

DESIGN AND DEVELOPMENT OF AN ONLINE ROBOT  
PROGRAMMING FRAMEWORK WITH ROBOT OPERATING  
SYSTEM (ROS)

YEOH RU SERN

RESEARCH REPORT SUBMITTED TO THE FACULTY OF  
ENGINEERING UNIVERSITY OF MALAYA, IN PARTIAL  
FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF  
MASTER OF ENGINEERING  
(MECHATRONICS)

2019

**UNIVERSITY OF MALAYA  
ORIGINAL LITERARY WORK DECLARATION**

Name of Candidate: Yeoh Ru Sern

Registration/Matric No: KQF170007

Name of Degree: Master of Engineering (Mechatronics)

Title of Research Report ("this Work"): Design and Development of an Online Robot Programming Framework with Robot Operating System (ROS)

Field of Study: Industrial Automation and Robotics

I do solemnly and sincerely declare that:

- (1) I am the sole author/writer of this Work;
- (2) This Work is original;
- (3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
- (4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
- (5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
- (6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature

Date

Subscribed and solemnly declared before,

Witness's Signature

Date

Name:

Designation:

## **ABSTRACT**

In the Fourth Industrial Revolution, robotics technology plays an increasingly important role in order to increase productivity through the use of cyber physical systems. However, industrial robotic arms require expertise in fields such as mechanical and software engineering in order to be used. Furthermore, modularity of robotic work cells could be improved. In this project, an online robot programming framework is developed in Robot Operating System (ROS). The framework includes a master and slave node that allows for teleoperation of the intended robotic arm. A graphical user interface (GUI) is provided on the master personal computer (PC) in order to receive a target coordinate point for the robotic arm end-effector from the user. The EezyBotArm Mk2 3-dimensional (3D) printed arm is used for control and testing. The kinematics study of the robotic arm is performed and based on the equations derived is used to convert the coordinate point into the corresponding joint variables. The joint variables are then transmitted from the master PC to the slave Raspberry Pi. The Raspberry Pi interfaces with an Arduino Uno board in order to control the servo motors on the robotic arm via pulse width modulation (PWM) signal.

## ABSTRAK

Dalam era Revolusi Industri Keempat ini, teknologi robotik memainkan peranan yang makin penting untuk meningkatkan produktiviti industri melalui penggunaan sistem siber-fizikal. Namun begitu, penggunaan lengan robot perindustrian memerlukan tahap kepakaran pekerja yang tinggi supaya sesuai digunakan. Tambahan pula, modulariti lengan robot perindustrian juga boleh dipertingkatkan. Dalam projek ini, satu rangka kerja pengaturcaraan robot atas talian diciptakan dengan menggunakan ROS. Rangka kerja ini termasuk menggunakan noda tuan dan noda hamba bagi menyokong penggunaan tele-operasi. Satu GUI juga disediakan di komputer noda tuan untuk mendapatkan kordinat sasaran daripada pihak pengguna. Lengan robot EezyBotArm Mk2 yang diperbuat menggunakan teknologi percetakan 3D pula digunakan untuk tujuan ujian dan kawalan. Satu kajian atas kinematik lengan robot tersebut dibuat bagi menukarkan kordinat sasaran ke pembolehubah sendi melalui kinematik inversi. Pembolehubah sendi tersebut kemudiannya dihantar ke noda hamba Raspberry Pi yang mengawal motor servo pada lengan robot melalui Arduino Uno.

## **ACKNOWLEDGEMENT/ DEDICATION**

First and foremost, I would like to thank my supervisor, Dr Yap Hwa Jen for the continuous support and feedback he provided throughout the course of this project. His invaluable help has contributed much to the success of this project, and has helped me gained much knowledge in the field of industrial robots. Secondly, I would like to thank my parents for continuously being by my side, providing me support and strength to pursue my studies. Also, I would like to thank the friends I have made throughout my time University Malaya, from whom I have shared much joy, learned much from. Not forgetting the many lecturers that have taught me, for making this learning experience an enjoyable journey. Last but not least, I give all glory to God for the success of this project.

## TABLE OF CONTENTS

### Contents

ABSTRACT .....	iii
ABSTRAK .....	iv
ACKNOWLEDGEMENT/ DEDICATION.....	v
TABLE OF CONTENTS .....	vi
LIST OF FIGURES .....	ix
LIST OF TABLES .....	xii
LIST OF ABBREVIATIONS .....	xiii
LIST OF APPENDICES .....	xiv
CHAPTER 1: INTRODUCTION .....	1
1.1 Introduction .....	1
1.2 Problem Statement .....	2
1.3 Objectives of Research .....	2
1.4 Scope of Research .....	3
1.5 Report Organization .....	3
CHAPTER 2: LITERATURE REVIEW .....	4
2.1 Anatomy of the Robotic Arm.....	4
2.2 Forward Kinematics .....	5
2.3 Inverse Kinematics.....	8
2.4 ROS Architecture .....	9
2.5 Arduino Interface .....	12

2.6 Teleoperation.....	13
CHAPTER 3: METHODOLOGY .....	16
3.1 Proposed Framework Block Diagram.....	16
3.1.1 Network Architecture Block Diagram .....	16
3.1.2 Software Architecture Block Diagram.....	17
3.2 Robotic Arm Model .....	19
3.3 Inverse Kinematic Analysis of Robotic Arm .....	21
3.4 Open-loop Chain Equivalent of Robotic Arm .....	27
3.4.1 Chain 1 .....	28
3.4.2 Chain 2 .....	29
3.4.3 Chain 3 .....	30
3.4.4 Chain 4 .....	31
3.5 HTM Derivation of Open-loop Chain by DH Convention .....	32
3.5.1 HTM of Chain 1 .....	32
3.5.2 HTM of Chain 2 .....	34
3.5.3 HTM of Chain 3 .....	36
3.5.4 HTM of Chain 4 .....	38
3.6 Electrical Wiring Diagram .....	40
CHAPTER 4: IMPLEMENTATION AND RESULTS.....	42
4.1 Implementation of Project.....	42
4.1.1 Overall Project .....	42
4.1.2 Organisation of Files .....	42

4.1.3 System Boot-up.....	43
4.2 Results and Discussion.....	48
4.2.1 Experiment 1: Inverse Kinematics Computational Time.....	48
4.2.2 Experiment 2: Testing of Data Transmission across WLAN.....	50
4.2.3 Experiment 3: Accuracy of Robot Actuation.....	54
CHAPTER 5: CONCLUSION.....	58
5.1 Summary .....	58
5.2 Further Improvements.....	58
REFERENCES.....	60
APPENDICES .....	1
Appendix A: eezybot.urdf.....	A-1
Appendix B: printxyz.py.....	B-1
Appendix C: roserial.py.....	C-1
Appendix D: servo_control.ino.....	D-1



## LIST OF FIGURES

Figure 2.1: Anatomy of robotic arm .....	4
Figure 2.2: Comparison of open and closed loop kinematic chains .....	5
Figure 2.3: Separate coordinate frames.....	6
Figure 2.4: Illustration of DH parameters between coordinate frames .....	8
Figure 2.5: Geometric breakdown of the kinematics of a manipulator.....	9
Figure 2.6: Concept of communication in ROS.....	10
Figure 2.7: VR based rehabilitation robot.....	11
Figure 2.8: Architecture of 6 DOF robotic arm proposed by Megalingam et al.....	12
Figure 2.9: Arduino interface .....	13
Figure 2.10: Robotic arm workcell server.....	14
Figure 2.11: Client side block diagram .....	14
Figure 2.12: Distributed network of multiple robots, sensors and tele-operated devices .....	15
Figure 3.1: Block diagram of master-slave network architecture in project.....	16
Figure 3.2: Block Diagram of Software Architecture .....	17
Figure 3.3: Node Organisation of Master and Slave.....	19
Figure 3.4: CAD model image of EezybotArm Mk 2.....	20
Figure 3.5: Exploded view of robotic arm .....	20
Figure 3.6: Robotic arm at home position.....	22
Figure 3.7: Joint variables for the robotic arm.....	23
Figure 3.8: Geometric analysis of robotic arm.....	23
Figure 3.9: Breakdown of geometric components .....	24
Figure 3.10: Derivation of variables in robot frame .....	25
Figure 3.11: Derivation of $\theta_5$ .....	26
Figure 3.12: Open-loop chain equivalent of closed-loop chain .....	28

Figure 3.13: Joint angles for Chain 1 .....	29
Figure 3.14: Joint angles for Chain 2 .....	30
Figure 3.15: Joint angles for Chain 3 .....	31
Figure 3.16: Joint variables for Chain 4.....	31
Figure 3.17: Frame assignment for Chain 1 .....	33
Figure 3.18: Axis of frames for Chain 1 .....	33
Figure 3.19: Frame assignment for Chain 2.....	35
Figure 3.20: Axis of frames for Chain 2 .....	35
Figure 3.21: Frame assignment for Chain 3.....	37
Figure 3.22: Axis of Frames for Chain 3 .....	37
Figure 3.23: Frame assignment for Chain 4.....	38
Figure 3.24: Frame assignment for Chain 4.....	39
Figure 3.25: System-level connections .....	40
Figure 3.26: Connections on the Arduino Uno board.....	41
Figure 4.1: Actual project layout .....	42
Figure 4.2: Organisation of files in ROS .....	43
Figure 4.3: Launch new terminal .....	43
Figure 4.4: Sourcing configuration file.....	44
Figure 4.5: Starting the ROS Master.....	44
Figure 4.6: Initializing IK node.....	45
Figure 4.7: Initializing RViz visualization and slider GUI.....	45
Figure 4.8: Visualization tool and GUI.....	46
Figure 4.9: Initializing roserial on slave .....	47
Figure 4.10: Summary of boot-up steps.....	47
Figure 4.11: Definition of computational time taken.....	48
Figure 4.12: Graph of frequency against computational time.....	49

Figure 4.13: Data transmission across multiple machines .....	50
Figure 4.14: Graph of frequency against time delay .....	53
Figure 4.15: Image of X-Y plane position 1 .....	54
Figure 4.16: Image of X-Y plane position 2 .....	54
Figure 4.17: Image of X-Y plane position 3 .....	55
Figure 4.18: Image of X-Y plane position 4 .....	55
Figure 4.19: Image of X-Y plane position 1, returned .....	55
Figure 4.20: Image of X-Z plane position 1 .....	56
Figure 4.21: Image of X-Z plane position 2 .....	56
Figure 4.22: Image of X-Z plane, position 3 .....	57
Figure 4.23: Image of X-Z plane position 4 .....	57
Figure 4.24: Image of X-Z plane, position 1 returned .....	57

## LIST OF TABLES

Table 1: 3D printed parts file list .....	20
Table 2: Table of standard parts .....	21
Table 3: Summary of joint variable equations .....	27
Table 4: Joint variables for Chain 1 .....	29
Table 5: Joint variables for Chain 2 .....	30
Table 6: DH parameters derivation for Chain 1 .....	34
Table 7: DH Parameters derivation for Chain 2 .....	36
Table 8: DH parameters derivation for Chain 3 .....	37
Table 9: DH parameters derivation for chain 4 .....	39
Table 10: Tabulated results for computational time .....	48
Table 11: Table of comparison between outgoing and incoming messages .....	50
Table 12: Table of comparison between outgoing and incoming messages after rounding down .....	51
Table 13: Tabulated results for time delay .....	52

## LIST OF ABBREVIATIONS

CAD	: Computer Aided Design
CPS	: Cyber Physical Systems
CSV	: Comma-separated Values
DH	: Denavit-Hartenberg
DOF	: Degrees of freedom
FK	: Forward kinematics
GUI	: Graphical user interface
IIOT	: Industrial Internet of Things
IK	: Inverse kinematics
I/O	: Input/output
LAN	: Local Area Network
OS	: Operating System
PC	: Personal computer
PWM	: Pulse Width Modulation
RasPi	: Raspberry Pi
RC	: Radio control
ROS	: Robot Operating System
SCARA	: Selective Compliance Articulated Robot Arm
STL	: Stereolithography
URDF	: Unified Robot Description Format
USB	: Universal Serial Bus
Wi-fi	: Wireless fidelity
WLAN	: Wireless Local Area Network
XML	: eXtensible Markup Language

## LIST OF APPENDICES

Appendix A: eezybot.urdf .....	63
Appendix B: printxyz.py .....	67
Appendix C: roserial.py .....	71
Appendix D: servo_control.ino .....	73

University of Malaya

## **CHAPTER 1: INTRODUCTION**

### **1.1 Introduction**

The idea of the fourth industrial revolution, or better known as Industry 4.0 was initially proposed in Germany in the year 2011 (Roblek, Meško, & Krapež, 2016). The key technological advancement in Industry 4.0 is the introduction of cyber physical systems (CPS) in the production floor, which would help to improve the effectiveness and efficiency of the industry. The nine key drivers in the ushering in of the fourth industrial revolution and integration of CPS are autonomous robots, simulations or digital twins, cloud computing, additive manufacturing, augmented reality, big data analytics, Industrial Internet of Things (IIOT), cyber security and system integration (Posada et al., 2015).

The use of autonomous robots will help to reduce manual labour costs, and increase the productivity of any given industry. Robots are often employed in areas which involve highly repetitive tasks, or involve great risk to human lives (Lawton, 2018). This is because unlike human labour, autonomous robots are able to operate for long hours without fatigue, and are easily replaceable when broken. Combined with the other key pillars of Industry 4.0 such as digital twins and IIOT, the flexibility of autonomous robots is further increased. With this, autonomous robots can be easily monitored remotely through the visualization of the data collected remotely from the robots.

However, the field of robotics is complex and is situated at the intersection of core engineering fields such as mechanical, electronics, control and software engineering. Traditionally, this meant that robotics projects are inherently complex and require expertise in multiple fields. Robot Operating System (ROS) is a popular open-sourced platform that is enabling rapid research in the field of robotics. The availability of various

modules and development tools has allowed a fast prototyping of ideas and a decrease in research costs (Barbosa et al., 2015). Also, ROS is available in many programming languages such as C++, Python, Octave and LISP which further cements its position as a flexible tool.

## **1.2 Problem Statement**

Traditionally, industrial robot programming requires a trained programmer to reconfigure the industrial robotic programming when deploying a robotic work cell or whenever there is a change to the robotic work cell. The programmer is also required to be present near the robotic work cell in order to have access to the industrial robot programming. Also, programming for the robotic work cell is usually rigid, requiring that the robotic work cell be taken offline whenever new modules are to be added or modified due to the software structure being centralised and not distributed into modular nodes. This reduces the ability of the robotic software framework to be changed and the flexibility for new modules and features to be added. Besides that, a robotic platform that is closed source discourages the reuse of software, prohibiting the fast development of new robotics technology.

## **1.3 Objectives of Research**

The objectives of this research are stated as below:

1. To derive the kinematics of an industrial robot.
2. To map an input from an input device coordinate space to the industrial robot end effector coordinate space.
3. To develop a robot programming framework and control in Robot Operating System (ROS).



## **1.4 Scope of Research**

The scope of this research project is to design a framework for programming and controlling an industrial robotic arm. A prototype industrial robotic arm is fabricated using additive manufacturing. The robotic arm is actuated by hobby-grade servo motors. Then, a software framework to program and control the robotic arm is designed using the ROS platform and is written using the Python programming language. The control of the robotic is achieved through the forward and inverse kinematics study of the robotic arm. This thesis includes the theory underlining the design of the framework and an analysis of the performance of the project.

## **1.5 Report Organization**

This research report is organized into five chapters. In Chapter 1, a general introduction and to the project and the objectives is given.

In Chapter 2, a brief literature review is presented, outlining the theory and past projects that are related to this project. Namely, the literature review covers the manipulator kinematics and software architecture that are related to the project.

Chapter 3 describes the methodology with which the overall system is to be developed, along with relevant equations to be used.

Chapter 4 gives the actual implementation of the project, showing the actual process flow and workings of the project. Also, experimentation is conducted to qualify the results of the project and a discussion on the results is presented.

Finally, in Chapter 5 a conclusion to the overall project is given, and future recommendations for the project are made.

## CHAPTER 2: LITERATURE REVIEW

### 2.1 Anatomy of the Robotic Arm

Figure 2.1 shows the anatomy of a typical industrial robotic arm or also known as a manipulator. In general, the robotic arm consists of a base that is rigidly attached to the ground, moving links and joints, and an end-effector attached to the free end of the robotic arm. It can be described through two main attributes: orientation and position (Craig, 2018). In order to fully describe the position and orientation of the robotic arm in space, a coordinate frame is typically attached to the base of the robotic arm. This then becomes the point of reference that describes the relative position and orientation of the robotic arm. A robotic arm is typically modelled with a combination of rigid links and joints, as shown in Figure 2.1. There are two types of joints that are used to allow relative movement between adjacent links: prismatic joints and revolute joints. For prismatic joints, the movement comes in the form of a sliding motion, which allows displacement between the two adjacent links. This motion is known as a translation, or a joint offset. With revolute joints, the joint becomes a pivot point for the adjacent links to revolve about one another, and the movement is categorised in terms of joint angles.

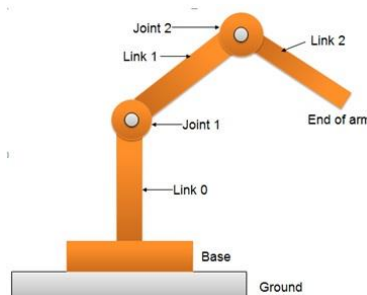


Figure 2.1: Anatomy of robotic arm

The number of degrees of freedom(DOF) that the robotic arm has is equal to the amount of independent position variables that are available(Lynch & Park, 2017). The degrees of freedom is a representation of the flexibility of the type of movement that is able to be achieved by the manipulator. The combination of joints and links that maps the robotic arm from the base to the end-effector is known as the kinematic chain. Kinematics is used to describe the position, velocity and the acceleration of the robotic arm in space. The forward kinematics(FK) analysis is finding the position of the end-effector given all the values of the joint variables, while the inverse kinematics(IK) analysis is to find the required joint variables to be achieved by the robot joints in order to bring the end-effector to a given point(Pandilov & Dukovski, 2014). There are two main types of kinematic chains; open-loop and closed-loop kinematic chains as shown in Figure 2.2. In the open loop configuration, the solution to the forward kinematics problem is relatively straight forward, while for the inverse kinematics problem is more complicated. However, the inverse is true for closed loop kinematic chains.

