

INDUSTRIAL ROBOTIC ARM MODELLING AND SIMULATION WITH ROBOT OPERATING SYSTEM

A Project Report Phase – II

Submitted in partial fulfilment of the requirements for the award of Bachelor of
Engineering Degree in Electronics and Communication Engineering

by

YATHAM SATYA LOKESH (39130524)

YEDLA SAI MAHESH (39130525)



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
SCHOOL OF ELECTRICAL AND ELECTRONICS**

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade “A” by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

APRIL 2023



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with “A” grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600 119

www.sathyabama.ac.in

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of YATHAM SATYA LOKESH (39130524) and YEDLA SAI MAHESH (39130525) who carried out the project entitled “**INDUSTRIAL ROBOTIC ARM MODELLING AND SIMULATION WITH ROBOT OPERATING SYSTEM**” under our supervision from November 2022 to April 2023.

Internal Guide

Dr. M. SUGADEV, M.Tech., Ph.D.,

Head of the Department

Dr. T. RAVI, M.E., Ph.D.,

Submitted for Viva voce examination held on 19.04.2023

Internal Examiner

External Examiner

DECLARATION

We, **YATHAM SATYA LOKESH (39130524)** and **YEDLA SAI MAHESH (39130525)** hereby declare that the Project Report entitled “**INDUSTRIAL ROBOTIC ARM MODELLING AND SIMULATION WITH ROBOT OPERATING SYSTEM**” done by us under the guidance of **DR. M. SUGADEV, M.Tech., Ph.D.**, is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Electronics and Communication Engineering.

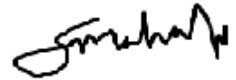
DATE: 25-04-23

PLACE: CHENNAI
Lokesh

1. Y. Satya



2. Y. Sai Mahesh



SIGNATURE OF THE CANDIDATES

ACKNOWLEDGEMENT

We are pleased to acknowledge our sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. We are grateful to them.

We convey our thanks to **Dr. N. M. NANDHITHA, M.E., Ph.D., Professor & Dean, School of Electrical and Electronics** and **Dr. T. RAVI, M.E., Ph.D., Professor & Head, Department of Electronics and Communication Engineering** for providing us necessary support and details at the right time during the progressive reviews.

We would like to express our sincere and deep sense of gratitude to our Project Guide **Dr. M. SUGADEV, M.Tech., Ph.D., Associate Professor, Department of Electronics and Communication Engineering** for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of the project work.

We wish to express our thanks to all teaching and Non-teaching staff members of the **Department of Electronics and Communication Engineering** who are helpful in many ways for the completion of the project.

ABSTRACT

A robot manipulator is an electronically controlled mechanism, consisting of multiple segments, that performs tasks by interacting with its environment. They are also commonly referred to as robotic arms. In robotics, a manipulator is a device used to manipulate materials without direct physical contact by the operator. The applications were originally for dealing with radioactive or biohazardous materials, using robotic arms, or they were used in inaccessible places. In more recent developments they have been used in diverse range of applications including welding automation, robotic surgery and in space. It is an arm-like mechanism that consists of a series of segments, usually sliding or jointed called cross-slides, which grasp and move objects with a number of degrees of freedom. In robotics, a manipulator is a device used to manipulate materials without direct physical contact by the operator. The applications were originally for dealing with radioactive or biohazardous materials, using robotic arms, or they were used in inaccessible places.

Robot Operating System (ROS) is an open-source robotics middleware suite. Although ROS is not an operating system (OS) but a set of software frameworks for robot software development, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message -passing between processes, and package management. The Robot Operating System (ROS) is an open-source framework that helps researchers and developers build and reuse code between robotics applications. ROS is also a global opensource community of engineers, developers and hobbyists who contribute to making robots better, more accessible and available to everyone. The objective of the work is to develop a software model of a 3 degree of freedom robotic manipulator using Robot Operating System and perform analysis of joint angles for different work space co-ordinates.

TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No.
	ABSTRACT	v
	LIST OF FIGURES	viii
	LIST OF TABLES	ix
1	INTRODUCTION	1
	1.1 Objective Of Robotic Arm	2
2	LITERATURE SURVEY	4
	2.1 Interference of Literature Review	4
3	SYSTEM ANALYSIS	11
	3.1 Existing Methodology	11
	3.2 Proposed Methodology	13
	3.2.1 System Requirements	15
	3.3 Robot Operating System (ROS)	16
	3.3.1 ROS Tools Packages	17
	3.3.2 ROS tools	17
	3.3.3 ROS Packages	18
	3.4 Alternative API	19
	3.5 API Architecture	21
	3.6 URDF	24
	3.7 Gazebo Simulation	27
4	EXPERIMENTAL METHODS	29
	4.1 Kinematics	29

CHAPTER No.	TITLE	PAGE No.
	4.2 Design Calculations	31
	4.3 Forward Kinematics	32
	4.4 Inverse Kinematics	35
5	DESIGN DETAILS OF ROBOTIC ARM	39
	5.1 Designing of Robotic Arm	41
	5.2 Simulation Pipeline	45
	5.3 Advantages of Industrial Robotic Arm modelling and Simulation with ROS	49
	5.4 Disadvantages of Industrial Robotic Arm modelling and Simulation with ROS	50
6	RESULTS AND DISCUSSION	51
	6.1 Simulation Results	51
	6.2 Discussion	52
	6.3 Graphs Plotted Based on The Above Simulation	53
7	CONCLUSION	56
	7.1 Conclusion	56
	7.2 Future Work	56
	REFERENCES	57
	APPENDIX	58
	A. IEEE STANDARDS	58
	B. CONSTRAINTS	59

LIST OF FIGURES

FIGURE No.	FIGURE NAME No.	PAGE
Fig 3.1	Proposed Block Diagram	14
Fig 3.2	Architecture of ROS	16
Fig 3.3	Execution of API	19
Fig 3.4	Components of ROS Interface Architecture	21
Fig 3.5	Architecture of URDF	24
Fig 3.6	Implementation of URDF	25
Fig 3.7	Trajectory of Gazebo	27
Fig 4.1	Torque of Elbow and Shoulder	31
Fig 4.2	Frame assignments for Kinematics Analysis	34
Fig 5.1	Steps to Define a Robotic Arm URDF file	43
Fig 5.2	Benefits of 3 DOF Robotic Arm	44
Fig 5.3	ROS Package File Structure	47
Fig 5.4	Node Operation	48
Fig 6.1	Simulation Results Of Robotic Arm	52
Fig 6.2	Robot Object Pick Graph	53
Fig 6.3	Robot Object Drop Graph	54
Fig 6.4	Robot Complete Trajectory Graph	54
Fig 6.5	Robot Trajectory Latency Graph	55

LIST OF TABLES

TABLE No.	TABLE NAME	PAGE No.
Table 4.1	D-H Parameters	32

CHAPTER 1

INTRODUCTION

The manufacturing sector is poised to undergo considerable change over the next decade. Driven by initiatives such as Industry 4.0, the Digital Agenda, and the Internet of Things, the introduction of new technologies and further digitalisation will lead to highly connected, and integrated workplaces. These changes will produce new ways of working, and open up new opportunities for innovation and process flexibility. In particular, developments in robotics will enable humans and robots to work collaboratively, maximising the benefits of manual and automated processes. This shift towards human-robot co-working is enabled by the recent development of collaborative robots, including the KUKA LBR iiwa. Such cobots are designed to operate alongside human users in shared environments without safety caging; back-drivable motors and compliant. available online ¹, which we have developed to support our experimental research work, and is now supporting development of new industrial processes.

The interface enables control and communication via the Robot Operating System (ROS), but with minimal installation requirement, and without compromising the inbuilt safety features. Controllers allowing humans to physically interact with the robots without harm. Whilst early adoption focuses on robots working un-caged in human-occupied spaces as assistive tools with little interaction, the full potential of this technology will only be realised through symbiotic human-robot collaborative processes. A major aim for future manufacturing is greater flexibility to support smaller batch sizes and more customisation. Whereas existing automated processes are highly repetitive, and difficult to reconfigure for new products or tasks, collaborative robots will support much greater variability through task switching. The added complexity of flexible processes and co-working with robots will require the up-skilling of the human workforce, and highly intuitive interfaces to support more variability in worker roles. To achieve these aims, greater integration of workers, robots, and systems are required. This requires development of robot control interfaces that can safely and flexibly connect to sensors and gather information from external sources; These issues will be exacerbated by the introduction, and up-skilling, of workers without robotics experience.

1.1 OBJECTIVE OF ROBOTIC ARM

The deployment of collaborative robotic systems in industry 4.0 raises the potential for complex human-robot interaction to create highly flexible processes. This brings a need for systems that can facilitate rapid programming and development of safe collaborative processes, without the need for extensive training. In this paper we introduce a novel Application Programming Interface (API) for the Intelligent Industrial Work Assistant (iiwa) Lightweight Robot (LBR) that enables fast development and integration of devices, using the popular Robot Operating System (ROS), without compromising the inherent safety features of the robot. We describe the API, released as a freely available download, and provide an example application of its use to support a large-scale interactive participant experiment. As flexible manufacturing technologies become ever more connected and complex, it is important to ensure compatibility between networked devices and provide tools to support system integration based on common platforms. Our API is one such tool, and has been designed to support faster and easier system integration and development, providing particular support to scientists in creating experiments for studying human-robot collaboration.

It provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message -passing between processes, and package management. Running sets of ROS-based processes are represented in a architecture where processing takes place in nodes that may receive, post, and multiplexer sensor data, control, state, planning, actuator, and other messages. Despite the importance of reactivity and low latency in robot control, ROS is not realtime operating system (RTOS). However, it is possible to integrate ROS with realtime code The lack of 0 for real time systems has been addressed in the creation of ROS 2, a major revision of the ROS API which will take advantage of modern libraries and technologies for core ROS functions and add support for real-time code and embedded system hardware.

- The objective of industrial robotic arm modelling and simulation with Robot Operating System (ROS) is to develop a virtual representation of a robotic arm that can be used to simulate various scenarios and test different control algorithms without the need for physical hardware.
- To optimize the design of robotic arms: Simulation allows for the rapid testing of different arm designs and configurations, which can help identify the most efficient and cost-effective options.
- To improve robot performance: Simulation can be used to test and refine control algorithms and motion planning strategies, leading to better overall robot performance.
- To reduce costs: Simulating robotic arm behaviour can help reduce the need for expensive physical prototypes and reduce overall development costs.
- To increase safety: Simulation can be used to test the safety of robotic arm operations in a controlled environment, before deploying them in real-world scenarios.
- To facilitate training: Simulation can be used to train operators on how to use and interact with robotic arms, allowing them to gain experience in a safe and controlled environment.
- By achieving these objectives, industrial robotic arm modelling and simulation with ROS can help organizations develop more efficient, effective, and safe robotic systems.
- To provide training and education opportunities for engineers and technicians to learn and practice the use of robotic arms in industrial settings.
- To reduce the development time and costs of robotic arm systems by using simulation and virtual prototyping techniques.
- To enhance the adaptability and flexibility of the robotic arm by using ROS-based software tools that support customization, modularity, and interoperability.
- To improve the overall performance and productivity of the industrial processes that involve the robotic arm, by optimizing the use of resources, reducing waste, and increasing efficiency.

CHAPTER 2

LITERATURE SURVEY

2.1 INTERFERENCE OF LITERATURE REVIEW

Iqbal et al (2012) presented a work on "Modelling and analysis of a 3 DOF robotic arm manipulator", Canadian Journal on Electrical and Electronics Engineering. The behaviour of physical systems in many situations may better be expressed with an analytical model. Robot modelling and analysis essentially involve its kinematics. For robotic manipulators having high Degrees Of Freedom (DOF) with multiple degrees in one or more joints, an analytical solution to the inverse kinematics is probably the most important topic in robot modelling. This paper develops the kinematic models a 3 DOF robotic arm and analyses its workspace. The proposed model makes it possible to control the manipulator to achieve any reachable position and orientation in an unstructured environment. The forward kinematic model is predicated on Denavit Hartenberg (DH) parametric scheme of robot arm position placement. Given the desired position and orientation of the robot end-effector, the realized inverse kinematics model provides the required corresponding joint angles. The forward kinematic model has been validated using Robotics Toolbox for while the inverse kinematic model has been implemented on a real robotic arm. Experimental results demonstrate that using the developed model, the end-effector of robotic arm can point to the desired coordinates within precision of $\pm 0.5\text{cm}$. The approach presented in this work can also be applicable to solve the kinematics problem of other similar kinds of robot manipulators.[1]

Nguyen.H.C (2016) presented a work on "Research on the application of Neural Networks in identification and control of the robotic arms-A nonlinear dynamic object," Journal of Science and Technology – the University of Da Nang. This paper presents a novel inverse kinematics solution for robotic arm based on artificial neural network (ANN) architecture. The motion of robotic arm is controlled by the kinematics of ANN. A new artificial neural network approach for inverse kinematics is proposed. The novelty of the proposed ANN is the inclusion of the feedback of current joint angles configuration of robotic arm as well as the desired position and orientation in the input pattern of neural network.

ANN has only the desired position and orientation of the end effector in the input pattern of neural network. In this paper, a Three DOF Denso robotic arm with a gripper is controlled by ANN. The comprehensive experimental results proved the applicability and the efficiency of the proposed approach in robotic motion control. The inclusion of current configuration of joint angles in ANN significantly increased the accuracy of ANN estimation of the joint angles output. The new controller design has advantages over the existing techniques for minimizing the position error in unconventional tasks and increasing the accuracy of ANN in estimation of robot's joint angles.[6]

Meng et al (2015) presented a work on "Recent development of mechanisms and control strategies for robot-assisted lower limb rehabilitation," *Mechatronics*. Robot-assisted rehabilitation and therapy has become more and more frequently used to help the elderly, disabled patients or movement disorders to perform exercise and training. The field of robot-assisted lower limb rehabilitation has rapidly evolved in the last decade. This article presents a review on the most recent progress (from year 2001 to 2014) of mechanisms, training modes and control strategies for lower limb rehabilitation robots. Special attention is paid to the adaptive robot control methods considering hybrid data fusion and patient evaluation in robot-assisted passive and active lower limb rehabilitation. The characteristics and clinical outcomes of different training modes and control algorithms in recent studies are analysed and summarized. Research gaps and future directions are also highlighted in this paper to improve the outcome of robot-assisted rehabilitation.[4]

Zlajpah and (2008) presented a work on "Simulation in robotics," *Mathematics and Computers in Simulation*. Simulation has been recognized as an important research tool since the beginning of the 20th century. However, the "good times" for simulation started with the development of computers and now the simulation is a powerful visualization, planning, and strategic tool in different areas of research and development. The simulation has also a very important role in robotics. Different tools are used for the analysis of kinematics and dynamics of robotic manipulators, for off-line programming, to design different control algorithms, to design mechanical structure of robots, to design robotic cells and production lines, etc. In the paper an overview of the simulation in robotics is given and some topics

like: how simulation makes things easier, advantages and back draws of the simulation in robotics, virtual and real world, are pointed out. The scope of the lecture is the role of the simulation, simulation in different fields of robotics.[9]

Topaz et al (2010) presented a work on "Dynamics simulation toolbox for industrial robot manipulators," Computer Applications in Engineering Education . A new robot toolbox for dynamics simulation based on ROS Graphical User Interface (GUI) is developed for educational purposes. It is built on the previous version named as ROBOLAB performing only the kinematics analysis of robot manipulators. The toolbox presented in this paper provides interactive real-time simulation and visualization of the industrial robot manipulator's dynamics based on Langrange-Euler and Newton-Euler formulations. Since most of the industrial robot manipulators are produced with three degrees of freedom (DOF) such as the PUMA 560, the Fanuc ArcMate 120iB and Stanford Arm, the library of the toolbox includes sixteen fundamental 3-DOF robot manipulators with Euler wrist. The software can be used to interactively perform robotics analysis and off-line programming of robot dynamics such as forward and inverse dynamics as well as to interactively teach and simulate the basic principles of robot dynamics in a very realistic way. In order to demonstrate the user-friendly features of the toolbox much better, simulation of the NS robot manipulator.[7]

