ROBOTIC ARM USING ROS2

By

OMKAR BHOGTE

SHUBHAM SUGIRE

DERRIAN PEREIRA

VIVEK JADHAV

MSC PART II ELECTRONICS

SCHOOL OF PHYSICAL AND APPLIED SCIENCE

GOA UNIVERSITY

2021 - 2022

ROBOTIC ARM USING ROS2

CERTIFICATE

This is to certify that the project entitled

"Robotic Arm using ROS2"

Is a record work done by

MR. OMKAR BHOGTE

MR. SHUBHAM SUGIRE

MR.DERRIAN PEREIRA

MR. VIVEK JHADAV

M.Sc. Part II Electronics For the year 2021 – 2022

The candidates themselves have worked on the project during the period of study under by guidance and to the best of my knowledge it has not previously formed the basis of award of any previous degree or diploma at Goa University or elsewhere.

H.O.D.	Examiner	Project Guide

Declaration

I student of "GOA UNIVERSITY" batch in M.Sc. ELECTRONICS here by solemnly declare that this project report under the title "ROBOTIC ARM USING ROS2" is a record of work done by us. And this report has not been submitted anywhere else for award of any diploma or degree to the best of my knowledge.

1. Derrian R. Pereira _____
2. Shubham D. Sugire _____
3. Vivek R. Jadhav _____
4. Omkar S. Bhogte _____

ACKNOWLEDGEMENT

First and foremost, we want to thank GOD for all of his graces and blessings, as well as the ongoing assistance he gives us in achieving our project goals through various mediums such as our mentors and friends.

Prof/ Vice-Dean Dr. R.S. Gad, Dr. J. S. Parab (HOD), Dr. Narayan Vetrekar, Dr. Aniketh Gaonkar and Dr. Marlon Sequeira from our teaching faculty deserve special thanks for their encouragement, help, and support with this project. Allowing us to use the department's facilities on a daily basis. The Registrar of Goa University for providing us with the necessary funds.

We would like to express our gratitude to Mr. Vishant Malik for taking time out of their busy schedules to lend a hand in setting up the lab and providing necessary components.

We would also like to express our gratitude to Mr. William D'Souza for his assistance with all of the paperwork and other office-related tasks. We would specially like to thank the non – teaching staff members Pushpa Andrade (Multi – Tasking Staff) and Ashwini Velip (Multi – Tasking Staff) for their invaluable assistance and encouragement.

Thank you to all of our colleagues, friends, and Goa University students who have assisted and collaborated with me. We appreciate all of our family members moral support and encouragement throughout the project.

Last but not least, many thanks to the internet contributors who provided references which we used in our project.

ABSTRACT

In this research work, we are presenting the design and development of a 5 DOF robotic arm using the Robot Operating System (ROS 2). Our project intends to create a low-cost 5 DOP robotic arm that can perform similarly to other developed arms on the market. Because of its easier hardware abstraction, ROS 2 was chosen to structure task architecture. The simulation system may analyse and compare the responses of the robotic arm to various algorithms and target poses. Forward kinematics explained how robots move in response to input angles. Calculating the angles of joints (i.e. the angles of the servo motors on a robotic arm) that will cause the end effector of a robotic arm (e.g. robotics gripper, hand, vacuum suction cup, etc.) to reach some desired point (x, y, z) in 3D space is what inverse kinematics is all about. 5 servo motors connect the parts and allow the robotic arm to move.

All the necessary calculations and parameters that we set in the URDF file are carried out by the ROS 2 packages. The parameters are set in Rviz and the simulation environment is developed in Gazebo and run on Robot Operating System (ROS 2) because of the complexity and high learning curve in using the ROS software system. The robot arm uses servo motors and encoders to drive the robot arm joints and was controlled using Micro-controller.

Table of Contents

Chapter 1: Introduction 1

1.1 Types of Robots ……3

1.2 Parts of Robotic Arm…….4

1.3 Working of Robotic Arm……..6

Project Goal……10

Chapter 2: Literature Survey ……..11

Chapter 3: Experimental Methodology……..25

3.1 System Block.....26

3.2 Software……28

3.2.1 What is ROS?……..……..29

3.2.2 ROS unique feature..…….32

3.2.3 ROS2…….38

3.2.4 Some key features differences between ROS and ROS2……..…….41

3.2.5 Visual Studio Code…….43

3.3 What is a Robot Simulator…….....47

3.4 What is Rviz?.....49

3.5 What is Gazebo?.....50

3.5.1 System requirements……..…..52

3.5.2 Development History……..…….52

.

3.5.3 Benefits of Gazebo Simulation…….……..53

3.5.4 Drawbacks of Gazebo Simulation…….……..54

3.6 RViz vs Gazebo…….……54

3.7 ROS Versions.....55

3.8 Hardware…..57

3.8.1 Arduino Mega 2560.....58

3.8.1.1 What is an Arduino Mega 2560?.....58

3.8.1.2 Communication……60

3.1.1.3 Programming……61

3.9 Encoders…….62

3.9.1 Rotary

Encoders…….....63

3.9.2 Different Mechanisms in encoder……68

3.9.3 Ways of encoding shaft position.....63

3.9.4 Keys KY-040 Rotary Encoder…….....75

3.10 MOTORS……..78

3.10.1 SG90 MICRO 9G……...78

3.10.2 MG-996R Servo Motors…….80

3.11 Wire Configurations…….....82

3.11.1 Applications…….....82

3.11.2 Types of Wires…….....83

3.12 Kinematics……85

3.12.1 Forward Kinematics……...86

3.12.2 Inverse Kinematics…….....87

3.12.3 The relationship between forward and inverse kinematics…….....90

3.12.4 IK Types……92

Chapter 4: Experimental Execution……..94

4.1 Robotic Operating System……95

4.1.1 Installing ROS2 via Debian Packages…….....96

4.2 Install ROS2 Packages…….....97

4.3 ROS2 Documentation: Foxy……....98

4.3.1 Core ROS2 Tutorial……....98

4.3.2 Create ROS2 Workspace……....100

4.4 Creating Your First ROS2 Package……....100

4.5 Unified Robotics Description Format(URDF)……....102

4.6 RViz Simulation For Joints tests…….106

4.7 Setup.py……...107

4.8 Gazebo and Physical properties…….....109

4.8.1 Pipeline of Launching Gazebo……...110

4.9 Calculating Inertia for Robot links……..111

4.10 Introduction to ROS2 control stack……..112

4.10.1 Types of Controllers…….....114

4.10.2 How to Install a Controller? …….115

4.10.3 Installing Joint trajectory Controller……116

Chapter 5: Result and Conclusion……..117

Chapter 6: Future Work……..120

Chapter 7: Bibliography……..122

Chapter 8: Appendix……128

8.1 Arduino……..129

8.1.1 Arduino Mega Specifications……...129

8.1.2 Arduino Mega Pin Configuration…….131

8.2 Rotary Encoder (KY-040)…….136

8.2.1 KY-040 Pin Outs……..136

8.2.2 Keyes Rotary Encoder Schematic……137

8.2.3 Module Connection to the Arduino……..137

8.3 Motor…….138

8.3.1 SG90 MICRO 9G…….....138

8.3.2 Mg-996R Servo Motor……140

CHAPTER 1:

INTRODUCTION

Robotics is an engineering discipline that deals with the conception, design, construction, and operation of robots. The goal of robotics is to develop intelligent devices that can help people in a number of ways. By definition, robotics is an interdisciplinary endeavour. Robots have been used in a wide range of applications, from the ocean floor to Mars and beyond. Robots are extensively utilised for a range of duties that are too risky for humans to accomplish. A robotic arm is a programmed robotic manipulator that performs functions comparable to a human arm. Because it resembles a human hand, the robotic arm is frequently referred to as anthropomorphic. The interaction of robots with their surroundings is an essential goal in robot development.

It's a kinematic-chain that's basically just a collection of links and joints. A tool, such as a gripper, a welder, or a drill, is usually at the end of this chain. The end-effector is the term for this. Your own human arm is an excellent example of connections and joints. Bones are the links in your arm, and joints are the connections between them: your elbow, shoulder, wrist, and so on.

Industrial robotic arms are one of the most prevalent types of robots in use today, with applications ranging from manufacturing to automotive to agricultural. Robots in industrial settings have typically relied on highly structured surroundings with properly placed materials. Due to technological advancements, industrial robot applications have recently been deployed to less organised situations.

1.1 Types of Robots

Mechanical bots come in a variety of shapes and sizes to perform the work for which they were created. Robots are evolving to perform tasks that humans simply cannot, from the 0.2 millimeter-long "RoboBee" to the 200-meter-long robotic shipping vessel "Vindskip." There are five sorts of robots in general:

I. Pre-Programmed Robots

Pre-programmed robots do simple, repetitive activities in a controlled setting. A mechanical arm in an automotive assembly line is an example of a pre-programmed robot. The arm has one job - weld a door shut, insert a part into the engine, etc. — and it's job is to do it longer, quicker, and more efficiently than a human could.

II. Humanoid Robots

Humanoid robots have the appearance and/or behaviour of humans. These robots typically do human-like tasks (such as running, jumping, and carrying goods) and are occasionally made to resemble humans, with human-like features and attitudes. Hanson Robotics' Sophia (seen above) and Boston Dynamics' Atlas are two of the most well-known humanoid robots.

III. Autonomous Robots

Human operators are not required for autonomous robots to function. These robots are typically built to perform jobs in open spaces without the need for human supervision. The Roomba vacuum cleaner is an example of an autonomous robot, as it uses sensors to move freely throughout a home.

IV. Tele-Operated Robots

Humans control mechanical robots known as teleoperated robots. These robots are typically used in extreme geographic situations, weather, and other circumstances. The Ohuman-controlled submarines used to repair underwater pipe leaks during the BP oil spill or drones used to locate landmines on a battlefield are examples of teleoperated robots.

V. Augmenting Robots

Augmenting robots can either improve current human capabilities or replace those that have been lost. Exoskeletons or robotic prosthetic limbs used to lift heavy weights are examples of augmenting robots.

1.2 Parts of Robotic Arm

What are the five components of a robotic arm?

There are five primary components to this design:

- Controller
- Sensors
- Robotic Arm
- End Effector
- Drive

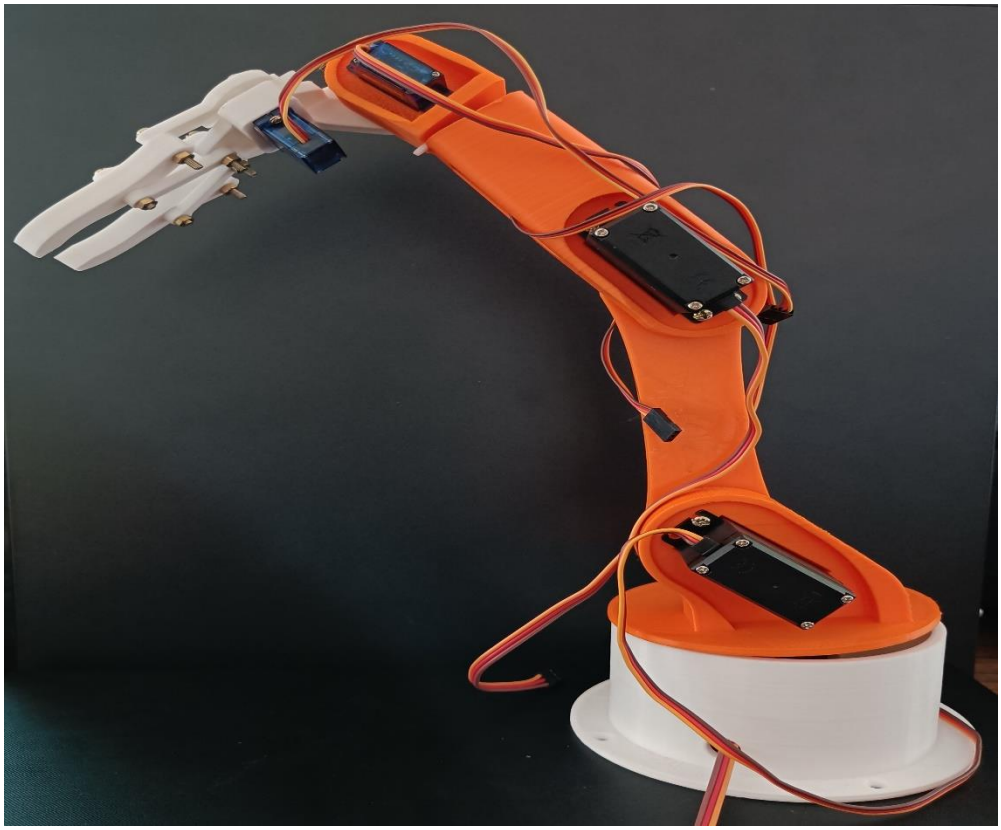


Figure1: Robotic Arm

Axis No.	Joint Name	Motion	Motor Type
1.	BASE	Rotates the Body	SERVO MOTOR (MG996R)
2.	SHOULDER	Raises and lowers the upper arm	SERVO MOTOR (MG996R)
3.	ELBOW	Raises and lowers the forearm	SERVO MOTOR (SG90)
4.	WRIST (WRIST PITCH)	Raises and lowers the end effector	SERVO MOTOR (SG90)
5.	WRIST (WRIST ROLL)	Rotates the end effector	SERVO MOTOR (SG90)

Table No.1: Parts of Robotic Arm

1.3 Working of Robotic Arm.

There are various manipulators; in robotics theory, a manipulator is a machine or robot that can move any object or physical object in 3D space without touching it. So you can use that machine to manipulate its location and mass using a manipulator. The most common manipulators are robotic arms, which

are hand-held and easy to handle. Serial/chain and parallel manipulators are the two types of manipulators.

i. Degrees of freedom

In mechanics, degrees of freedom are particular, defined modes in which a mechanical device or system can move. The total number of independent displacements or features of motion equals the number of degrees of freedom. A machine with more than three degrees of freedom can operate in two or three dimensions. The phrase is commonly used to describe robot motion capabilities.