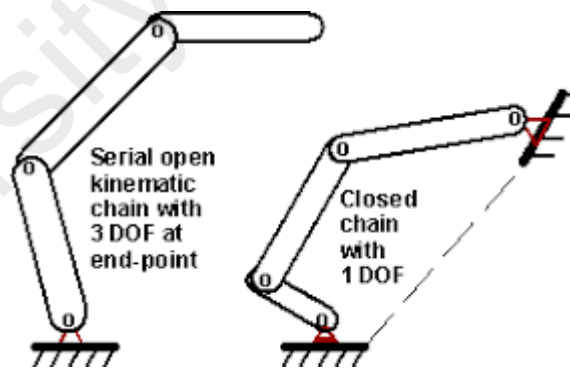


Figure 2.2: Comparison of open and closed loop kinematic chains

## 2.2 Forward Kinematics

In the forward kinematics analysis, all the joint variables are known, and the analysis is conducted in order to derive the location of the end-effector relative to the base frame. In a coordinate system, any point located in space can be identified through the use of a 3

x1 position vector as shown in equation (2.1). The position vector gives a location of a point with the respect to the reference frame A, with displacements in the  $P_x$ ,  $P_y$ , and  $P_z$  unit vectors(Spong, Hutchinson, & Vidyasagar, 2006).

$${}^A P = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} \quad (2.1)$$

However, a complete description of a point in space also requires information of the orientation of the point. In order to achieve this, a separate coordinate system is attached to the point, and the description of the orientation of this point can then be given relative to the reference plane as shown in Figure 2.3. With these two coordinate systems, the orientation of the frame B can then be given relative to the frame A.

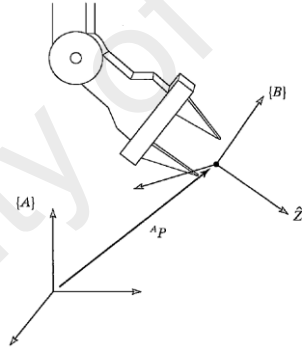


Figure 2.3: Separate coordinate frames

A rotation matrix is used in order to transform a point located in coordinate system B to coordinate system A. The rotation matrix transforms the unit vectors of the three principal axes in B to A as shown in equation (2.2).

$${}^A R_B = \begin{bmatrix} {}^A \hat{x}_B & {}^A \hat{y}_B & {}^A \hat{z}_B \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (2.2)$$

Equation (2.3) shows a general equation that gives a transformation of a vector that is described in frame B into a vector that is described in frame A. The combination of the position vector and the rotation matrix in the right hand side of equation (2.3) yields a 4x4 matrix known as the homogeneous transformation matrix (HTM). With this, the FK analysis of a robotic arm can be derived through successive multiplications of the homogeneous transform of one coordinate frames from the base to the end-effector.

$$\begin{bmatrix} {}^A P \\ 1 \end{bmatrix} = \begin{bmatrix} {}^A_B R & {}^A P \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^B P \\ 1 \end{bmatrix} \quad (2.3)$$

One popular method in performing the FK analysis is through the Denavit-Hartenberg(DH) Convention. The procedures outlined through this procedure simplify the process of transforming from one coordinate frame to a subsequent coordinate frame through the use of DH parameters. Through this convention, each transformation  $T_i$  can be represented as a product of four subsequent transformations as shown in equation (2.4) (Spong et al., 2006). These transformations are a rotation  $\theta_i$  around  $Z_{n-1}$  to align  $X_{n-1}$  to  $X_n$ , followed by a translation  $d_i$  along  $Z_{n-1}$ , followed by a rotation  $\alpha_i$  around  $X_n$  to align  $Z_{n-1}$  to  $Z_n$ , and lastly a translation  $a_i$  along  $X_n$ .

$$\begin{aligned} T_i &= Rot_{Z_i \theta_i} Trans_{Z_i d_i} Trans_{X_i a_i} Rot_{X_i \alpha_i} \\ &= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} c_{\alpha_i} & s_{\theta_i} s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i} c_{\alpha_i} & -c_{\theta_i} s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \quad (2.4)$$

Figure 2.4 shows an illustration of the application of the DH parameters between two adjacent coordinate frames. In order to apply the DH convention for kinematic analysis, there are two assumptions made. First, the axis  $x_1$  is perpendicular to  $z_0$ . Secondly, the axis  $x_1$  intersects  $z_0$ .

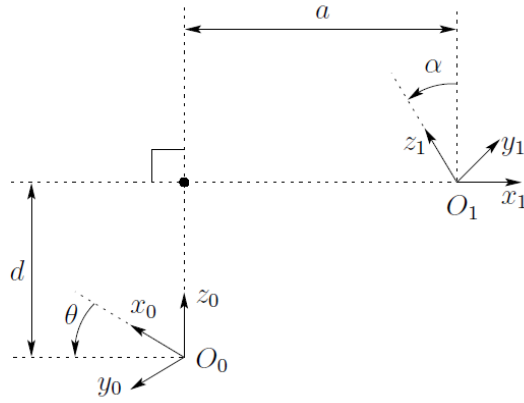


Figure 2.4: Illustration of DH parameters between coordinate frames

### 2.3 Inverse Kinematics

Inverse kinematics (IK) analysis is the task of finding the combination of joint variables that could bring the end-effector to a specific coordinate point. Depending on the number of degrees of freedom of a given manipulator under analysis, the IK analysis may include controlling the orientation of the end-effector. While the FK analysis has unique solutions, the IK analysis may have no solutions, a unique solution or even multiple solutions. This makes the IK problem generally harder to solve when compared to the forward kinematics analysis.

The existence of a solution for the IK problems ties closely to the workspace of the robotic arm. If a goal point lies within the workspace of the robotic arm, then the solution to the inverse kinematics analysis exists. The workspace of a robotic arm is defined as the amount of space that is reachable by it. A robotic arm that has less than 6 DOF will not be able to fully control the orientation of its end-effector within its reachable workspace. In general, there are two methods to solving inverse kinematics analysis: closed-form solutions and numerical solutions. Numerical solutions are typically slower when compared to closed-form solutions, thus more emphasis will be given to closed-form solutions. One such form of numerical solutions to the inverse kinematics problem is

through the use of genetic algorithms where the solution to the IK problem is modelled as search problem, and the solution is found through natural selection of successive generations of different models (Bjorlykhaug, 2018). A closed-form solutions are analytical methods, and can be obtained either algebraically or geometrically.

(Issa, Aqel, Albelbeisi, Elaila, & Mortaja, 2017) described one method of obtaining the inverse kinematics of a manipulator, which is through successive inverse transformations of the transformation  $T_i$  obtained originally in equation (2.4). Besides that, another method of solving the inverse kinematics problem is described by (Tao, Chen, & Xiong, 2015) is through a geometric breakdown of the angles and lengths of the links and joints of manipulator into its component, and then solving each term geometrically as shown in Figure 2.5. A similar approach was also utilised by (Sarkar, 2018) and (Mohammed Abu, Abuhadrous, & Elaydi, 2010).

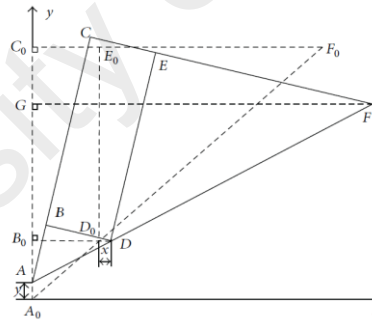


Figure 2.5: Geometric breakdown of the kinematics of a manipulator

## 2.4 ROS Architecture

ROS is a software framework that is used for writing software for robotics application. It provides a collection of tools that can aid in the process of creation of complex and robust robotics systems across multiple platforms (Lentin, 2015). The ROS filesystem is organized into packages, where each package contains important executables, libraries and configuration files.

In the ROS architecture, the main data structure that is being used to through the ROS frame work is through the use of messages. Each functionality in the robotic system can be represented as a node in ROS as shown in Figure 2.6. This feature allows the ROS framework to be distributed and modular. A ROS Master then manages the communications between each ROS node, without which the ROS nodes would not be able to send or receive messages. Communication between each ROS node then occurs through the concept of publishing and subscribing, where a ROS node may make information computed from the node public by publishing messages to a ROS topic, and other ROS nodes may have access to the information by subscribing to the same ROS topic.

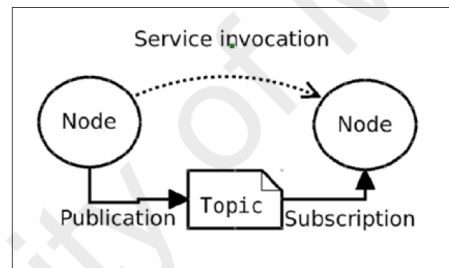


Figure 2.6: Concept of communication in ROS

There are many projects that have been implemented through the use of the ROS architecture. One such project is a limb rehabilitation robot that was proposed by (Du, Sun, Su, & Dong, 2014) in Figure 2.7. The project involves a master and slave type of architecture where a master personal computer (PC) is used to host the ROS software framework that then interfaces with various peripherals such as the robot itself and input devices. A simulation software called Gazebo is also run on the master PC in order to simulate the movements of the patient and provide important metrics in the training programme.



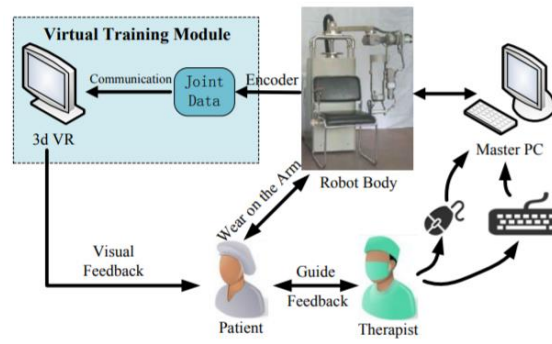


Figure 2.7: VR based rehabilitation robot

Besides that, another project by (Qian et al., 2014) made use of the Gazebo simulator in ROS to run simulations on the manipulation task of a robotic arm. The robotic arm was modelled in Gazebo using the Unified Robot Description Format (URDF), which is a type of eXtensible Markup Language (XML) file used to provide information on the joints and links of a manipulator. The URDF format makes use of stereolithography (STL) file which are generated from computer aided design (CAD) models in order to provide visualization of the robotic arm in Gazebo. Another simulation tool that is available in the ROS framework is RViz. While Gazebo is able to simulate the physics involved in the operations of a manipulator or robot, Rviz is a lighter weight tool that simply provides visualization of the robot in the simulation.

(Megalingam et al., 2018) proposed a 6 DOF robotic arm that was also designed in the ROS framework as shown in Figure 2.8. The system used the RViz tool for visualization, and the Moveit package within ROS to solve the FK and IK problems. The developed system makes use of a graphical user interface (GUI) for the user to give an input of the desired point in the 3D workspace for the end-effector to move to. Then, the IK solver in the Moveit package is used to compute the necessary joint variables need to bring the end-effector of the robotic arm to the desired location. The motion planner is able to generate the optimum trajectory and FK analysis is used to display the simulated path of

the movement of the robotic arm in RViz. Similar work was also conducted by (Ergur & Ozkan, 2014) in designing a simulator for a 5 DOF Selective Compliance Articulated Robot Arm (SCARA) used in deburring process. The proposed system utilized a linear trajectory between desired end-effector locations, and performed trajectory generation calculations in a customized ROS node.

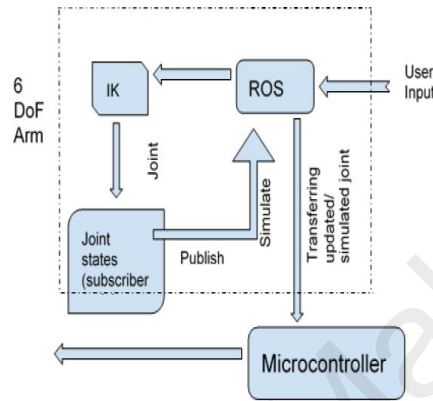


Figure 2.8: Architecture of 6 DOF robotic arm proposed by Megalingam et al.

## 2.5 Arduino Interface

The Arduino microcontroller is a low cost input/output (I/O) board. (Kruthika, Kumar, & Lakshminarayanan, 2016) proposed a robotic arm design that used an Arduino Mega2560 board as an interface to the motors and sensors on the robotic arm as shown in Figure 2.9. The PC does not have I/O pins, and instead needs an intermediary device to handle the connections to the sensors and motors for the robotic arm. In this project, the robotic arm is actuated by stepper motors, and are controlled by pulse width modulation (PWM) signals generated by the Arduino board. A PC is connected to the Arduino board through a serial interface, and regularly sends the desired motor angles that are calculated through IK to the Arduino board. Potentiometers are used as a feedback mechanism on the robot joints in order to obtain the angles. (Bhargava & Kumar, 2017) took a slightly different approach, replacing the combination of stepper motors and potentiometers at the robot joints with radio control (RC) servos. This way, the closed-loop control of the position of

the motors becomes extracted away, and the angle of the motors could be controlled directly through PWM signals corresponding to the desired servo angle. However, the disadvantage is that RC servos typically have a limited range of actuation.

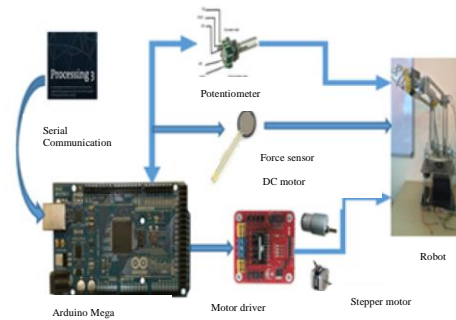


Figure 2.9: Arduino interface

The project proposed by (Megalingam et al., 2018) and (Hernandez-Mendez et al., 2017) also made use of an Arduino board in order to handle the interface with the robot motors and sensors. Simulation and the IK calculations were first done on the PC, and the information on the desired joint angles were sent to the Arduino board through the roserial package. The roserial package is a package within ROS that handles the communication between the master node and the Arduino node within the ROS environment.

## 2.6 Teleoperation

Teleoperation is the ability to remotely control or interface with a robotics system. Robotics systems are often deployed in unstructured environments where human presence may endanger human life (Mortimer, Horan, & Joordens, 2016). In the case of robotic arms in the factory setting, production lines may prove to be inaccessible to operators and engineers while they are in operation. Therefore, the incorporation of teleoperation as a feature may help to increase accessibility to the robotics systems, while improving safety measures on the production floor (Chen, Yan, Yuan, Yao, & Hu, 2018).

Figure 2.10 and Figure 2.11 shows the server and client block diagrams of the client-server architecture proposed by (Rozman, Luža, & Zbořil, 2014). The server side of the system is responsible for IK computation, simulation, collection and processing of input data from sensors, and sending control signals to the robotic arm. On the client side of the system, an interface is presented in order to collect input commands from users. Also, the high level path planning and image processing of the image data collected from the cameras is conducted.

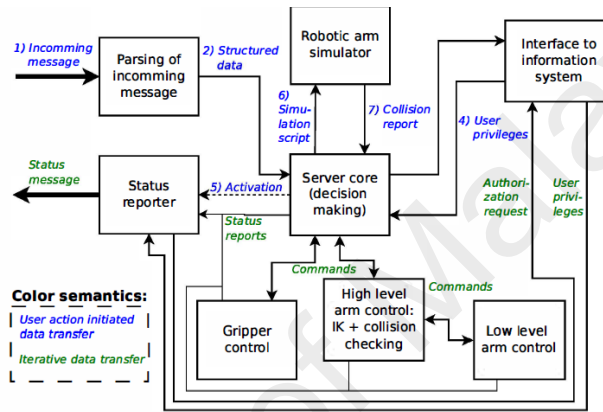


Figure 2.10: Robotic arm workcell server

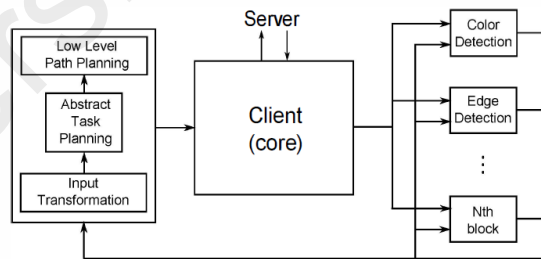


Figure 2.11: Client side block diagram

On the other hand, (Araújo, Portugal, Couceiro, & Rocha, 2013) implemented a master-slave type of network architecture in creating a distributed network to control mobile robots. A PC served as the master in the system, while multiple mobile robots that were modelled as the slave nodes were present in the system. Communication was achieved across a wireless-fidelity (Wi-Fi) network, with the ROS framework being hosted on the

master PC while Xbee Shields that connected to the Wi-fi network were placed on Arduino Uno boards that controlled the mobile robots. Once all the devices were connected to the same Wi-fi network, a serial communication was used to establish connection between the Arduino and the ROS/PC. Each device and component can then be modelled as separate nodes as shown in Figure 2.12.

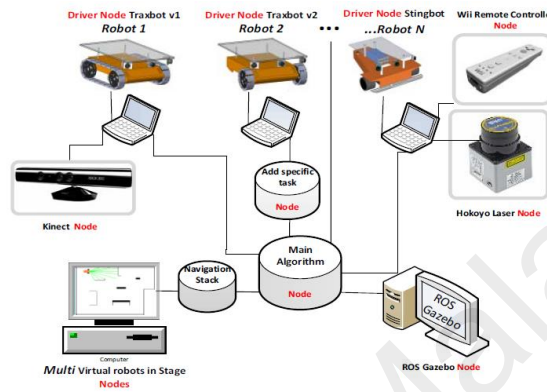


Figure 2.12: Distributed network of multiple robots, sensors and tele-operated devices

## CHAPTER 3: METHODOLOGY

### 3.1 Proposed Framework Block Diagram

#### 3.1.1 Network Architecture Block Diagram

In this project, master-slave type of network architecture will be implemented similar to the model implemented by (Araújo et al., 2013). Figure 3.1 shows the illustration of the implemented network. Only a single master and a single slave node is used. The entire project will be built upon the ROS framework. It is possible to add additional slave nodes to be controlled, but due to the scope of the project only a single slave node with a single robotic arm is used. The distributed network type of the master-slave model allows teleoperation of the robotic arm, within the boundaries of the wireless local area network (WLAN). It is also possible to expand the connection through the use of the internet, but it is beyond the scope of this project.