Wang et al (2012) presented a work on "Kinematical analysis and simulation of an industrial robot based on ROS," Jidian Gongcheng/Mechanical & Electrical Engineering Magazine . The history of industrial automation is characterized by quick update technology, however, without a doubt, the industrial robot is a kind of special equipment. With the help of ROS matrix and drawing capacity in the ROS environment each link coordinate system set up by using the d-h parameters method and equation of motion of the structure. Robotics, Toolbox programming Toolbox and GUIDE to the joint application is the analysis of inverse kinematics, so as to achieve the aim of reservation.[8]

Kang et al (2010) presented a work on "Robot manipulator modelling in ROS-SimMechanics with PD control and online gravity compensation," In International Forum on Strategic Technology . The objective of this article is to present a method to model the mechanics of robot manipulators.[2]

Kim et al (2013) presented a work on "Aerial manipulation using a quadrotor with a two DOF robotic arm," In IEEE/RSJ International Conference Intelligent Robots and Systems. This paper presents aerial manipulation using a quadrotor with a two-DOF robot arm. By considering a quadrotor and robot arm as a combined system, the kinematic and dynamic models are developed, and an adaptive sliding mode controller is designed. With the controller, an autonomous flight experiment is conducted including picking up and delivering an object, which requires accurate control of a quadrotor and robot arm. Overall result shows that the proposed approach demonstrates satisfactory performance as a potential platform which can be utilized in various applications such as inspection, manipulation, or transportation in remote places. Mohammed Abu Qassem, I.M. Abuhadrous, H. Elaydi, "Modeling and Simulation of 5 DOF educational robot arm". Many universities and institutes experience difficulty in training people to work with expensive equipment's. A common problem faced by educational institutions concerns the limited availability of expensive robotics equipment's, with which students in the didactic program can work, in order to acquire valuable "hands on" experience. Therefore, the Robot Simulation Software (RSS) nowadays is paramount important. This paper presents the development of a visual software package where a AL5B Robot arm has been taken as a case study.[3]

P.Hemalatha et al (2015) presented a work on "Real-time Image Processing based Robotic Arm Control Standalone System using Raspberry pi" SSRG International Journal of Electronics and Communication Engineering. This paper proposes Realtime Image Processing based Robotic Arm Control Standalone System using Raspberry pi. In the present era we made a robot capable of surveillance and also with an alternate.Application in detecting and following a pre specified object. The detection and recognition have been done using open CV library. The whole code for Object detection written in ROS. And this all processing has been done on raspberry pi which works on Raspbian OS based on Debian which is Linux OS. To program the Controlling of one arm robot using Raspberry pi for the identification of objects and tracking object operations without any manual control. The total programming model is developed in ROS. Simulink support package for raspberry pi hardware.

Jamshed et al (2012) presented a work on "Modelling and analysis of a 3 DOF robotic arm manipulator", Canadian Journal on Electrical and Electronics Engineering. widely used 3 DOF robotic arm manipulator, ED7220C has been kinematically modelled followed by the analysis of its workspace. Forward kinematic model has been validated using ROS. The results from the derived forward kinematic model match exactly with that of ROS. The IK model of the robot has also provided correct joint angles to move the arm gripper to any position and orientation within its workspace. The IK model has been implemented on the real robotic platform.

Results obtained from the model have been compared with the actual performance of the robot in accomplishing a task e.g. pick and place. It has been found that with the joint angles computed by IK model, the robot achieves position precision within $\pm 0.5\text{cm}$. This little Canadian Journal on Electrical and Electronics Engineering Vol. 3, No. 6, July 2012 306 deviation is because of many reasons namely, reported platform precision ($\pm 0.5\text{mm}$ as mentioned in TABLE I), mechanical coupling of the joints, non-linearity in mapping angles to low-level encoder ticks.

Nguyen.H.C.(2016) presented a work on "Research on the application of Neural Networks in identification and control of the robotic arms-A nonlinear dynamic object," Journal of Science and Technology – the University of Da Nang. This study introduced a very accurate solution for inverse kinematics by using the artificial neural network to overcome the drawbacks of traditional ANN controller. A new design of artificial neural network ANN has been proposed for the optimal robot motion control in Cartesian coordinates. evaluate the integral performance of the system, the current joint angles information was added to the traditional ANN based inverse kinematics solution. The proposed design showed improvement in performance of end effector in some aspects. The motion of robot has been executed; it satisfies the constraint of robotic arm motion by the designed structure. Robot kinematics are analyzed, and position/orientation of end effector in different configurations are studied. A comparison has been carried out between two ANNs. The parameters of motion and the errors were calculated. The results showed that the proposed ANN has superior Performance in terms of the joint angles estimation.

Meng et al (2015) presented a work on "Recent development of mechanisms and control strategies for robot-assisted lower limb rehabilitation," *Mechatronics*. Robot-assisted lower limb rehabilitation has a variety of advantages over traditional manual therapy and training, and shows encouraging clinical outcomes and recovery efficiency. Existing studies have also demonstrated the effectiveness of robot-assisted lower limb training. In this review, various studies are conducted to compare different robotic mechanisms, training modes and control strategies. As the rehabilitation robot directly contacts with the human limb, its workspace and movement features must be considered when designing a mechanical structure. A superior feature of exoskeletons for rehabilitation is the possibility to control the segments of the lower limb since it can be worn by the participant. While the end-effector robots usually contact with the patient's body at a certain point, making this kind of robots more easily adaptable to different patients.

In order to improve clinical outcomes, rehabilitation robots should have various operation modes and be adaptable to patients' recovery conditions. The patient-active mode taking into account participant's active intention and voluntary efforts is supposed to be more effective than traditional passive and repetitive training. Contrary to active assist mode, challenge-based robotic training aims at rendering the task more challenging and demanding higher efforts so as to improve motor function. As for the control strategies, impedance control becomes more and more popular in the control of lower limb exoskeletons and platforms, and EMG signals have also been widely used to estimate the human intention prior to the system control. Evidences from many studies show that human-robot interaction is very important in encouraging patient's recovery. An adaptive controller that can sense patient's status and tune the robot compliance to match particular patients is believed to be the most efficient to realize "patient-cooperative" robotic therapy. In the meantime, the assist-as-needed control concept has emerged to encourage the maximum participation of the patient by providing only as much assistance as necessary.

Although most exiting rehabilitation robots are able to provide systematic and prolonged treatment and training sessions, there are drawbacks associated with their designs. In terms of actuators and mechanical designs, artificial pneumatic muscle represents a good choice for wearable robots because of its relatively low

weight, high efficiency, and intrinsic safety. However, low control bandwidth and complex algorithms are main challenges that hinder its applications. Trajectory tracking control is the basis and is applied in almost all available rehabilitation robotic trainers. However, this control strategy guides the patient's lower limb on a predefined trajectory without taking into account patient's movement intention, thus reducing user's active participation and motivation.

Impedance control based on force feedback allows the patient to deviate from the reference trajectory according to the dynamic relationship between position and interaction force, and thus is more suitable for patients. However, as the impaired limb should be treated with different impedance values and robot compliance, how to determine the impedance parameters is still an open problem. In this situation, robotic training with adaptive impedance model and assist-as-needed control concept is supposed to be the most appropriate method, as the amount of robot assistance can be adjusted to suit patient's recovery conditions and training progresses.

As stated previously, the implementation of adaptive control algorithm ought to its ability to generate different training modes adaptable to patients with different recovery conditions. Thus, one important factor is to evaluate and quantify the patient's movement ability, muscle activity or training progress. To the authors' best knowledge, the patient's muscle activity and recovery status cannot be monitored online in most existing lower limb rehabilitation robots . Most current approaches used kinematic parameters, such as the interaction force, joint angle/trajectory tracking errors, to determine the patient's movement ability, but these data are hard to accurately reflect patient's actual ability [83,99]. It is expected in the future that the bio-signals (e.g. EMG) must be introduced in the whole lifecycle of robot control, so that the patient's movement intention can be perceived and the muscle activity level can be reflected.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING METHODOLOGY

In 2017, robot sales increased by 30% to 381,335 units. Robot sales in the automotive industry increased by 22% and remain still the major customer of industrial robots with a share of 33%. The electrical/electronics industry has been catching up with 32% of the total share. Many of these robots use commercial softwares. The reason for this is safety features that robot manufacturers can guarantee by remaining close to end-users. Open-source software today doesn't offer real safety features, and developers/end-users need to develop their safety systems. Without these safety systems, robots cannot be part of the factory. When it comes to service robots, there are a lot of these robots, but still, we are not seeing many of them in our homes and in our everyday use. This type of robot must deal with the unstructured human environment. If they are capable of dealing with these conditions, they can perform tasks like vacuum cleaning, cooking, or hosting people in hotels. A technological breakthrough, especially in the field of artificial intelligence (AI), gives robots a possibility to work in the human's environment. Besides technology readiness, robots need to be cheaper and more open to users in order to be widely adopted. Nowadays, modern factories work on the popular concept of Smart Industry that focuses on flexible and smart automatization.

ROS (Robot Operating System) community is trying to overcome all these obstacles to introduce open-source concepts to robotics. The main concept of ROS, besides being open-source, is modular and reusable software in order to make robot programming accessible to everyone. In this way, ROS promotes flexibility and accelerates deploy time. Because of its good features ROS is a part of academic research for a while. For it to become a part of an industrial environment standardization is very important. ROS community recently that has increased safety and synchronization being one step closer to real-time framework.

The behavior of physical systems in many situations may better be expressed with an analytical model. Robot modeling and analysis essentially involve its kinematics. For robotic manipulators having high Degrees Of Freedom (DOF) with multiple degrees in one or more joints, an analytical solution to the inverse kinematics is probably the most important topic in robot modeling. The foundations of systems modelling Numerical analysis the development of algorithms for simulation They also include considerations about computer hardware for simulation and about special software and compilers. The journal also publishes articles concerned with specific applications of modelling and simulation in science and engineering, with relevant applied mathematics, the general philosophy of systems simulation, and their impact on disciplinary and interdisciplinary research.

The software can be used to interactively perform robotics analysis and off-line programming of robot dynamics such as forward and inverse dynamics as well as to interactively teach and simulate the basic principles of robot dynamics in a very realistic way. In order to demonstrate the user-friendly features of the toolbox much better, simulation of the NS robot manipulator (Stanford Arm) is provided as an example. Many universities and institutes experience difficulty in training people to work with expensive equipment's. A common problem faced by educational institutions concerns the limited availability of expensive robotics equipment's, with which students in the didactic program can work, in order to acquire valuable "hands on" experience. Therefore, the Robot Simulation Software (RSS) nowadays is paramount important. This paper presents the development of a visual software package where a AL5B Robot arm has been taken as a case study. It utilizes ROS/Simulink and AutoCAD as the tools for testing motional characteristics of the AL5B Robot arm. A robot model will be developed; the Forward, Inverse Kinematics, Velocity Kinematics "Jacobian" and Path Planning problems will be implemented and tested. developed package will be used as an educational tool. We think that this work will increase the education, training, research and development possibilities for robotics classes in graduate and under graduate studies.

3.2 PROPOSED METHODOLOGY

This paper mainly deals with robot arm modeling for industrial polishing applications. From this model, we analyze the forward kinematics of the multi-step free controller and calculate the angle of the robot arm's joints based on the inverse kinematic model, thus deducing the controller's dynamics solution. ROS provides a set of software libraries and tools to help developers to build robot applications. Since it is an open-source platform, the main goal of the community is to introduce ROS as a standard in robotics. ROS allows researchers and developers to use different programming languages for creating their applications. ROS works with C++, Python, and Lisp. There is a beta version of client libraries that supports Java, C#, R, and other languages. ROS software is organized in packages. A package might contain nodes, independent libraries, configuration files, third-party software, or anything else that constitutes a useful module.

The main goal of organizing software in packages is to provide easy-to-consume and reusable software. ROS packages follow a “Goldilocks” principle: enough functionality to be useful, but not too much that the package is heavy-weight and difficult to use from other software. In order to control a robot from ROS, appropriate drivers need to be installed. Also, all drivers for any device are organized in packages. One of the most valuable ROS packages for industrial robots is Ros control. The Ros control is a set of packages that includes controller interface, controller managers, transmissions, and hardware interface. The package takes as input joint state data from the robot's encoders and an input set point. It uses a generic control loop feedback mechanism to control the output sent to actuators. Also, tracking of coordinate frames is very important in robotics. Program is called a node. Nodes can communicate with each other. Communication between ROS nodes is peer-to-peer. There are three methods of communication through topics, services, and action services. Integrate the robotic arm model and control algorithms with other systems and devices, such as sensors, actuators, and user interfaces, using ROS-based software tools. Deploy the robotic arm model and control algorithms in a real-world scenario, and compare the simulation results with the actual performance of the robotic arm.

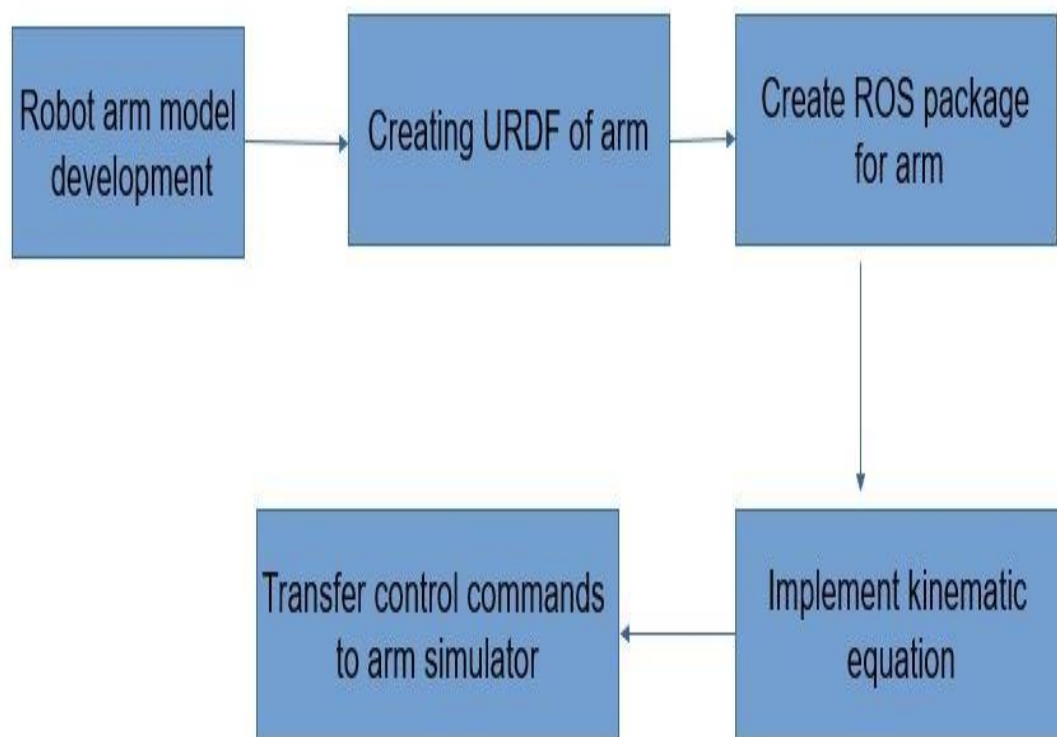


Fig: 3.1: Proposed Block Diagram

Robot arm development involves several stages, including designing the arm, creating a simulation model, and implementing kinematic equations to control the arm's movement. The first step in creating a URDF model is to design the robot arm. This involves defining the number of links and joints, their dimensions, and their connections. Once the design is complete, the next step is to create a URDF file that describes the arm's physical properties.

The URDF file is created using an XML editor and consists of several elements, including robot, link, joint, and transmission. The robot element is the top-level element and contains all the other elements. The link element describes the physical properties of a robot link, such as its mass, shape, and inertia. The joint element describes the type of joint connecting two links, such as a revolute joint or a prismatic joint.

The transmission element describes the relationship between a joint and its associated actuator. It defines the actuator's parameters, such as its type and control signals. The transmission element also defines the motor's performance characteristics, such as torque and speed. Once the URDF model is created, the next step is to create a ROS package that contains the arm's control software. The ROS package includes several components, including the ROS node that communicates with the arm simulator, the kinematics solver that computes the arm's joint angles, and the controller that generates the control commands. The kinematics solver uses the URDF model to compute the arm's joint angles based on the desired end-effector position and orientation. creating a URDF model of a robot arm and developing a ROS package to control its movement involves several steps, including designing the arm, creating a URDF file that describes the arm's physical and kinematic properties, and developing the ROS package that contains the arm's control software.