Consider a human-like robot arm. Pitch (up and down) and yaw (left and right) shoulder motions are both possible (left and right). Elbow motion can only be caused by pitch. Wrist motion includes pitch and yaw. It's also possible to rotate the wrist and shoulder (roll). These robot arms have between five and seven degrees of freedom. A complicated robot with two arms has twice the amount of degrees of freedom.

ii. 5 DOF Modelling

We constructed a 5 DOF robotic arm for this project that can do the following motions:

I. Shoulder: Raising and lowering the upper arm
II. Body: Rotating the arm's body
III. Elbow: Forearm raising and lowering
IV. Wrist Pitch: Changing the end effector's position
V. Rotating the end effector with the wrist roll

The 5 DOF robotic arm has five revolute joints and is vertically articulated. It is an instructional robotic system that is both dependable and safe. Students can obtain theoretical and practical experience in robotics, automation, and control systems with this adaptable system.

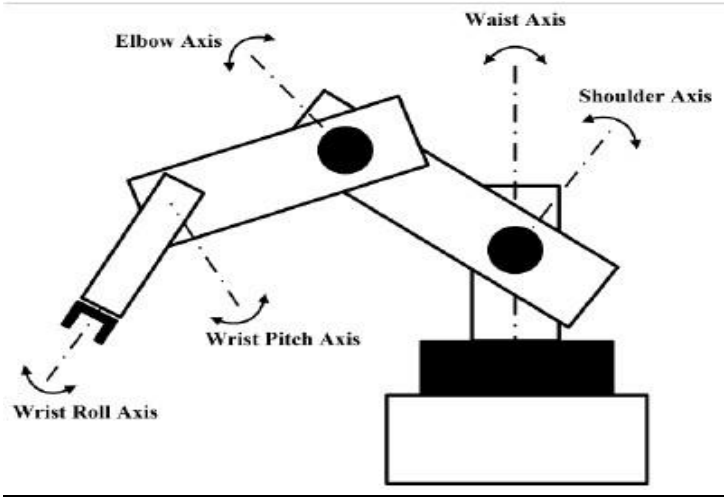


Figure 2: 5 DOF Robotic Arm

By attaching a gripper or other attachment to the end effector link, the arm can be turned to a 6DOF arm. The Gripper can be used to imitate holding motions, such as pick and place.

Project goal

The main goal of this project is to develop a 5 DOF robotic arm using the forward and inverse kinematic and can perform various operations like pick and place the objects on ROS2 platform. The purpose of using robotic arm is to reduce errors and human efforts. As the robotic field is having an application in various department of engineering such as production, inspection, material handling etc.

Project aims at building a cost effective 5 DOF Robotic arm which can replicate the working of similar developed arms available in the market. Also to integrate the arm with ROS to control and to give better movements using Kinematics so that the arm is moved precisely to a particular location and provide path planning for the arm.

CHAPTER 2: LITERATURE SURVEY

This toolbox generates the trajectory for the pick place operation. The Rigid body tree kinematics Simulink block is used to determine joint configurations for a desired end-effector pose. The model is subjected to dynamic simulations to assess properties such as torque change at the joints and manipulator trajectory. The manipulator should respond similarly to the physical model, according to the simulation results. The articulated robotic arm requires a maximum joint torque of roughly 3 Nm and a minimum joint torque of about 0.04 Nm. The Unified Robot Description Format (URDF) file simplifies and streamlines robotic modelling. The needed methods for robot modelling and control are supported by the MATLAB Robotic System Toolbox. Euler-Lagrange equations are employed in this procedure. The structural modelling is accomplished by defining the connection parameters and dynamics of each element, as well as creating the Simulink dynamic equations required to solve the problem. Simscape Multibody can validate kinematic analysis, dynamic analysis, and control design without the necessity for co-simulation with other software packages. This work proposes a new method for analysing the inverse kinematics of an articulated robotic arm with an open chain mechanism.[1]

The Leap Motion controller and the 6-DOF Jaco robotic arm were created to provide a human-machine communication interface. An algorithm was created to allow for the best possible mapping between the user's hand movement and the Leap Motion controller's tracking. The approach should allow for more natural human-computer interaction as well as smooth robotic arm manipulation. The uses of this human-robot interaction were examined in relation to Ambient Assisted Living, and specific use case scenarios were presented.[2]

A robotic arm is a mechanical arm that can execute a task. Artificial arms are becoming increasingly necessary in today's society for a variety of inhuman scenarios where human connection is difficult or impossible. Humans pick up objects without considering the procedures necessary. As a result, the robotic arm is operated manually via wired and wireless connections. In this research, numerous ways for controlling a robotic arm were investigated, as well as its downsides. Robotic Arm, Depth of Field, Human Arm.[3]

We present the design and development of a 6 Degrees of Freedom (DOF) robotic arm control for agricultural applications utilising the Robot Operating System in this research paper (ROS). We would require the robotic arm to operate at a specific height from the ground level to harvest a crop, especially when gathering fruits from tall trees. Space and weight are not as important elements to consider in applications that require the robotic arm to work on the ground level, but they become essential difficulties in applications that require the robotic arm to work at an altitude. In terms of design and implementation, the six DOF robotic arm suggested in this research takes these factors into account. The robotic arm is implemented in the study utilising ROS's built-in inverse kinematic support capabilities. The experimental results suggest that a 6 DOF robotic arm with a robust construction can be used in agricultural applications, particularly for fruit harvesting.

People will be displaced from harmful jobs in perilous settings by autonomous robots. This will also save energy and increase the work's accuracy. Automation would close the gap between the robot and the client. Because they are sufficiently adaptive to complete jobs with minimal effort and greater precision, automated robotics is frequently employed in industrial processes in industries.

In this study, we use the Robot Operating System to explain the design and implementation of a 6DoF robotic arm for a tree climbing robot (ROS).

The user can choose between two modes for providing input. One method is to place a 3D point in the robotic arm's workspace zone, prompting the user to enter X, Y, and Z coordinates such that the end effector moves to the point. Another option is to adjust the end effector position in the simulation such that the robotic arm follows suit. The second technique of delivering input cannot be employed when the arm is further developed into a fully automated control system since the system will become completely automated when the robot works without any human interference. As a result, the first way of supplying coordinates is favoured over the second. The first approach can be advanced by applying image processing algorithms to discover the coordinates of the desired point and feeding the coordinates to the robotic arm. The user can enter the relevant parameters and click the execute button to manoeuvre the robotic arm using the QT-developed GUI. The Robot Visualizer (Rviz) module in ROS, its technicalities, and implementation for 3D interfaces are all covered in depth in Paper [17]. They compared traditional user interfaces to interactive makers that added minimal overhead to an application.

The input data must be in the form of a three-dimensional coordinate system with x, y, and z components. This information would be fed into the Moveit simulation programme. KDL and IK (Inverse Kinematics) solvers are available in Moveit. These solvers would assist in solving forward and inverse kinematics by determining the arm's joint angles. The joint states node, which will publish the joint angles as subjects, has these joint angles. A node is a computation-performing process. Both the Moveit and the microcontroller that controls the robotic arm must subscribe to the published joint angles. The Moveit must

subscribe to the joint states node, which aids in the simulation's updating of joint position. Similarly, the microcontroller will subscribe to these topics and send these joint angles to each servo using a pulse width modulation variation that corresponds to the angles of each joint motor. The microcontroller in our situation was an Arduino Mega. Moveit updates the joint angles in the simulation environment at the same time as the microcontroller updates the joint angles in servo motors. The simultaneous execution of the two operations produces the impression of a live simulation of the actual arm in the computer. The ROS package is explained in detail in [18]. According to Moveit, this package will help to enable safer, more capable robots that can work effectively in human situations.

Using a SolidWorks add-in (SolidWorks to URDF exporter) that makes it easy to convert SolidWorks part and assembly files to URDF format. This exporter will generate a ROS directory containing all of the necessary files, such as models, robot textures, and URDF files. The SolidWorks assembly hierarchy would be used to create all of the linkages, joint types, joint transformations, and rotation axes. The next crucial challenge was to simulate a robotic arm and use kinematic equations to control it. Both forward and inverse kinematics have been implemented. Forward kinematics yielded positive results, however inverse kinematics was unsuccessful due to a lack of inverse kinematic solutions in the workspace. Increase the degrees of freedom or improve the inverse kinematics process with additional modifications and iterations in the equations to increase inverse kinematics solutions over a workspace. Because modifying the algorithm and inverse kinematics for a certain workspace is a time-consuming and difficult procedure, the other alternative was to increase the degrees of freedom of the robotic arm.[4]

This study proposes an analytical technique to solve the forward kinematics problem of a serial robot manipulator with six degrees of freedom and a specific combination of joints and links to define the gripper's location using a set of joint angles. In addition, in order to determine the robot's joint angles to pose the gripper in a given coordinate, a direct geometrical solution for the robot's inverse kinematics problem will be defined. Furthermore, by comparing the results in a built simulation software that uses the Unified System for Automation and Robot Simulation, the accuracy of the two solutions will be demonstrated (USAR Sim).5]

Artificial arms are becoming increasingly necessary in today's society for a variety of inhuman scenarios where human connection is difficult or impossible. They could include anything from taking readings from an active volcano to defusing a weapon. We propose to develop a robotic arm that is controlled by normal human arm movements, with data collected using accelerometers. The output of the accelerometer is smoothed using a good averaging method for proper control and to limit the amount of noise coming in from the sensors. This arm's development is based on the ATmega32 and ATmega640 platforms, as well as a personal computer for signal processing, all of which will be connected via serial communication. Finally, this arm prototype should be able to solve problems like placing or picking dangerous or non-hazardous objects that are far away from the user. [6]

This study describes the design and implementation of a low-cost, user-friendly interface for controlling a slave tele-operated anthropomorphic robotic arm with six degrees of freedom. Six single-axis revolute joints are used to articulate the robotic arm: one for each shoulder abduction adduction (abd-add), shoulder flexion-extension (flx-ext), elbow flexion-extension (flx-ext), wrist flexion-ext,

wrist radial-ulnar (rad-uln), and gripper open-close. The master tele-operator uses the Man Machine Interface (MMI) to control the robotic arm in real time. Simple motion capture devices in the MMI convert motion into analogue voltages, which are translated into actuation signals in the robotic arm.[7]

Robotics frequently uses planar two and three-link manipulators as testbeds for various algorithms or ideas. The instance of a three-link planar manipulator is considered in this work. A feed-forward neural network is used to find a solution to the inverse kinematics problem, which is required for creating desired trajectories in Cartesian space (2D) for this sort of robot.[8]

Modeling and animating the caterpillars of crawler UGVs is a difficult challenge that has yet to be solved in ROS/Gazebo simulators. In this research, we offer an approximation of a crawler UGV's track-terrain interaction, model and simulate the Russian crawler robot "Engineer" using ROS/Gazebo, and visualise its movements using ROS/RViz. Finally, we put the suggested model to the test in an unpredictable Gazebo environment with diverse robot groups. The owner/author owns the copyright (s). ACM has been granted publication rights.[9]

This study presents the design analysis of a Remote Controlled "Pick and Place" Robotic vehicle. This research reveals that humans will always want to follow safety procedures at work and in their surroundings in order to perform specific jobs, such as sending a robotic vehicle into a hazardous environment to collect samples for chemical analysis. A typical Robotic Vehicle is capable of navigating obstacles and traversing diverse terrains. The design in this work incorporates a five-degree-of-freedom robotic arm with its base lying directly on top of the vehicle, as well as a body with four driving wheels attached to the

ends. To propel the vehicle, the wheels are selectively energised. The hardware, software, and implementation of both designs are all part of the design technique. To test design parameters, a prototype of the Remote Controlled "Pick and Place" Robotic vehicle was created. The final results were satisfactory. Robotics are highly suggested for industries, particularly for reasons of safety and productivity. [10]

In many circumstances, an analytical model can better express the behaviour of physical systems. The kinematics of a robot are crucial to its modelling and analysis. An analytical solution to the inverse kinematics is perhaps the most essential topic in robot modelling for robotic manipulators with high degrees of freedom (DOF) and numerous degrees in one or more joints. This study examines the workspace of a 6 DOF robotic arm and creates kinematic models for it. In an unstructured environment, the suggested paradigm allows the manipulator to be controlled to any attainable position and orientation. Denavit Hartenberg (DH) parametric system of robot arm position placement underpins the forward kinematic model. The achieved inverse kinematics model offers the required appropriate joint angles given the desired position and orientation of the robot end-effector. The forward kinematic model was tested in MATLAB using Robotics Toolbox, and the inverse kinematic model was implemented on a real robotic arm. Experiments show that the created model allows the robotic arm's end-effector to point to the specified coordinates with a precision of 0.5cm. The approach given in this paper can be used to tackle the kinematics problem of other robot manipulators of similar types.[11]

In this paper, we discuss how to use ROS and RViz with a specific plugin called MoveIt to build forward kinematics of a six-axis robotic arm. We generated an arm model that is compatible with RViz for executing kinematics operations

(forward or reverse) utilising the above specified set up. Later, for motion planning purposes, we produced a MoveIt configuration package of the same model. Following the creation of a configuration package, we ran forward kinematics on the arm model and compared the results to the manual calculations in RViz.

