to able to yield faster and accurate results, In order to make it useful for real-time applications.
- vii. We can propose an alternative optimization formula in order to avoid the problem of DH model matrix-based numerical solution of the inverse kinematic. The numerical solution poses several challenges, including the stability of the solution, which increases as the number of degrees of freedom of the manipulator increases.

8.4 Future work

A major concern for many students writing their thesis is the inverse kinematic problem. Researchers over the last few decades have been searching for general solution methods for different configurations, as well as for N-DOF manipulators, with different degrees of success [10]. This thesis lays the foundation on which future researchers will develop a general solution to the problem. It is believed that methods such as intelligence-based approaches, conventional approaches, and an optimization algorithm can provide inverse kinematics with a basic tool for solving the problem. As a result of the current performance, we have laid the foundation stone for future kinematic inversion studies so that hopefully other researchers will be motivated to explore innovative procedures in this field. There is also the possibility that a thesis could be examined on how to implement ANN models to reduce the DH model and inverse kinematic errors.

The physical robot arm may also be expanded upon by implementing inverse kinematics, as well as some complex trajectory tracking. For example, tightening the screw in a sophisticated environment, robots movement synchronization including vision input by using camera. Some methods from the neural network models of robot manipulator are required to be able to solve the problem of mathematical operations of inverse kinematics. There are quite a lot of inverse kinematic inversion problems that can be solved using ANN-based approaches. There is a complex and non-linear organizational structure that the ANN provides for the input and output

data structure and architecture. To generate the data sets used for training, forward kinematic equations that describe the movement of the manipulator can be used. In addition to this, it is important that the generated data sets are as big as possible to minimize the network's learning error. ANN models are prone to poor performance since they require different optimization strategies for the training of the model and are usually stuck at local optimum points. Inverse kinematics problems can be successfully solved using conventional methods as well as gradient descent learning algorithms.

Chapter 9

9.1 References

- [1] T. Robotics, "ROS Thesis Arms," 26 9 2021. [Online]. Available: <https://www.trossenrobotics.com/robotic-arms/ros-thesis-arms.aspx>.
- [2] R. Sharma, "Lecture Notes for the course IIA 4117: Model Predictive Control," 2019, p. 11.
- [3] S. Sivakumar, "GrabCAD Community," STRATASYS, 14 6 2020. [Online]. Available: <https://grabcad.com/library/2-dof-robot-arm-1> . [Accessed 18 04 2022].
- [4] Interbotix, "GitHub/Interbotix Ros Manipulators," 15 09 2021. [Online]. Available: https://github.com/Interbotix/interbotix_ros_manipulators/tree/main/interbotix_ros_xsarms. [Accessed 25 04 2022].
- [5] Interbotix, "GitHub/DYNAMIXEL SDK," 25 09 2021. [Online]. Available: <https://github.com/ROBOTIS-GIT/DynamixelSDK/tree/master/matlab>. [Accessed 25 04 2022].
- [6] Robotis, "Protocol2," 2021. [Online]. Available: <https://emanual.robotis.com/docs/en/dxl/protocol2/>. [Accessed 27 04 2022].
- [7] Robotis, "XM430-W350," 2021. [Online]. Available: <https://emanual.robotis.com/docs/en/dxl/x/xm430-w350/>. [Accessed 27 04 2022].
- [8] Rick, "xseries_arms," ROS, 11 02 2020. [Online]. Available: http://wiki.ros.org/xseries_arms. [Accessed 29 04 2022].
- [9] "ReactorX 150 Robot Arm," Trossen Robotics, [Online]. Available: <https://www.trossenrobotics.com/reactorx-150-robot-arm.aspx>. [Accessed 14 04 2022].
- [10] Panchanand jha, Inverse Kinematic Analysis of Robot Manipulators. ROURKELA, 2015.
- [11] J. Lenar i , T. Bajd and M. Stani i , Robot Mechanisms. Dordrecht: Springer, 2013.
- [12] Mohammad Saifuddin Chowdhury, Kevin Skogstad and Syed Sami , FM4017 Project, Porsgrunn, 2021

Appendices

Appendix A: Thesis Description

FMH606 Master's Thesis

Title: Real time control of robotic arm manipulators

USN supervisor: Roshan Sharma (USN)

Task background:

Robotic arm manipulators are widely used in industries for various applications. They are used in automotive, aerospace, electronic/electrical industries, shipping and trade etc. (just to name a few), for example, for performing repetitive tasks like those involved in an assembly line. USN has recently purchased several units of a ReactorX 150 robotic arm manipulators from Trossen Robotics. These robotic arms are planned to be used in teaching and research activities here at USN. The ReactorX 150 offers 5 degrees of freedom and a full 360 degree of rotation. At the heart of the ReactorX150 is the Robotis DYNAMIXEL X-Series smart servo motors and DYNAMIXEL U2D2 which enables easy access to Dynamixel software development kit. Figure 1 shows the ReactorX150 robotic arm manipulator.



Figure 1: ReactorX 150 robotic arm manipulator

Aim:

It is of interest to use this robotic arm in a MATLAB/Simulink platform for forming various tasks like position/trajectory control, direct/inverse kinematics, some advanced model based control by making use of robot dynamics and advanced observer/estimator design. Furthermore collaborative tasks involving interaction of two or more ReactorX150 robot arm manipulators is also considered to be one of goals of this project.

Task description:

The following are the main tasks:

- a) Set up, configure and test the ReactorX 150 robotic arm to be used with Simulink/MATLAB for various functionalities. In this task you should create necessary communication interfaces between the physical robotic arm and Simulink/MATLAB.

- b) Use the robotic arm for direct and inverse kinematics. For the inverse kinematic, the end-effector trajectory can either be (i) specified using polynomials, or (ii) it can also be created by simply moving the robot arm manually by hand along the desired trajectory.
- c) Use the robotic arm for pick and place operations. For this the gripper mechanism should be used.
- d) Using at least two or more ReactorX 150 robot arms, perform collaborative or interactive tasks. In this task, you may need to use additional hardware/software support.
- e) If time permits, interface a camera (for e.g. raspberry pi camera) to the robot arm and perform some tasks involving computer vision like classifying/separating different coloured objects.
- f) Document the work in a report. Presentation of the work.

Student category: Reserved

Reserved for Mohammad Saifuddin Chowdhury.

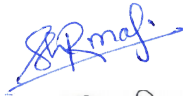
The task is suitable for online students (not present at the campus): No

Practical arrangements:

ReactorX 150 robotic arm manipulators will be provided to the student.

Signatures:

Supervisor (date and signature):

 25.01.2022

Students (date and signature):