3.2.1 System Requirements

The processor requirement of a core i3, i5, or i7 is a relatively standard requirement for most software development tasks. However, the amount of RAM required, which is 2-4GB, can vary depending on the complexity of the project. Generally, the more RAM you have, the smoother your system will run when working with larger files or running multiple programs simultaneously. The hard drive requirement of 256GB is a reasonable amount of storage for most software development projects.

Pycharm IDE is an Integrated Development Environment (IDE) that is commonly used for developing Python code. Gazebo is a 3D simulator that is often used in robotics and automation applications. Both of these software tools will be necessary for your project. In terms of programming languages, you will need to be familiar with Python, which is a popular high-level programming language, as well as XML and URDF. XML is a markup language that is often used for structuring data and describing the content of documents, while URDF (Unified Robot Description Format) is a file format used to describe a robot's physical characteristics and its behavior.

3.3 ROS (Robot Operating System)

ROS (Quigley et al., 2009) provides a unified platform independent of languages and platforms for publishing and subscribing to data streams. It also comprises a large collection of commonly used functionality and applications for robot software development such as hardware drivers, robot models, simulation tools, data-types, planning, perception and other algorithms. It provides a useful architecture for developing and deploying robot systems, which can be easily modelled using graph-based techniques (Aitken et al., 2014).

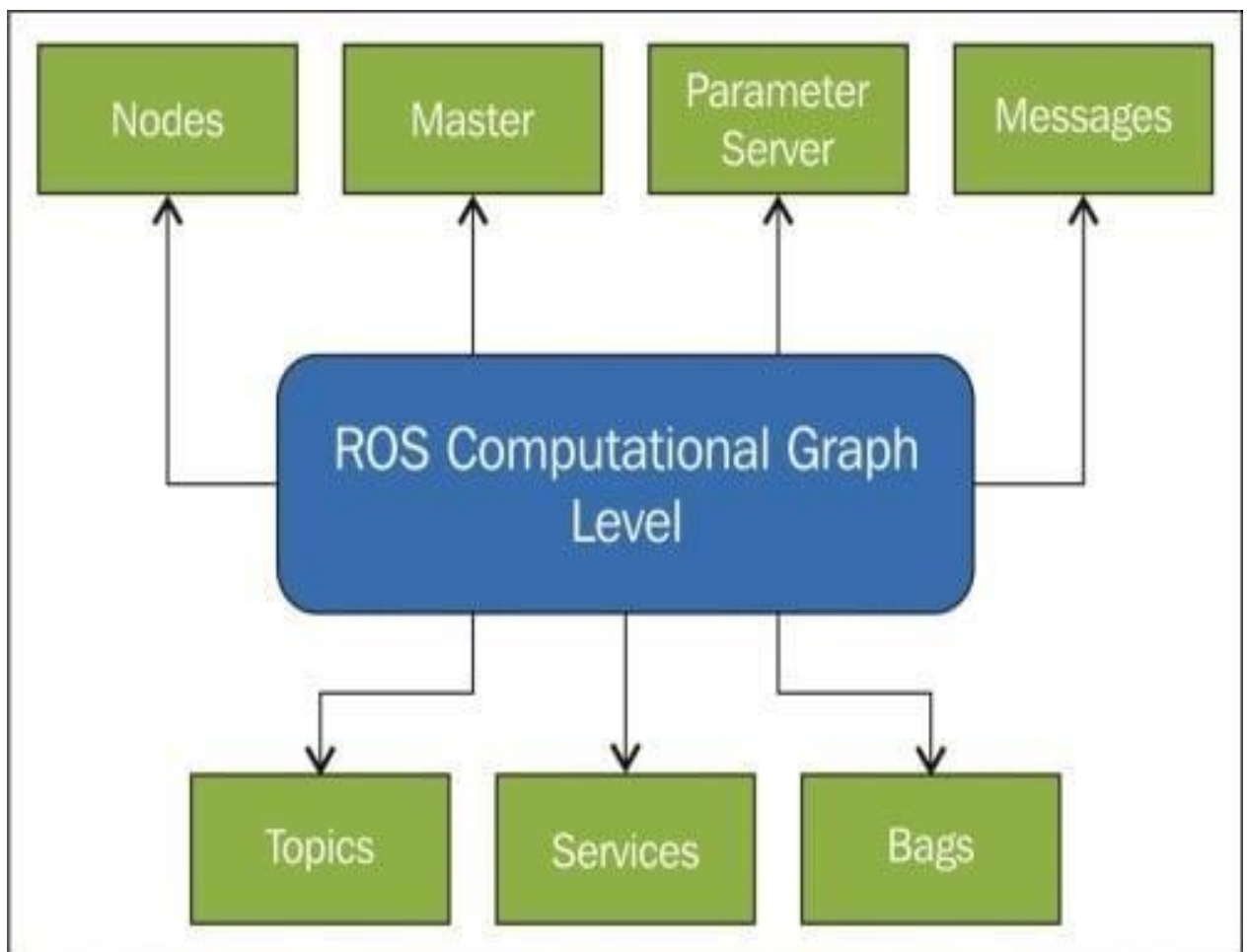


Fig: 3.2: Architecture of ROS

Software in ROS is organized in *packages*. A package might contain a library, a dataset, configuration files or anything else that logically constitutes a useful module. The goal of these packages is to provide this functionality in an easy-to-consume manner so that software can be easily reused. In general, ROS packages follow a “Goldilocks” principle: *enough* functionality to be useful, but not too much that the package is heavyweight and difficult to use from other software.

3.3.1 ROS tools and packages

ROS provides a set of software libraries and tools to help developers to build robot applications. Since it is an open-source platform, the main goal of the community is to introduce ROS as a standard in robotics. ROS allows researchers and developers to use different programming languages for creating their applications. ROS works with C++, Python, and Lisp. There is a beta version of client libraries that supports Java, C#, R, and other languages. ROS program is called a node. Nodes can communicate with each other. Communication between ROS nodes is peer-to-peer. There are three methods of communication: through topics, services, and action services.

3.3.2 ROS tools

The main tools ROS provides are rviz, rosbag, rqt_bag, rqt_plot, rqt_graph, command-line tool, and other. For 3D visualization rviz is used as presented. This tool can let you combine sensor data, robot models, and for example work cell 3D model for a better understanding of the ongoing scenario. For data logging and visualization of sensor data, ROS uses rosbag and rqt_bag. The tool for making plots is rqt_plot. By selecting the desired topic, this tool automatically generates a plot from its data. And to see what nodes, topics, and services are running on the system rqt_graph is needed. Gazebo software can be used to run physics simulation on the ROS platform.

3.3.3 ROS Packages

ROS software is organized in packages. A package might contain nodes, independent libraries, configuration files, third-party software, or anything else that constitutes a useful module. The main goal of organizing software in packages is to provide easy-to-consume and reusable software. ROS packages follow a “Goldilocks” principle: enough functionality to be useful, but not too much that the package is heavy-weight and difficult to use from other software. In order to control a robot from ROS, appropriate drivers need to be installed. Also, all drivers for any device are organized in packages. One of the most valuable ROS packages for industrial robots is `ros_control`. The `ros_control` is a set of packages that includes controller interface, controller managers, transmissions, and hardware interface. The package takes as input joint state data from the robot’s encoders and an input set point. It uses a generic control loop feedback mechanism to control the output sent to actuators. Also, tracking of coordinate frames is very important in robotics. The package that lets users keep tracking of multiple coordinate frames over time is `tf`. In a robotic system, there are many 3D coordinate frames that change over time, such as a world frame, base frame, gripper frame, etc. This is very important if there is a need, in an application, for calculating pose of the object in robot gripper relative to its base. Or if information about the current pose of the base frame in the map frame is needed. For mobile robotics, there are a couple of important packages, such as `move_base`, `navigation`, `robot_pose_ekf`, `gmapping`, etc.

The `move_base` provides an implementation of the algorithm that, for a given goal in the world, will try to reach it with a mobile robot base. The navigation package combines information from odometry, sensor streams, and tries to compute safe velocity commands. These commands use as input to the `move_base` package. Because of noise, slippage, and incorrect modeling of robot geometry odometry might not give a correct robot position. The `robot_pose_ekf` package applies Extended Kalman Filter (EKF) to sensor data to improve “the belief” of robot position. In the end, the robot needs to know the environment and use `gmapping` package for map building. This package uses OpenSlam Gmapping algorithm for mapping. As the output from this package user gets 3-D occupation grid map.

3.4 ALTERNATIVE API

There are several existing alternative APIs available for the KUKA LBR iiwa. Each has been built with a slightly different focus and consequently is customised to its own domain. In this section we provide a brief summary of the most prominent. ROS Industrial is a working group developing interfaces aimed at wide-scale industrial usage, typically the frame- work focuses on the larger capacity arms that are part of the KUKA range (Edwards and Lewis, 2012). At present this is only an experimental package so is subject to regular alterations, however, the main focus of the API is not on co-working, but capturing more general industrial use.

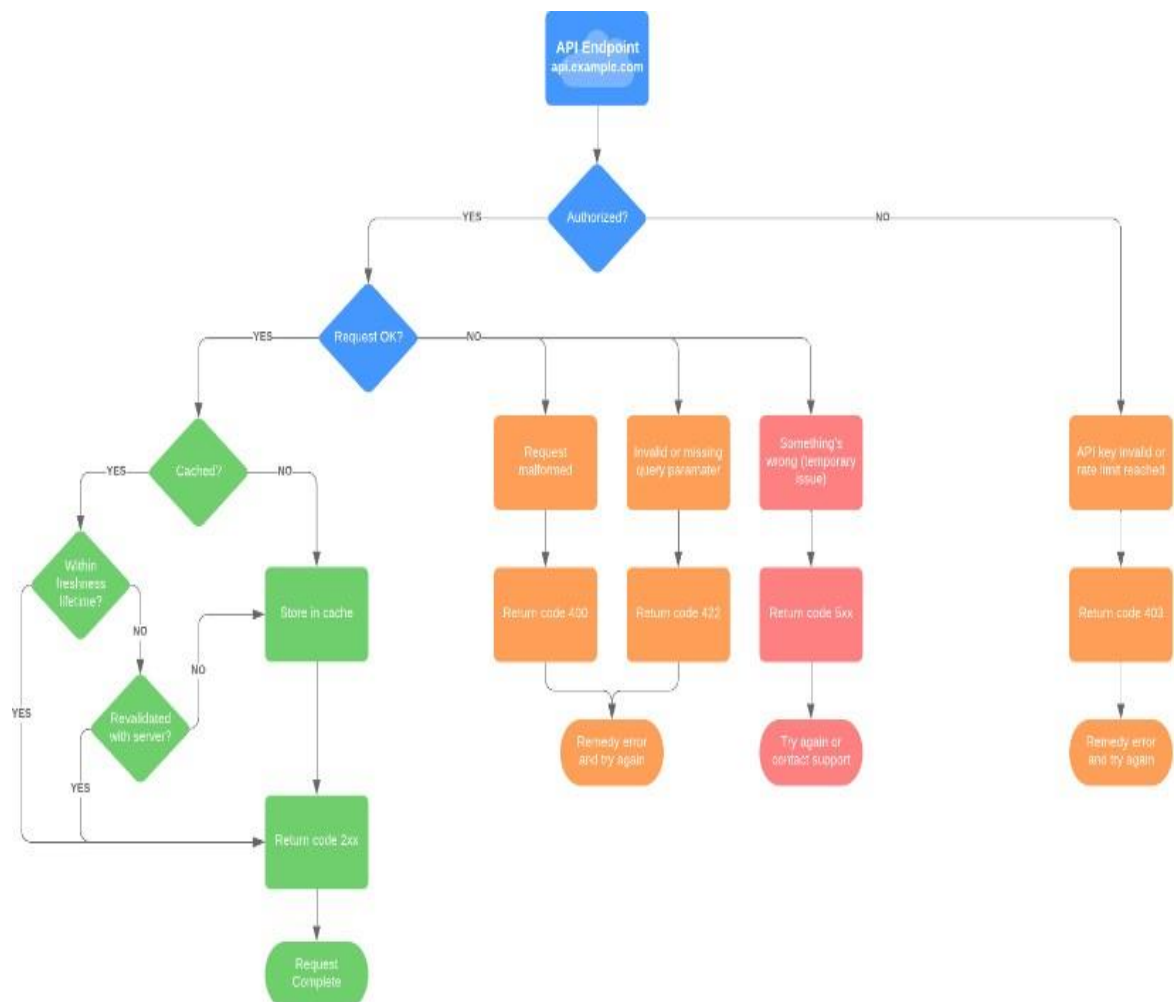


Fig: 3.3: Execution of API

There are several reasons why alternative APIs may be developed. For example, some developers may prefer using a particular programming language or platform that is not supported by the original API. Alternatively, an alternative API may be created to provide better documentation, error handling, or performance optimizations than the original API.

- **Additional functionality:** Alternative APIs may offer additional features or capabilities that are not available in the original API, allowing developers to build more complex applications or perform tasks more efficiently.
- **Better documentation:** Alternative APIs may offer more detailed documentation, making it easier for developers to understand and use the API.
- **Cross-platform support:** Alternative APIs may support multiple platforms, allowing developers to use the same API across different operating systems or hardware architectures.
- **Better error handling:** Alternative APIs may provide better error handling, making it easier to debug and troubleshoot issues in the application.
- **Improved performance:** Alternative APIs may be optimized for performance, offering faster response times or lower resource usage than the original API.
- However, there may also be some disadvantages to using an alternative API, such as:
- **Lack of compatibility:** Alternative APIs may not be fully compatible with the original API, making it more difficult to switch between them or integrate them into existing code.
- **Limited support:** Alternative APIs may have a smaller user base or developer community than the original API, making it harder to find support or resources.
- **Security risks:** Alternative APIs may have security vulnerabilities that are not present in the original API, potentially exposing the application or user data to security threats.
- **Integration challenges:** Alternative APIs may require additional configuration or integration work, increasing the complexity of the application.

3.5 API ARCHITECTURE

The API architecture focuses on breaking out the functionality that would normally be available within the KUKA Sunrise controller run on the Smartpad. The architecture can be viewed as extending the capability of the KUKA LBR iiwa, using the generic structure shown in Figure 1. The API exposes an interface to operation on a network of machines. This allows different sensing methods and extra computing resources to be easily deployed and exploited in operations of the KUKA LBR iiwa.

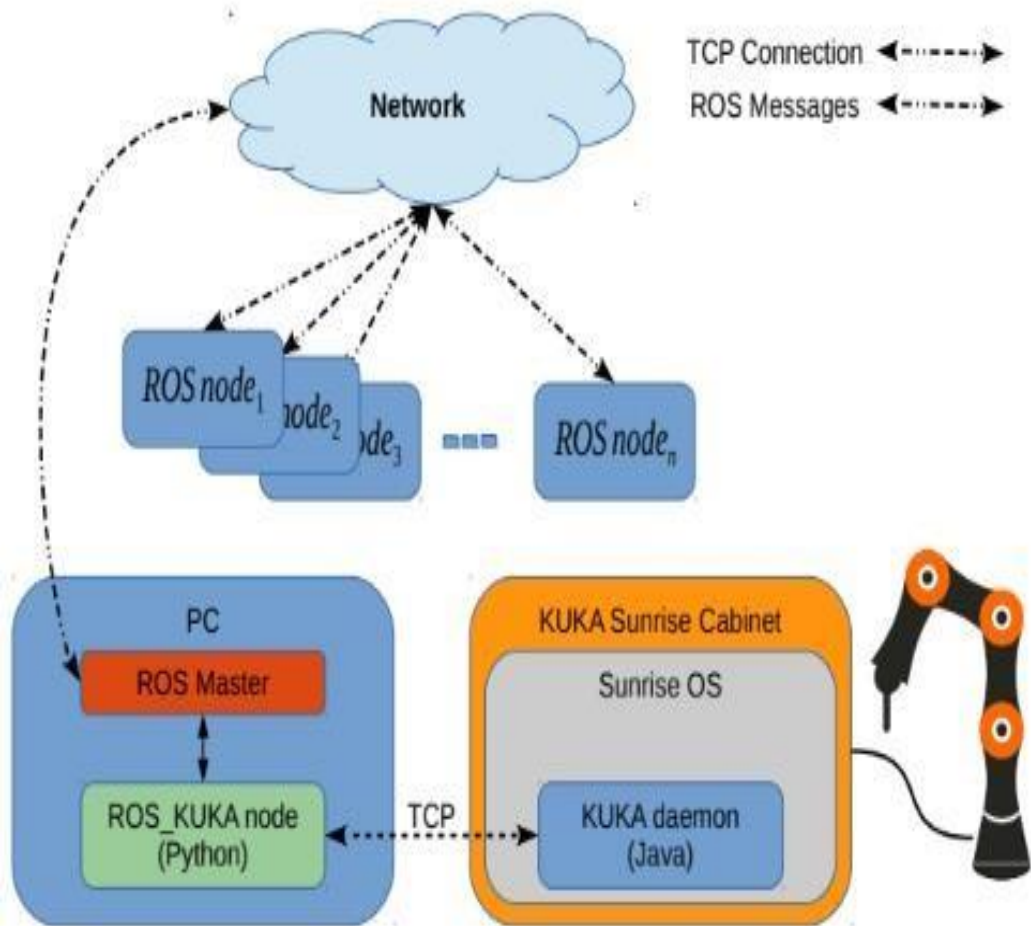


Fig: 3.4: Components of ROS Interface Architecture

API architecture refers to the design and structure of an API (Application Programming Interface). It encompasses the various components of an API, such as its endpoints, methods, parameters, data formats, authentication mechanisms, and documentation. A well-designed API architecture is crucial for creating a reliable, scalable, and user-friendly interface that developers can easily use to interact with the underlying system or service.