RVIZ is a ROS graphical interface that lets you see a lot of data with the help of plugins for a variety of topics. Move It is a motion planning framework that must work in conjunction with ROS and RViz. Because they can deliver cost-effective and high-speed services, robotic manipulators have been widely used in industrial production lines.

This study's methodology may be broken down into four steps. The first is the modelling step, in which we used URDF (Unified Robot Description Format) standards for a ROS package to design a simple kinematic linkage of the arm. A MoveIt configuration package was later created. After that, we used the RViz visualiser to load the configuration package and ran forward kinematics on the robotic arm. The flowchart below displays the identical steps that were involved in conducting this research. [12]

This study describes a multistage technique for modelling a vision-based 6-DOF (Degrees of Freedom) robotic arm manipulator. The main purpose of this project is to integrate a vision system with a 6-DOF robotic arm in order to enhance the combined camera-robot system's capabilities. Because the robotic arm lacks an integrated vision system, the necessary results were achieved by mounting a camera above the robot's workspace. The key problems with this system were designing an appropriate sequence of operations, implementing

acceptable communication between the camera and the robot, and integrating with the system components in ROS.

The URDF file is imported into the ROS Move-it helper programme, which includes joint state controllers and kinematics planners. The Move-it configuration file, together with the created URDF, is viewed in Gazebo, a simulation environment, after the Move-it assistant setup is complete. Finally, the 2D position of the object is determined via image processing and sent to the Moveit Commander, which moves the robot's end effector to the location where the object is positioned for pick and place application.[13]

The design and development of a robust 5-DoF robotic arm is presented in this study. Linear actuators and DC brushed motors are used to articulate the robotic arm. Nvidia Jetson Nano is used for onboard processing, paired with off-the-shelf motor drivers and microcontrollers for feedback and control. The sub-system is designed with the safety of electronic components and battery management in mind. To support networked computing, the software stack is based on Robot Operating System (ROS2). [14]

In order to control the robot, the robot controller is crucial. The controller's primary goal is to reduce or eliminate the influence of unknown circumstances on the control robot. Furthermore, there are many various sorts of controllers, each with its own set of functions. This article examines some controllers from the basic and advanced controller categories to see how they differ. This article uses pre-determined tests to benchmark the selected controller. The most typical pick and place task is the test.[15]

ROS-based keyboard control and simulation of a 6-DOF robotic arm In this study, we propose the design and simulation of a 6 DOF robotic arm for usage in disaster-stricken locations for search and rescue missions. Also mentioned is an algorithm for keyboard-based interaction with the 6 DOF robotic arm that makes it more user-friendly. RVIZ was used to build and visualise the 6-DOF articulated robotic arm, and Moveit was utilised as a control interface with the Robot Operating System (ROS). The joint control mechanism prevents us from using the joint when we manipulate the arm with an end effector. The approach described in this study is both affordable and simple to use. The first three DOF of the planned 6 DOF robotic arm are used to position the arm, while the remaining three DOF are employed to manipulate the gripper. In this study, we propose the design and simulation of a 6 DOF robotic arm for usage in disaster-stricken locations for search and rescue missions. Also mentioned is an algorithm for keyboard-based interaction with the 6 DOF robotic arm that makes it more user-friendly.

RVIZ was used to build and visualise the 6-DOF articulated robotic arm, and Moveit was utilised as a control interface with the Robot Operating System (ROS). The joint control mechanism prevents us from using the joint when we manipulate the arm with an end effector. The approach described in this study is both affordable and simple to use. The first three DOF of the planned 6 DOF robotic arm are used to position the arm, while the remaining three DOF are employed to manipulate the gripper. In this study, we propose the design and simulation of a 6-DOF robotic arm for usage in disaster-stricken areas for search and rescue missions. Also mentioned is an algorithm for keyboard-based interaction with the 6 DOF robotic arm that makes it more user-friendly. RVIZ was used to build and visualise the 6-DOF articulated robotic arm, and Moveit was utilised as a control interface with the Robot Operating System (ROS). The

joint control mechanism prevents us from using the joint when we manipulate the arm with an end effector. The approach described in this study is both affordable and simple to use.

The first three DOF of the planned 6 DOF robotic arm are used to position the arm, while the remaining three DOF are employed to manipulate the gripper. In this study, we propose the design and simulation of a 6 DOF robotic arm for usage in disaster-stricken locations for search and rescue missions. Also mentioned is an algorithm for keyboard-based interaction with the 6 DOF robotic arm that makes it more user-friendly. RVIZ was used to build and visualise the 6-DOF articulated robotic arm, and Moveit was utilised as a control interface with the Robot Operating System (ROS). The joint control mechanism prevents us from using the joint when we manipulate the arm with an end effector. The approach described in this study is both affordable and simple to use. The first three DOF of the planned 6 DOF robotic arm are used to position the arm, while the remaining three DOF are employed to manipulate the gripper. [16]

Soft robotic systems open up a slew of new possibilities for tackling difficult issues. Soft robotic grippers, for example, can reduce the difficulty of tasks such as grabbing irregular and delicate objects. Soft robotics adoption has been delayed in academia and industry, in part due to the large quantity of hardware and software that must be designed from scratch for each use of soft system components. The design, manufacture, and validation of an open-source framework to lower the barrier to entry for integrating soft robotic subsystems are detailed in this research. We use this framework, which is based on ROS (Robot Operating System), to demonstrate a modular, soft-hard hybrid system capable of picking and placing items. We anticipate that by decreasing the

entrance barrier, system designers and researchers would find it simple to incorporate soft components into their existing ROS-enabled robotic systems. [17]

As many publications and studies have shown, virtual reality (VR) has a bright future in the field of rehabilitation because of the benefits it brings to the training process. We offer a unique way for creating a virtual training environment for a 5 degrees of freedom (DOF) upper limb rehabilitation robot that has been built to assist patients who have survived a stroke but are still hemiplegic in completing rehabilitation exercises. The method uses ROS (Robot Operating System) and Gazebo (a multi-robot simulator) to create an engaging virtual scene of everyday life in a 3D world that allows patients to move their damaged arms with simultaneous visual feedback and participate with the virtual training task. [18]

The work given here shows how to use the Robot Operating System to construct an autonomous navigation system on wheeled mobile robots (ROS). We examined location difficulties, map building, and autonomous navigation packages in this work because ROS has numerous reusable software stacks. The hardware and software architectures for our own designed robot are shown. Furthermore, the four-wheel differential drive mechanism of our robot has been explored in terms of robotic kinematics of motion. During the navigation process, an inertial measurement unit is employed to help provide a more accurate odometry model and precisely localise the robot within the world. [19]

Periods of rapid change in popular approaches characterise the history of industrial automation. Such periods of change in automation techniques appear to be closely linked to world economics, either as a cause or, maybe, as an

effect. The use of industrial robots, which became distinguishable as a distinct device in the 1960s [1], as well as computer-aided design (CAD) and computer-aided manufacturing (CAM) systems, marks the most recent advancements in manufacturing process automation. These technologies are ushering in a new era in industrial automation.

The positioning of items in three-dimensional space is a persistent topic in robotics research. The manipulator's linkages, the parts and tools with which it interacts, and other objects in the manipulator's environment are all examples of these items. These objects are characterised by only two qualities at a basic but crucial level: location and orientation. Naturally, one of the first things that comes to mind is how we represent and manage these numbers mathematically. Kinematics is a branch of physics that studies motion without taking into account the forces that create it. Position, velocity, acceleration, and all higher order derivatives of the position variables (with respect to time or any other variable(s)) are all studied in kinematics. As a result, the study of manipulator kinematics encompasses both geometrical and time-based aspects of motion.[20]

CHAPTER 3: EXPERIMENTAL METHODOLOGY

3.1 System Block:

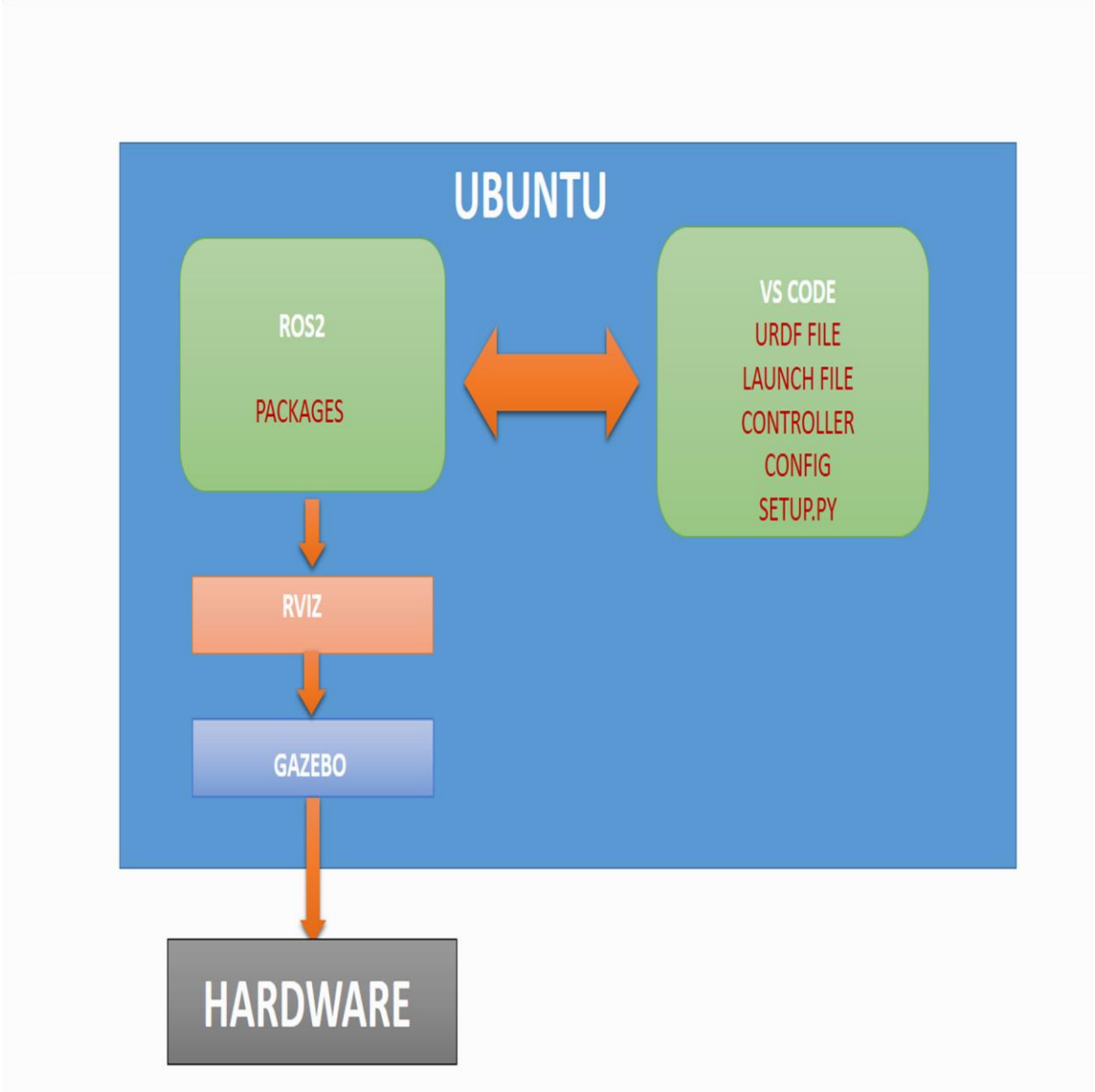


Figure 3: System Block Diagram

Block Diagram Desciption :

Ubuntu is linux operating system which is installed on our system, for our purpose we used the 20.04 foxy fitzroy version of Ubuntu. All our softwares and packages, that we require for our robotic arm rest on this system. ROS2 is a sub operating system which is installed on Ubuntu.