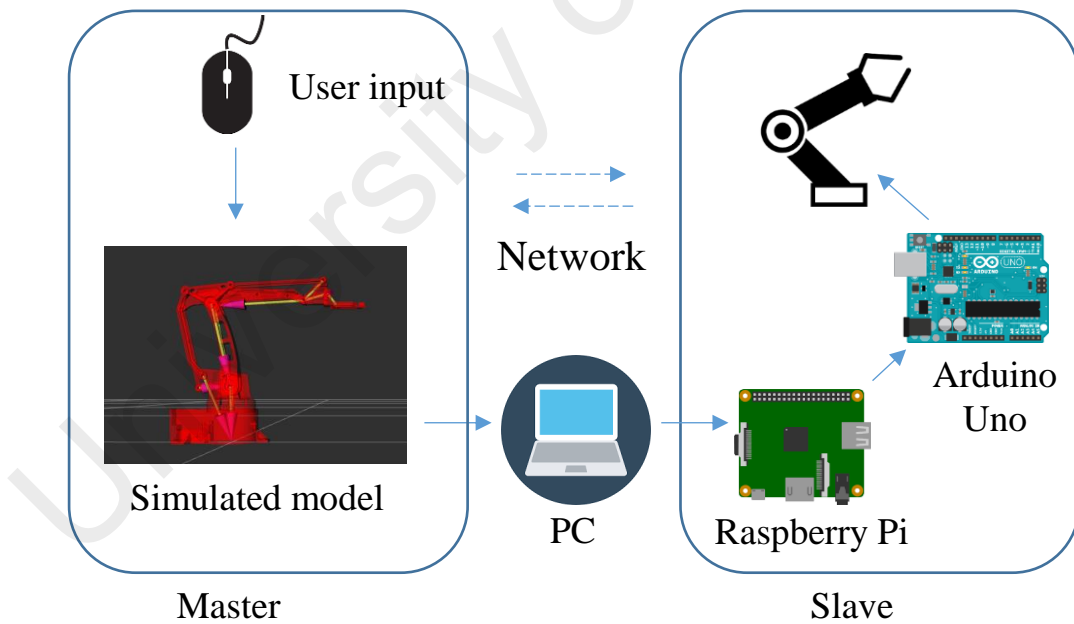


Figure 3.1: Block diagram of master-slave network architecture in project

The master node hosts the ROS master node, provides a GUI for receiving a target coordinate input from the user, performs calculation for the inverse kinematics, and

visualization of the robot model. The master node is hosted by a PC which is running on an Ubuntu operating system (OS). The installed ROS version is the Kinetic Kame.

The slave node is hosted by a Raspberry Pi embedded computer. The Raspberry Pi runs on the Ubuntu Mate OS, and the ROS Kinetic Kame is also installed. The Raspberry Pi is connected to an Arduino Uno board through the Universal Serial Bus (USB) connector which is responsible for controlling the servo motors on the robotic arm.

### 3.1.2 Software Architecture Block Diagram

Figure 3.2 shows the block diagram of the software architecture employed in this project. The rectangular boxes represent the topics, while the ovals represent the nodes where the nodes publish and subscribe messages. An explanation of the structure of ROS nodes and topics can be given below:

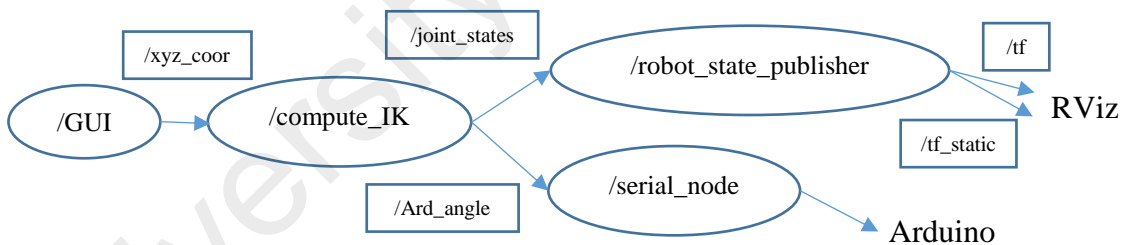


Figure 3.2: Block Diagram of Software Architecture

1. **/GUI** – Represents the GUI node that interacts with the user in order to obtain a target coordinate in the form of a X,Y and Z-axis coordinate points which will be published to the **/xyz\_coor** ROS topic. The GUI to be employed here is a simple slider where the user can control the X, Y and Z points individually.

2. **/compute\_IK** – The ROS node that receives the target coordinate point from the GUI via the **/xyz\_coor** topic. The computation for the inverse kinematics is performed on this node using the equations derived in section 3.3 Inverse Kinematic Analysis of Robotic Arm and section 3.4 Open-loop Chain Equivalent of Robotic Arm. The output angles for the robot model control is published to the **/Ard\_angle** topic, while the output angles for the robot visualization in RViz is published to the **/joint\_states** topic.
3. **/serial\_node** – The ROS node that is subscribed to the **/Ard\_angle** topic and is responsible for communication with the Arduino board.
4. **/robot\_state\_publisher** – The ROS node that is responsible for calculation of the vectors to represent the robot model in RViz. Calculates the necessary transformations needed based on the information on the URDF file and publishes the information to RViz via the **/tf** and **/tf\_static** topics.

Figure 3.3 shows the organisation of nodes between the Master and the Slave. All the computation is performed on the master node, which is the PC, while the slave node functions only to connect the Arduino board to the ROS environment and execute the servo actuations on the robot model. The Master contains the **GUI**, **robot\_state\_publisher** and the **compute\_IK** nodes, while the Slave contains the **serial\_node**. However, all these nodes appear to be on the same environment due to the use of the same ROS Master. When creating the nodes, the nodes are explicitly subscribed to the ROS Master which is hosted on the Master by specifying the **ROS\_MASTER\_URI**. This enables the ROS environment to be distributed across multiple machines but functioning as a single environment.



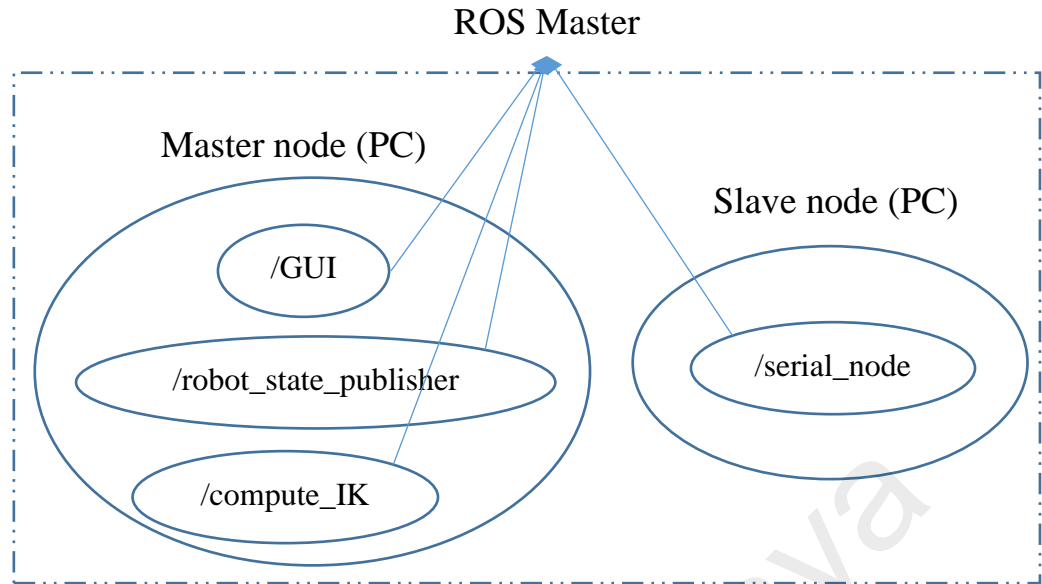


Figure 3.3: Node Organisation of Master and Slave

### 3.2 Robotic Arm Model

In this project, the EezybotArm Mk2 robotic arm is selected for use. Figure 3.4 shows the CAD model of the manipulator. The EezybotArm mk2 is an open source robotic arm design that was modelled after the ABB IRB460 robotic arm. It has 3 DOF, and a gripper end-effector. It uses a closed-loop kinematic design, and RC servo motors for actuation. The robotic arm is designed to be fabricated by additive manufacturing or 3-Dimensional (3D) printing. It consists of 19 3D printed parts, and 55 non-printed parts. A breakdown of the parts is given below in Table 1 and Table 2. An exploded view detailing the 3D printed parts is given in Figure 3.5.

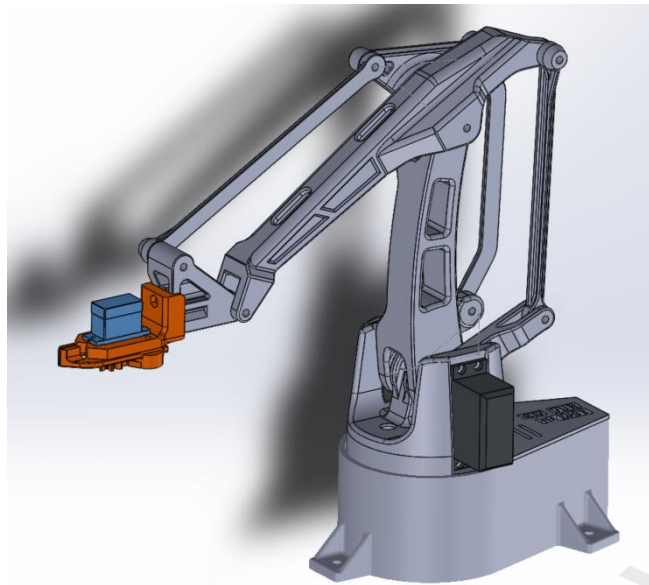


Figure 3.4: CAD model image of EezybotArm Mk 2

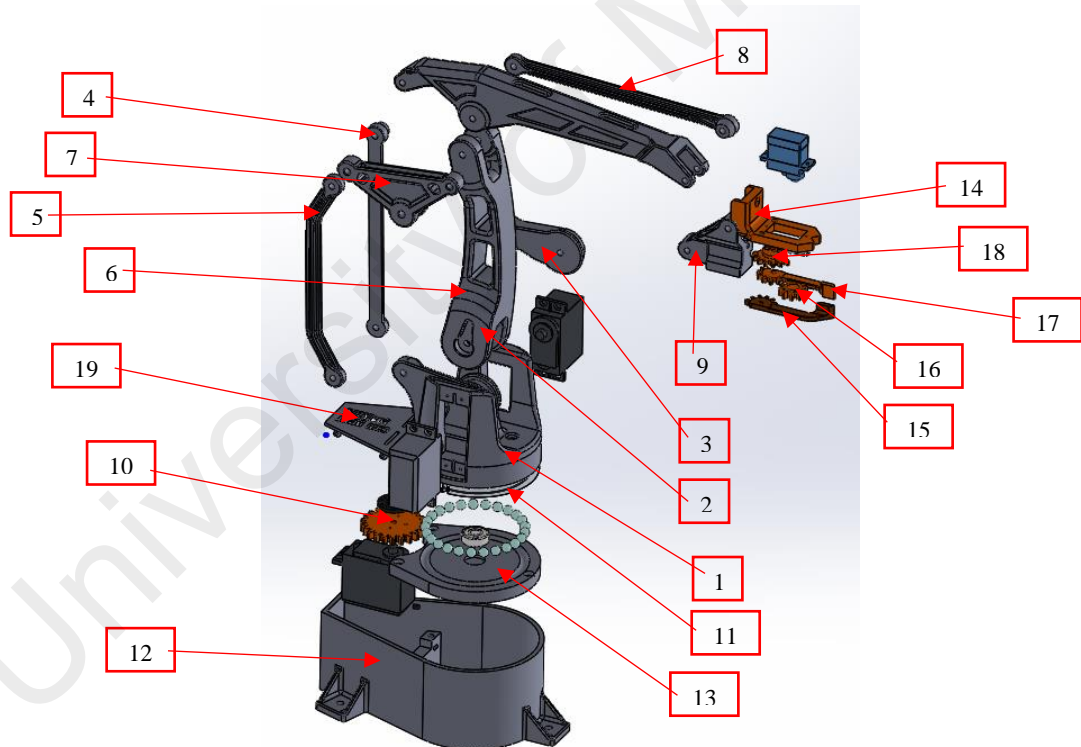


Figure 3.5: Exploded view of robotic arm

Table 1: 3D printed parts file list

No	Item Name	Quantity
1	001_base	1
2	002_mainarm	1
3	003_VArm	1
4	004_link135	1

5	005_link135angled	1
6	006_horarm	1
7	007_trialink	1
8	008_link147	1
9	009_trialinkfront	1
10	010_gearservo	1
11	011_gearmast	1
12	012_base	1
13	013_lowerbase	1
14	014_clawbase	1
15	015_clawfingersx	1
16	016_gear	1
17	017_clawfingersx	1
18	018_clawgeardriven	1
19	019_cover	1

Table 2: Table of standard parts

No	Item name	Quantity
1	MG995 servo	3
2	SG90 servo	1
3	M6 selflocking nut	1
4	M6x25 screw	1
5	M3 selflocking nut	2
6	M3x20 screw	2
7	M2x10 hex recessed head screw	1
8	M4 selflocking nut	9
9	M4x40 screw	1
10	M4x30 screw	1
11	M4x20 screw	5
12	M4x60 threaded rod	1
13	M4x32 threaded rod	1
14	Φ6 mm ball sphere	25
15	606 bearing	1
16	M4 washer	miscellaneous

### 3.3 Inverse Kinematic Analysis of Robotic Arm

In this project, the robotic arm should be able to bring the end-effector to a desired location given a set of coordinate points as the input. In order to achieve this, the inverse kinematics of the EezybotArm Mk2 must be analysed. This section explains in further details the derivation of the inverse kinematics.

First, the home or default position where all the joint angles are at  $0^\circ$  is taken as shown in Figure 3.6. The coordinate frame is then assumed as follows: x-axis points to the right along the robotic arm, y-axis goes into the page, and z-axis points upwards with a reference coordinate frame located at the position indicated in Figure 3.6. This follows the 'right-hand rule' convention for the axis naming. Next, a geometric analysis of the components is conducted.

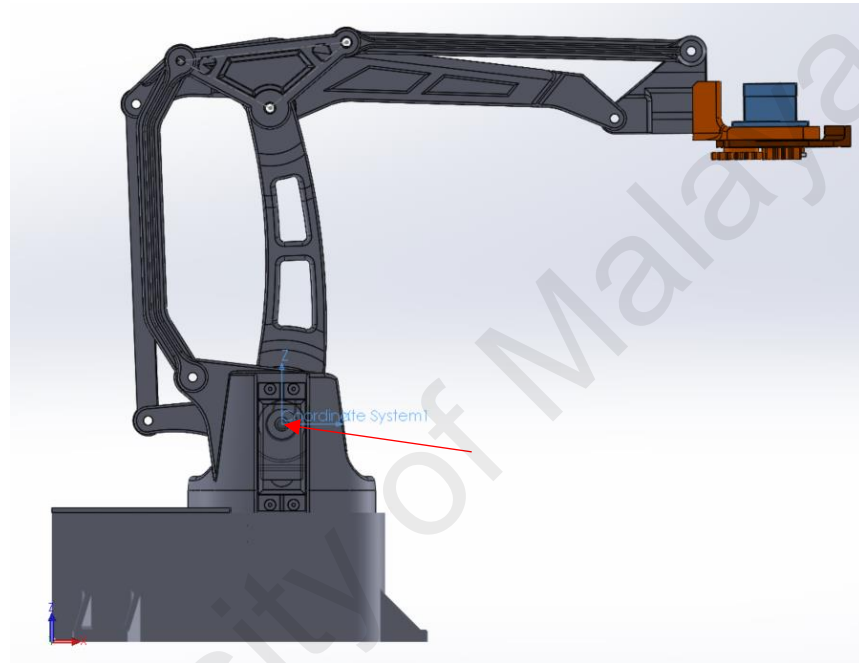


Figure 3.6: Robotic arm at home position

Figure 3.7 shows the joint variables to be controlled in the robotic arm model. The joint variable  $\theta_1$  is a rotation of the robot base piece about the z-axis. The joint variable  $\theta_4$  is the rotation of the robot main arm piece about the y-axis. Lastly, the joint variable  $\theta_5$  is the rotation of the robot V-arm piece about the y-axis.

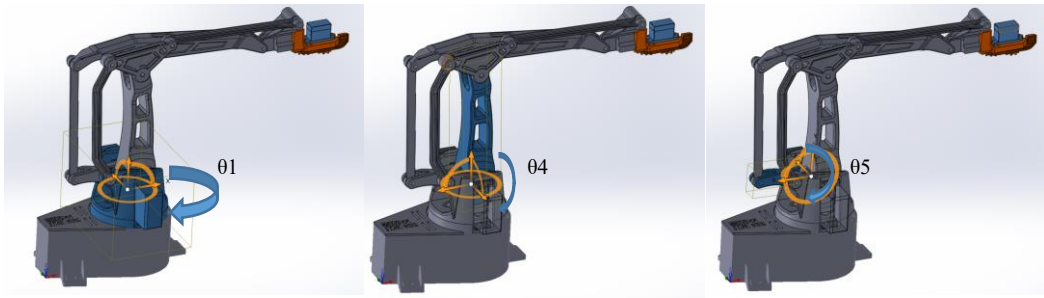


Figure 3.7: Joint variables for the robotic arm

In order to simplify the analysis of the robotic arm, the model of the robotic arm can be simplified geometrically into a set of equivalent lines as shown in Figure 3.8.

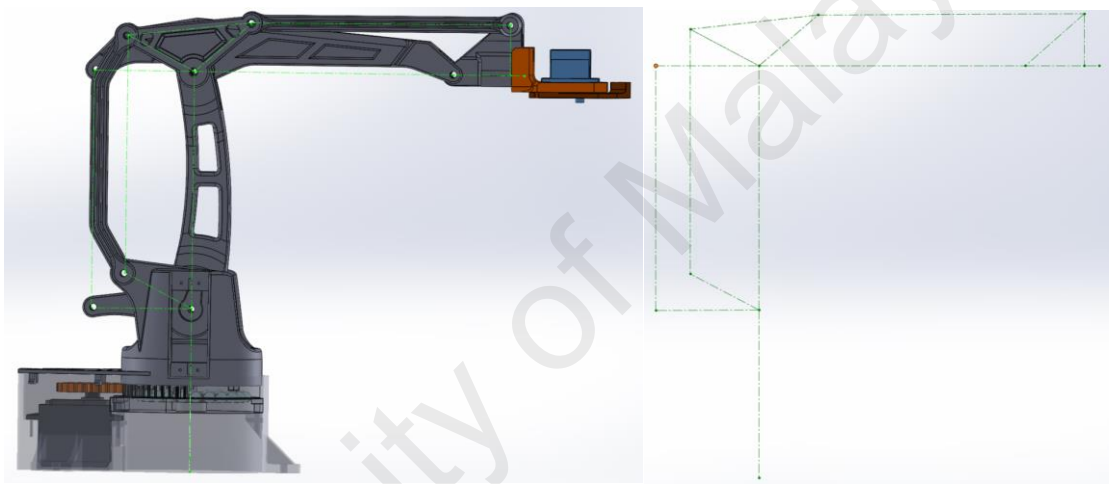


Figure 3.8: Geometric analysis of robotic arm

Given a desired coordinate point for the end-effector location, the coordinate point can be broken down into several geometric components as shown in Figure 3.9. The coordinate point is described by the projection in the X, Y and Z axis.