- **REST (Representational State Transfer):** REST is a popular architectural style used for building web-based APIs. It is based on the principles of statelessness, client-server architecture, and cacheability, and it emphasizes the use of HTTP verbs (GET, POST, PUT, DELETE) to perform CRUD (Create, Read, Update, Delete) operations on resources.
- **Microservices:** Microservices architecture is an approach to building APIs that emphasizes the creation of small, independent, and loosely-coupled services that can be easily managed and scaled. Each service is responsible for a specific function or task and communicates with other services through a lightweight messaging protocol.
- **GraphQL:** GraphQL is a query language for APIs that allows clients to request only the data they need and nothing more. It provides a flexible and efficient way to fetch data from a server, and it can reduce network latency and bandwidth usage by minimizing over-fetching and under-fetching of data.
- **SOAP (Simple Object Access Protocol):** SOAP is an older architectural style for building web-based APIs. It uses XML-based messages to transmit data between the client and server and provides a standardized set of rules for performing operations on resources.

API architecture is important because it determines the quality and usability of the API, as well as its impact on the software ecosystem in which it operates. A well-designed API architecture can facilitate the development of robust and interoperable software systems, reduce the time and cost of integration and maintenance, and enhance the user experience and satisfaction.

In addition to these principles, there are several other factors that can influence API architecture, including security, scalability, performance, and usability. For example, APIs may use authentication mechanisms such as OAuth or JWT to ensure that only authorized users can access protected resources. APIs may also implement caching mechanisms to reduce response times and improve scalability, or they may use rate limiting to prevent overloading of the system. Documentation is also an important part of API architecture, as it provides developers with the information they need to use the API effectively. API documentation should include detailed descriptions of endpoints, methods, parameters, data formats, and error handling, as well as examples and tutorials that demonstrate how to use the API in practice. API architecture is an important aspect of software development that involves designing and structuring the components of an API. By following established principles and best practices, developers can create reliable, scalable, and user-friendly APIs that enable efficient interaction with the underlying system or service. An API (Application Programming Interface) architecture is a set of guidelines, standards, and protocols used to design and build APIs. It is a framework that defines how different components of an API work together to provide a seamless and efficient experience for developers and end-users.

Endpoints define the location of resources and specify the type of action to be taken. Request and response formats specify how data is structured, with common formats being JSON and XML. Authentication and authorization are essential to ensure that only authorized users can access and manipulate data, with various methods available, such as OAuth2 and API keys. Rate limiting restricts the number of requests that can be made to prevent abuse and ensure that the API remains stable and responsive. Error handling should provide clear and informative messages that help developers understand and fix problems. Documentation should be comprehensive and include examples of requests and responses, making it easier for developers to use the API.

3.6 URDF

Unified Robotics Description Format, URDF, is an XML specification used in academia and industry to model multibody systems such as robotic manipulator arms for manufacturing assembly lines and animatronic robots for amusement parks. URDF is especially popular with users of Robotics Operating System, ROS. You can import URDF models into the Sims cape Multibody environment for simulations, analyses, or control design tasks. See the humanoid robot featured example for a simple use case.

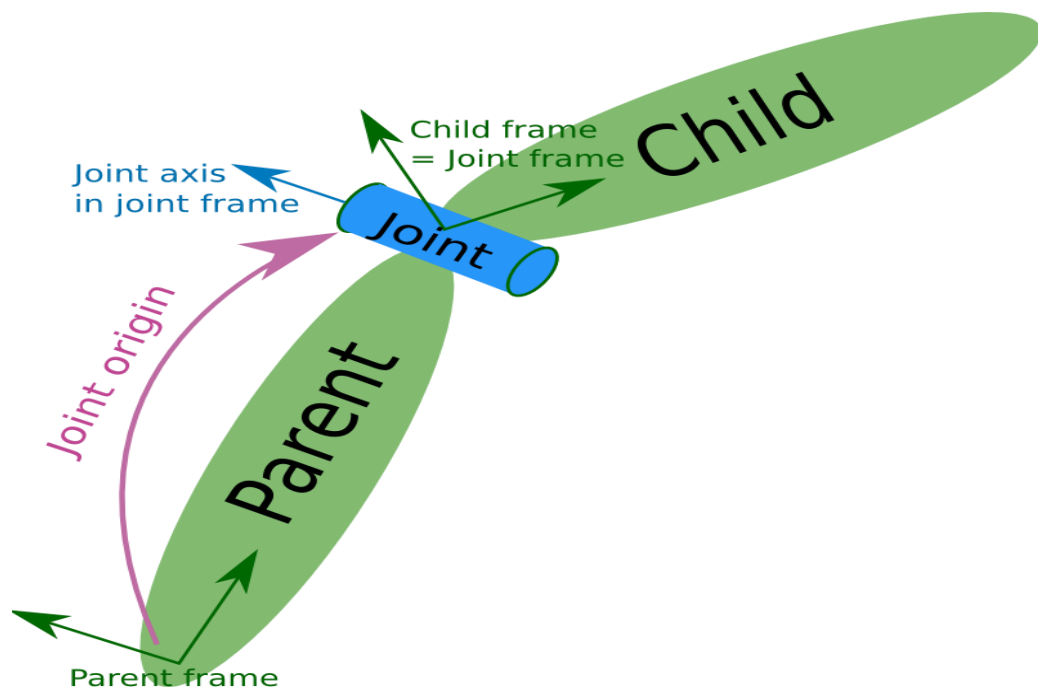


Fig: 3.5: Architecture of URDF

In URDF robot description model, The whole elements between tag and tag are the description of one specified link, the same as joint description between tag and tag. Stereo Lithography (STL) [9] and Collada files can be imported as meshes into URDF. The recommended format for best texture and colour support is Collada files, though STL files are also supported. What we need to do is to make scattered links together. Commonly, a joint is the connection of two different links. And the two links can be divided into parent link and child link. Since every link has its fixed own coordinate frame. The transformation information from parent link to child link must be filled into joint property block. RViz is a extremely useful visualization tool of ROS which can be used to display the topics communicated between nodes. RViz has a Graphical User Interface (GUI) to allow users to configure and modify robots. Launching RViz node and other essential nodes to establish a ROS network which can control the position of each joint provides a simple method to check the joint connection between links. Here are some brief introduction about main active nodes and topics transported between each other.

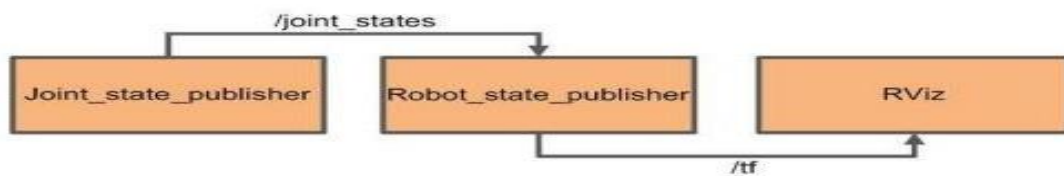


Fig: 3.6: Implementation of URDF

joint state publisher: This node reads the robot description parameter and publishes all non-fixed joints' information such as position, velocity, acceleration, at a specified frequency. It also provides a control GUI to set the joint position of manipulators. robot state publisher: This node subscribes to current joint positions of the manipulators and publishes transform data of coordinate frames attached to each link by the method called forward kinematics. tf: The tf library was designed to provide a standard way to keep track of coordinate frames and transform data within an entire system such that individual component users can be confident that the data is in the coordinate frame that they want without requiring knowledge of all the coordinate frames in the system.

URDF (Unified Robot Description Format) is a popular open-source file format used to describe the physical and kinematic properties of robots in a machine-readable format. It is widely used in the robotics community for modeling and simulating robots and robot systems. URDF is an XML-based file format that describes the geometry, kinematics, dynamics, and sensors of a robot. The geometry section defines the visual and collision properties of the robot, including the shape, size, and orientation of each link and joint. The kinematics section defines the joint types and their limits, which control the motion of the robot. The dynamics section defines the mass, inertia, and friction properties of each link, which affect how the robot moves and responds to external forces. Finally, the sensors section defines the sensors attached to the robot, such as cameras, lidars, or tactile sensors. URDF files are typically used in conjunction with robotics software frameworks, such as ROS (Robot Operating System), Gazebo, or MoveIt!, to simulate and control robots. They can also be used to generate robot models for use in simulation or physical hardware systems.

URDF files are human-readable, making it easy to edit and modify the robot description. They can also be validated using tools such as the URDF Validator, which checks the syntax and structure of the URDF file for errors. URDF has become an important tool in the field of robotics, allowing researchers and developers to easily model and simulate robots. Its open-source nature and wide adoption in the community have made it a standard format for describing robots, which has led to the development of a wide range of tools and resources for working with URDF files.

In URDF, a robot model is represented as a hierarchical tree structure, where each node corresponds to a robot link or joint, and each edge represents a transformation or constraint between two adjacent nodes. URDF allows the definition of various types of geometric primitives, such as boxes, cylinders, and spheres, as well as the import of more complex geometries from external files, such as STL or COLLADA. URDF is a flexible and extensible format that can accommodate different types of robots, from simple manipulators to humanoid robots or mobile robots. URDF can also be combined with other ROS-based software tools, such as robot state publishers, motion planners, or controllers, to provide a comprehensive solution for robot modelling and simulation.

3.6 GAZEBO SIMULATION

Gazebo is a multi-robot simulator for outdoor environments. Like Stage [12] (part of the Player project), it is capable of simulating a population of robots, sensors and objects, but does so in a three-dimensional world. It generates both realistic sensor feedback and physically plausible interactions between objects (it includes an accurate simulation of rigid body physics). By realistically simulating robots and environments code designed to operate a physical robot can be executed on an artificial version. Numerous researchers have also used.

Gazebo to develop and run experiments solely in a simulated environment. Controlled experimental setups can easily be created in which subjects can interact with manipulators in a realistic manner. There is a big difference between RViz visualisation and Gazebo simulation, the former one is used to display relative position of links, but the later one can be regarded as a experimental copy of real robot in virtual world. A transmission is an element in a control pipeline that transforms efforts/flow variables. Transmission-specific code (not robot-specific) implementing bidirectional effort and flow maps under a uniform interface shared across transmission types. Gazebo controllers need the corresponding transmission interface in URDF file.

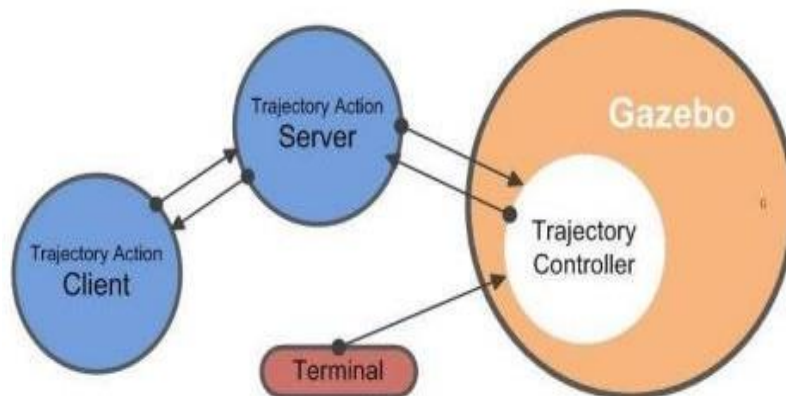


Fig: 3.7: Trajectory of Gazebo

Gazebo is a powerful open-source robot simulation software that allows users to model and simulate robots in a virtual environment. It is widely used in robotics research, education, and development to test and validate algorithms and control strategies before deploying them on physical hardware. Gazebo provides a highly realistic simulation environment that models the physical properties of objects, such as gravity, friction, and collisions. It includes a library of models and sensors that can be used to construct complex robot systems, including robots, sensors, and environments. The simulation process in Gazebo involves several components, including the simulation world, robot models, and controllers. The simulation world defines the environment in which the robot operates, including objects such as walls, obstacles, and terrain. The robot model describes the physical and kinematic properties of the robot, including its joints, links, sensors, and actuators. Controllers provide the logic and algorithms that control the behavior of the robot, including its movements and interactions with the environment.

One of the key features of Gazebo is its ability to simulate sensors, such as cameras, lidars, and sonars. This allows users to test and validate sensor configurations and algorithms in a controlled environment before deploying them on real hardware. Gazebo also provides a range of tools and APIs for controlling and interacting with the simulation. This includes APIs for scripting and programming, as well as plugins for integrating with other software frameworks, such as ROS (Robot Operating System). Gazebo is designed to be extensible and customizable, allowing users to add their own models, sensors, and controllers to the simulation. It also supports multi-robot simulations, allowing users to simulate and test interactions between multiple robots and objects in a single environment. Gazebo simulation is a powerful and flexible tool that can be used for a variety of robot simulation tasks, such as testing and validating new robot designs, evaluating the performance of different control algorithms, and training robot models for various tasks, such as navigation, grasping, or manipulation. The availability of Gazebo simulation as an open-source tool also facilitates the collaboration and sharing of knowledge and resources within the robotics community.

CHAPTER 4

EXPERIMENTAL METHODS

4.1 KINEMATICS

Many methods can be used in the direct kinematics calculation. The Denavit-Hartenberg analyses is one of the most used, in this method the direct kinematics is determinate from some parameters that have to be defined, depending on each mechanism. However, it was chosen to use the homogeneous transformation matrix. This transformation specifies the location (position and orientation) of the hand in space with respect to the base of the robot, but it does not tell us which configuration of the arm is required to achieve this location. Kinematics is an important concept in many fields, including physics, engineering, and robotics. In robotics, kinematics is used to describe the motion of robots and their components, such as arms, legs, and grippers.

The study of kinematics involves several key concepts, including position, displacement, velocity, and acceleration. Position describes the location of an object in space, often relative to a fixed reference point. Displacement is the change in position of an object over time, and velocity is the rate at which an object changes its position. Acceleration is the rate at which the velocity of an object changes over time. Kinematics can be further divided into two subfields: forward kinematics and inverse kinematics. Forward kinematics involves calculating the position and orientation of the end-effector of a robot arm based on the joint angles. Inverse kinematics, on the other hand, involves calculating the joint angles required to position the end-effector at a desired location and orientation. Robot kinematics is important in robotics because it provides a framework for modeling, planning, and controlling the motion of robots in various applications, such as manufacturing, healthcare, and space exploration. Kinematic models can be used to optimize the design and performance of robot systems, as well as to enable robots to perform complex tasks with high accuracy and efficiency.