ROS2: The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications. Software tools like hardware drivers, networking modules, communication architecture and several robot algorithms are needed to build our robotic arm. It contains packages for robot control, navigation and simulation.

We created a workspace for our project, the concept of the workspace is that the workspace is a space that contains a basic structure for packages and stores in a workspace. We created a package called big_bazu which contains multiple directories or file such as urdf, launch, controller, setup and config all this have their specific function and is designed specifically for python module. All of the paths required in this packages are stored in the setup.py file. This all files are coded inside vscode software. The basic structure or a design of robotic arm where each joint and link will be connected is designed in a URDF file.

Rviz is graphical interface that visualize information and algorithm. The joints and the links that we created in the urdf file are tested and simulated in rviz simulator, we can also adjust the parameters in the rviz according how we require the robotic arm to move. ROS2 controllers were installed and different trajectories were performed and is simulated in gazebo.

3.2 Software:

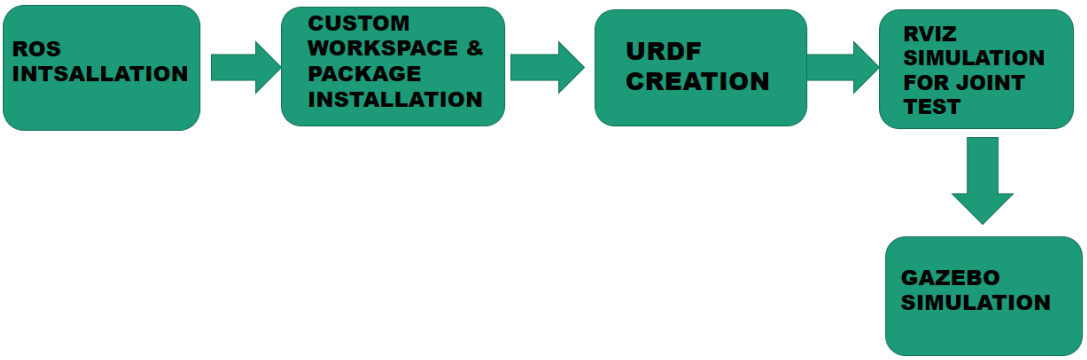


Figure 4: Software Block Diagram

3.2.1 What is ROS?

ROS is a free and open-source robot operating system. ROS (Robotic Operating System) is a robotics middleware (i.e. collection of software frameworks for robot software development). It's a set of tools, libraries, and conventions aimed at making building complicated and reliable robot behaviour easier on a range of robotic systems. Although ROS is not an operating system, it provides services such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management for a heterogeneous computer cluster.

It also comes with tools and libraries for getting, constructing, writing, and running programmers on many computers. ROS is similar to 'robot frameworks like Player, YARP, Orocos, CARMEN, Orca, MOOS, and Microsoft Robotics Studio in various ways. The ROS runtime "graph" is a loosely linked peer-to-peer network of processes (possibly spread across machines). ROS supports synchronous RPC-style communication over services, asynchronous data streaming over topics, and data storage on a Parameter Server.

Active clusters of ROS-based processes are represented in a graph architecture, having nodes receiving, posting, and multiplexing sensor, control, state, planning, actuator, and other signals. Despite the importance of reactivity and low latency in robot control, ROS is not a real-time operating system in and of itself (RTOS). However, it is possible to connect ROS with real-time code. Pr2 ether CAT is a technology used by the Willow Garage PR2 robot to transfer ROS messages in and out of a real-time operation. The Orocos Real-time

Toolkit also is seamlessly integrated with ROS. The creation of ROS 2.0 addressed the utter lack of support for real-time systems.

The ROS Ecosystem's software can be divided into three categories:

- Tools for developing and distributing ROS that are language and platform diagnostic based computer software:
- Implementations of ROS client libraries such as roscpp, rospy, and roslisp:
- packages containing code for applications that use one or more ROS clients libraries

Both the language-independent tools and the primary client libraries (C++, Python, and Lisp) are released under the BSD licence, making them open source software that can be used for commercial or academic purposes. The majority of the remaining packages are released under various open source licences. Hardware drivers, robot models, datatypes, planning, perception, simultaneous localization and mapping, simulation tools, and other techniques are all implemented in these additional packages.

Because of their reliance on enormous collections of open-source software dependencies, the main ROS client libraries (C++, Python, and Lisp) are designed for a Unix-like environment. Ubuntu Linux is categorized as "Supported" for these client libraries, whereas other variations such as Fedora Linux, macOS, and Microsoft Windows are listed as "Experimental" and are supported by the community. The rosjava native Java ROS client library, on the other hand, does not have these limitations, allowing ROS-based software to be

created for Android. ROS has also been integrated into an officially supported MATLAB toolbox for Linux, macOS, and Microsoft Windows, thanks to rosjava. Roslibs is a JavaScript client library that allows software to be integrated into a ROS system using any standards-compliant web browser. Microsoft released Core ROS for Windows 10 in September 2018.

Because developing truly stable, general-purpose robot software is challenging, ROS is generally recognized and used. Problems that seem insignificant to humans can differ dramatically between activities and environments from the robot's perspective. Dealing with these differences is so difficult that no single person, laboratory, or institution can hope to do it alone.

As a result, ROS was created from the bottom up to foster the development of collaborative robotics software. One laboratory, for example, might have experts in mapping interior spaces and could donate a world-class mapping system. Another group might be experts at utilizing maps to navigate, and yet another might have devised a computer vision method for spotting little items amid chaos that works effectively. As mentioned throughout this site, ROS was created particularly for groups like this to collaborate and build on each other's work.

3.2.2 ROS's Unique Features:

1. Modular and distributed design

ROS was created to be as distributed and flexible as possible, allowing users to utilise as much or as little of the software as they want. The components that make up ROS will be covered elsewhere, but ROS' modularity allows you to pick and choose which pieces are valuable to you and which parts you'd rather implement yourself.

Because of its distributed nature, ROS has a vast community of user-contributed packages that offer a lot of value to the core system. At last count, the ROS ecosystem has over 3,000 packages, and that's just the ROS packages that individuals have taken the trouble to publicise. The fidelity of these packages varies, ranging from proof-of-concept implementations of new algorithms to industrial-grade drivers and capabilities. The ROS user community creates an integration point for hardware drivers, generic robot capabilities, development tools, valuable external libraries, and more.

2. A thriving community

ROS has attracted a big user group from throughout the world in recent years. Historically, the majority of users were in research labs, but we are increasingly seeing commercial adoption, especially in industrial and service robotics.

ROS has a thriving community. According to our statistics, the ROS community has over 1.500 members on the ros-users mailing list, over 3,300 on the

collaborative documentation wiki, and over 5,700 on the community-driven ROS Answers Q&A website at the time of writing. There are about 22,000 wiki pages on the wiki, with around 30 wiki page modifications per day. To date, 13,000 questions have been submitted to the Q&A website, with a 70% response rate.

3. Licensing with Permission

ROS's core is released under the standard BSD three-clause licence. This is a fairly permissive open licence that allows commercial and closed source products to reuse it. While the core of ROS is licenced under the BSD licence, community packages frequently use additional licences, such as the Apache 2.0 4.

4. A Collaborative Setting

ROS provides a lot of value to most robotics projects on its own, but it also allows you to network and cooperate with world-class roboticists in the ROS community. Shared development of common components is one of ROS' main ideas.

5. Essential Elements

Infrastructure for Robots | Robot-Specific Features | Tools
While we can't present an entire list of everything in the ROS ecosystem, we can point 0you to some of the most important components and discuss their

functionality, technical specifications, and quality to help you understand what ROS can do for your project.

6. Infrastructure for Communication

At its most basic level, ROS provides a message passing interface for interprocess communication, which is referred to as middleware.

These features are provided by the ROS middleware:

- anonymous message passing publishing/subscription
- recording and playback of messages
- Remote procedure requests and responses
- system with scattered parameters

7. Message Transmission

When developing a new robot application, one of the initial requirements is usually a communication system. The anonymous publish/subscribe technique of ROS's built-in and welltested messaging system saves you time by managing the specifics of communication between distributed nodes. Another advantage of utilising a message passing system is that it compels you to create clear interfaces between your system's nodes, which improves encapsulation and encourages code reuse. The message IDL defines the structure of various message interfaces (Interface Description Language).

8. Compatibility with Other Libraries

Do you want to use OpenCV with MoveIt! to detect and pick up an object with your robot? ROS has you covered, with easy interface with these and other

popular open source projects, as well as a message passing system that lets you switch between different data sources, such as live sensors and log files.

GAZEBO

Gazebo is a multi-robot 3D indoor and outdoor simulator with a pluggable physics engine and dynamic and kinematic mechanics. Integration between ROS and Gazebo is possible thanks to a set of Gazebo plugins that support a variety of existing robots and sensors. Because the plugins use the same message protocol as the rest of the ROS ecosystem, you can construct ROS nodes that are compatible with simulation, logged data, and hardware. You can design your application in simulation and then deploy it to the physical robot with little or no code changes.

OpenCV

OpenCV is the most widely used computer vision library, with applications in academics and in products all around the world. Many common computer vision techniques and tools are available in OpenCV, which you can use and extend. ROS has a close relationship with OpenCV, allowing users to easily

feed data from a variety of cameras into OpenCV algorithms like segmentation and tracking. ROS uses OpenCV to provide libraries for camera calibration, monocular image processing, stereo image processing, and depth image processing, among other things. If your robot has USB, Firewire, or Ethernet-connected cameras. ROS with OpenCV will simplify your life.

Point Cloud Library (PCL)

The Point Cloud Library (PCL) is a perceptual library that specialises in manipulating and processing three-dimensional data and depth images. Filtering, feature detection, registration, kd-trees, octrees, sample consensus, and other point cloud methods are available in PCL. PCL and ROS will enable you gather, transform, process, visualise, and act on rich 3D data whether you're using a three-dimensional sensor like the Microsoft Kinect or a scanning laser.

Movelt!

Movelt! is a motion planning toolkit that provides efficient, well-tested implementations of cutting-edge planning techniques that have been utilised on everything from simple wheeled platforms to walking humanoids. Movelt! contains everything you need for motion planning, whether you wish to use an existing algorithm or create your own. Movelt! may be used with any ROS-compatible robot thanks to its ROS integration. The rviz and rqt plugins can be used to visualise planning data, and the ROS control system can be used to execute plans.

ROS industrial:

ROS-Industrial is an open-source project that applies ROS' enhanced capabilities to robotics and manufacturing automation. Interfaces for typical industrial manipulators, grippers, sensors, and device networks are available in the ROS-Industrial repository. It also offers software libraries for automatic 2D/3D sensor calibration, process path/motion planning, Scan-N-Plan applications, developer tools like the Qt Creator ROS Plugin, and manufacturer-specific training courses. An international consortium of industry and research participants supports ROS-I.

3.2.3 ROS 2

The initial ROS (Robot Operating System) was launched in 2007 by Open Robotics, and it was aimed to provide a set of software libraries and tools that developers could use to create robot applications. A ROS system is made up of a number of autonomous nodes that communicate with one another over a publish/subscribe messaging network.

A sensor's driver, for example, may be implemented as a node that publishes sensor data as a stream of messages. Filters, loggers, maps, navigation, and other nodes could then read those messages.

To perform a function, such nodes don't even have to be on the same system or be part of the same architecture. An Arduino system might send messages, a laptop could subscribe to them, and an Android phone could control motors or activate sensors. As a result, ROS is adaptive to the needs of the user. Furthermore, the platform is open source and maintained by a diverse group of individuals, some of whom have contributed code that others might utilise in their own projects.

Furthermore, ROS is a framework that sits on top of an OS, allowing it to separate the hardware from the software, allowing all of the robot's hardware to be thought of as software. This means that users can programme robots without having to have physical hardware on hand.

ROS' design structure is broken down into several essential parameters and tools, one of which is a computational graph model, in which processes are represented as nodes in a graph structure connected by topics. ROS nodes can communicate with one another via topics, make service calls to other nodes, provide a service to other nodes, and set or receive shared data from the parameter server, a community database.

Each node represents a single ROS graph-running process. Before taking any other actions, each node has a name that it registers with the ROS master. Multiple nodes with different names can exist in various namespaces, or a node

can be designated as anonymous, in which case an additional identifier will be generated at random and added to its provided name.