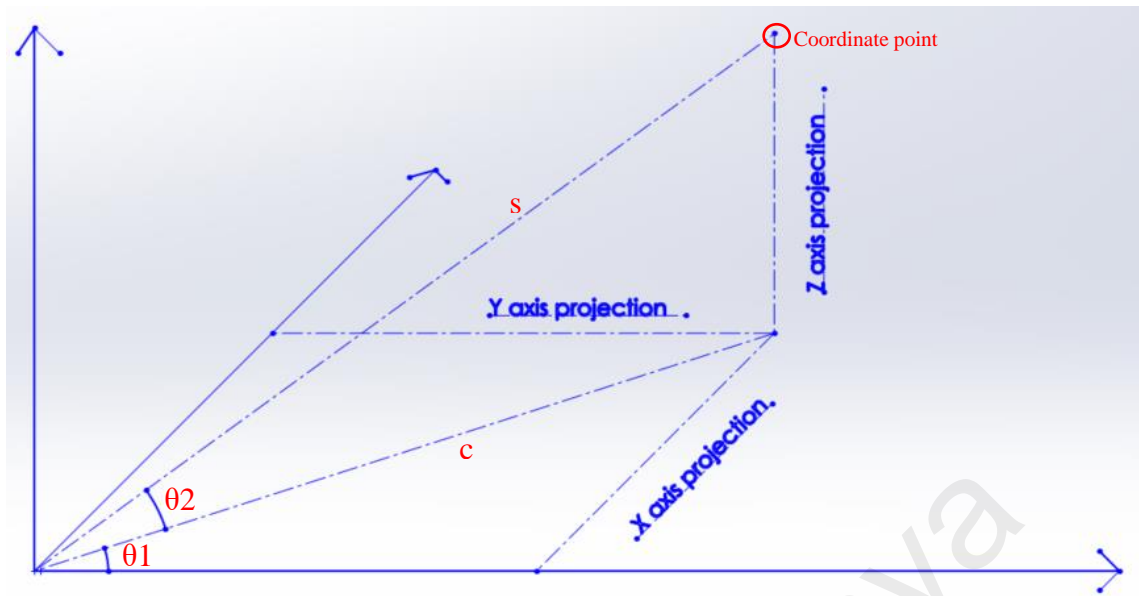


Figure 3.9: Breakdown of geometric components

Besides that, the point can also be described by a rotation  $\theta_1$  about the Z-axis, and planar projection entities  $c$  and  $s$ . The equations for  $c$ ,  $s$ , and  $\theta_1$  can be given as follows:

$$\begin{aligned} c^2 &= x^2 + y^2 \\ c &= \sqrt{x^2 + y^2} \end{aligned} \quad (3.1)$$

$$\begin{aligned} s^2 &= c^2 + z^2 \\ &= x^2 + y^2 + z^2 \\ s &= \sqrt{x^2 + y^2 + z^2} \end{aligned} \quad (3.2)$$

$$\theta_1 = \cos^{-1} \left( \frac{y}{c} \right) \quad (3.3)$$

$$\theta_2 = \cos^{-1} \left( \frac{c}{s} \right) \quad (3.4)$$

Figure 3.10 shows the derivation of the  $s$ ,  $c$ ,  $z$  and  $\theta_2$  variables in the planar robot frame.  $\theta_4$  and  $\theta_5$  represent are two of the joint variables that are to be controlled, and  $\theta_3$  is the  $90^\circ$  complement angle of the sum of  $\theta_2$  and  $\theta_4$ . The last portion of the robotic arm remains parallel to the ground in every configuration due to the parallel links being used in the design. Therefore, the kinematic analysis is performed up to the end-effector wrist to simplify the analysis.

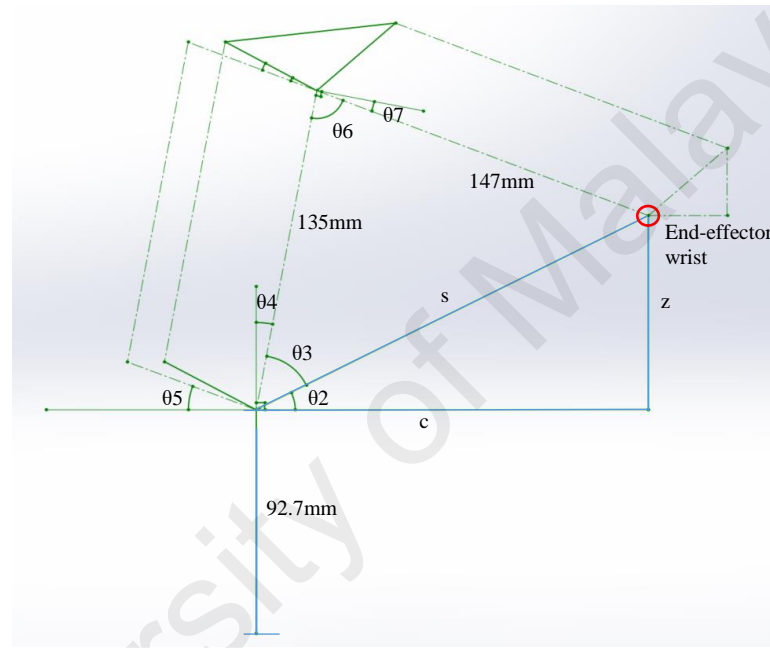


Figure 3.10: Derivation of variables in robot frame

Given Figure 3.10, the variables can then be calculated by the following equations:

Using cosine rule,

$$147^2 = s^2 + 135^2 - 2(s)(135)(\cos\theta_3)$$

$$\cos\theta_3 = \frac{147^2 - s^2 - 135^2}{-2(s)(135)}$$

$$\theta_3 = \cos^{-1}\left(\frac{147^2 - s^2 - 135^2}{-2(s)(135)}\right)$$

$$\theta_4 = 90^\circ - \theta_2 - \theta_3$$

(3.5)

$$s^2 = 147^2 + 135^2 - 2(147)(135)(\cos\theta_6)$$

$$\theta_6 = \cos^{-1} \left( \frac{s^2 - 147^2 - 135^2}{-2(147)(135)} \right)$$

$$\theta_7 = 90^\circ - \theta_6$$

(3.6)

From the illustration shown in Figure 3.11, the joint variable  $\theta_5$  can be derived through a use of a series of solutions for complementary angles and parallel lines.

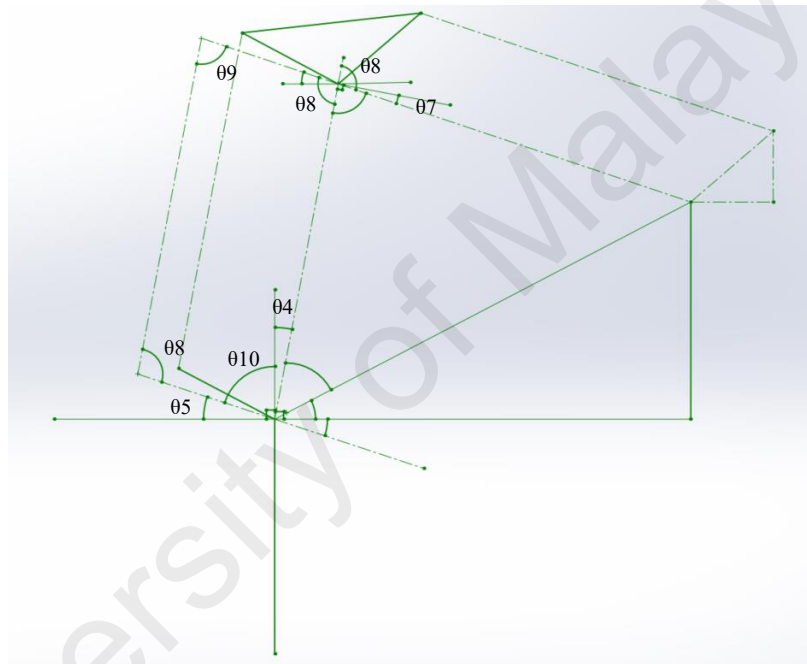


Figure 3.11: Derivation of  $\theta_5$

The derivations will be further outlined in the equations below:

$$\theta_8 = 90^\circ - \theta_7$$

$$\theta_9 = 180^\circ - \theta_8$$

$$\theta_{10} = \theta_9 - \theta_4$$

$$\theta_5 = 90^\circ - \theta_{10}$$



$$\theta_5 = 90^\circ - ((180^\circ - \theta_8) - \theta_4)$$

$$\theta_5 = 90^\circ - ((180^\circ - (90^\circ - \theta_7)) - \theta_4)$$

$$\theta_5 = \theta_4 - \theta_7 \quad (3.7)$$

Table 3 shows the summary of the equations used to calculate the joint angles to be controlled from the inverse kinematics analysis. With these joint variables, control is able to be performed to bring the end-effector to any location within the robot workspace.

*Table 3: Summary of joint variable equations*

Joint variable	Description	Equation
$\theta_1$	Base rotation about the z-axis	$\theta_1 = \cos^{-1}\left(\frac{y}{c}\right)$
$\theta_4$	Mainarm rotation about the y-axis	$\theta_4 = 90^\circ - \theta_2 - \theta_3$
$\theta_5$	Varm rotation about the y-axis	$\theta_5 = \theta_4 - \theta_7$

### 3.4 Open-loop Chain Equivalent of Robotic Arm

The ROS visualization tool RViz does not support the mechanics of closed-loop chain type manipulators. Therefore, in order to successfully visualise the EezybotArm Mk2 within RViz, the robotic arm model must be converted to an equivalent group of open-loop chains. The joint angles from the open-loop equivalent chains will then be used back to control the visualized model in RViz in order for it to properly display the correct configurations given the joint angles  $\theta_1$ ,  $\theta_4$  and  $\theta_5$ . Figure 3.12 shows the open-loop chain equivalent of the robotic arm model. Next, the 4 open-loop kinematic chains will be analysed.

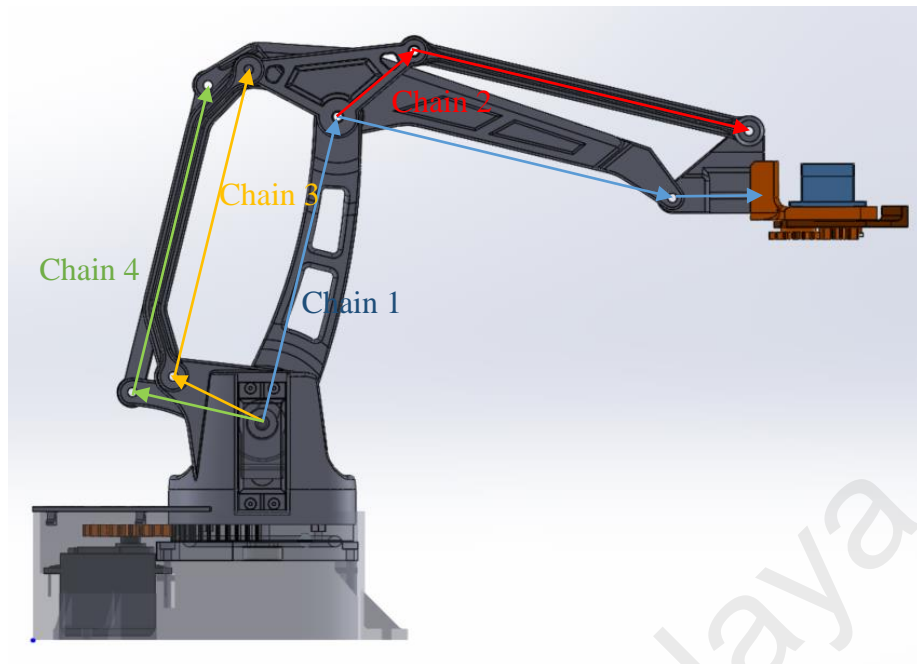


Figure 3.12: Open-loop chain equivalent of closed-loop chain

### 3.4.1 Chain 1

Figure 3.13 shows the joint angles needed for the visualisation of chain 1 within RViz. The joint angles  $\theta_4$  and  $\theta_7$  have already been found in section 3.3 Inverse Kinematic Analysis of Robotic Arm. The angle at the last joint of the end-effector can be found by solving for parallel lines. Then, it can be seen that the angle is equivalent to the angle of joint variable  $\theta_5$ . Table 4 shows a summary of the equations for the joint variables in Chain 1.

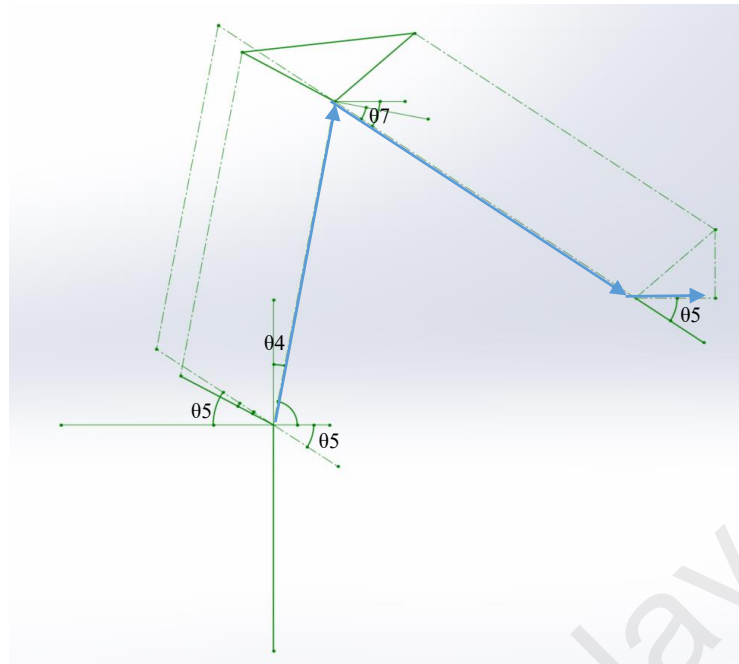


Figure 3.13: Joint angles for Chain 1

Table 4: Joint variables for Chain 1

Joint variable	Equation
$\theta_4$	$\theta_4 = 90^\circ - \theta_2 - \theta_3$
$\theta_5$	$\theta_5 = \theta_4 - \theta_7$
$\theta_7$	$\theta_7 = 90^\circ - \theta_6$

### 3.4.2 Chain 2

Figure 3.14 shows the joint angles needed for chain 2. From illustration, it can be seen that the two joint variables that are governing chain 2 are variables  $\theta_4$  and  $\theta_7$ . The first link of the chain is the mainarm piece that rotates by the angle  $\theta_4$ . The joint variable for the second link is then a counter rotation of  $\theta_4$ , and the last link rotates by joint variable  $\theta_7$ . Table 5 shows a summary of the equations to be used in Chain 2.

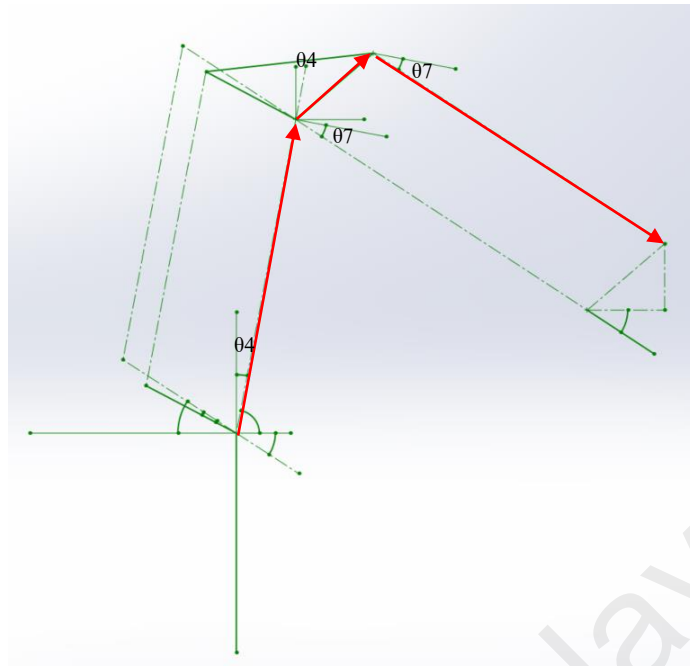


Figure 3.14: Joint angles for Chain 2

Table 5: Joint variables for Chain 2

Joint variable	Equation
$\theta_4$	$\theta_4 = 90^\circ - \theta_2 - \theta_3$
$\theta_7$	$\theta_7 = 90^\circ - \theta_6$

### 3.4.3 Chain 3

Figure 3.15 shows the joint variable needed to represent the open-loop chain 3. The first link in the chain is fixed, and the second link is parallel to that of the robot mainarm due to geometric constraints, therefore it can be represented by the joint variable  $\theta_4$ , as given in equation (3.5) .

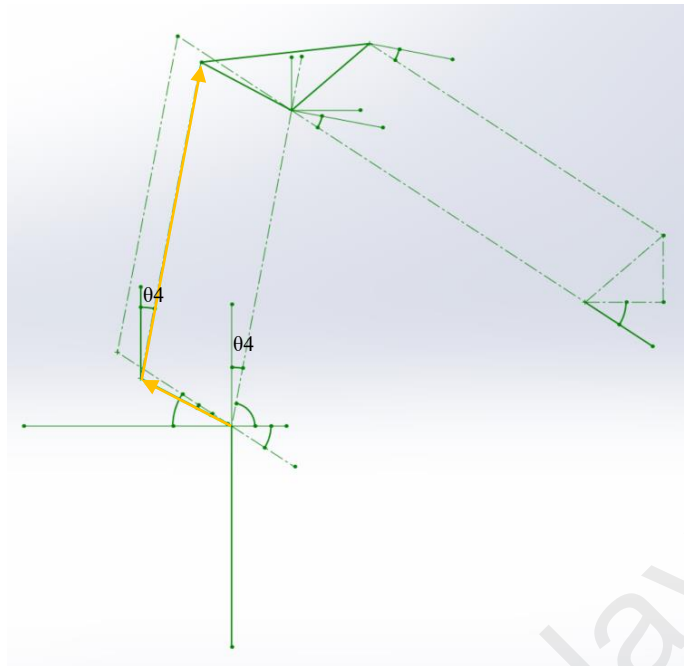


Figure 3.15: Joint angles for Chain 3

#### 3.4.4 Chain 4

Figure 3.16 shows the joint variables used in Chain 4. The first link in the chain is controlled by the joint variable  $\theta_5$ . The second link is then controlled by the joint variable  $\theta_{12}$ .