Forward kinematics is used to determine the position and orientation of the end-effector of a robot arm given the joint angles. This can be done using geometric and trigonometric methods, such as the Denavit-Hartenberg notation or the Euler angles. By knowing the position and orientation of the end-effector, it is possible to plan and control the movement of the robot arm to perform tasks such as grasping objects, moving them to a desired location, and manipulating them in a specific way. Inverse kinematics, on the other hand, involves calculating the joint angles required to position the end-effector at a desired location and orientation. This is a more complex problem than forward kinematics, as there can be multiple solutions or no solutions at all. Inverse kinematics is used in robotics to plan and control the movement of the robot arm to perform tasks such as picking up objects or manipulating them in a specific way. There are various methods for solving inverse kinematics problems, including numerical methods, analytical methods, and optimization-based methods. These methods can be used to calculate the joint angles required to position the end-effector at a desired location and orientation.

Kinematics is also used in the design and control of robotic systems. By understanding the kinematics of a robot, it is possible to optimize its design and control strategies to improve its performance and accuracy. Overall, kinematics is a fundamental concept in mechanics and robotics, providing a framework for understanding the motion of objects and the design and control of robotic systems. Forward kinematics and inverse kinematics are important tools in robotics, allowing engineers and researchers to plan and control the movement of robots to perform complex tasks in a variety of environments. To understand kinematics in robotics, it's important to first define some key terms. A robot joint is a point of articulation where two or more robot links meet and can rotate or move relative to each other. A robot link is a rigid element that connects two or more robot joints, and the end-effector is the part of the robot that interacts with the environment, such as a gripper or a tool. Kinematic models can be used to determine the relationship between the joint angles and the position and orientation of the robot end-effector. This is known as forward kinematics. For example, given the joint angles of a robotic arm, a forward kinematics model can be used to determine the position and orientation of the end-effector.

4.2 DESIGN CALCULATIONS

In order to determine the most appropriate cross sectional shape of the robotic arm's links, three possible geometries were considered; a solid cylindrical beam, a hollow cylindrical beam and a set of two rectangular beams. The section modulus, S , is a geometrical property, which indicates the strength of beams based on their cross sectional shapes or areas.

$$S = I/c \quad (4.1)$$

where I is the second moment of area (mm^4), and c is the maximum distance from the neutral axis (mm). It can be seen from the generated graph, shown in Figure 3, that for any value of the cross sectional area, the set of two parallel rectangular beams has the highest value of S , followed by the hollow cylindrical beam and then lastly, the solid cylindrical beam. This means that the set of two parallel rectangular beams have the highest structural strength to weight ratio out of the three links, considering they are all made of the same material. Hence, rectangular beams were used for the robotic arm's links.

The following calculation was carried out in order to ensure that the torque of the elbow and shoulder motors will be enough to rotate the joints with a maximum payload of 200g. The maximum required torque corresponding to the worst case scenario with the arm fully stretched out is shown in Figure. From the moments equilibrium equations, Eqs. 2 and 3, the results showed that a minimum of 0.351N.m for the elbow and 1.259N.m for the shoulder are required. Consequently, based on the available servomotors in the market, the selected servomotors for the elbow and shoulder joints provide a maximum torque of 0.941N.m and 2.422N.m, respectively.



Fig: 4.1: Torque of Elbow and Shoulder

$$T_{elbow} = [F_{load} \times L_2] + \left[W_2 \times \left(\frac{L_2}{2} \right) \right] \quad (4.2)$$

$$T_{shoulder} = [F_{load} \times (L_1 + L_2)] + \left[W_2 \times \left(\frac{L_2}{2} + L_1 \right) \right] + (W_{m3} \times L_1) + \left(W_1 \times \frac{L_1}{2} \right) \quad (4.3)$$

4.3 FORWARD KINEMATICS

Five frames were assigned to the robotic arm as shown in Figure. The frame assignment procedure followed can be summarized in three steps. Assign the Z_i axis pointing along the i -th joint axis (either one of two choices). It is better to assign Z_0 and Z_{n+1} parallel to Z_1 and Z_n , respectively. Assign the X_i axis pointing along the common perpendicular from Z_i to Z_{i+1} . Assign the Y_i axis according to the right hand rule. The non-rotating reference Frame $\{0\}$ was attached to a fixed point at the center of the rotary base. Frame $\{1\}$.

corresponding to Link (1) was attached at its upper end. Note that the origin of this frame could have been appointed to any location along Link (1) since Z_0 and Z_1 are parallel to each other. Frame $\{2\}$ rotates with Link (2) and is located at the intersection between Link (1) and Link (2), thus having X_2 along the common perpendicular pointing from Z_2 to Z_3 . Frame $\{3\}$ rotates with Link (3) and is attached at its intersection with Link (2). Finally, Frame $\{4\}$ corresponding to the end-effector is located midway between the two grippers. Based on the defined frames, the Denavit-Hartenberg (D-H) parameters [2] were determined as shown in Table 1.

Table : 4.1 : D-H PARAMETERS

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0°	0	L_1	θ_1
2	90°	0	0	θ_2
3	0°	L_2	0	θ_3
4	0°	L_3	0	0°

The definition of each of the D-H parameters is as follows:

α_i : Link twist is the angle about X_i between (Z_i, Z_{i+1})

a_i : Link length is the distance along X_i between (Z_i, Z_{i+1})

d_i : Link offset is the distance along Z_i between (X_{i-1}, X_i)

θ_i : Joint angle is the angle about Z_i between (X_{i-1}, X_i)

The D-H parameters were then used to compute the homogenous transform matrices (1T_0 , 2T_1 , 3T_2 and 4T_3) by substituting in the general formula for ${}^iT_{i-1}$.

$${}^{i-1}T_i = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & \alpha_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1}d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.4)$$

$${}^0T_1 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.5)$$

$${}^1T_2 = \begin{bmatrix} c_2 & -s_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s_2 & c_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.6)$$

$${}^2T_3 = \begin{bmatrix} c_3 & -s_3 & 0 & L_2 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.7)$$

$${}^3T_4 = \begin{bmatrix} 1 & 0 & 0 & L_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.8)$$

where c_i is the cosine of the angle θ_i , and s_i represents the sine. By multiplying the homogeneous transformation matrices in the right order, 4T_0 describing the end effector frame with the reference frame may be obtained.

$${}^0T_4 = {}^0T_1 {}^1T_2 {}^2T_3 {}^3T_4 \quad (4.9)$$

$${}^0T_4 = \begin{bmatrix} c_1 c_{23} & -c_1 s_{23} & s_1 & c_1 c_{23} L_3 + c_1 c_2 L_2 \\ s_1 c_{23} & -s_1 s_{23} & -c_1 & s_1 c_{23} L_3 + s_1 c_2 L_2 \\ s_{23} & c_{23} & 0 & s_{23} L_3 + s_2 L_2 + L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.10)$$

The first three entries of the last column in 4T_0 represent the Cartesian coordinates of the position of the origin of the end effector frame {4} with respect to the reference frame {0}.

$$x = c_1 c_{23} L_3 + c_1 c_2 L_2 \quad (4.11)$$

$$y = s_1 c_{23} L_3 + s_1 c_2 L_2 \quad (4.12)$$

$$z = s_{23} L_3 + s_2 L_2 + L_1 \quad (4.13)$$

Notice that c_{23} is the cosine of the sum of θ_2 and θ_3 and s_{23} is the sine of the sum of these two angles. The procedure for obtaining the analytical expression for the x, y and z coordinates is called forward kinematics analysis. The forward kinematics, Eqs. 11 to 13, produce the position of the end effector as a function of the joint angles θ_1 , θ_2 and θ_3 .

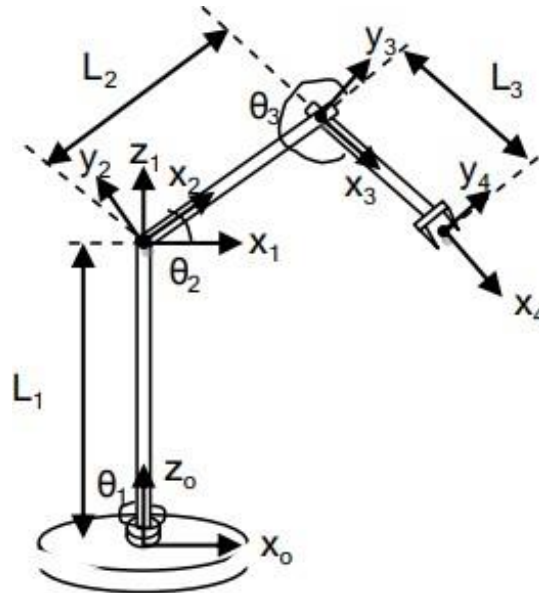


Fig: 4.2: Frame Assignments For The Kinematics Analysis

4.4 INVERSE KINEMATICS

In the inverse kinematics, the position coordinates equations, Eqs. 11 - 13, were used to find the analytical expressions of the rotational angles (i.e. θ_1 , θ_2 and θ_3) for the waist, shoulder and elbow motors as a function of the desired x, y and z coordinates. The derivation of the inverse kinematic equations is summarized as follows:

Firstly, dividing Eq. 12 by Eq. 11 gives

$$\frac{y}{x} = \frac{s_1 c_{23} L_3 + s_1 c_2 L_2}{c_1 c_{23} L_3 + c_1 c_2 L_2} = \frac{s_1 (c_{23} L_3 + c_2 L_2)}{c_1 (c_{23} L_3 + c_2 L_2)} = \frac{s_1}{c_1} = \tan \theta_1 \quad (4.14)$$

As a result θ_1 is explicitly expressed as

$$\theta_1 = \text{atan2} \left(\frac{y}{x} \right) \quad (4.15)$$

The (atan2) function generates the arctangent of the argument and the proper quadrant based on the signs of both the denominator and the numerator.

Furthermore, by summing up the squares of Eqs. 11 to 13 and rearranging one can obtain an explicit expression for the cosine of θ_3 as a function of the robotic arm geometry and the coordinates of the target point.

$$c_3 = \frac{x^2 + y^2 + z^2 - (L_1^2 + L_2^2 + L_3^2) - 2L_1(z - L_1)}{2L_2L_3} \quad (4.16)$$

From the Pythagorean trigonometric identity

$$s_3 = \pm \sqrt{1 - c_3} \quad (4.17)$$

Notice the ambiguity in algebraic sign in the above equation, Eq. 17, is an indication of multiple solutions. The two solutions correspond to the two different configurations that the arm could take to reach the same point (i.e. elbow up and elbow down). Accordingly from the previous,

$$\theta_3 = \text{atan2}\left(\frac{s_3}{c_3}\right) \quad (4.18)$$

Considering Eq. 13, applying the angle addition trigonometric identity, and defining the following variables

$$\begin{aligned} k_1 &= (c_3 L_3 + L_2) \\ k_2 &= (s_3 L_3) \end{aligned}$$

it can be shown that

$$z - L_1 = k_1 s_2 + k_2 c_2 \quad (4.19)$$

By rearranging and squaring both sides of the equation,

$$(z - L_1 - k_1 s_2)^2 = (k_2 c_2)^2 \quad (4.20)$$

Then by expanding the left side and regrouping the terms, the following quadratic equation can be obtained

$$(k_1^2 + k_2^2)s_2^2 + 2k_1(L_1 - z)s_2 + (z^2 + L_1^2 - k_2^2 - 2zL_1) = 0 \quad (4.21)$$

Using the quadratic formula, θ_2 can be expressed in terms of the robotic arm geometry, the target point z - coordinate, and the value of θ_2 as obtained from Eq. 18

$$\theta_2 = \sin^{-1}\left(\frac{A \pm \sqrt{B}}{2(k_1^2 + k_2^2)}\right) \quad (4.22)$$

Where,

$$\begin{aligned} A &= -2k_1(L_1 - z) \\ B &= (2k_1(L_1 - z))^2 - 4(k_1^2 + k_2^2)(z^2 + L_1^2 - k_2^2 - 2zL_1) \end{aligned}$$

It can be noticed from Eq. 22 that there are two solutions, roots, for θ_2 . It is noteworthy that there is another approach to find θ_2 . To obtain this alternative formula for θ_2 , the x and y coordinates equations, Eqs. 11 and 12, should be rewritten as

$$x = c_1(k_1c_2 - k_2s_2) \quad (4.23)$$

$$y = s_1(k_1c_2 - k_2s_2) \quad (4.24)$$

In obtaining Eqs. 23 and 24, the trigonometric angle addition identity was used. Now by subtracting Eq. 24 from Eq. 23.

$$\frac{x-y}{c_1-s_1} = k_1c_2 - k_2s_2 \quad (4.25)$$

Together with Eq. 19, Eq. 25 constitute a system of two transcendental equations that can be solved by transformation to polar coordinates using

$$r = \sqrt{(k_1)^2 + (k_2)^2} \quad \text{and} \quad \gamma = \text{atan2}\left(\frac{k_2}{k_1}\right)$$

to give

$$k_1 = r\cos\gamma \quad \text{and} \quad k_2 = r\sin\gamma$$

Thus,

$$\frac{x-y}{c_1-s_1} = r\cos\gamma c_2 - r\sin\gamma s_2 \quad (4.26)$$

Applying the angle addition trigonometric identity,

$$\frac{x-y}{r(c_1-s_1)} = \cos(\gamma + \theta_2) \quad (4.27)$$

Notice that Eq. 19, using the same transformation and the angle addition trigonometric identity, can be written as

$$\frac{z-L_1}{r} = \sin(\gamma + \theta_2) \quad (4.28)$$

Dividing Eq. 28 by Eq. 27 and substituting for the values of k_1 , k_2 , r and γ ,

$$\theta_2 = \text{atan2} \left[\frac{(z-L_1)(c_1-s_1)}{(x-y)} \right] - \text{atan2} \left[\frac{s_3 L_3}{(c_3 L_3 + L_2)} \right] \quad (4.29)$$

Equation 29 offers another way to explicitly compute θ_2 . However, both methods are supposed to produce the same result. Therefore, just by substituting the target x , y and z coordinates into Eqs. 15, 18, and 22 (or 29) will generate θ_1 , θ_2 and θ_3 , after which the robotic arm links will rotate accordingly to reach that location in space.

In this section the workspace is identified analytically by investigating the equations obtained from the forward kinematics calculations. The cosine of an angle lies between a value of -1 and 1, therefore the following mathematical inequality holds true based on the formula obtained in Eq. 16,

$$(L_2-L_3)^2 \leq x^2 + y^2 + (z-L_1)^2 \leq (L_2+L_3)^2 \quad (4.30)$$

The result of this design is represented as an equation of a sphere with a center at $(0, 0, L_1)$ and inner and outer radii of $(L_2 - L_3)$ and $(L_2 + L_3)$, respectively. This equation describes the physical limitations of the arm's movement due to its design constraints. For example, the arm cannot move beyond the outer radius of the sphere without compromising its structure. Finally, the analysis shows that the arm's minimum and maximum horizontal reach is ± 270.8 mm and ± 399.5 mm, respectively. To further analyze the arm's capabilities, a 3D plot has been generated that shows its workspace. The workspace is the area where the arm can reach and perform its intended tasks. This information is useful in determining the arm's capabilities in performing specific tasks that require a specific range of motion. Overall, the design and analysis of the robotic arm provide valuable information for understanding its capabilities and limitations in performing its intended tasks.

CHAPTER 5

DESIGN DETAILS OF ROBOTIC ARM

Robots are usually expensive and quite susceptible to damage and undesired interaction with themselves or their surroundings. Three-dimensional simulation is a sustainable method to test and rapidly iterate during robot application development. Robot Operating System (ROS) is a middleware for multiple robot software tools and also provides simulation platforms like Rviz and Gazebo. Robotic arms are the most common robots to exist in the last century. They are built out of multiple actuators and passive links forming an actuated chain that can be used to manipulate objects using different end-effectors. They are extremely useful in industry as well as personal settings for assembly, drilling, machining, pick-and-place, and other applications. A robot arm could have revolute or prismatic joints for rotation or linear motion respectively.