Nodes can send and receive messages via topic buses, which are user-defined and can comprise anything from sensor data to actuator orders. The nodes can also promote services or operations that result in a single outcome with a beginning and a finish, such as taking a single frame photograph. A parameter server is a shared database that allows nodes to access both static and non-static information, such as data that seldom changes.

Developers can also use the ROS core capabilities to view data, browse package structures, and write scripts to automate complex setups and setup processes. "rviz," a 3D visualizer for imaging robots, sensor data, and their work environs, "rosbag," a command-line tool for recording and replaying message data, and the catkin ROS build system are among them.

Mapping and localization (2D/3D SLAM), navigation, perception, coordinate frame representation, and simulation are some of the add-on packages for the ROS core platform. "rosbash," an app that provides a subset of tools to expand the capabilities of a Unix-based bash shell and command language, is also included in the tool set. Finally, "roslaunch" is a tool for launching many nodes both locally and remotely, as well as configuring the servers where the nodes broadcast and receive topics.

3.2.4 Some key features differences between ROS and ROS2

Features	ROS	ROS 2
Platforms	Tested on Ubuntu Maintained on other Linux flavors as well as OS X	ROS2 is currently being CI tested and supported on Ubuntu Xenial, OS X El Capitan as well as Windows 10
C++	C++03 // don't use C++11 features in its API	Mainly uses C++11 Start and plan to use C++14 & C++17
Python	Target Python 2	>= Python 3.5
Middleware	Custom serialization format (transport protocol + central discovery mechanism)	Currently all implementations of this interface are based on the DDS standard.
Unify duration and time types	The duration and time types are defined in the client libraries, they are in C++ and Python	In ROS2 these types are defined as messages and therefore are consistent across languages.
Components with life cycle	In ROS every node usually has its own main function.	The life cycle can be used by tools like roslaunch to start a system composed of many components in a deterministic way.

Threading model	In ROS the developer can only choose between singlethreaded execution or multithreaded execution.	In ROS2 more granular execution models are available and custom executors can be implemented easily.
Multiple nodes	In ROS the developer can only choose between singlethreaded execution or multithreaded execution.	In ROS2 it is possible to create multiple nodes in a process.
Roslaunch	In ROS roslaunch files are defined in XML with very limited capabilities.	In ROS2 launch files are written in Python which enables to use more complex logic like conditionals etc.

Table 2: Differences between ROS and ROS2

3.2.5 Visual Studio Code

In layman's terms, Visual Studio Code is a code editor. Visual Studio Code is a "free-editor that assists programmers in writing code, debugging, and code correction utilising the intelli-sense method." In simple terms, it allows users to write code quickly and easily. Many people claim it's half an IDE and half an editor, but the coders have the last word. Any programme or software that we see or use is based on background code. Coding was traditionally done in traditional editors or even in simple editors like notepad! These editors used to offer basic assistance to developers.

3.2.5.1 What can you accomplish using Visual Studio Code?

Visual Studio Code provides a number of distinct features. The following is a list of them:

- **Multiple programming languages are supported:** Multiple programming languages are supported. Previously, programmers need Web-Support: a separate editor for each language, but it now comes with built-in multi-language support. This also means that if there's a problem or a cross-language reference, it'll be able to recognise it quickly.

- **Intelli-Sense:** It can determine whether any code snippets are missing. Variable declarations and common variable syntaxes are also automatically generated. For example, if a variable is utilised in a programme and the user forgets to define it, intelli-sense will declare it for the user.

- **Cross-Platform Support:** Editors have traditionally supported either Windows, Linux, or Mac systems. Visual Studio Code, on the other hand, is cross-platform. As a result, it is compatible with all three platforms. Furthermore, the code runs on all three platforms; previously, the open-source and proprietary software codes were separate.

- **Extensions and Support:** Most programming languages are supported, however if a user/programmer wants to use a programming language that isn't, he can download and use an extension. Furthermore, because the addon runs as a separate process, it does not slow down the editor.

- **Repository:** With the growing demand for code, a secure and fast repository is essential. For pulling and saving instances, it is linked to Git or can be linked to any other repository.

- **Web support:** Web application support is already built-in. Web applications can thus be created and maintained in VSC.

- **Hierarchy Structure**: Files and directories contain the code files. Some files in the required code files are also required for other sophisticated projects. These files can be removed whenever it is convenient.

- **Improving Code:** Some code snippets can be declared in a slightly different way, which may aid the user in the code. This function advises the user to modify it to the suggested option whenever necessary.

- **Terminal Support:** When a user needs to start with a specific activity from the root of the directory, the in-built terminal or console allows them to do so without having to switch between two windows.

- **Multi-Projects:** You can open numerous projects with multiple files/folders at the same time. These projects/folders could be related or unrelated to one another.

- **Git Support:** Resources can be retrieved from the Git Hub Repo online and vice versa, as well as saved. Resource pulling also refers to cloning code that has been made publicly available on the internet. This code can be modified and saved at a later time.

- **Commenting:** This is a frequent feature, but not all languages offer it. Commenting on the code aids the user in recalling or tracking the desired sequence.

3.2.5.2 Advantages and Why should we use Visual Studio Code

- Freeware: The best aspect of all for all programmers, and even more so for companies, is that it is free.

- Many users will use it or have used it for desktop programmes solely, but it also supports Web technologies such as HTML, CSS, and JSON.

- Light-weight

- Robust Architecture

- Cross-platform support : Linux, Windows, Mac

3.3 What is a Robot Simulator?

A robot simulator enables virtual interaction with a robot and its environment without the need of a real robot. It's utilised to make an application for a physical robot without relying on the machine itself, saving money and time. In some cases, these apps can be directly transferred (or rebuilt) onto the physical robot.

3.3.1 Robot simulators often provide the following functionality:

- Ability to modify the physical layout of the robot's environment (e.g. adding a wall or moving a person around),
- Choice of a robot model (which impacts the availability of sensors, actuators, mass, etc.),
- Simulated sensor data (i.e. what would the selected robot model observe if it were the real world),
- Means to execute motor commands,
- Visualization of the simulation state.

The term "robotics simulator" can apply to a variety of robotics simulation programmes. Behavior-based robotics simulators, for example, allow users to

design rudimentary worlds of stiff objects and light sources and programme robots to interact with them in mobile robotics applications. When compared to binary or computational simulators, behavior-based simulation allows for more biologically inspired activities. Furthermore, behavior-based simulators are capable of "learning" from mistakes and displaying the anthropomorphic attribute of tenacity.

The 3D modelling and visualisation of a robot and its environment is one of the most prominent uses for robotics simulations. This sort of robotics software includes a virtual robot simulator that can simulate the motion of a real robot in a real work environment.

A physics engine is used in some robotics simulators to generate more realistic robot motion. Regardless of whether or not a real robot is accessible, using a robotics simulator to design a robotics control programme is highly recommended. The simulator makes it possible to write and debug robotics programmes off-line before having the final version tested on a real robot. This is primarily true for industrial robotics applications, as the effectiveness of off-line programming is determined by how well the robot's real environment matches the simulated environment. Because the robot motion is dependent on real-time sensor data, sensor-based robot movements are far more difficult to replicate and/or programme off-line.

Virtual simulations are one of the most cutting-edge programming methods available today. Both the organisation and the programmer can benefit from simulations that use virtual models of the working environment and the robots themselves. Costs are lowered by using simulations, and robots can be programmed off-line, eliminating any assembly line downtime. Months before

prototypes are made, robot operations and assembly pieces can be visualised in a 3D virtual environment. It's also easier to write programming for a simulation than it is for a genuine robot. While the shift to virtual simulations for programming robots is a step forward in user interface design, many of these applications are still in their early stages.

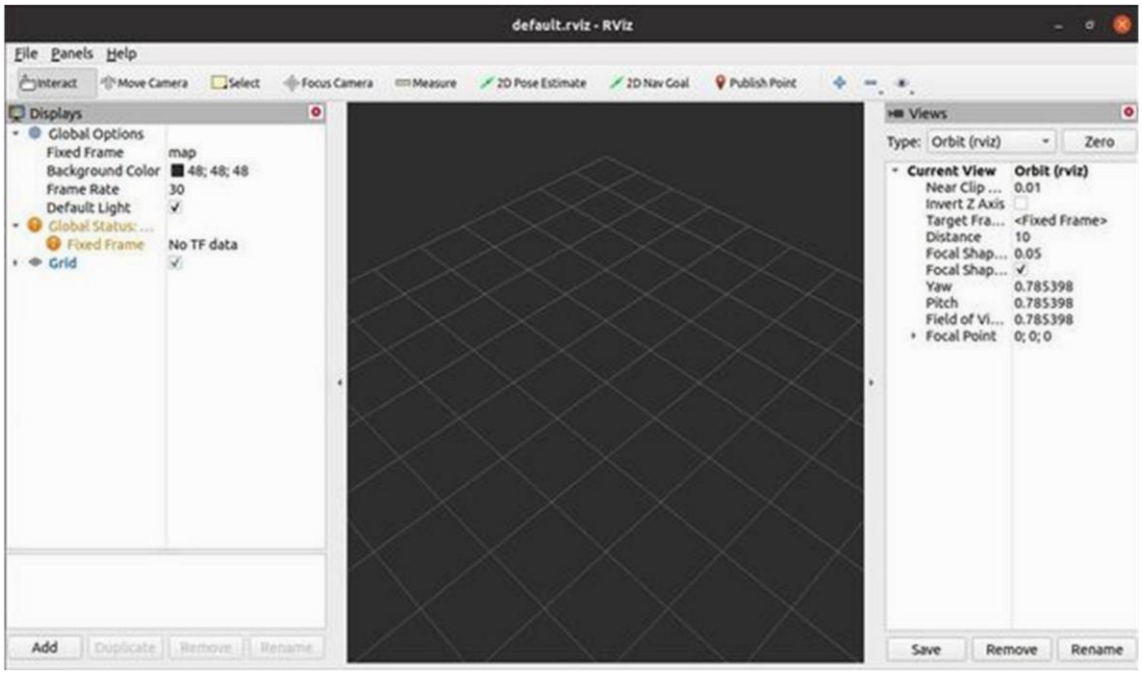
3.4 What is Rviz?

Rviz (short for "ROS visualisation") is a robot, sensor, and algorithm visualisation software tool. It allows you to observe how the robot sees the world (real or simulated). The goal of rviz is to allow you to visualise a robot's status. It makes use of sensor data to try to construct an accurate representation of the robot's surroundings.

There is a strong necessity in computational science and computer graphics to represent and show information in the real world, and numerous visualisation data structures and methods have been suggested to accomplish this goal. Unfortunately, because this method can only accept specific data structures, the dataflow model that is frequently chosen to handle this issue in visualisation systems is not flexible enough to depict newly generated data structures and algorithms. We offer RViz, a new visualisation tool that is independent of the input information data structures, to address this challenge.

Any scientific information visualisation algorithms are easier to develop than the dataflow model since there are no additional efforts required to manage the flow networks and the interface to abstracted information is easy in RViz.

Here is the screen you should see when you launch rviz:



Rviz layout

The left panel is the Displays panel. It has a list of plugins. These plugins enable you to view sensor data and robot state information. To add a plugin, you would click the **Add** button on the bottom left of the window.

3.5 What is Gazebo?

Gazebo is a three-dimensional dynamic simulator that can correctly and effectively model robot populations in both indoor and outdoor situations.

While similar to game engines, Gazebo offers physics simulation at a much higher degree of fidelity, a sensor suite, and user and programme interfaces.
While Gazebo is presently the market leader in robotic simulation, other major corporations like as Unity and Nvidia are catching up and developing their own robotic simulation software. It allows people to experiment with robots without making large investments. We used Unity's sophisticated game engine to develop our robotic simulation simulator. The beauty of Gazebo is that it allows you to create a dynamic simulation utilising a variety of high-performance physics engines such as ODE, Sim Body, Bullet, and DART. It aids with the replication of gravity, friction, torques, and any other real-world conditions that may affect the effectiveness of your simulation. This is critical to your robotics research and development. It would be dreadful if you created a perfect robot that couldn't function in the absence of gravity.

It included the ODE physics engine, OpenGL graphics, and sensor simulation and actuator control support code. Gazebo supports a variety of high-performance physics engines, including ODE, Bullet, and others (the default is ODE). It renders surroundings realistically, with high-quality lighting, shadows, and textures. It may imitate sensors like laser range finders, cameras (including wide-angle), Kinect-style sensors, and others that "see" the simulated environment. The OGRE engine is used by Gazebo for 3D rendering.

Typical uses of Gazebo include:

- testing robotics algorithms,
- designing robots,
- performing regression testing with realistic scenarios,
- A few key features of Gazebo include:

- multiple physics engines,
- a rich library of robot models and environments,
- a wide variety of sensors,
- convenient programmatic and graphical interfaces

3.5.1 System requirements

Gazebo works best with Ubuntu, a Linux distribution. You'll need a machine with the following features: A dedicated GPU; Nvidia cards perform well under Ubuntu. At least an Intel I5 or similar processor Ubuntu Trusty or later installed, with at least 500MB of free storage space.