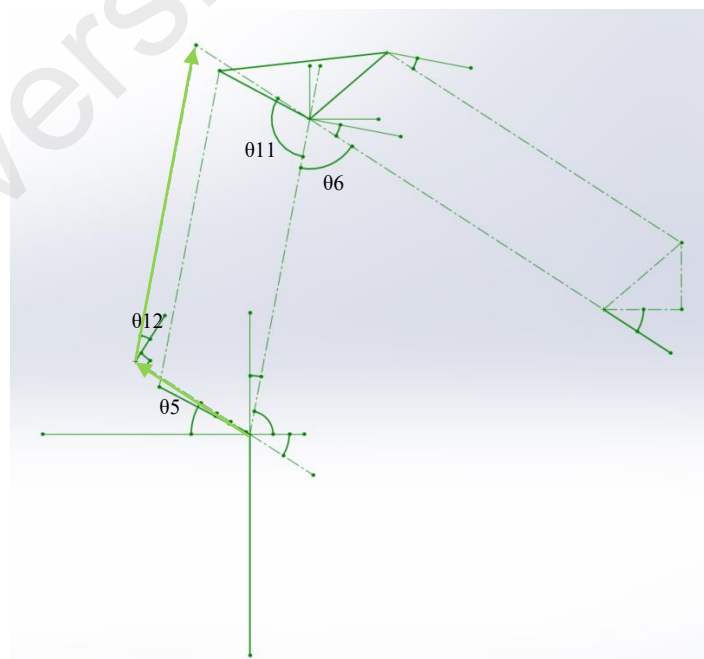


Figure 3.16: Joint variables for Chain 4

The derivation for the joint variable  $\theta_{11}$  can be described as:

$$\theta_{11} = 180^\circ - \theta_6 \quad (3.8)$$

$$\begin{aligned} \theta_{12} &= \theta_{11} - 90^\circ \\ &= 90^\circ - \theta_6 \end{aligned} \quad (3.9)$$

### 3.5 HTM Derivation of Open-loop Chain by DH Convention

The DH parameters can be derived for the open-loop chain equivalents in order to solve the FK analysis. Implicitly in RViz, this analysis is done based on the configurations provided in the URDF file. Then, based on the results of the FK calculations, RViz is able to provide the correct visualization of the configuration of the robotic arm based on the joint variables provided.

#### 3.5.1 HTM of Chain 1

Figure 3.17 and Figure 3.18 show the DH Convention analysis for Chain 1. Coordinate frames 0 and 1 lie directly on top of one another, coordinate frame 2 is offset from coordinate frame 1 along the Y1-axis, coordinate frame 3 is offset from coordinate frame 2 along the X2-axis, and coordinate frame 4 is offset from coordinate frame 3 along the X3-axis. The derivation of the DH parameters can be given in Table 6.

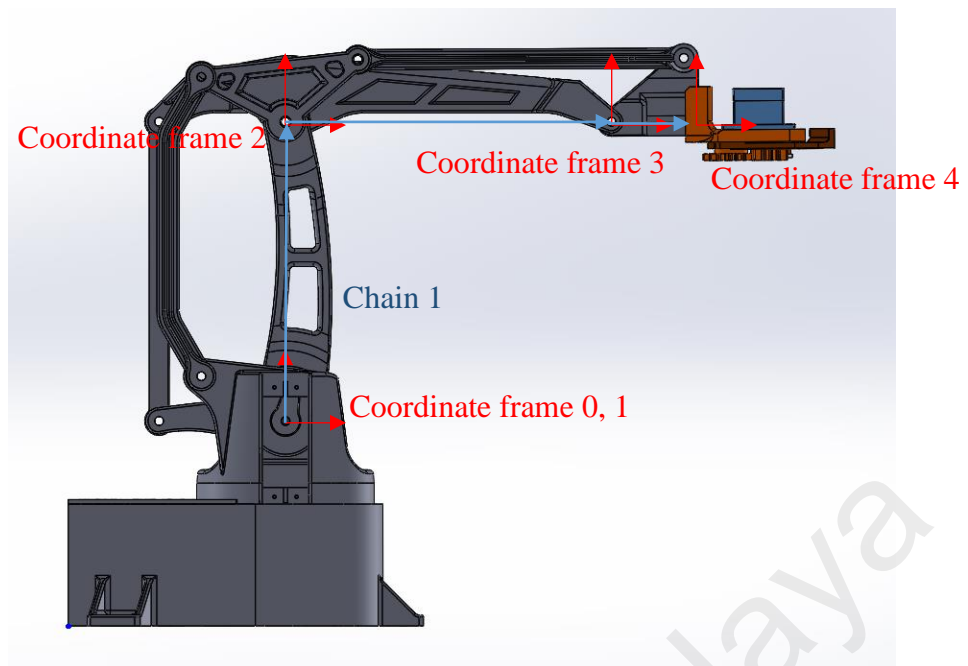


Figure 3.17: Frame assignment for Chain 1

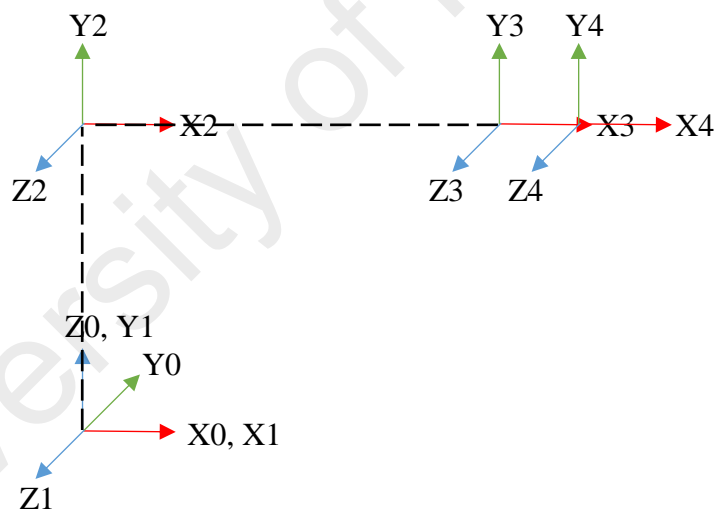


Figure 3.18: Axis of frames for Chain 1

Table 6: DH parameters derivation for Chain 1

	$\theta$	$\alpha$	a	d
1	$\theta_1$	90	0	0
2	$\theta_4$	0	0	0
3	$\theta_7$	0	0.147	0
4	$\theta_5$	0	0.0393	0

Based on the DH parameters found in

Table 6, the derivation of the HTM can then be given as:

$$\begin{aligned}
 HTM_4^0 &= \begin{bmatrix} c_{\theta_1} & 0 & s_{\theta_1} & 0 \\ s_{\theta_1} & 0 & -c_{\theta_1} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_{\theta_4} & -s_{\theta_4} & 0 & 0 \\ s_{\theta_4} & c_{\theta_4} & 0 & 0 \\ 0 & s_{\alpha_4} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 &\begin{bmatrix} c_{\theta_7} & -s_{\theta_7} & 0 & 0.147c_{\theta_7} \\ s_{\theta_7} & c_{\theta_7} & 0 & 0.147s_{\theta_7} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_{\theta_5} & -s_{\theta_5} & 0 & 0.0393c_{\theta_5} \\ s_{\theta_5} & c_{\theta_5} & 0 & 0.0393s_{\theta_5} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.10)
 \end{aligned}$$

### 3.5.2 HTM of Chain 2

Figure 3.19 and Figure 3.20 and show the DH Convention analysis for Chain 2. Coordinate frames 0 and 1 lie directly on top of one another, coordinate frame 2 is offset from coordinate frame 1 along the Y1-axis, coordinate frame 3 is offset from coordinate frame 2 along the X2-axis, and coordinate frame 4 is offset from coordinate frame 3 along the X3-axis. The derivation of the DH parameters can be given in Table 7.



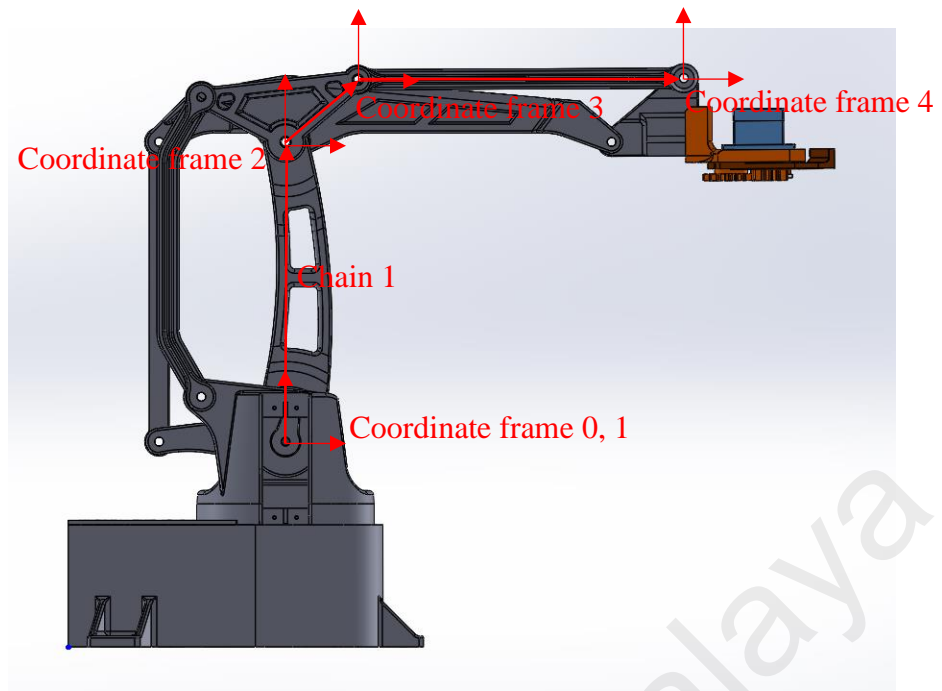


Figure 3.19: Frame assignment for Chain 2

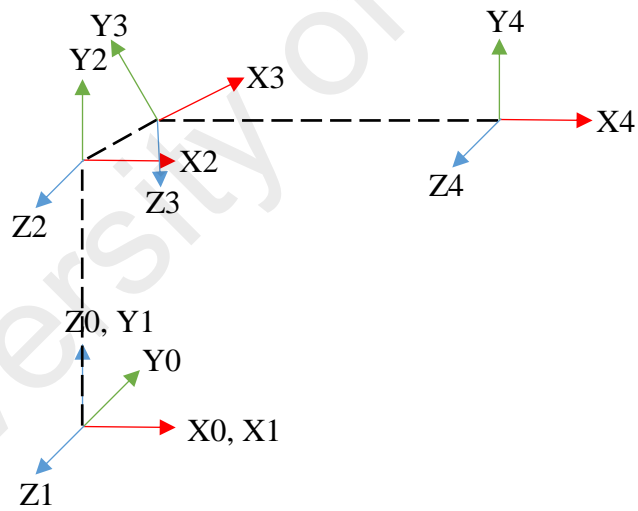


Figure 3.20: Axis of frames for Chain 2

Table 7: DH Parameters derivation for Chain 2

	$\theta$	$\alpha$	a	d
1	$\theta_1$	90	0	0
2	$\theta_4$	0	0	0
3	$40.70 + \theta_7$	0	0.043	0
4	$-40.70 + \theta_7$	0	0.147	0

Based on the DH parameters found in Table 7, the derivation of the HTM can then be given as:

$$HTM_4^0 = \begin{bmatrix} c_{\theta_1} & 0 & s_{\theta_1} & 0 \\ s_{\theta_1} & 0 & -c_{\theta_1} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_{\theta_4} & -s_{\theta_4} & 0 & 0 \\ s_{\theta_4} & c_{\theta_4} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} c_{40.70 + \theta_7} & -s_{40.70 + \theta_7} & 0 & 0.043c_{40.70 + \theta_7} \\ s_{40.70 + \theta_7} & c_{40.70 + \theta_7} & 0 & 0.043s_{40.70 + \theta_7} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0.147c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i} & 0 & 0.147s_{\theta_i} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.11)$$

### 3.5.3 HTM of Chain 3

Figure 3.21 and Figure 3.22 show the DH Convention analysis for Chain 3. Coordinate frame 1 is offset from coordinate frame 0 along both the X0 and Z0 axis. Coordinate frame 2 is offset from coordinate frame 1 along the Y1 axis. The derivation of the DH parameters can be given in Table 8.

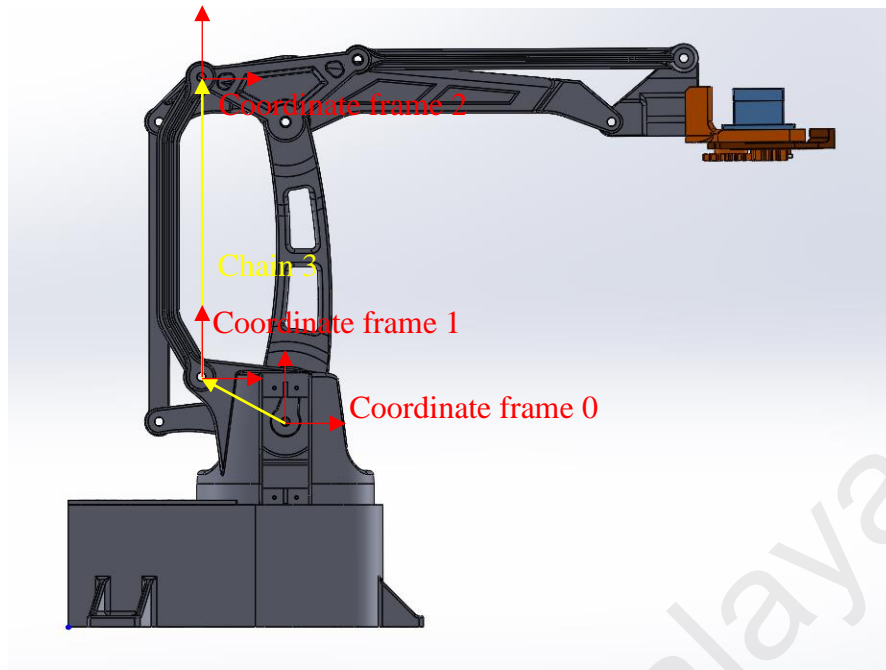


Figure 3.21: Frame assignment for Chain 3

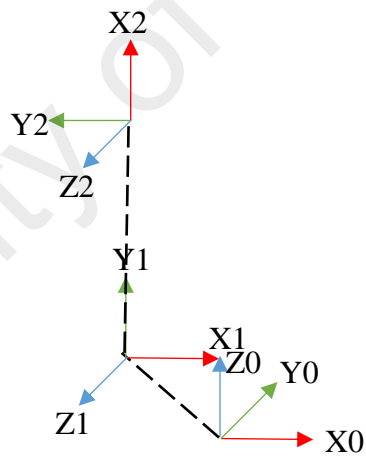


Figure 3.22: Axis of Frames for Chain 3

Table 8: DH parameters derivation for Chain 3

	$\theta$	$\alpha$	$a$	$d$
1	$\theta_1$	-90	-0.038	0.0202
2	$\theta_4$	0	0.135	0

Based on the parameters derived in Table 8, the HTM can be given as:

$$HTM_2^0$$

$$= \begin{bmatrix} c_{\theta_1} & 0 & -s_{\theta_1} & -0.038c_{\theta_1} \\ s_{\theta_1} & 0 & c_{\theta_1} & -0.038s_{\theta_1} \\ 0 & -1 & 0 & 0.0202 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -s_{\theta_4} & 0 & 0.135c_{\theta_4} \\ 0 & c_{\theta_4} & 0 & 0.135s_{\theta_4} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.12)$$

### 3.5.4 HTM of Chain 4

Figure 3.23 and Figure 3.24 show the DH Convention analysis for Chain 4. Coordinate frame 1 lies directly on top of coordinate frame 0. Coordinate frame 2 is offset from coordinate frame 1 along the X1 axis. Coordinate frame 3 is offset from coordinate frame 1 along the Y2 axis. The derivation of the DH parameters can be given in Table 9.

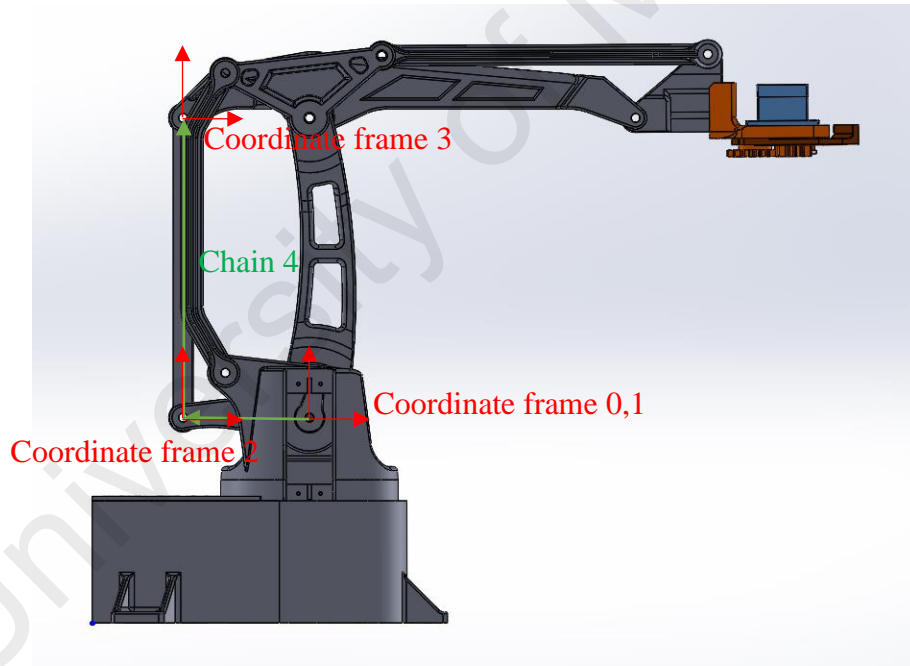


Figure 3.23: Frame assignment for Chain 4

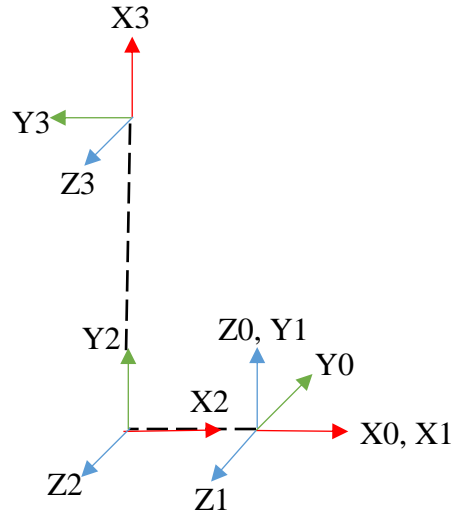


Figure 3.24: Frame assignment for Chain 4

Table 9: DH parameters derivation for chain 4

	$\theta$	$\alpha$	a	d
1	$\theta_1$	90	0	0
2	$\theta_4$	0	0.057	0
3	$90 + \theta_{12}$	0	0.135	0

Based on the parameters derived in Table 9, the HTM can be given as:

$$\begin{aligned}
 HTM_3^0 &= \begin{bmatrix} c_{\theta_1} & 0 & s_{\theta_1} & 0 \\ s_{\theta_1} & 0 & -c_{\theta_1} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_{\theta_4} & -s_{\theta_4} & 0 & 0.057c_{\theta_4} \\ s_{\theta_4} & c_{\theta_4} & 0 & 0.057s_{\theta_4} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 &\quad \begin{bmatrix} c_{90+\theta_{12}} & -s_{90+\theta_{12}} & 0 & 0.135c_{90+\theta_{12}} \\ s_{90+\theta_{12}} & c_{90+\theta_{12}} & 0 & 0.135s_{90+\theta_{12}} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.13)
 \end{aligned}$$

### 3.6 Electrical Wiring Diagram

Figure 3.25 shows the system-level connections in this project. The PC which acts the master node is supplied with a 19V power supply. A mouse is attached as a peripheral to the PC as an input device for the user. The PC also functions to provide the visualization through RViz of the current configuration of the robotic arm. The PC is wirelessly connected to the WLAN. The PC is wirelessly connected to the WLAN.