A robotic arm is a mechanical device that is designed to perform tasks similar to that of a human arm. It is a type of manipulator that is used in a wide range of industries, including manufacturing, aerospace, and healthcare. Robotic arms are capable of performing repetitive tasks with high precision and accuracy, making them ideal for use in industries where precision is important. This paper will provide a detailed explanation of robotic arms, their components, types, applications, and future prospects. A typical robotic arm consists of several components, including the base, shoulder, elbow, wrist, and end effector. The base is the foundation of the arm, while the shoulder allows the arm to rotate in the horizontal plane. The elbow joint allows the arm to bend vertically, while the wrist allows the end effector to rotate. The end effector is the tool or device that is attached to the end of the arm and is used to perform the task. There are several types of robotic arms, including cartesian, cylindrical, spherical, and articulated. Cartesian arms are the most common type of robotic arm, and they move in a straight line along three axes. Cylindrical arms move in a straight line along one axis and rotate around another. Spherical arms move in a spherical motion, while articulated arms have multiple joints that allow them to move in a wide range of directions. Robotic arms are used in a wide range of applications.

Including manufacturing, aerospace, healthcare, and entertainment. In manufacturing, they are used to assemble products, weld components, and paint surfaces. In aerospace, they are used to assemble and maintain spacecraft and satellites. In healthcare, they are used in surgery, rehabilitation, and prosthetics. In entertainment, they are used to create special effects and animatronics.

The future of robotic arms is promising, with advancements in technology allowing for more precise and complex tasks to be performed. The development of artificial intelligence and machine learning algorithms has allowed for robotic arms to be programmed to learn and adapt to new tasks, making them even more versatile. There is also a growing trend towards the development of collaborative robotic systems, where robotic arms work alongside human operators to improve efficiency and safety. In conclusion, robotic arms are mechanical devices that are designed to perform tasks similar to that of a human arm. They consist of several components, including the base, shoulder, elbow, wrist, and end effector. There are several types of robotic arms, including cartesian, cylindrical, spherical, and articulated, each with its own unique features and capabilities. Robotic arms are used in a wide range of applications, including manufacturing, aerospace, healthcare, and entertainment. The future of robotic arms is promising, with advancements in technology allowing for more precise and complex tasks to be performed, and the development of collaborative robotic systems allowing for improved efficiency and safety. The implementation of a robotic arm requires precise engineering and programming to ensure that it operates safely and accurately. The arm must be programmed with algorithms that control its movement and allow it to perform specific tasks. This programming can be done using various software tools and languages, depending on the application and the manufacturer's preferences.

The robotic arm's implementation also involves selecting appropriate sensors to monitor the arm's position and detect any obstacles that could interfere with its movement. These sensors may include encoders, accelerometers, and proximity sensors. Additionally, safety features such as emergency stop buttons, limit switches, and collision detection systems must be incorporated to ensure safe operation.

5.1 DESIGNING OF ROBOTIC ARM

In this article, we design a simple robot arm with four revolute joints and one prismatic joint. A Universal Robot Description Format (URDF) file with the robotics kinematic, visual, and collision model representation is needed for the robotics interaction with ROS. The inertial model may also be required for applications dealing with the robot dynamics. The use of SimMechanics as a tool to model the mechanics of the robot allows the possibility to verify model, as in real application, is unknown for the designer so that the ability of the control method to match the real model can be easily proven. That would allow the testing of techniques like the dynamic model-based control but also learning techniques that extract the model of the plant under consideration. Robotic arms are one of the most versatile and essential components of industrial automation systems. They are used in a wide range of applications such as manufacturing, assembly, packaging, material handling, welding, and painting, among others. The design of a robotic arm involves several stages, including conceptualization, mechanical design, electrical and control system design, and testing. In this article, we will discuss the various stages of designing a robotic arm in detail. The first step in designing a robotic arm is conceptualization. This stage involves identifying the requirements of the robotic arm, such as its payload capacity, reach, and precision.

The conceptualization stage also involves determining the type of robotic arm to be used. There are several types of robotic arms available, including Cartesian, cylindrical, spherical, articulated, and hybrid robotic arms. The type of robotic arm chosen will depend on the specific application requirements. Once the requirements and type of robotic arm are determined, the next step is mechanical design. This stage involves designing the structure of the robotic arm, including the arm segments, joints, and end-effector. The arm segments are typically designed using aluminum or steel tubes, which are lightweight and strong. The joints of the robotic arm can be either revolute or prismatic. The end-effector is the component of the robotic arm that interacts with the environment. It can be designed for a specific application, such as a gripper for material handling or a welding torch for welding applications.

such as the type of material being handled and the precision required. The electrical and control System design is another critical stage in designing a robotic arm. This stage involves designing the electrical components, such as motors, sensors, and actuators, and the control system that controls the movement of the robotic arm. The motors used in the robotic arm are typically servo motors, which provide precise control over the movement of the arm. The sensors used in the robotic arm include encoders, which provide feedback on the position and speed of the arm, and force sensors, which measure the force applied by the arm. The control system of the robotic arm can be either open-loop or closed-loop. In an open-loop control system, the control signals are sent to the motors without any feedback on the actual position or speed of the arm. In a closed-loop control system, the control signals are adjusted based on the feedback from the sensors, which allows for more precise control over the movement of the arm.

The final stage in designing a robotic arm is testing. This stage involves testing the performance of the robotic arm to ensure that it meets the requirements of the application. The testing can be done using simulation software, which allows for the virtual testing of the robotic arm, or using a physical prototype. The testing should include testing the accuracy and precision of the robotic arm, as well as its speed and payload capacity. The design of a robotic arm involves several stages, including conceptualization, mechanical design, electrical and control system design, and testing. Each stage is critical in ensuring that the robotic arm meets the requirements of the application. The design of a robotic arm should take into account the specific requirements of the application, including its payload capacity, reach, and precision. With the proper design and testing, a robotic arm can greatly improve the efficiency and productivity of industrial automation systems. Robotic arm designing is a complex process that involves multiple stages to ensure that the arm can perform its intended task accurately and efficiently. The first step in designing a robotic arm is to determine its application, which will dictate the necessary specifications for the arm's size, weight capacity, range of motion, and end effector or gripper. Once the application is determined, the arm's structure and components must be designed, such as the number of joints, type of actuators, sensors, and control systems needed. The end effector or gripper must also be designed to handle the intended task accurately. After designing the structure and components.

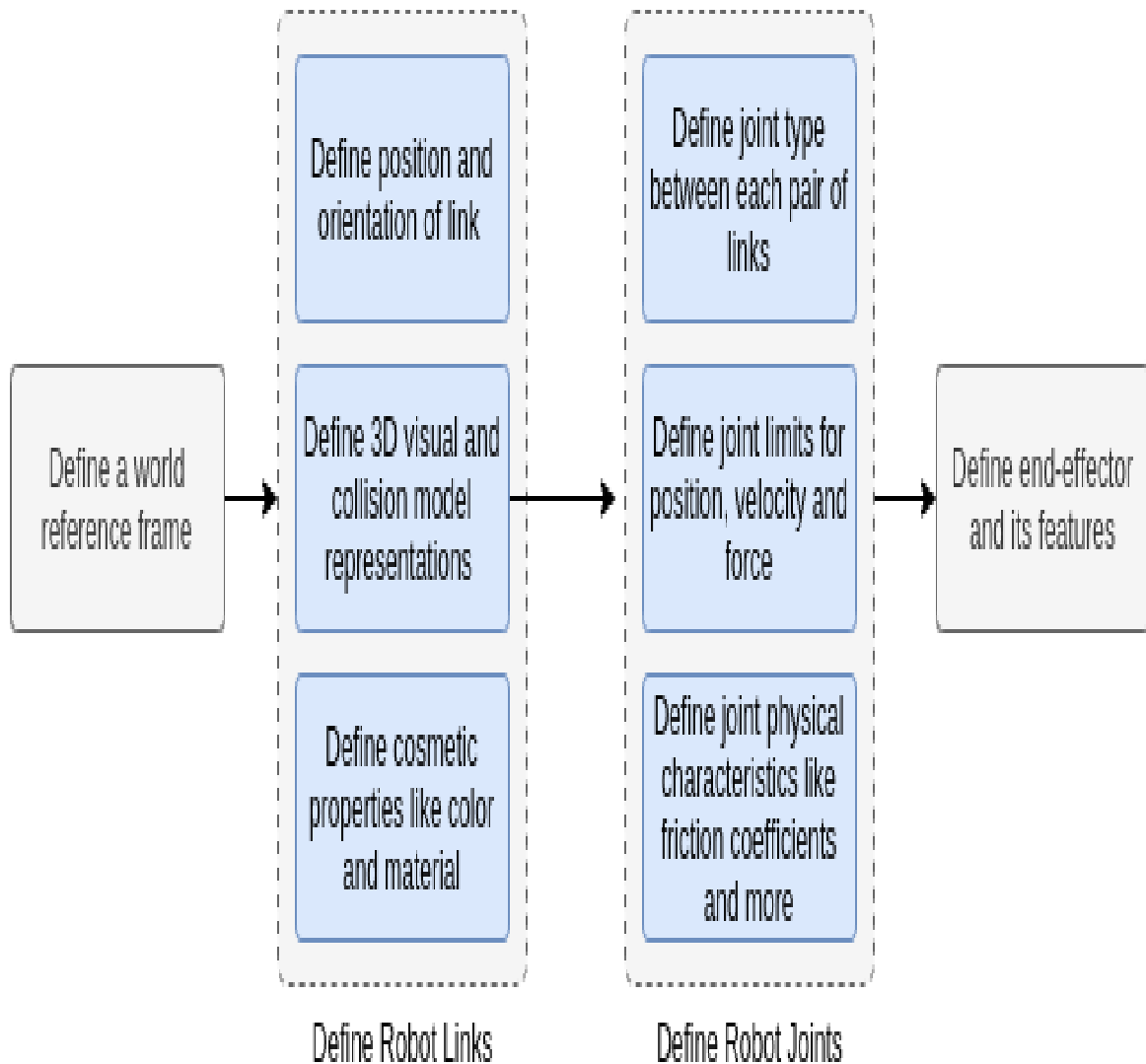


Fig: 5.1: Steps To Define a Robotic Arm URDF File

The URDF file is an XML file with dedicated tags for physical features like the material and colour of the links. For the 3D visual representation of the robot links, the `<visual></visual>` tags are used. They carry information about the geometry (within `<geometry></geometry>` tags) which can be a primitive shape usually centered at its geometric center or a mesh with the center defined in the 3D modelling software itself. URDFs support visual representation using primitive shapes or 3D meshes like COLLADA and STL files. The collision model is similar to the visual model within `<collision></collision>` tags but ideally with a coarser representation of the 3D model. The inertial model of the link is defined within `<inertial></inertial>` tags and has information about the mass and the inertia matrix.

The joints are defined within the `<joint></joint>` with information about the origin, axis of rotation, parent and child links, and limits for positions, velocities, accelerations, and forces. They often have friction and damping coefficient values as well. The URDF file provided in the package, `robot_arm.urdf`, is formed of primitive boxes (cuboids) and cylinders. It does not have any end-effector attached. After the robot description is ready, follow the steps to create a ROS package to simulate the design.

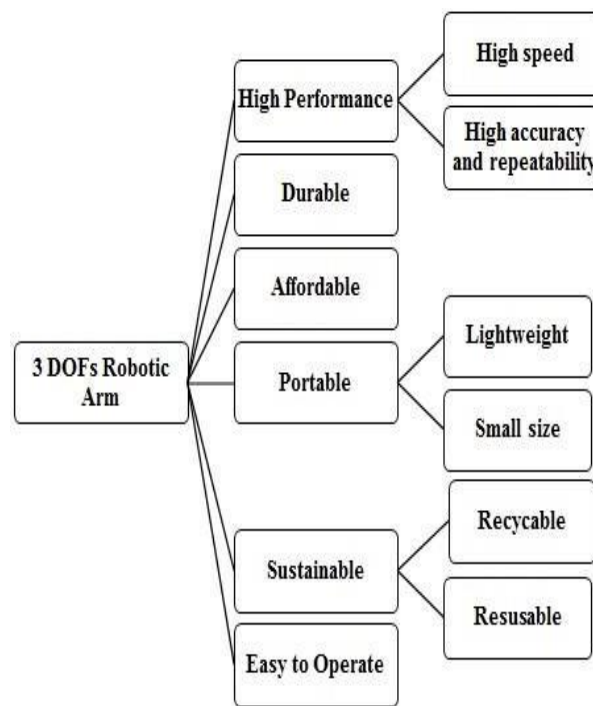


Fig: 5.2: Benefits of 3 DOF Robotic Arm

The robotic arm consists of three joints; the waist joint represented by rotation of the rotary table, the shoulder joint represented by the rotation of Link (1) and the elbow joint represented by Link (2)'s rotation. Each joint employs a specific servomotor to provide the torque required for the motion. Each motor receives the shaft position controlling sign. The inverse kinematics equations are then used to find the required joint angles corresponding to the aforementioned coordinates. Before constructing the robotic arm prototype, it was necessary to carry out certain calculations such as to determine the required torque, the forward and inverse kinematics, and the robotic arm workspace.

5.2 SIMULATION PIPELINE

Simulation is an essential tool for the design, development, and testing of robotic systems. The simulation pipeline in robotics involves several stages, including the creation of a virtual environment, the design of the robot model, the implementation of the control algorithm, and the analysis of the simulation results. In this article, we will discuss the various stages of the simulation pipeline in robotics in detail. The first stage in the simulation pipeline in robotics is the creation of a virtual environment. This stage involves creating a virtual 3D environment that accurately models the physical environment in which the robot will operate. The virtual environment should include all the necessary elements, such as obstacles, terrain, and lighting, that are required for the simulation. The virtual environment can be created using a variety of software tools, such as Unity or Unreal Engine. These tools provide a user-friendly interface for creating the virtual environment, and they also include a library of pre-built models that can be used to create the environment quickly.

The second stage in the simulation pipeline in robotics is the design of the robot model. This stage involves creating a 3D model of the robot that accurately reflects its physical characteristics, such as its shape, size, and weight. The robot model should also include the necessary sensors, such as cameras and lidar sensors, that are required for the simulation. The robot model can be created using 3D modeling software, such as SolidWorks or Blender. These tools provide a user-friendly interface for creating the robot model, and they also include a library of pre-built components that can be used to create the model quickly. The third stage in the simulation pipeline in robotics is the implementation of the control algorithm. This stage involves programming the control algorithm that will be used to control the movement of the robot in the virtual environment. The control algorithm should be designed to take into account the physical characteristics of the robot, such as its mass, inertia, and friction, as well as the dynamics of the virtual environment. The control algorithm can be programmed using a variety of programming languages, such as C++ or Python. The programming environment should include a simulation interface that allows the control algorithm to communicate with the virtual environment.

This stage involves analyzing the simulation results to determine if the robot behaves as expected in the virtual environment. The analysis should include a variety of metrics, such as the accuracy of the robot's movements, its speed, and its ability to navigate around obstacles. The simulation results can be analyzed using a variety of software tools, such as Python. These tools provide a user-friendly interface for analyzing the simulation results, and they also include a library of pre-built analysis functions that can be used to analyze the results quickly. Simulation is an essential tool for the design, development, and testing of robotic systems. There are several advantages to using simulation in robotics, including:

- **Cost-Effective:** Simulation is a cost-effective way to test and develop robotic systems without the need for expensive physical prototypes.
- **Time-Efficient:** Simulation allows for quick iterations of the design and control algorithm, which can save time in the development process.
- **Safety:** Simulation provides a safe environment for testing and developing robotic systems without the risk of damage to the robot or the environment.
- **Flexibility:** Simulation allows for testing of the robot in a variety of environments and scenarios, which can be difficult to replicate in the real world.

Simulation is an essential tool in the design, development, and testing of robotic systems. The simulation pipeline in robotics involves several stages, including the creation of a virtual environment, the design of the robot model, the implementation of the control algorithm. The pipeline typically involves four main stages: modeling, verification and validation, implementation, and execution. In the modeling stage, the simulation model is conceptualized and designed, including defining the system to be simulated, identifying key variables and inputs, and selecting the appropriate simulation methodology. The verification and validation stage involves testing the model to ensure that it accurately represents the real-world system and produces valid results. In the implementation stage, the model is translated into computer code and integrated into a simulation software package. Finally, the execution stage involves running the simulation model and analyzing the results to draw conclusions about the behavior of the system being studied. The simulation pipeline is an iterative process, as each stage may reveal issues that require revisions to earlier stages, leading to refinements and improvements to the simulation model.

ROS is highly modular in nature, allowing the use of multiple tools and services for each individual aspect of robotics. For this simulation task, execute the following steps on Ubuntu 16.04 environment with ROS Kinetic.