3.5.2 Development History

From 2004 to 2011, Gazebo was a part of the Player Project. Gazebo became a standalone project with Willow Garage in 2011. The Gazebo project was taken over by the Open Source Robotics Foundation (OSRF) in 2012. In 2018, OSRF changed its name to Open Robotics.

Version 11 is Gazebo's most recent and final major release, a long-term support release with an end-of-life date of January 2025. All prior versions of Gazebo got long-term maintenance, with minor upgrades for Gazebo 9 and 10 issued simultaneously with the release of Gazebo 11.0.0. Open Robotics moved its attention to developing Ignition, a "collection of open source software libraries meant to simplify creation of high-performance applications," with a target audience of robot developers, designers, and educators, after Gazebo 11 was launched in January 2020.

Ignition's first version was released on February 2019. The standalone Gazebo simulator is referred to as Gazebo Classic on the Ignition website to distinguish it from Ignition Gazebo, which is now included in the current Ignition releases. The need for major updating in Gazebo's code, as well as the possibility to migrate from a monolithic architecture to a set of loosely connected libraries, were both mentioned by Open Robotics.

3.5.3 Benefits Of Gazebo Simulation

Gazebo helps you integrate a multitude of sensors, and it gives you the tools to test these sensors and develop your robots to best use them. Consider the case when you don't have access to robotic hardware and need to test hundreds of robots at once. Without a robotic simulator like Gazebo, it's impossible. Even if you have hardware, Gazebo is useful because it allows you to test your robotic idea before putting it into action in the real world. This is why so much money is being spent on robotic simulators and digital twins. They want to improve the workflow and speed of their production processes without spending too much money on technology. Because Gazebo is open-source software, many third-party plugins and solutions are available to assist you address specific problems or speed up your process. Gazebo is constantly evolving, with the most recent version being Gazebo 11.

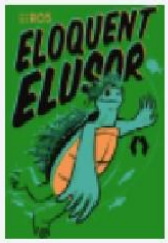
3.5.4 Drawbacks Of Gazebo Simulation

Gazebo lacks several of the features seen in 3D simulators such as Nvidia Isaac or our Zero SIM. It's difficult to import 3D models into Gazebo, and finding someone who can prepare the files for Gazebo can be challenging if you're not a 3D modeller. It will be much easier to import these models and have more realistic testing conditions if you use more popular tools like Unity. In Unity, you may re-create your entire warehouse. Gazebo installation is also a difficult task. Its Windows installation includes a total of 18 stages, which will be challenging for someone who isn't a developer and isn't used to installing software with code.

3.6 RViz vs Gazebo

While RViz and Gazebo both have graphical user interfaces that appear to be similar, they are very distinct in nature. A simulator is Gazebo. It creates synthetic sensor data and sets robots in a virtual world where people may give them commands. This sensor data isn't exactly user-friendly (e.g. some numbers and binaries). RViz merely visualises the sensor data generated by the robot so that human users may understand what it sees.

3.7 ROS Versions:

Distro	Release date	Logo	EOL date
Galactic Geochelone	May 23rd, 2021		November 2022
Foxy Fitzroy	June 5th, 2020		May 2023
Eloquent Elusor	November 22nd, 2019		November 2020
Dashing Diademata	May 31st, 2019		May 2021
Crystal Clemmys	December 14th, 2018		December 2019

Distro	Release date	Poster	EOL date
Lunar Loggerhead	May, 2017		May, 2019
Kinetic Kame	May 23, 2016		**2021-05-30**
Jade	May 23, 2015		2017-05-30
Indigo	July 22, 2014		2019-04-30
Hydro	September 4, 2013		2014-05-31
Groovy Galapagos	December 31, 2012		2014-07-31
Fuerte Turtle	April 23, 2012		
Electric Emys	August 30, 2011		
Diamondback	March 2, 2011		
C Turtle	August 2, 2010		
Box Turtle	March 2, 2010		

Old version | Older version, still supported | **Latest version** | Future release

3.8 HARDWARE:

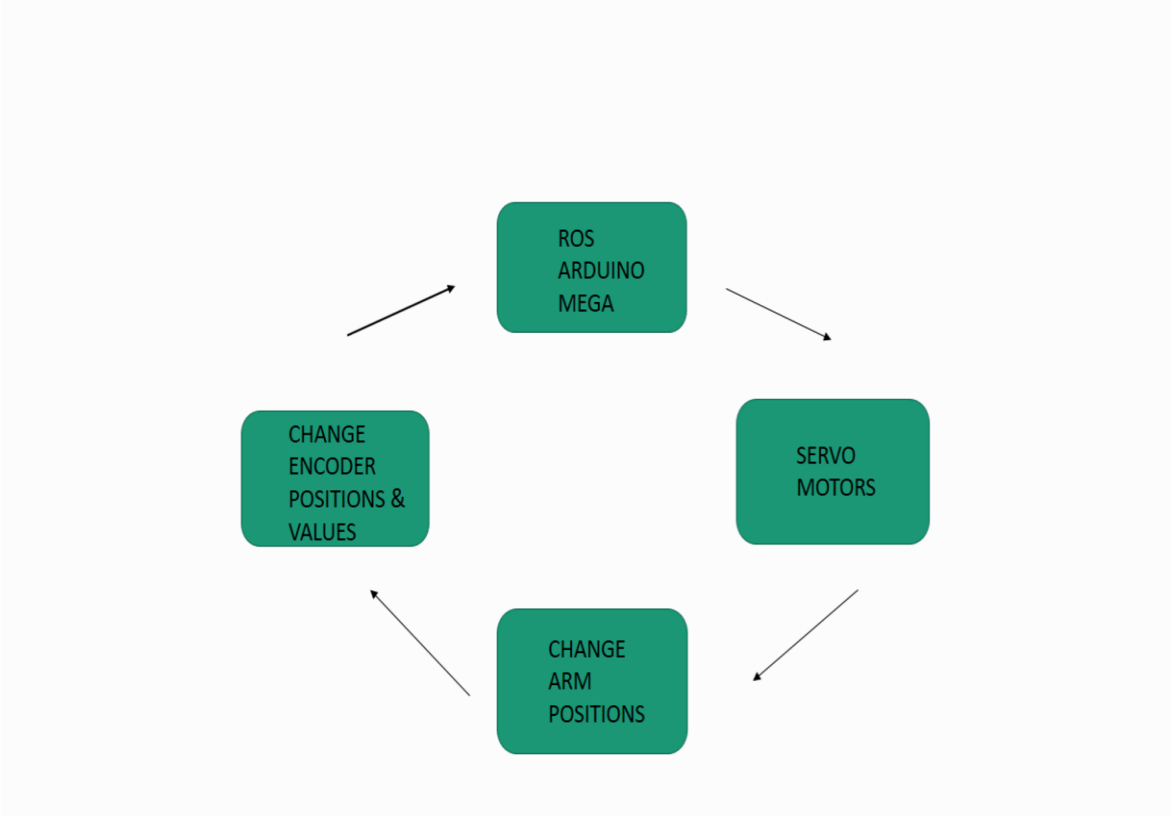


Figure 5: Hardware Block Diagram

3.8.1 ARDUINO MEGA 2560

The Arduino board is a microcontroller board based on the Atmega 2560 microcontroller. The processing or wiring language is executed by this board's growing environment. These boards have re-energized the automation sector with their user-friendly platform, which allows anyone with a limited or no technical background to get started learning how to programme and control the Arduino board.These boards are used to link to software on your PC such as Max MSP, Processing, and Flash, or to extend independent interactive items.

3.8.1.1 What is an Arduino Mega 2560?

The ATmega2560 microcontroller is used in microcontroller boards such as the "Arduino Mega." It has 54 digital input/output pins, 16 of which are analogue inputs and 14 of which are utilised as PWM outputs, 4 hardware serial ports (UARTs), an ICSP header, a power jack, a USB connection, and a RST button. This board primarily contains everything required to support the microcontroller. This board's power can be supplied by connecting it to a PC via a USB cable, a battery, or an AC-DC adapter. A base plate can be used to shield this board from an unexpected electrical discharge.

Figure 6: Arduino Mega 2560

The Mega 2560 R3 board's SCL and SDA pins are connected to the AREF pin. In addition, there are two most recent pins near the RST pin. The IOREF is a pin that allows the shields to alter the voltage provided by the Arduino board. Another pin is unrelated and will be retained for future use. These boards operate with any old shield, but they can be adjusted to work with newer shields that use these extra pins.

3.8.1.2 Communication:

The Arduino Mega2560 can communicate with a computer, another Arduino, or other microcontrollers. The ATmega2560 has four hardware UARTS for TTL (5V) serial transmission. One of these is channelled over USB by an Atmega16U2 (Atmega 8U2 on revision 1 and revision 2 boards) on the board, which gives a virtual com port to software on the computer (Windows machines will need an inf file, but OSX and Linux machines will identify the board as a COM port immediately). The Arduino software has a serial monitor that allows simple textual data to be sent to and from the device. The RX and TX LEDS on the board will blink when data is communicated via the Atmega8U2/Atmega16U2 chip and USB connection to the PC (but not for serial communication on pins 0 and 1).

Using the Software Serial library, serial communication can be done on any of the Mega2560's digital pins. The Atmega2560 also supports TWI and SPI connectivity. The Arduino software includes a Wire library that simplifies using the TWI bus.

3.8.1.3 **Programming:**

The Arduino Mega 2560 can be programmed using an IDE (Arduino Software), which supports the C programming language. The sketch is the software code that is burned into the programme and then transferred to the Arduino board via USB connection. The boot loader on an Arduino mega board eliminates the need for an external burner to burn the programme code onto the Arduino board. The boot loader can communicate using the STK500 protocol in this case.

After compiling and burning the Arduino software, we can disconnect the USB cable and remove the Arduino board's power supply. If you plan to use the Arduino board for your project, the power source can be delivered through a power jack or the board's Vin pin.

Another advantage is the ability to multitask wherever an Arduino mega board is available. Instead, the Arduino IDE Software does not enable multi-tasking; however, other operating systems such as RTX and Free RTOS can be used to build C-programs for this purpose. With the help of an ISP connector, you can utilise this in your own custom build programme.

This concludes the datasheet for the Arduino Mega 2560. It's a replacement for the older Arduino Mega board. Because of the large number of pins, it is rarely used in normal projects; however, it may be found in difficult projects like as temperature sensors, 3D printers, IoT applications, radon detectors, real-time data monitoring, and so on.

3.9 Encoders

A device or process that translates data from one format to another is known as an encoder. An encoder is a device that can detect and transform mechanical motion into an analogue or digital coded output signal in position sensing. It particularly measures position, from which velocity, acceleration, and direction can be calculated in either linear or circular motion.
Different physical principles of operation, outputs, communication protocols, and other factors influence encoder functionality.

What Is an Encoder?

Encoders are used in machinery across the board. Cut-to-length applications, plotters, robotics, packing, conveying, automation, sorting, filling, imaging, and many other applications employ encoders. An encoder is a feedback-giving sensing device. Encoders transform motion into an electrical signal that can be read by a motion control system's control device, such as a counter or a PLC. The encoder generates a feedback signal that can be used to calculate position, count, speed, and direction. This information can be used by a control device to transmit a command for a certain function. For instance:

- An encoder with a measuring wheel tells the control device how much material has been supplied in a cut-to-length application, so the control device knows when to cut.
- By delivering positioning feedback, encoders in an observatory tell actuators what position a moving mirror is in.
- Encoders offer accurate motion feedback on railroad-car raising jacks, allowing the jacks to lift in lockstep.

- The encoder signal is used by the PLC to control the timing and speed of bottle rotation in a precision servo label application system.
- The encoder triggers a print head to make a mark at a precise position in a printing application.
- Encoders installed on a motor shaft offer location feedback to a huge crane, allowing it to know when to pick up or release its load.
- Feedback tells the filling machines where the bottles or jars are in an application where they are being filled.
- Encoders in an elevator tell the controller when the car has arrived at the correct floor and position. That is, elevator doors open level with the floor due to encoder motion feedback to the elevator's controller. Without encoders, you could have to climb into or out of an elevator instead of strolling out onto a level floor.
- Encoders provide motion feedback to robots on automated assembly lines. This can entail ensuring that the robotic welding arms on an automotive assembly line have the correct information to weld in the proper places.

3.9.1 Rotary Encoders

The angular position or motion of a shaft or axle is converted to analogue or digital output signals by a rotary encoder, also known as a shaft encoder. Absolute and incremental rotary encoders are the two most common varieties. An absolute encoder is an angle transducer since its output indicates the current shaft position. An incremental encoder's output offers information on the shaft's motion, which is often processed into information like position, speed, and distance.