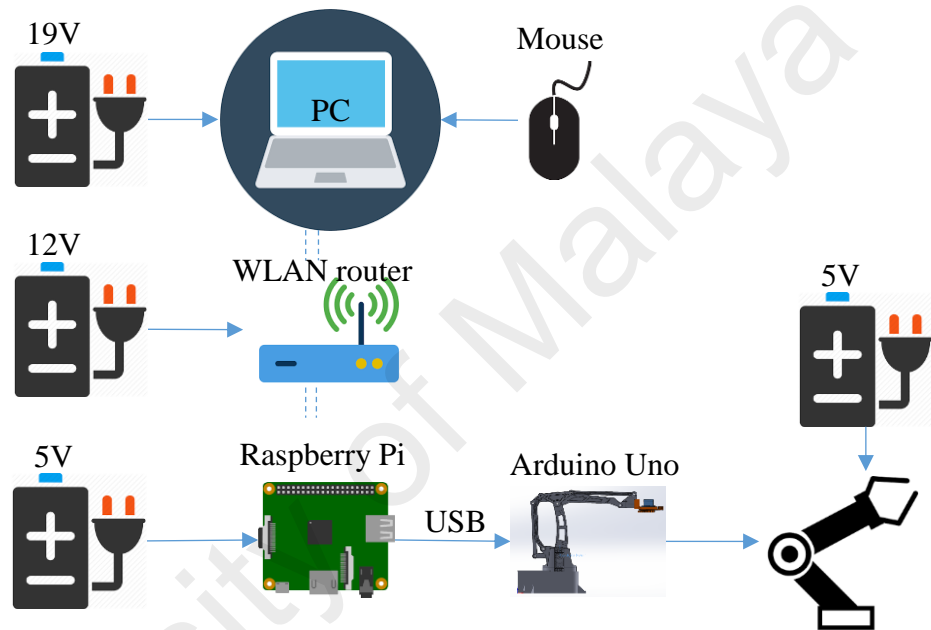


Figure 3.25: System-level connections

Besides that, there is a WLAN router that is connected with a 12V power supply. The WLAN router provides the WLAN connection that allows communication between the PC and the Raspberry Pi.

The Raspberry Pi is powered by a 5V power supply. It acts as the slave node in the network and is connected to the Arduino board via a USB cable. Like the master node, it is wirelessly connected to the WLAN. The Arduino Uno is then connected to the servo motors on the robotic arm which are powered by 5V power supply.

Figure 3.26 shows the circuit connections on the Arduino Uno board. The Arduino Uno board is connected to the Raspberry Pi via the USB cable. This USB connection allows the Raspberry Pi to transmit the information on the desired servo motor angles to the Arduino Uno board. Also, the Arduino Uno board draws the required power from this USB connection. The Arduino Uno is connected to 4 servo motors that are located on the robotic arm through the digital I/O pins 8, 9, 10, and 11. These digital I/O pins transmit the control PWM signal to the servo motors. The servo motors are powered by a separate 5V power supply, and the neutral pin on the servo motors are grounded back to the GND pin of the Arduino Uno board.

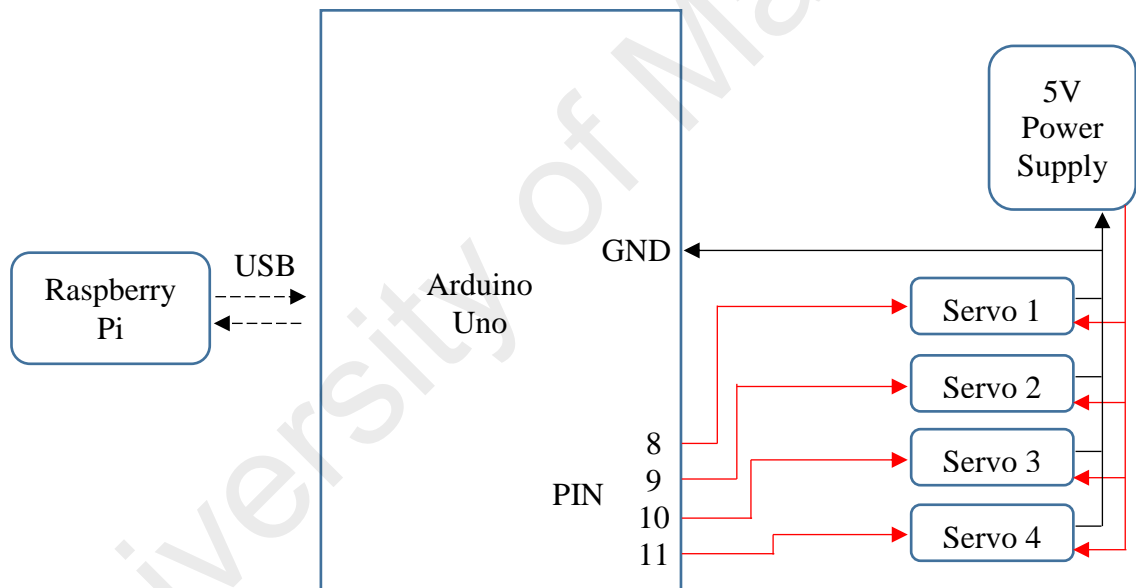


Figure 3.26: Connections on the Arduino Uno board

## CHAPTER 4: IMPLEMENTATION AND RESULTS

### 4.1 Implementation of Project

#### 4.1.1 Overall Project

Figure 4.1 shows the implementation of the actual project. On the PC, there is a GUI that allows the user to given an input of the desired coordinate point of the robot arm end-effector. Then, a simulation of the configuration of the robot arm is presented to the user. The data on the required joint angles to achieve the configuration is transmitted wirelessly through the WLAN router to the Raspberry Pi. The Raspberry then interfaces with the Arduino Uno board in order to control the servo motors at the robot arm joints. The robot arm was fabricated using additive manufacturing and then assembled together manually.

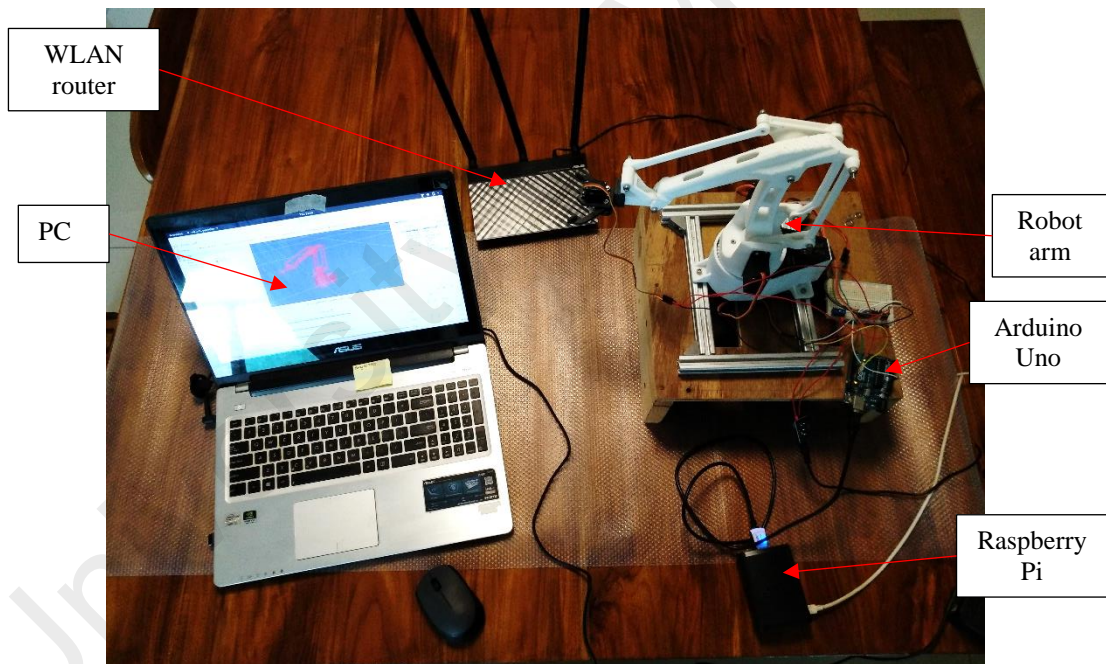


Figure 4.1: Actual project layout

#### 4.1.2 Organisation of Files

Figure 4.2 shows the organisation of files in the ROS system. The files on the ROS system are organised using *catkin* workspaces. In Figure 4.2, the *catkin* workspace used is named as *catkin\_ws*. The under each workspace, it is further subdivided into different folders



such as *src*, *build*, and *devel*. The *src* subfolder contains the different packages which are make up the ROS project. The software package contains different modules that provide additional functionality to the system. All developed code for the project is stored in the packages.

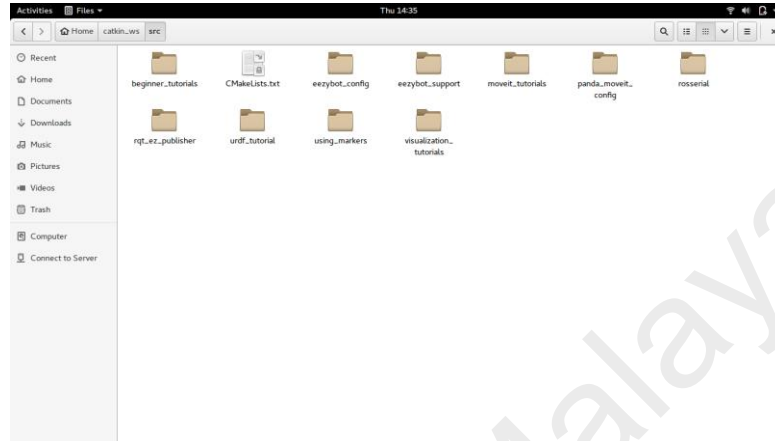


Figure 4.2: Organisation of files in ROS

#### 4.1.3 System Boot-up

This section outlines the procedure in booting-up the system. First, the master PC is switched on. The PC is then connected to the LAN called *ROSNetwork*. Then, a new terminal is launched, and the current directory is switched to *catkin\_ws* which contains all the project code as shown in Figure 4.3.

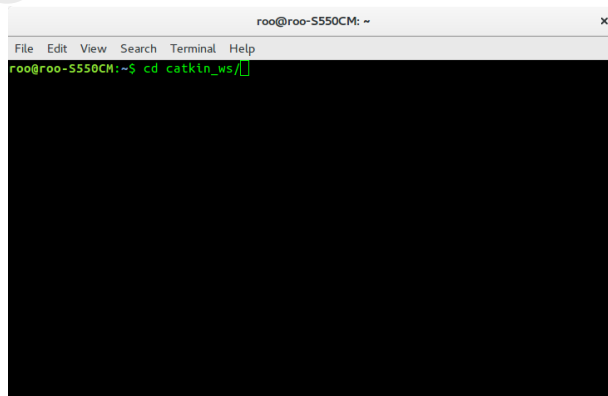
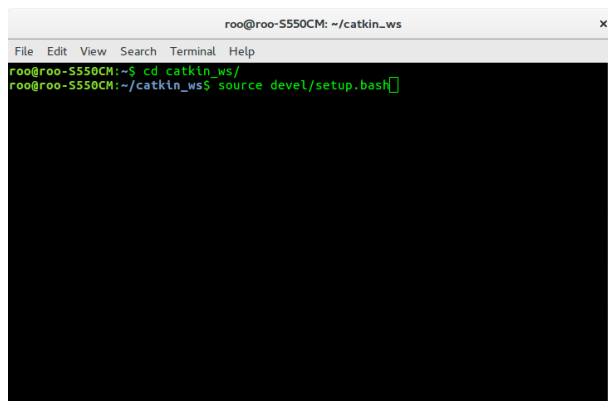


Figure 4.3: Launch new terminal

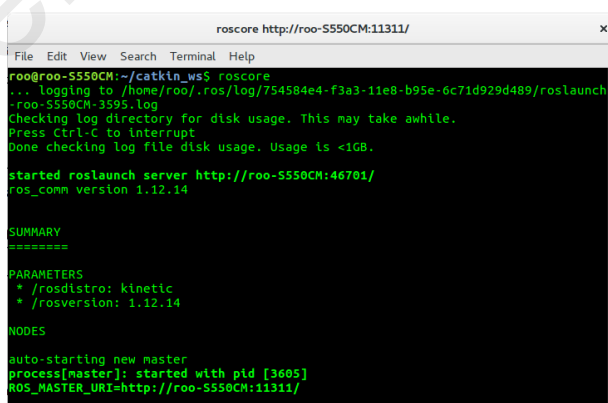
Next, the configuration files for the catkin workspace needs to be sourced by using the command `'source devel/setup.bash'`. The `setup.bash` file provides all the necessary information to locate the packages within the workspace. Every time a new terminal is launched, the current directory needs to be changed to `catkin_ws` and the `setup.bash` file be sourced again as shown in Figure 4.4.



```
roo@roo-S550CM: ~/catkin_ws
File Edit View Search Terminal Help
roo@roo-S550CM:~$ cd catkin_ws/
roo@roo-S550CM:~/catkin_ws$ source devel/setup.bash
```

Figure 4.4: Sourcing configuration file

When initializing the framework, the first piece of software that needs to be started is the ROS master. Without it, all other ROS nodes would not be able to communicate with one another. The ROS master is hosted on the Master PC, and can be initialized with the `roscore` command as shown in Figure 4.5.



```
roscore http://roo-S550CM:11311/
File Edit View Search Terminal Help
roo@roo-S550CM:~/catkin_ws$ roscore
... logging to /home/roo/.ros/log/754584e4-f3a3-11e8-b95e-6c71d929d489/roslaunch
-roo-S550CM-3595.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://roo-S550CM:46701/
ros_comm version 1.12.14

SUMMARY
=====
PARAMETERS
* /rostdistro: kinetic
* /rosversion: 1.12.14

NODES
auto-starting new master
process[master]: started with pid [3605]
ROS_MASTER_URI=http://roo-S550CM:11311/
```

Figure 4.5: Starting the ROS Master

Next a new terminal is launched and the terminal with the ROS master is kept running in the background. In the ROS architecture, each terminal is generally used to hold a

separate node. The ROS\_IP of the master PC is exported, so that the node would become discoverable by other nodes distributed across the slave. A static IP address of 192.168.1.148 is assigned to the master PC. The inverse kinematics calculator node is launched from the *printxyz.py* file in the *beginner\_tutorials* package using the *roslaunch* command as shown in Figure 4.6.

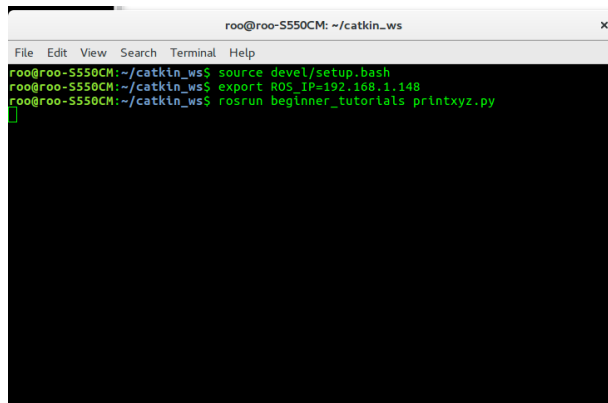
A terminal window titled 'roo@roo-S550CM: ~/catkin\_ws' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows three lines of commands: 'source devel/setup.bash', 'export ROS\_IP=192.168.1.148', and 'roslaunch beginner\_tutorials printxyz.py'. The prompt is 'roo@roo-S550CM:~/catkin\_ws\$'.

Figure 4.6: Initializing IK node

Next, another new terminal is launched. Then, the *roslaunch* command is used to launch the RViz visualization tool and the slider GUI as shown in Figure 4.7. The *display.launch* file is simply a collection of nodes and services to be launched simultaneously. The launch file is located in the *urdf\_tutorial* directory. The robot model data used in the visualization is sourced from the *eezybot.urdf* file located in the *eezybot\_support* package.

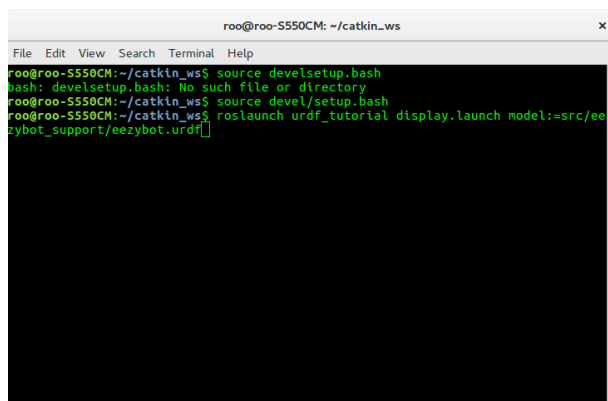
A terminal window titled 'roo@roo-S550CM: ~/catkin\_ws' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows three lines of commands: 'source develsetup.bash', 'source devel/setup.bash', and 'roslaunch urdf\_tutorial display.launch model:=src/eezybot\_support/eezybot.urdf'. The first command results in an error: 'bash: develsetup.bash: No such file or directory'. The prompt is 'roo@roo-S550CM:~/catkin\_ws\$'.

Figure 4.7: Initializing RViz visualization and slider GUI

Figure 4.8 shows the launched window containing the robot visualization and the slider GUI to input the desired coordinate location of the robot end-effector and also to control the actuation of the robot gripper. With these tools, the user is able to perform teleoperation on the robotic arm.