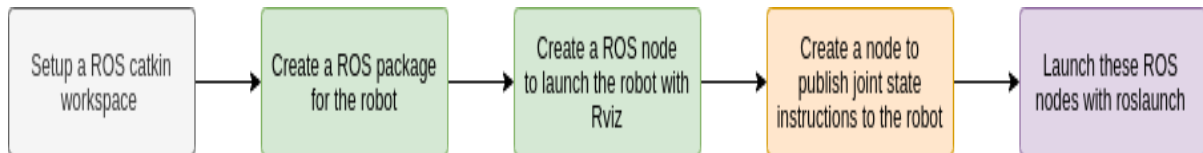


Fig: 5.3: ROS Package File Structure

Launch - Contains all the launch files

SRC - Contains the node that publishes the joint instructions

URDF- Contains the URDF model of the robot

RVIZ - Contains the default configurations for the Rviz simulator

Launch files are used to start up and configure ROS nodes and other components that make up a robot system. The launch directory typically contains files that define the launch configuration for the various nodes, topics, parameters, and other ROS components.

The "src" directory typically contains the source code for the ROS nodes, libraries, and other executables that are used in the robot system. In this case, it contains the node that publishes the joint instructions, which is likely responsible for sending commands to the robot's actuators to move its joints.

URDF (Unified Robot Description Format) is an XML file format used to describe the physical properties of a robot, including its joints, links, sensors, and other components. The URDF directory typically contains the URDF file that describes the robot's geometry, kinematics, and dynamics, which is used by other ROS components to visualize and interact with the robot.

RViz is a 3D visualization tool used to display and interact with robot models and sensor data in ROS. The RViz directory typically contains configuration files that define the default visualization settings for RViz, such as the camera view, lighting, and object properties. These files can be customized to suit the needs of the specific robot system being developed.

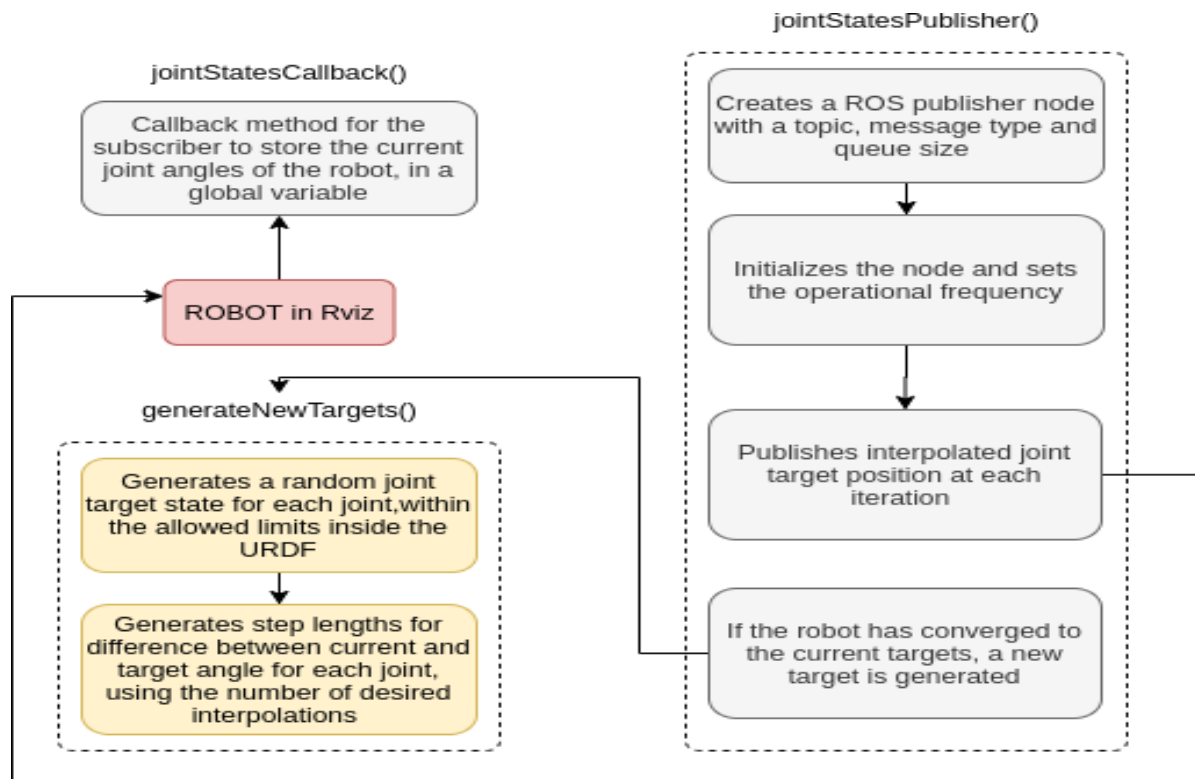


Fig: 5.4: Node Operation

To run the simulation, follow these steps:

1. In the terminal, `roslaunch panda_moveit_config demo.launch`
2. In another terminal, `roslaunch moveit_tutorials satyabama_robotic_arm.py`.

This runs the custom ROS node explained above and the robot starts executing the random targets it is commanded to.

Industrial robotic arms are widely used in various industrial applications, including manufacturing, assembly, and material handling. The robotic arms are typically used to automate repetitive tasks, increase productivity, and improve efficiency. Modelling and simulation of industrial robotic arms can help to improve the design, performance, and safety of the robotic systems. Robot Operating System (ROS) is an open-source software framework for developing robotic applications. ROS provides a set of tools and libraries for building and testing robotic systems. ROS has become a popular platform for modelling and simulating industrial robotic arms due to its flexibility, modularity, and extensibility.

Three degrees of freedom (DOF) refers to the number of independent motion axes of the robotic arm. A three-DOF robotic arm can move in three directions: up and down, left and right, and forward and backward. The three-DOF robotic arm is commonly used in applications that require simple movements, such as pick-and-place operations.

5.3 ADVANTAGES OF INDUSTRIAL ROBOTIC ARM MODELLING AND SIMULATION WITH ROS

- **Reduced Time and Cost:** Modelling and simulation of industrial robotic arms with ROS can reduce the time and cost of developing and testing the robotic system. Simulation can help to identify design issues and optimize the performance of the system before physical implementation.
- **Improved Accuracy:** Simulation can help to improve the accuracy of the robotic system by testing the system under various conditions and identifying potential errors in the system.
- **Increased Safety:** Simulation can help to increase the safety of the robotic system by identifying potential hazards and testing safety features of the system.
- **Flexibility:** ROS provides a flexible platform for modeling and simulating industrial robotic arms. The modular architecture of ROS allows users to customize the system to meet their specific needs.
- **Reusability:** ROS provides a library of reusable components that can be used to develop and test various robotic systems. This can reduce the time and cost of developing new systems.

5.4 DISADVANTAGES OF INDUSTRIAL ROBOTIC ARM MODELLING AND SIMULATION WITH ROS

- **Complexity:** Modelling and simulation of industrial robotic arms with ROS can be complex, especially for users who are not familiar with the software.
- **Limited Realism:** Simulation may not accurately reflect the real-world conditions and limitations of the robotic system. This can lead to unrealistic expectations and potential errors in the system.
- **Hardware Limitations:** Simulation may not accurately reflect the hardware limitations of the robotic system. This can lead to incorrect assumptions about the performance of the system.
- **Software Bugs:** ROS is an open-source software framework, and as such, may contain software bugs and errors that can affect the performance of the robotic system.
- **Limited Support:** ROS is an open-source software framework, and as such, may not have the same level of support and documentation as commercial software packages.

In conclusion, modelling and simulation of industrial robotic arms with ROS can provide significant advantages in terms of time, cost, accuracy, safety, flexibility, and reusability. However, there are also several disadvantages, including complexity, limited realism, hardware limitations, software bugs, and limited support. Therefore, it is essential to carefully evaluate the advantages and disadvantages of using ROS for modelling and simulating industrial robotic arms and to determine whether it is the most appropriate tool for the specific application. It is important to carefully evaluate whether ROS is the most appropriate tool for modelling and simulating industrial robotic arms in a specific application. Factors such as the level of complexity required, the required level of realism, the availability of expertise and resources, and the cost-benefit tradeoffs must all be taken into account when making this determination. This allows for early identification and resolution of design flaws and reduces the risk of costly errors during the physical testing phase. ROS also enables developers to run numerous simulations in parallel, which can lead to significant time savings.

CHAPTER 6

RESULTS AND DISCUSSION

6.1 SIMULATION RESULTS

To evaluate the performance of a robotic arm, we utilized a virtual simulation environment that comprised a 3D model of the robotic arm and the object it had to pick and place. We controlled the simulation using various inputs, such as the object's position, the robotic arm's orientation, and the joint angles.

- **Inputs and Outputs:**

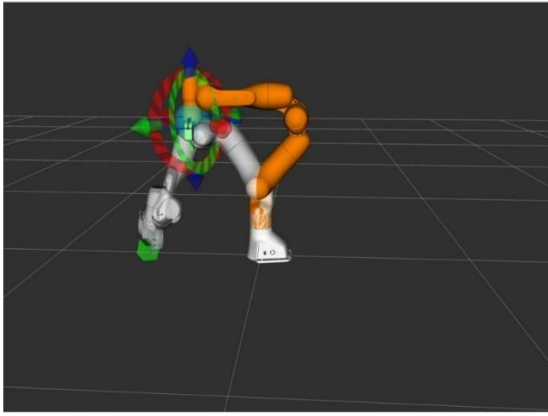
During the simulation, we tested various inputs to observe the behavior of the robotic arm when picking and placing the object. These inputs included different object positions, angles of the robotic arm, and combinations of joint angles. The output of the simulation was the object's position and orientation after the robotic arm had picked and placed it.

- **Metrics or Performance Indicators:**

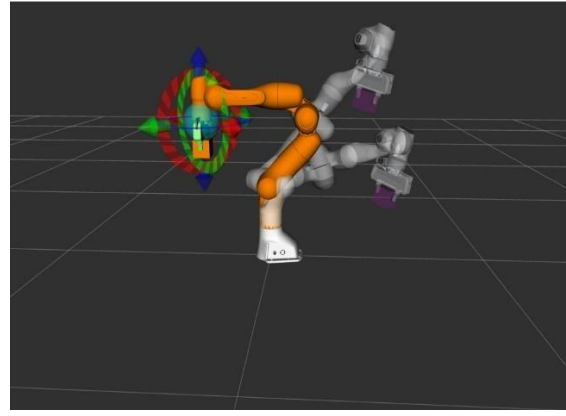
We monitored several metrics and performance indicators during the simulation to assess the robotic arm's performance. These metrics included the time taken to complete the task, the accuracy of the placement, and the smoothness of the robotic arm's movements.

- **Simulation Results:**

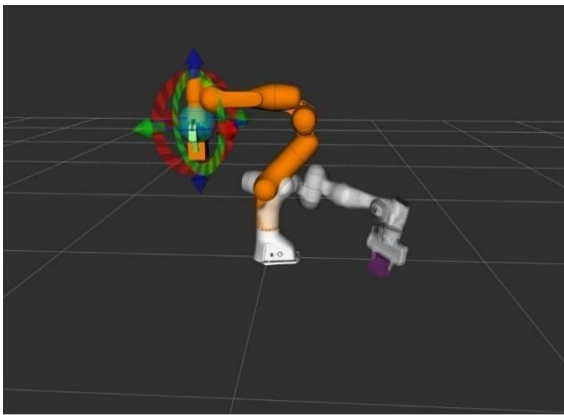
The simulation revealed that the robotic arm could pick and place the object accurately in any direction. However, the time taken to complete the task varied depending on the object's position, with some positions taking longer than others. While the robotic arm's movements were sufficiently smooth, they could still be improved in future iterations of the simulation.



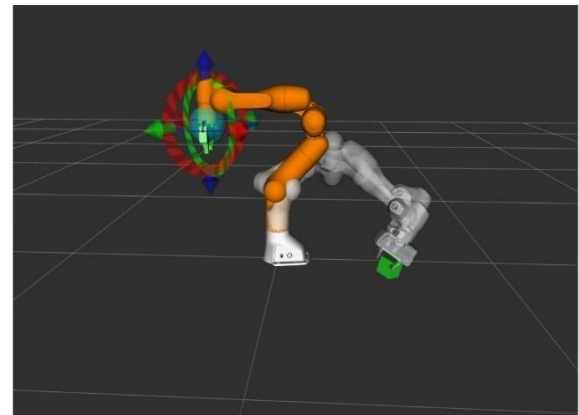
(A)



(B)



(C)



(D)

Fig: 6.1: Simulation Results Of Robotic Arm

6.2 DISCUSSION:

- **Performance and Behavior:**

Based on the simulation results, we concluded that the robotic arm could accurately pick and place the object in any direction. However, further optimization might be necessary to reduce the overall time taken to complete the task. Also, the robotic arm's movements could be made more fluid and efficient.

- **Strengths and Limitations:**

Our approach's strengths lie in the successful simulation of the robotic arm's behavior in a virtual environment, allowing us to test various inputs without physical testing. However, the approach's limitations are that we did not perform physical testing to validate the simulation results, and the simulation might not fully reflect the robotic arm's real-world behavior, necessitating further testing and refinement.

- **Comparison to Other Similar Projects:**

Our project is comparable to other projects aiming to develop robotic systems for object manipulation and transportation. However, we specifically focused on developing a system that could pick and place objects in any direction, which we believe could have significant applications in various industries, from manufacturing to logistics and warehousing.

6.3 GRAPHS PLOTTED BASED ON THE ABOVE SIMULATION

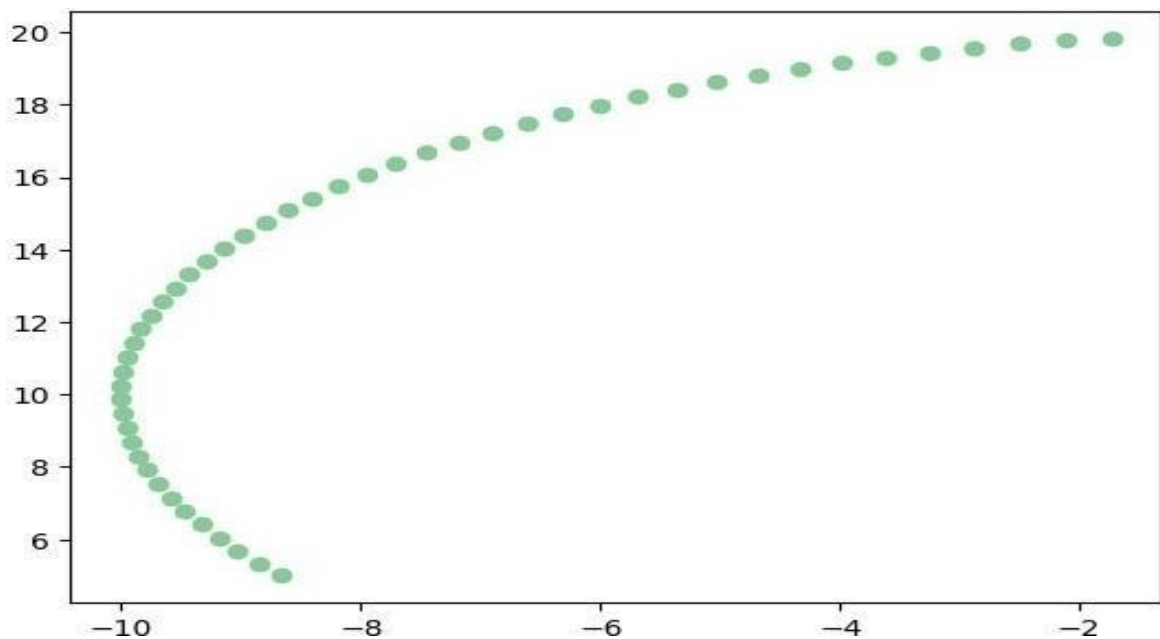


Fig: 6.2: Robot Object Pick Graph

Robot Object Pick Graph (Fig 6.2) is a graph that represents the trajectory of a robot's end effector to pick up an object from its home position in Cartesian space. The X-axis shows the robot's end effector position, and the Y-axis shows the distance travelled by the end effector in the vertical direction. The Y-axis values range from 2.23 to 18.65, while the X-axis values range from 2.66 to -4.65. The trajectory seems to follow a parabolic path with a peak at approximately $X=3.66$ and $Y=7.66$. The robot starts at the home position, moves towards the object, picks it up, and returns to the starting position.