Rotary encoders are used in a variety of applications that require mechanical system monitoring, control, or both, such as industrial controls, robotics, photographic lenses, computer input devices like optomechanical mice and trackballs, controlled stress rheometers, and rotating radar platforms, to name a few.

3.9.1.1 Basic Types of Rotary Encoders

- **Absolute Encoder**

When power is disconnected from an absolute encoder, the location information is preserved. When power is applied, the encoder position is immediately available. The relationship between the encoder value and the physical location of the controlled machinery is established during assembly, and the system does not require calibration to maintain position accuracy. An absolute encoder uses numerous code rings with different binary weightings to generate a data word that represents the encoder's absolute position inside one revolution. A parallel absolute encoder is the name given to this sort of encoder.

Additional code wheels and toothed wheels are included in a multi-turn absolute rotary encoder. A high-resolution wheel records the number of complete shaft revolutions, while a lower-resolution geared code wheel records the fractional rotation.

- **Incremental encoder**

An incremental encoder will notify changes in position instantly, which is a need in some applications. It does not, however, report or track absolute

position. As a result, to establish absolute position measurement, the mechanical system monitored by an incremental encoder may need to be homed (moved to a fixed reference point).

Absolute encoders

- **Mechanical absolute encoders**

An insulating disc is tightly fixed to the shaft, and a metal disc with a set of concentric rings of apertures is fixed to it. A stationary object is attached to a row of sliding contacts, each of which wipes against the metal disc at a varied distance from the shaft. Some of the contacts touch metal as the disc rotates with the shaft, while others fall into gaps where the metal has been carved out. Each contact is connected to a separate electrical sensor, and the metal sheet is connected to an electric current source. The metal pattern is designed to provide a unique binary code for each potential position of the axle, with certain contacts connected to the current source (i.e. switched on) and others not (i.e. switched off).
Mechanical encoders are often used in low-speed applications such as manual volume or tuning controls in a radio receiver since brush-type contacts are prone to wear.

- **Optical absolute encoders**

The disc of an optical encoder is constructed of glass or plastic and has transparent and opaque portions. The optical pattern that comes from the disc's position at any given time is read by a light source and photo detector array. The Gray coding is frequently employed. A controlling device, such as a

microprocessor or microcontroller, may read this code to compute the shaft's angle. The absolute analogue type generates a one-of-a-kind dual analogue code that may be converted into a shaft angle.

- **Magnetic absolute encoders**

The magnetic encoder represents the encoder position to a magnetic sensor using a series of magnetic poles (2 or more) (typically magneto-resistive or Hall Effect). Magnetic pole locations are read by the magnetic sensor.
Similar to an optical encoder, this code can be read by a controlling device, such as a microprocessor or microcontroller, to calculate the angle of the shaft. The absolute analog type generates a one-of-a-kind dual analogue code that may be converted into a shaft angle. These encoders may be ideal for use in settings where other types of encoders may fail owing to dust or particle accumulation due to the nature of recording magnetic effects. Vibrations, small misalignment, and shocks are also relatively insensitive to magnetic encoders.

- **Brushless motor commutation**

In permanent magnet brushless motors, which are extensively used on CNC machines, robotics, and other industrial equipment, built-in rotary encoders are utilised to detect the angle of the motor shaft. In these situations, the encoder acts as a feedback mechanism that is critical to proper equipment operation. Electronic commutation is required for brushless motors, which is frequently accomplished in part by employing rotor magnets as a low-resolution absolute encoder (typically six or twelve pulses per revolution). The servo drive receives the shaft angle information and uses it to energise the proper stator winding at any given time.

- **Capacitive absolute encoders**

Within the encoder, an asymmetrical shaped disc rotates. The capacitance between two electrodes will change as a result of this disc, which may be measured and converted to an angular value.

- **Absolute multi-turn encoder**

More than one revolution can be detected and stored using a multi-turn encoder. When an encoder detects shaft movements without external power, it is referred to as an absolute multi turn encoder.

- **Battery-powered multi-turn encoder**

For preserving counts through power cycles, this sort of encoder needs a battery. It detects movement using an energy-saving electrical architecture.

- **Geared multi-turn encoder**

The number of revolutions is mechanically stored in these encoders using a train of gears. One of the technologies listed above is used to detect the location of the single gears.

- **Self-powered multi-turn encoder**

These encoders create energy from the moving shaft using the energy harvesting method. This concept, which was first proposed in 2007, employs a Wiegand sensor to generate enough current to operate the encoder and write the turns count to non-volatile memory.

3.9.2 Different Mechanisms in encoders

- **Mechanical**: Conductive encoders are another name for them. The information is encoded using contact brushes sensing the conductive areas via a sequence of circumferential copper tracks etched onto a PCB. Mechanical encoders are inexpensive, but they are vulnerable to wear. Human interfaces, such as digital multimeters, frequently use them.

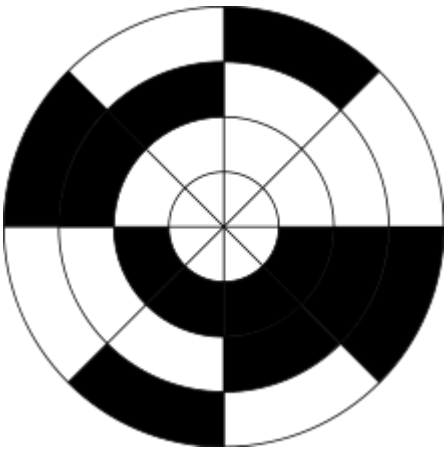
- **Optica**l: A photodiode is illuminated by light passing through slits in a metal or glass disc. There are also reflective variations. One of the most widely used technologies. Optical encoders are extremely dust-sensitive.

- **On-Axis Magnetic**: A particularly magnetised 2 pole neodymium magnet attached to the motor shaft is commonly used in this technology. It can work with motors with only one shaft projecting out of the motor body because it can be fastened to the end of the shaft. The precision might range from a few degrees to less than one degree. The resolution can range from 1 degree to 0.09 degree (4000 CPR, Count per Revolution). Internal interpolation that is poorly constructed can produce output jitter, but this can be mitigated by using internal sample averaging.

- **Off-Axis Magnetic**: Rubber-bonded ferrite magnets are commonly attached to a metal hub in this technology. This provides design freedom and low cost for unique applications. Many off-axis encoder chips are flexible enough to accommodate any number of pole widths, allowing them to be put in any position required for the application. Magnetic encoders work in extreme conditions when optical encoders would fail.

3.9.3 Ways of encoding shaft position

- **Standard binary encoding**



An example of a binary code, in an extremely simplified encoder with only three contacts, is shown below

Standard Binary Encoding

Sector	Contact 1	Contact 2	Contact 3	Angle
0	off	off	off	0° to 45°
1	off	off	ON	45° to 90°
2	off	ON	off	90° to 135°
3	off	ON	ON	135° to 180°
4	ON	off	off	180° to 225°
5	ON	off	ON	225° to 270°
6	ON	ON	off	270° to 315°
7	ON	ON	ON	315° to 360°

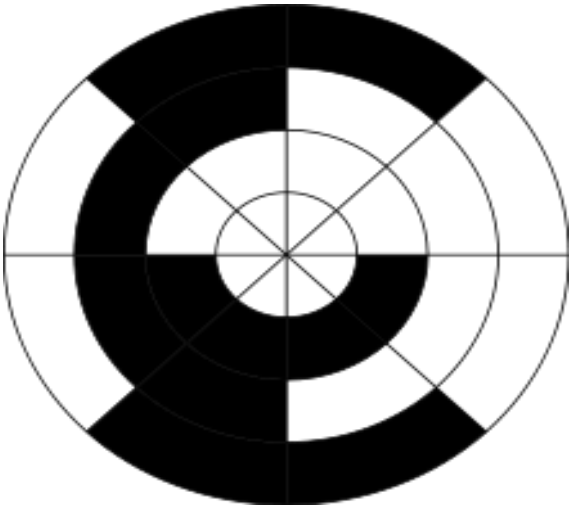
The number of unique positions of the shaft is 2n in general when there are n contacts. Because n is 3 in this case, there are 2^3 or 8 positions.

As the disc rotates in the example above, the contacts produce a conventional binary count. The disadvantage is that if the disc stops between two adjacent sectors or the contacts are not completely aligned, determining the angle of the shaft can be difficult. Consider what occurs when the shaft angle changes from 179.9° to 180.1° to illustrate this issue (from sector 3 to sector 4). According to the preceding table, the contact pattern shifts from off-on-on to on-off-off at some point. In fact, however, this is not the case. The contacts in a practical device are never perfectly aligned, thus each one switches at a different time. If, for example, contact 1 is activated first, then contact 3, then contact 2, the actual code sequence is:

off-on-on (starting position)
on-on-on (first, contact 1 switches on)
on-on-off (next, contact 3 switches off)
on-off-off (finally, contact 2 switches off)

Examine the table's sectors that correspond to these codes. They are 3, 7, 6, and 4, in that order. The shaft appears to have jumped from sector 3 to sector 7, then backwards to sector 6, then backwards again to sector 4, which is where we expected to find it, based on the coding sequence. This behaviour is undesirable in many instances and may lead the system to fail. If the encoder were used in a robot arm, for example, the controller would believe the arm was in the wrong position and attempt to fix the error by turning it through 180 degrees, perhaps damaging the arm.

- **Gray encoding**



3-bit binary-reflected Gray code rotary encoder for angle-measuring devices (BRGC). Contact 1 in the table refers to the inner ring. The black areas are active. The angle increases counter-clockwise from zero degrees on the right. Gray coding is used to avoid the difficulty mentioned above. Any two neighbouring codes differ by only one bit location in this binary counting method. The Gray-coded version of the three-contact example presented before would be as follows.

Gray Coding

Sector	Contact 1	Contact 2	Contact 3	Angle
0	off	off	off	0° to 45°
1	off	off	ON	45° to 90°
2	off	ON	ON	90° to 135°
3	off	ON	off	135° to 180°
4	ON	ON	off	180° to 225°
5	ON	ON	ON	225° to 270°
6	ON	off	ON	270° to 315°
7	ON	off	off	315° to 360°

In this case, the transition from sector 3 to sector 4 includes simply one of the contacts changing its state from on to off or vice versa, as with all other transitions. This means that the wrong code sequence illustrated in the previous illustration cannot occur.

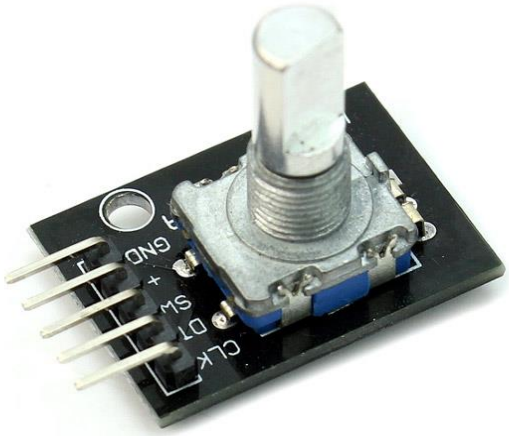
Single-track Gray encoding

If a contact is moved to a new angular point (while remaining the same distance from the centre shaft), the matching "ring pattern" must be rotated at the same angle to provide the same output. If the most significant bit is rotated far enough, the following ring out will be identical. The inner ring can then be eliminated, and the sensor for that ring can be relocated to the remaining, identical ring (but offset at that angle from the other sensor on that ring). A quadrature encoder with a single ring is made up of those two sensors on a single ring.

It is possible to arrange several sensors around a single track (ring) so that consecutive positions differ at only a single sensor; the result is the single-track Gray code encoder.

3.9.4 Keyes KY-040 Rotary Encoder

The Keyes KY-040 rotary encoder is a rotary input device (like in a knob) that shows how far the knob has been rotated and in which direction it is rotating. It's an excellent device for controlling stepper and servo motors. It might also be used to operate electronic devices such as digital potentiometers.



Rotary Encoder Basics :

The number of locations per revolution of a rotary encoder is xed. As you turn the encoder, these positions are easily sensed as little "clicks."

Figure 7

Three pins are located on one side of the switch. A, B, and C are the common names for them. They are orientated in the instance of the KY-040 (as shown in fig. 7)

There are two switches inside the encoder. Pin A is connected to pin C by one switch, while pin B is connected to C by the other switch.

Both switches are either open or closed in each encoder position. These switches change states as a result of each click:

- If both switches are closed, rotating the encoder one position clockwise or counter clockwise will open both switches.
- When both switches are open, rotating the encoder one position clockwise or counterclockwise closes both switches.