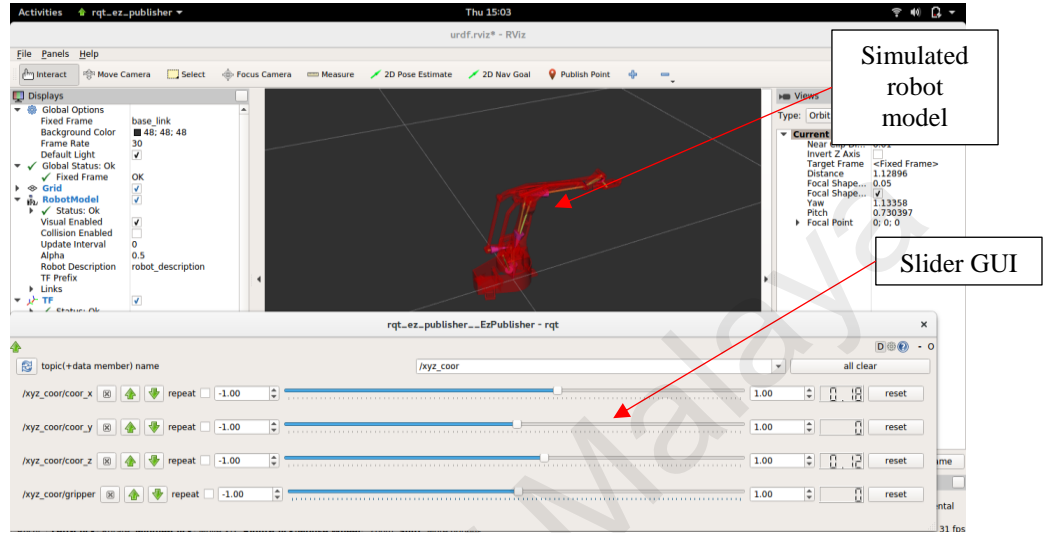


Figure 4.8: Visualization tool and GUI

The boot-up of the master PC is now complete. Next, the Raspberry Pi slave is booted-up. The slave is also connected to the *ROSNetwork* LAN. Then, a new terminal is launched. Now, this terminal must explicitly subscribe to the ROS master node on the master PC in order for communication between the master and slave to be established. This is achieved through exporting the *ROS\_MASTER\_URI* as *http://192.168.1.148:11311* which is the static IP address of the master PC and the local host at 11311. Next, the *rosserial* node is created in order to establish connection with the Arduino Uno board. The node is created from the *serial\_node.py* which is located in the *rosserial\_python* package. The serial connection is established on the */dev/ttyACM0* port as shown in Figure 4.9.

```

roosern@roosern-desktop: ~/catkin_ws
File Edit View Search Terminal Help
roosern@roosern-desktop:~/catkin_ws$ source devel/setup.bash
roosern@roosern-desktop:~/catkin_ws$ export ROS_MASTER_URI=http://192.168.1.148:11311
roosern@roosern-desktop:~/catkin_ws$ export ROS_IP=192.168.1.146
roosern@roosern-desktop:~/catkin_ws$ roslaunch roserial_python serial_node.py /dev/ttyACM0
[INFO] [1543483596.867788]: ROS Serial Python Node
[INFO] [1543483598.148365]: Connecting to /dev/ttyACM0 at 57600 baud
[INFO] [1543483600.472986]: Requesting topics...
[INFO] [1543483600.975861]: Note: subscribe buffer size is 280 bytes
[INFO] [1543483600.979388]: Setup subscriber on Ard_angle [beginner_tutorials/xyz]

```

Figure 4.9: Initializing roserial on slave

Figure 4.10 shows a graphical summary of the boot-up steps of the system.

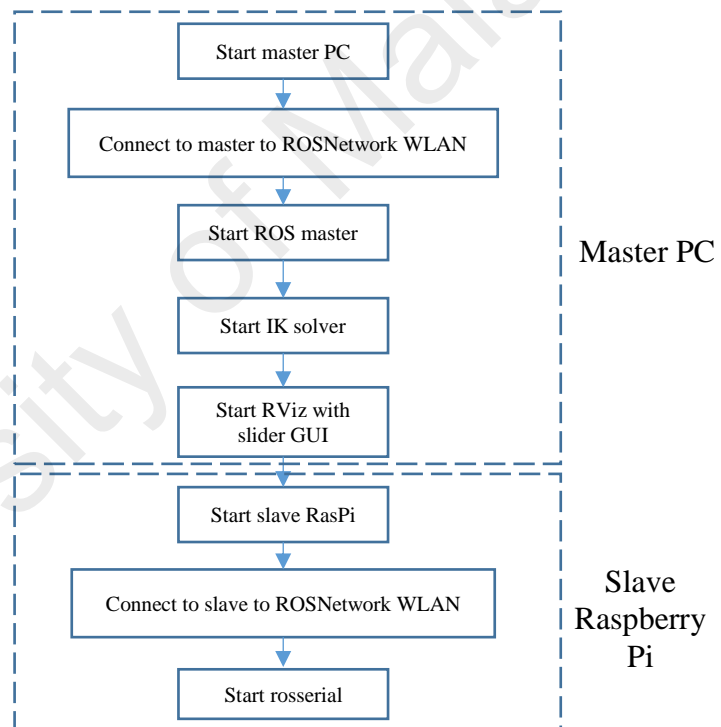


Figure 4.10: Summary of boot-up steps

## 4.2 Results and Discussion

### 4.2.1 Experiment 1: Inverse Kinematics Computational Time

Figure 4.11 shows the definition of the computational time of inverse kinematics taken. In this experiment, the time from a coordinate point being published to the `/xyz_coor` to the time the joint variables calculated from the inverse kinematics is published to the `/joint_states` and `/Ard_angle` topics are taken. This is computational time required to calculate the necessary joint variables through inverse kinematics.

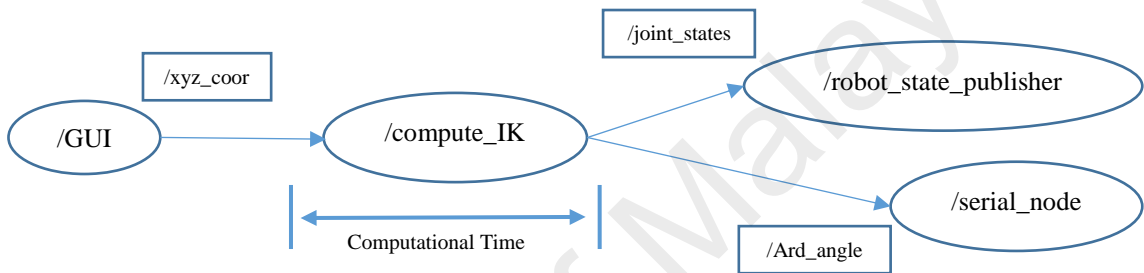


Figure 4.11: Definition of computational time taken

In setting up the experiment, the `printxyz.py` file which creates the `/compute_IK` node is modified. The system time is recorded when a message from the `/xyz_coor` topic is received, and the system time is recorded again when calculated output is published to the `/joint_states` and `/Ard_angle` topics. The two system times are compared and recorded into a comma-separated values (csv) file. The data is collected for 1000 random coordinate point calculations, and the results are plotted.

Table 10: Tabulated results for computational time

Time range (ns)	Frequency
0 - 474930	1
474930 - 615514.5	192
615514.5 - 756099	241

756099 - 896683.5	119
896683.5 - 1037268	158
1037268 - 1177852.5	139
1177852.5 - 1318437	118
1318437 - 1459021.5	73
1459021.5 - 1599606	67
1599606 - 1740190.5	35
1740190.5 - 1880775	6

Figure 4.12 shows the graph of frequency against the computational time being plot from the data collected. The computational time in the experiment ranged from 474930 nanoseconds to 1880775 nanoseconds. The mean can then be calculated as 963706.47 nanoseconds, the mode as 741005 nanoseconds, and median as 915051 nanoseconds. This shows that the average computational time is less than 0.001 second or 1 millisecond. In the system, all the nodes are set to publish to the respective topics at a maximum frequency of 10Hz. This ensures that there is ample time for the computation to be completed between receiving coordinate inputs from the GUI.

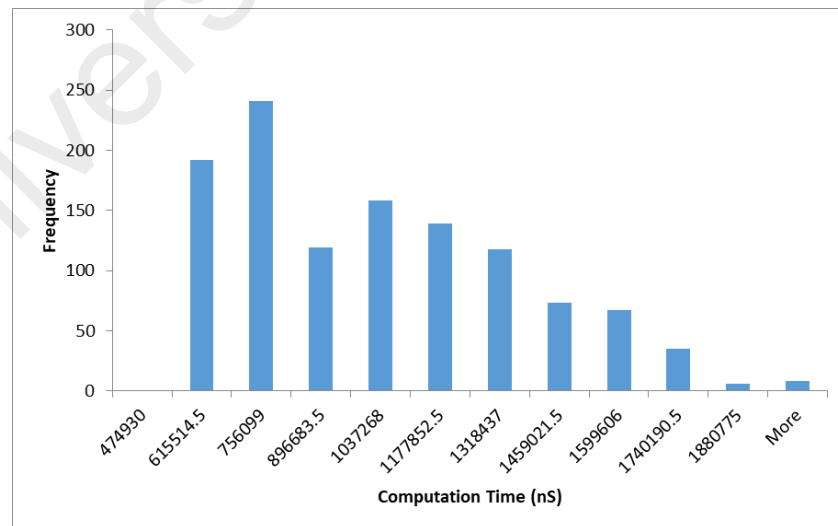


Figure 4.12: Graph of frequency against computational time

#### 4.2.2 Experiment 2: Testing of Data Transmission across WLAN

Figure 4.13 shows the transmission of data from the `/compute_IK` node to the `/serial_node` that happens across the master and slave. The first part of this experiment is conducted in order to ensure that there is no packet loss in the transmission of data across the master and slave. In the second part of the experiment, the time delay between the message being transmitted from the `/compute_IK` node to the `/serial_node` is recorded.

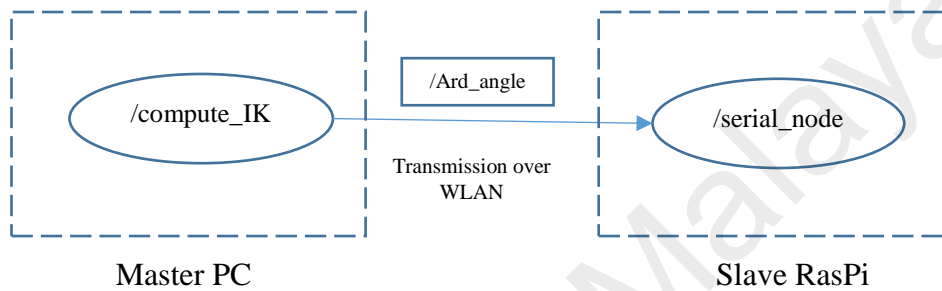


Figure 4.13: Data transmission across multiple machines

In setting up the first part of the experiment, the message being sent out of the `/compute_IK` node and the message being received by the `/serial_node` are compared. First, the system is set up. Then, both the outgoing and the incoming message are logged into a csv file for comparison. The data log from both the outgoing and incoming messages are compared for similarity, and the data is tabulated in Table 11.

Table 11: Table of comparison between outgoing and incoming messages

Similar	1530
Dissimilar	3698

From the 5228 data points collected, it can be seen that only 1530 of the messages being sent and received are similar. The percentage of times where the sent and received messages are dissimilar is:



$$\text{percentage dissimilarity} = \frac{3968}{5228} \times 100\% = 75.899\% \quad (4.1)$$

Equation (4.1) shows that the percentage of times the sent and received messages are different is 75.899%.

Next, the message data is rounded down to 4 decimal points and checked again. The percentage of times dissimilar is calculated again, and it is found that the similarity between sent and received messages is now:

*Table 12: Table of comparison between outgoing and incoming messages after rounding down*

Similar	5228
Dissimilar	0

This shows that there is a loss of transmission data due to the use of the floating point numbers in the joint variable data. However, after rounding down to 4 decimal points, the data being sent and received has become the same. This shows that the information in the transmitted data is being preserved up to 4 decimal places.

In the second part of the experiment, the latency of the data transfer across the network is measured. The message structure in ROS makes use of headers in order to embed some metadata. One type of metadata that can be embedded is the time at which the message is published to the topic. Using this, the header time can be compared to the system time when the message is received by the `/serial_node` to determine the latency in the transmission across the master and slave.

Table 13 shows the tabulated data from the experiment. Figure 4.14 show the histogram that is plotted from the collected data. From the analysis being conducted, the mean delay time is 47664370.6 ns, the median delay time is 44825077 ns and the mode delay time is 44711113 ns. This shows that the average delay time between the transmission of the message from the `/compute_IK` node to the `/serial_node` node is about 0.045s. With a computational time of 0.001s and a publishing frequency of 10Hz, this ensures that no message that is published will be missed.

*Table 13: Tabulated results for time delay*

Delay time (ns)	Frequency
0 – 40489912	1
40489912 – 61939621	1528
61939621 – 83389330	20
83389330 – 104839039	7
104839039 – 126288748	5
126288748 – 147738457	7
147738457 – 169188166	8
169188166 – 190637875	3
190637875 – 212087584	0
212087584 – 233537293	1
233537293 - 254987002	3

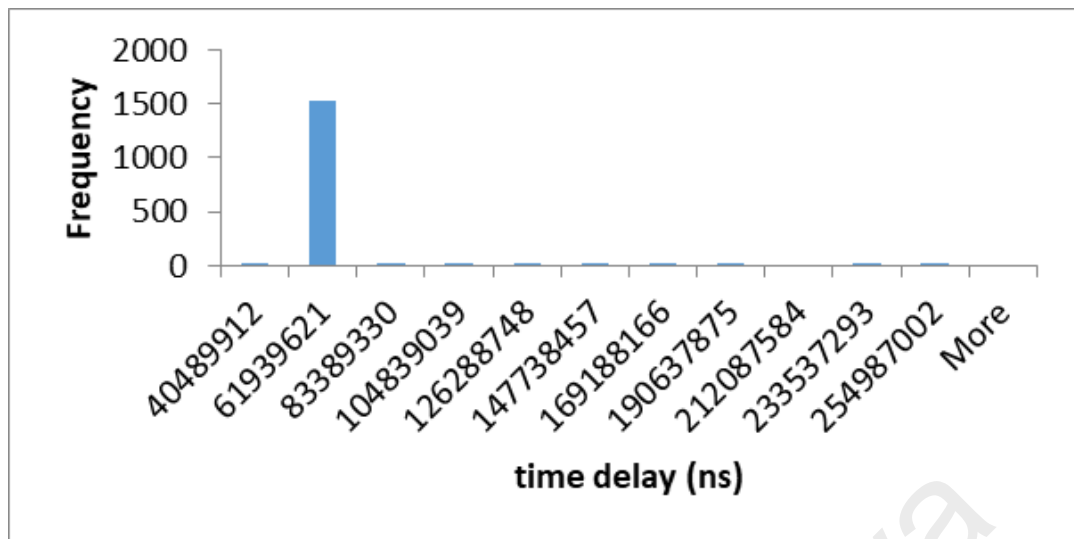
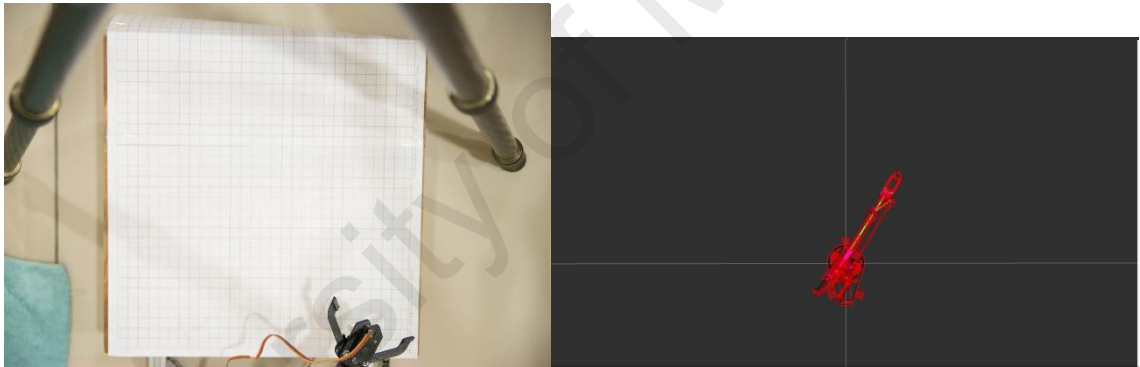


Figure 4.14: Graph of frequency against time delay

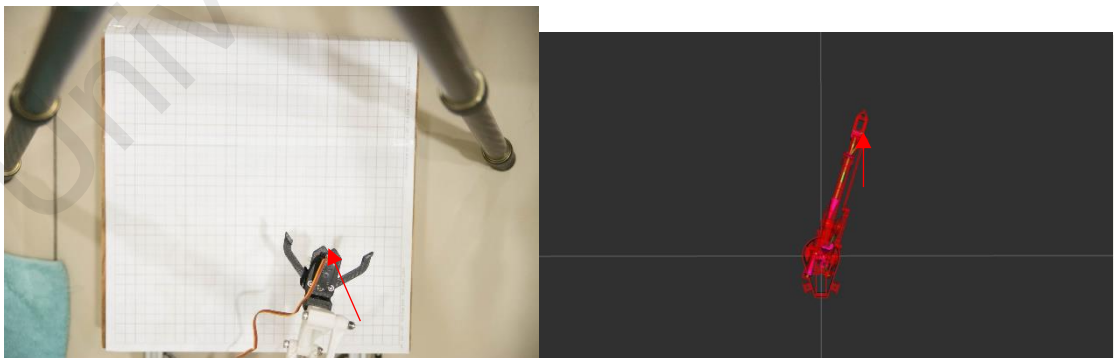
### 4.2.3 Experiment 3: Accuracy of Robot Actuation

In this experiment, the accuracy of the robotic arm is inspected. In setting up the experiment, a grid of 1cm boxes is used to track the movement of the robotic arm. Then, the movement is compared to the visualized model on the Master PC for verification. The robotic arm is made to trace a box. The image of the robotic arm at the four vertices of the box is captured. The experiment is repeated for both the X-Z plane and the X-Y plane.

Figure 4.15 to Figure 4.19 shows the sequence of movements tested on the robotic arm in the X-Y plane. The image of the physical robot arm is compared to the simulated movement of the robotic arm. From the images, it can be seen that the physical robot arm has poor accuracy. The robotic arm is unable to track a straight movement.