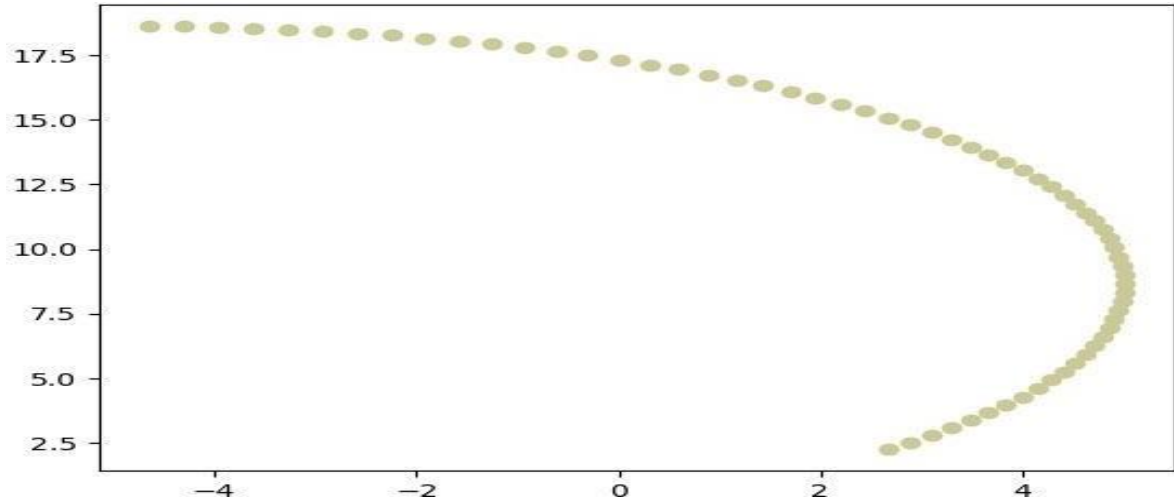


Fig: 6.3: Robot Object Drop Graph

The Robot Object Drop Graph (Fig 6.3) represents the trajectory of the robot's end effector while dropping the picked-up object. The X and Y axes denote the position of the end effector in cartesian space. The Y axis values range from 19.84807753 to 5, while the X axis values range from 1.736481777 to -8.660254038. The graph shows a gradual decrease in the Y axis values and a slight zig-zag pattern in the X axis values. Overall, the graph indicates that the robot successfully completed the task of dropping the object at a designated location.

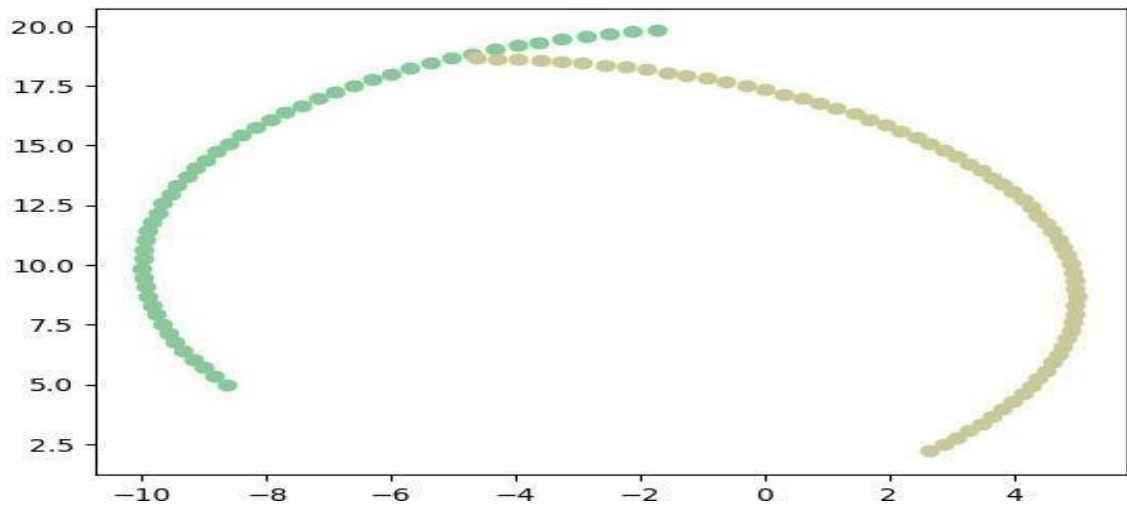


Fig: 6.4: Robot Complete Trajectory Graph

The given information describes a graph (Fig 6.4) depicting the complete trajectory of a robot's end effector during an operation of object pick and placement. The graph represents the positions of the end effector in Cartesian space over time, with the x-axis denoting the position of the end effector along the horizontal axis, and the y-axis representing its position along the vertical axis. The x-axis values in the graph range from 2.66 to -4.65, while the y-axis values range from 2.23 to 18.65. The graph captures the entire movement of the end effector during the operation and provides a visual representation of the trajectory followed by the robot throughout the task.

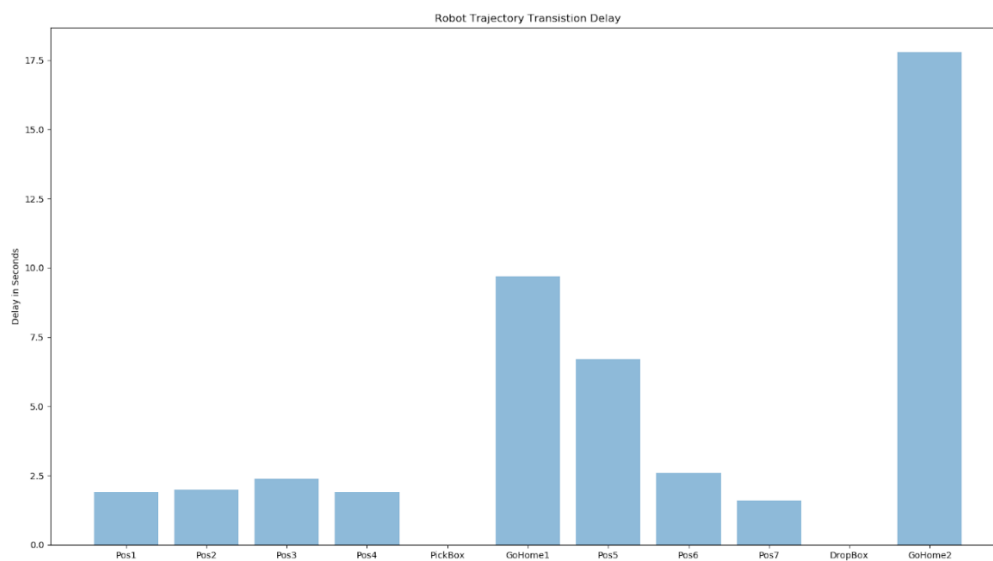


Fig: 6.5: Robot Trajectory Latency Graph

The Robot Trajectory Latency Graph (Fig 6.5) provided is a plot of the time taken for each movement of a robot, where the X-axis represents each position and action of the robot, and the Y-axis represents the time taken in seconds. The graph shows that most of the movements take around 1.8 to 2.4 seconds to complete, with the exception of the Pick Box action, which takes only 0.003 seconds. However, the robot takes significantly longer to complete the GoHome1 and GoHome2 actions, taking 9.688 and 11.087 seconds respectively, suggesting that these positions may be further away from the robot's starting point. Overall, the Robot Trajectory Latency Graph can be used to evaluate the performance of the robot's movements and identify areas for optimization to improve its overall efficiency.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

This paper presented how ROS can lead to open-source robotics. Although, ROS is not on a level like a commercial manufacturer, because of safety standards and real-time features, the ROS community is making efforts to overcome these problems. This paper also presented basic tools and packages that can accelerate the process of developing and deploying. Use case for industrial robots shown that ROS poses capabilities for running applications in which a force/torque sensor integration. In the case of mobile robots, it has been shown that students can easily learn basic postulates with ROS. Using ROS packages developing specific algorithms becomes an independent task. The objective of the project was to model a in a component-based manner, and implement its dynamic model from Gazebo in a effort to set up the means for simulating a robot with co-simulation support. With Gazebo, simulated robots can be ran with the same software as their real counterparts. A model of a robotic arm was realized using the serial-link structure approach, upholding the component-based goal.

7.2 FUTURE WORK

Future work first includes examining possible solutions to simulation performance to handle real-time requirements. It's possible the dynamic model may somehow be optimized to improve simulation spend stability. Use of powerful computing hardware may be required for real-time simulations. Second is exploring the dynamic model's integrity with the real arm. More experiments may give more insight into how the dynamic model compares, Understanding the dynamic compensation and being able to read the correct torque applied to the joints allow for correct use of the model. future research could focus on optimizing the robotic arm's movements to reduce the time taken to pick and place objects. Additionally, developing a physical prototype to validate the simulation results and testing the system in a real-world environment could be another area of focus. Lastly, we could explore integrating machine learning techniques to enable the system to adapt to new objects and environments.

REFERENCES

- [1] Iqbal, Jamshed, R. Ul Islam, and Hamza Khan, "Modelling and analysis of a 3 DOF robotic arm manipulator", *Canadian Journal on Electrical and Electronics Engineering* 3, no. 6, pp. 300-306, 2012.
- [2] Kang, Hee-Jun, and Young-Shick Ro, "Robot manipulator modelling in ROS-SimMechanics with PD control and online gravity compensation," In *International Forum on Strategic Technology 2010*, pp. 446-449. IEEE, 2010.
- [3] Kim, Suseong, Seungwon Choi, and H. Jin Kim, "Aerial manipulation using a quadrotor with a two DOF robotic arm," In *2013 IEEE/RSJ International Conference Intelligent Robots and Systems*, pp. 4990-4995. IEEE, 2013.
- [4] Meng, Wei, Quan Liu, Zude Zhou, Qingsong Ai, Bo Sheng, and Shengquan Shane Xie, "Recent development of mechanisms and control strategies for robot-assisted lower limb rehabilitation," *Mechatronics* 31, pp. 132-145, 2015.
- [5] Mohammed Abu Qassem, I.M. Abuhadrous, H. Elaydi, "Modelling and Simulation of 5 DOF educational robot arm", In *Conference: Advanced Computer Control (ICACC), 2010 2nd International Conference on*, Volume. 5 April 2010.
- [6] Nguyen H.C, "Research on the application of Neural Networks in identification and control of the robotic arms-A nonlinear dynamic object," *Journal of Science and Technology – the University of Da Nang*, vol. 5, pp. 14-18, 2016.
- [7] Topaz, Metin, and Serdar Kucuk, "Dynamics simulation toolbox for industrial robot manipulators," *Computer Applications in Engineering Education* 18, no. 2, pp. 319-330, 2010.
- [8] Wang, Zhi-Xing, Wen-Xin Fan, Bao-Cheng Zhang, and Yuan-Yuan Shi, "Kinematical analysis and simulation of an industrial robot based on ROS," *Jidian Gongcheng/Mechanical & Electrical Engineering Magazine* 29, no. 1, pp. 33-37, 2012.
- [9] Žlajpah, Leon, "Simulation in robotics," *Mathematics and Computers in Simulation* 79, no. 4, pp. 879-897, 2008.

APPENDIX

A. IEEE STANDARDS

- [1] IEEE 1872-2015: Recommended Practice for Safety and Performance of Industrial Robotics Systems. This standard provides guidelines for ensuring the safety and performance of industrial robotic systems, including those that are modelled and simulated using ROS.
- [2] IEEE 1872.2-2015: Recommended Practice for Robot Programming. This standard provides guidelines for programming robots, including those that are simulated using ROS.
- [3] IEEE 1872.3-2015: Recommended Practice for Robot System Integration. This standard provides guidelines for integrating robotic systems, including those that use ROS, with other industrial systems.
- [4] IEEE 1872.4-2017: Recommended Practice for Robot Teach Pendant User Interfaces. This standard provides guidelines for designing user interfaces for teach pendants, which are used to program and control industrial robots, including those that are simulated using ROS.
- [5] IEEE 1872.5-2019: Recommended Practice for Robot Joint Position Sensing. This standard provides guidelines for the design and use of joint position sensors in industrial robots, including those that are modeled and simulated using ROS.
- [6] IEEE 1872.6-2019: Recommended Practice for Robot Control Systems. This standard provides guidelines for designing and implementing control systems for industrial robots, including those that are simulated using ROS.
- [7] IEEE 1240-2019: Standard for Model and Data Exchange in Distributed Simulation of Aerospace Systems. This standard provides guidelines for the exchange of models and data in distributed simulation environments, which can be used for simulating and testing industrial robotic systems.
- [8] IEEE 1872.7-2020: Recommended Practice for Robot Vision Systems. This standard provides guidelines for designing and implementing vision systems for industrial robots, including those that are modelled and simulated using ROS.
- [9] IEEE 1872.9-2021: Recommended Practice for Robot Motion Planning. This standard provides guidelines for the design and implementation of motion planning algorithms for industrial robots, including those that are simulated using ROS.

B. CONSTRAINTS

- **Physical Constraints:** The physical constraints of the robotic arm, such as the arm's joint limits, the maximum range of motion of the arm, the weight capacity of the arm, and the arm's structural limitations, must be taken into account during the modelling and simulation process.
- **Sensor Constraints:** The sensor constraints refer to the sensors that are used to control the robotic arm. These sensors include cameras, force sensors, and position sensors. The sensor constraints include their accuracy, their range, and their response time.
- **Environmental Constraints:** The environmental constraints refer to the environment in which the robotic arm operates. This includes the presence of obstacles, the lighting conditions, and the temperature.
- **Computational Constraints:** The computational constraints refer to the limitations of the hardware and software used for modelling and simulation. This includes the processing power of the computer, the memory available, and the speed of the simulation.
- **Time Constraints:** The time constraints refer to the amount of time available for modelling and simulation. These constraints can include deadlines for project completion, time required for testing and debugging, and the time required to implement any changes or modifications to the robotic arm.
- **Cost Constraints:** The cost constraints refer to the budget available for modelling and simulation. This includes the cost of hardware and software, the cost of any necessary upgrades or modifications, and the cost of any required training for personnel involved in the project.
- **Safety Constraints:** The safety constraints refer to the safety requirements that must be met during the modelling and simulation process. This includes ensuring that the robotic arm operates within safe limits, and that any potential hazards are identified and addressed.
- **Actuator Constraints:** The actuator constraints refer to the motors that control the movement of the robotic arm. These motors have constraints such as their maximum torque, speed, and power consumption. These constraints need to be considered when modelling the robotic arm and ensuring that the control signals sent to the motors are within the allowable range.

- **Communication Constraints:** The communication constraints refer to the limitations of the communication between the robotic arm and other devices or systems. This includes the latency, bandwidth, and reliability of the communication channels. These constraints can impact the accuracy and responsiveness of the robotic arm's control system.
- **Software Constraints:** The software constraints refer to the limitations of the software used to develop the robotic arm's control system. This includes the programming languages used, the libraries and frameworks available, and the development tools used. These constraints can impact the flexibility and scalability of the control system.
- **Regulatory Constraints:** The regulatory constraints refer to the regulations and standards that must be met when developing and operating the robotic arm. These may include safety regulations, environmental regulations, and industry-specific standards. Failure to meet these constraints can result in legal and financial consequences.
- **Maintenance Constraints:** The maintenance constraints refer to the requirements for maintaining the robotic arm's performance and reliability over time. This includes regular inspections, cleaning, and calibration of sensors, actuators, and other components. These constraints must be considered during the design and development of the robotic arm to ensure that maintenance requirements are manageable and cost-effective.
- **Power Constraints:** The power constraints refer to the limitations of the power supply used to operate the robotic arm. This includes the voltage, current, and power capacity of the power supply. These constraints need to be considered when selecting components and designing the control system to ensure that they operate within the allowable power limits.
- **User Constraints:** The user constraints refer to the limitations of the human operators who will interact with the robotic arm. This includes their training, skills, and experience operating robotic arms. These constraints need to be considered when designing the user interface and control system to ensure that they are easy to use and do not require specialized knowledge or training.