*Figure 4.15: Image of X-Y plane position 1*



*Figure 4.16: Image of X-Y plane position 2*

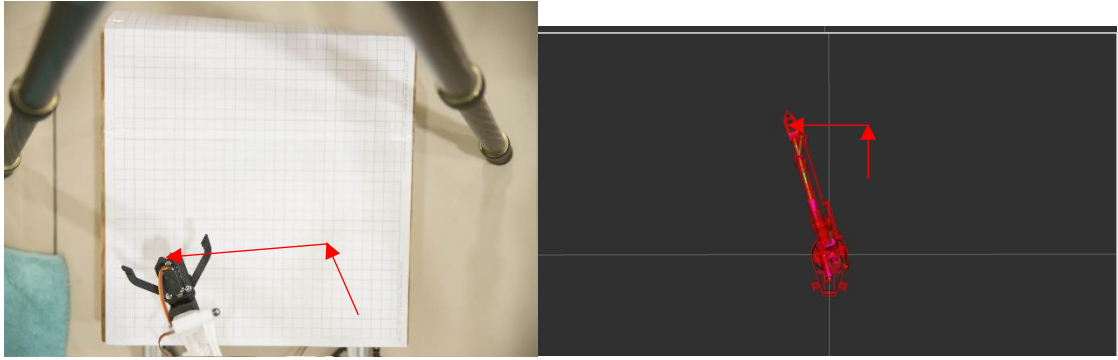


Figure 4.17: Image of X-Y plane position 3

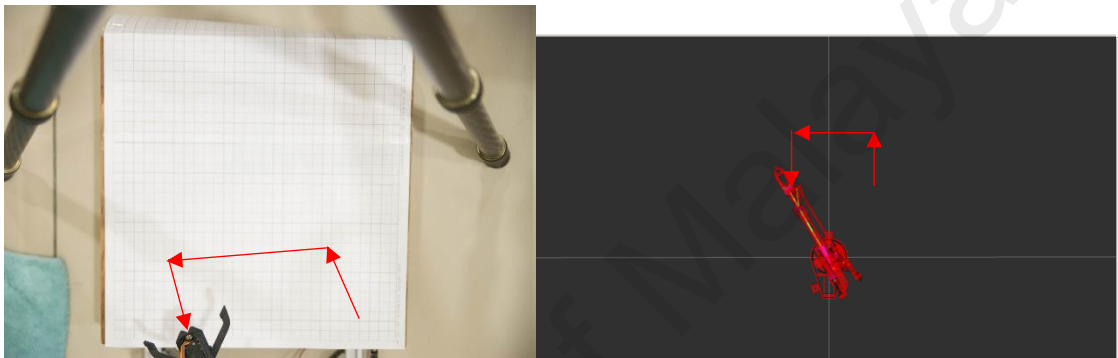


Figure 4.18: Image of X-Y plane position 4

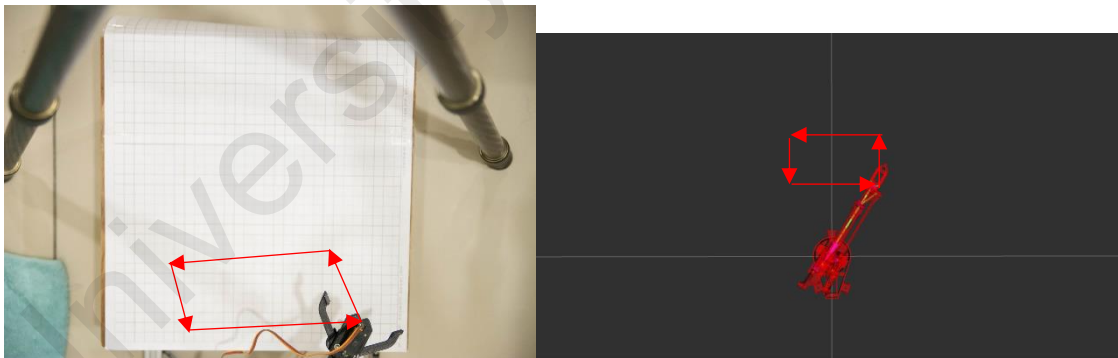
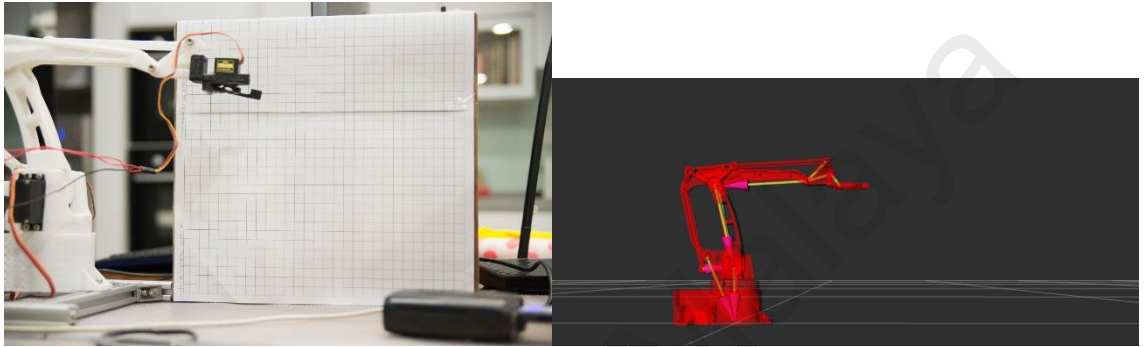


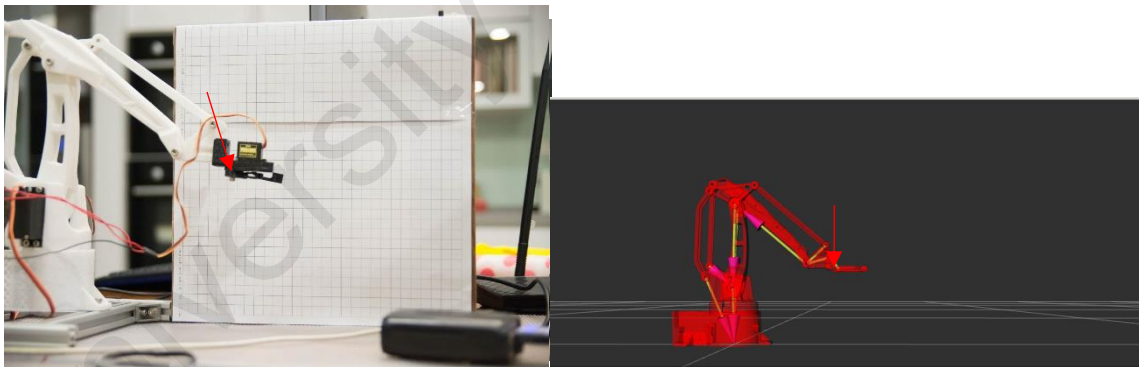
Figure 4.19: Image of X-Y plane position 1, returned

Figure 4.20 to Figure 4.24 shows the sequence of images tested on the robotic arm in the X-Z plane. Once again, the movement of the physical robotic arm is compared to that of the simulated robotic arm. Once again, the physical robotic arm shows a poor ability to track the movement in a straight line. However, this is not due to the incorrect joint variables being calculated by the IK. This is because the simulated robot is able to track

straight movements using the similar joint variables as shown. Therefore, the inaccuracy is due to a physical limitation of the servo motors and the build of the robotic arm itself. During the assembly of the robotic arm parts, some of the holes which held the pivot joints of the robotic arm parts were too small due to the additive manufacturing process. Therefore, they had to be manually drilled through with a hand drill, thus introducing tolerance errors.



*Figure 4.20: Image of X-Z plane position 1*



*Figure 4.21: Image of X-Z plane position 2*

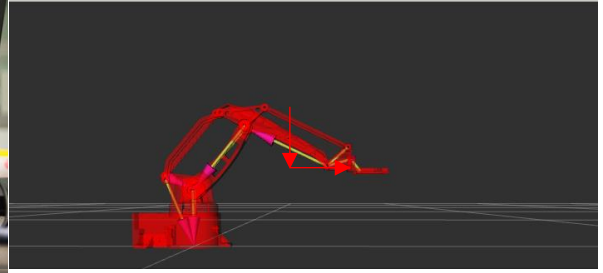
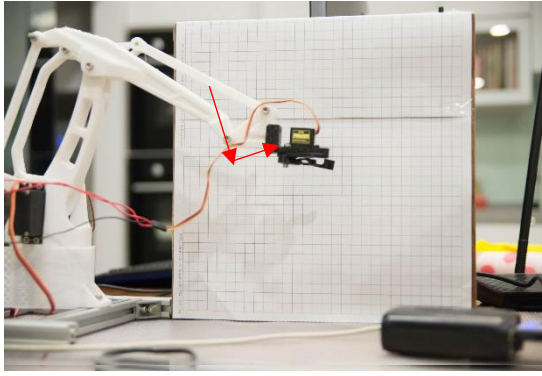


Figure 4.22: Image of X-Z plane, position 3

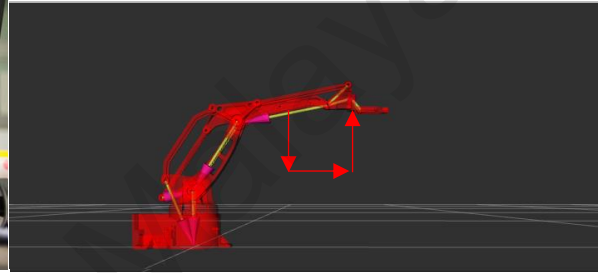
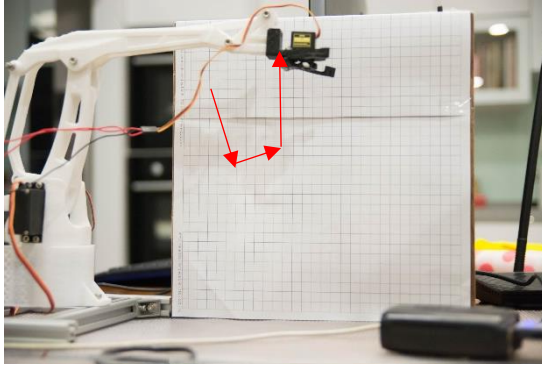


Figure 4.23: Image of X-Z plane position 4

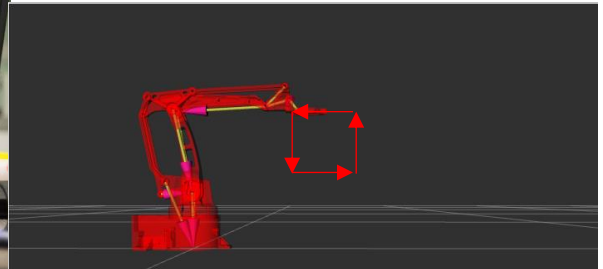
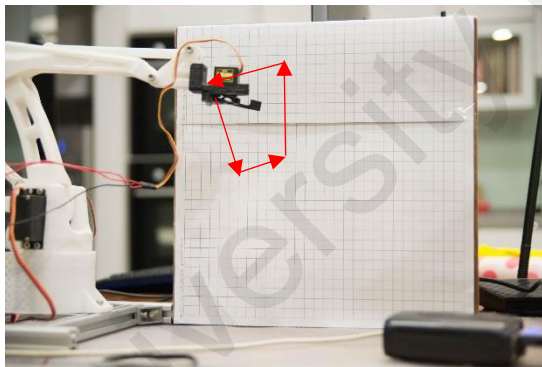


Figure 4.24: Image of X-Z plane, position 1 returned

## **CHAPTER 5: CONCLUSION**

### **5.1 Summary**

As a summary, an online robot programming framework was developed in the ROS environment (objective-3). The project used a PC as a master node, which provided a GUI for the user to provide a target coordinate to move the robotic arm. The IK analysis for the robotic arm model was obtained using the geometric approach to solve for the closed-loop kinematic chains and verified visually through visualization of the robotic arm movement (objective-1). The IK calculation was performed on the master PC in order to obtain the required joint variables to bring the robotic arm end-effector to the desired location (objective-2). A FK analysis of the robotic arm model was also performed using the DH convention. A visualization of the configuration of the robotic arm was provided in RViz to aid the use of teleoperation. A Raspberry Pi is used as the slave node, which is connected to the master node through a WLAN connection. The Raspberry Pi interfaced with an Arduino Uno board, which controlled the servo motors on the robotic arm by PWM signals. In experimentation, the IK calculations were able to be performed in under 1 millisecond. The transmission of data across showed some losses, but was able to preserve the joint variables data with up to 4 decimal point precision. The transmission speed per message took about 45 ms each. However, the physical robot could not be controlled accurately due to physical limitations.

### **5.2 Further Improvements**

1. The physical robot model needs to be improved. The RC grade servo motors used are not accurate, and should be switched to stepper motors with encoders for better accuracy.



2. In this project, the framework is only tested with one master and one slave node.

The number of slave nodes could be increased to test out multiple robot teleoperation.

3. The framework built in this project did not include trajectory planning. A trajectory planner could be developed in order to support robotic arms with more degrees of freedom.

University of Malaya

## REFERENCES

- Araújo, A., Portugal, D., Couceiro, M. S., & Rocha, R. P. (2013). *Integrating Arduino-based educational mobile robots in ROS*. Paper presented at the 2013 13th International Conference on Autonomous Robot Systems.
- Barbosa, J. P. d. A., Lima, F. d. P. d. C., Coutinho, L. d. S., Leite, J. P. R. R., Machado, J. B., Valerio, C. H., & Bastos, G. S. (2015). *ROS, Android and cloud robotics: How to make a powerful low cost robot*. Paper presented at the 2015 International Conference on Advanced Robotics (ICAR).
- Bhargava, A., & Kumar, A. (2017). *Arduino controlled robotic arm*. Paper presented at the 2017 International conference of Electronics, Communication and Aerospace Technology (ICECA).
- Bjorlykhaug, E. J. R. (2018). A Closed Loop Inverse Kinematics Solver Intended for Offline Calculation Optimized with GA. 7, 7.
- Chen, Z., Yan, S., Yuan, M., Yao, B., & Hu, J. (2018). Modular Development of Master-Slave Asymmetric Teleoperation Systems With a Novel Workspace Mapping Algorithm. *IEEE Access*, 6, 15356-15364. doi:10.1109/ACCESS.2018.2809860
- Craig, J. J. (2018). *Introduction to robotics : mechanics and control* (Fourth edition. ed.). Ny Ny: Pearson.
- Du, Z., Sun, Y., Su, Y., & Dong, W. (2014). A ROS/Gazebo based method in developing virtual training scene for upper limb rehabilitation. *IEEE International Conference on Progress in Informatics Computing*, 307-311.
- Ergur, S., & Ozkan, M. (2014). *Trajectory planning of industrial robots for 3-D visualization a ROS-based simulation framework*. Paper presented at the 2014 IEEE International Symposium on Robotics and Manufacturing Automation (ROMA).
- Hernandez-Mendez, S., Maldonado-Mendez, C., Marin-Hernandez, A., Rios-Figueroa, H. V., Vazquez-Leal, H., & Palacios-Hernandez, E. R. (2017). *Design and implementation of a robotic arm using ROS and MoveIt!* Paper presented at the 2017 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC).
- Issa, A. I., Aqel, M. O. A., Albelbeisi, M. M., Elaila, M. O., & Mortaja, M. A. J. I. C. o. P. E. T. (2017). Palletizing Manipulator Design and Control Using Arduino and MATLAB. 60-65.
- Kruthika, K., Kumar, B. M. K., & Lakshminarayanan, S. (2016). *Design and development of a robotic arm*. Paper presented at the 2016 International Conference on Circuits, Controls, Communications and Computing (I4C).
- Lawton, J. (2018). The Role Of Robots In Industry 4.0. Retrieved from Forbes website:
- Lentin, J. (2015). *Learning robotics using Python : design, simulate, program, and prototype an interactive autonomous mobile robot from scratch with the help of Python, ROS, and Open-CV!* In *Community experience distilled* (pp. 1 online resource (1 volume)). Retrieved from Knovel. Restricted to UCB, UCI, UCLA, UCM, UCR, UCSB, and UCSD <http://uclibs.org/PID/277475>
- Lynch, K. M., & Park, F. C. (2017). *Modern robotics : mechanics, planning, and control*. Cambridge, UK: Cambridge University Press.
- Megalingam, R. K., Sivanantham, V., K Sai Kumar, S. G., Teja, P. S., Gangireddy, R., M, S. K., & Gedela, V. V. (2018). Design and Development of Inverse kinematics Based 6 DoF Robotic Arm Using ROS. *International Journal of Pure and Applied Mathematics*, 118(18), 2597-2603.

- Mohammed Abu, Q., Abuhadrous, I., & Elaydi, H. (2010). *Modeling and Simulation of 5 DOF educational robot arm*. Paper presented at the 2010 2nd International Conference on Advanced Computer Control.
- Mortimer, M., Horan, B., & Joordens, M. (2016). *Kinect with ROS, interact with Oculus: Towards Dynamic User Interfaces for robotic teleoperation*. Paper presented at the 2016 11th System of Systems Engineering Conference (SoSE).
- Pandilov, Z., & Dukovski, V. (2014). Comparison of the Characteristics between Serial and Parallel Robots. *Acta Technica Corviniensis*.
- Posada, J., Toro, C., Barandiaran, I., Oyarzun, D., Stricker, D., Amicis, R. d., . . . Vallarino, I. (2015). Visual Computing as a Key Enabling Technology for Industrie 4.0 and Industrial Internet. *IEEE Computer Graphics and Applications*, 35(2), 26-40. doi:10.1109/MCG.2015.45
- Qian, W., Xia, Z., Xiong, J., Gan, Y., Guo, Y., Weng, S., . . . Zhang, J. (2014). *Manipulation task simulation using ROS and Gazebo*. Paper presented at the 2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014).
- Roblek, V., Meško, M., & Krapež, A. (2016). A Complex View of Industry 4.0. 6(2), 2158244016653987. doi:10.1177/2158244016653987
- Rozman, J., Luža, R., & Zbořil, F. V. (2014). *ROS-based remote controlled robotic arm workcell*. Paper presented at the 2014 14th International Conference on Intelligent Systems Design and Applications.
- Sarkar, B. K. (2018). Modeling and validation of a 2-DOF parallel manipulator for pose control application. *Robotics and Computer-Integrated Manufacturing*, 50, 234 - 241. doi:<https://doi.org/10.1016/j.rcim.2017.09.017>
- Spong, M. W., Hutchinson, S., & Vidyasagar, M. (2006). *Robot modeling and control*. Hoboken, NJ: John Wiley & Sons.
- Tao, Y., Chen, F., & Xiong, H. (2015). Kinematics and Workspace of a 4-DOF Hybrid Palletizing Robot. *Advances in Mechanical Engineering*, 6. doi:10.1155/2014/125973