The figure below depicts how the switch is put together (in fig. 8)

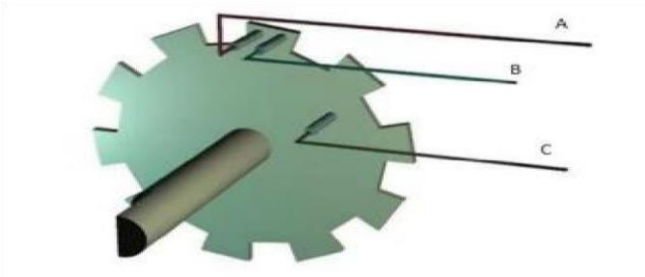


Figure 8

As you can see, the A and B terminals are in such an angle relationship that:

The switch linking A and C will change states first when you rotate it clockwise. When you turn the switch counterclockwise, the switch linking B and C will be the first to change states. If the opening and closing of the switches were represented as wave shapes (as shown fig7)

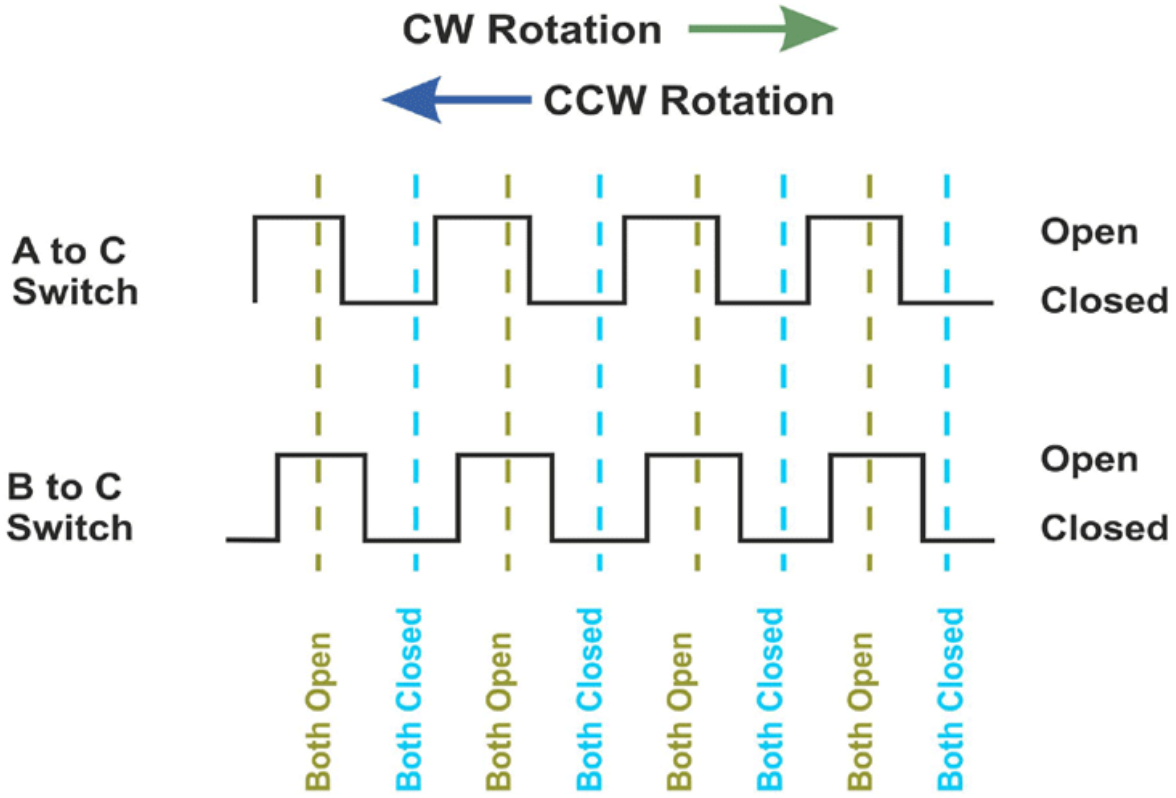


Figure 9

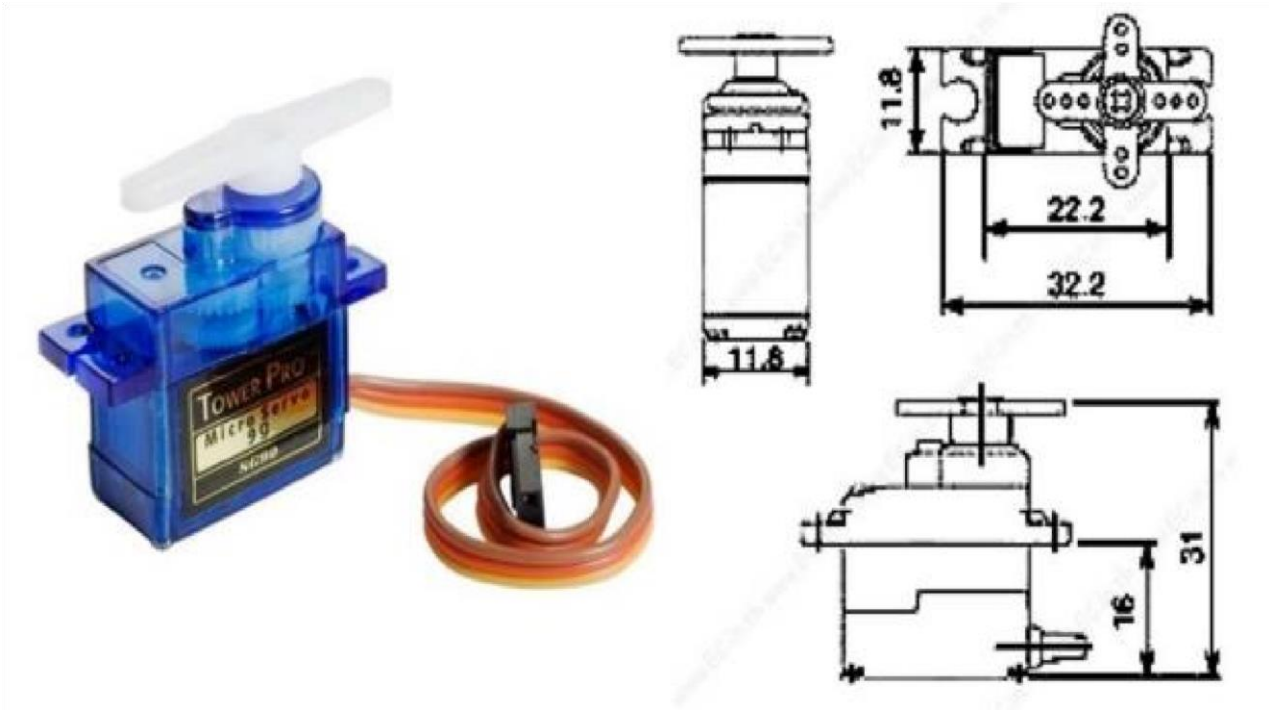
The direction of rotation is essentially determined by determining which switch changed states.

The switch rotates in a clockwise manner if A changed states first.

The switch rotates in a counter clockwise manner if B changed states first.

3.10 MOTORS

3.10.1 SG90 MICRO 9G:



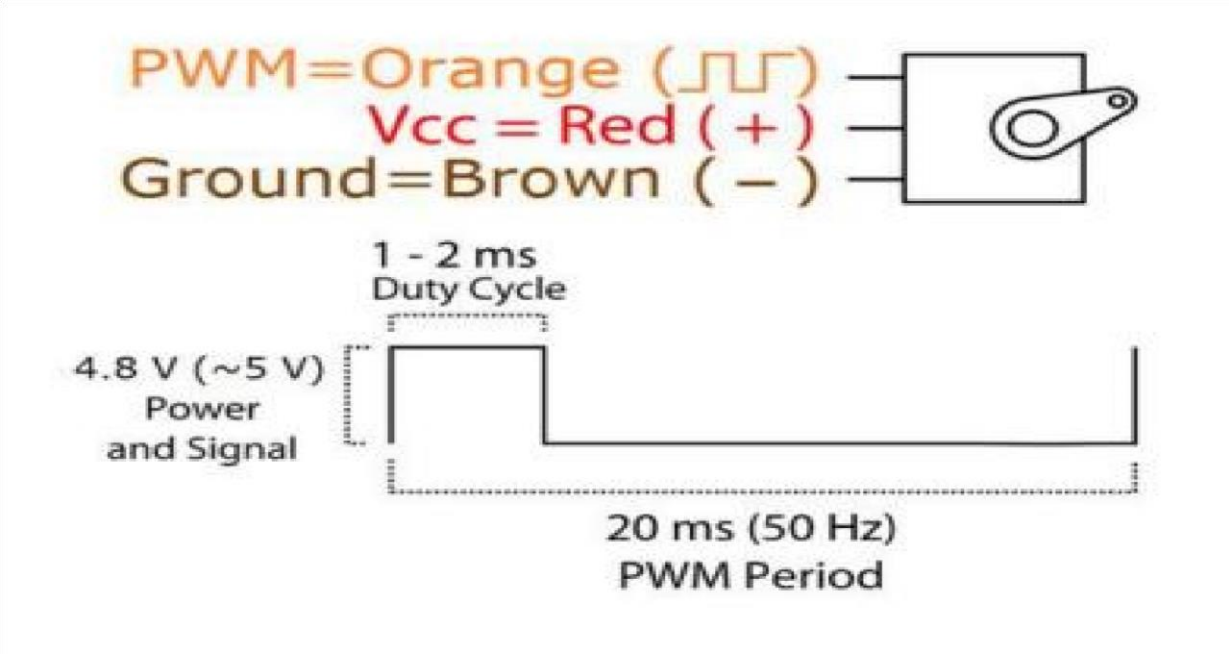
Small and light, but with a lot of power. Servos can spin 180 degrees (90 degrees in each direction) and work similarly to regular types but are smaller. To control these servos, you can use any servo code, hardware, or library. It includes three horns (arms) as well as hardware.

The TP SG90 is a solid choice for most park flyers and helicopters, as it is close in size and weight to the Hitec HS-55.

The TP SG90 servo is 0.32 ounces in weight (9.0 grams). With the wire and connector, the total weight is 0.37 ounces (10.6 grams).

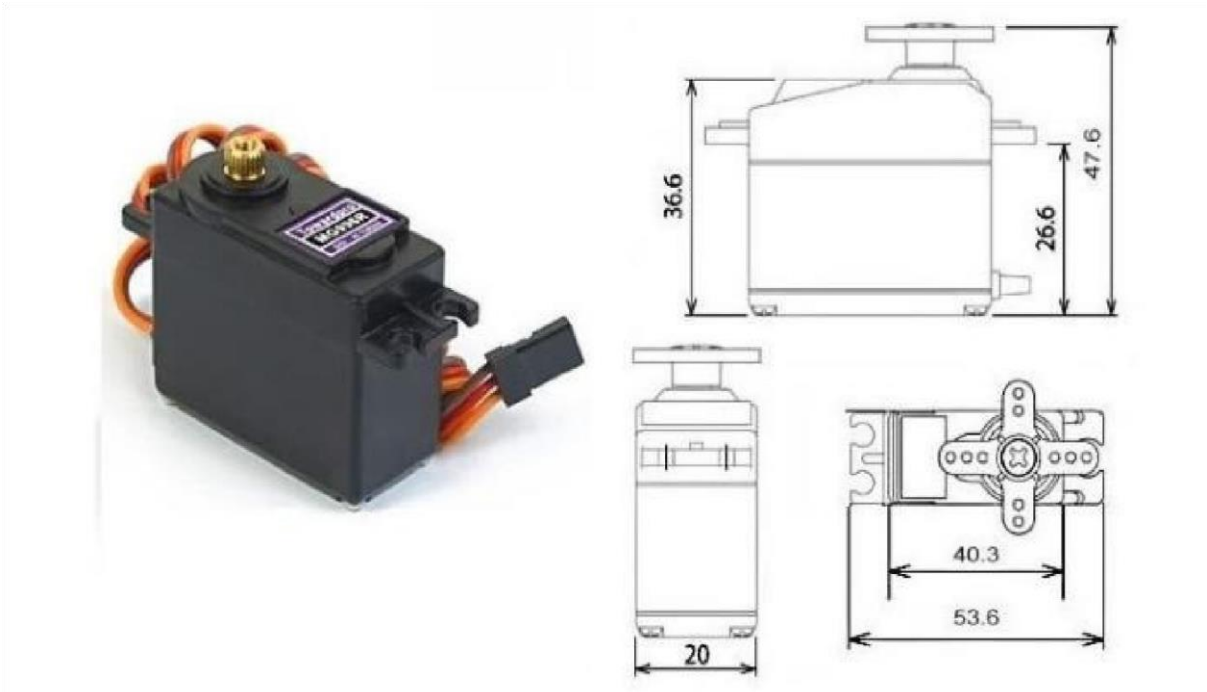
Most receivers, including Futaba, JR, GWS, Cirrus, Blue Bird, Blue Arrow, Corona, Berg, and Hitec, use the universal "S" type connector on the TP SG90.

The wire colors are Red = Battery (+) Brown = Battery (-) Orange = Signal



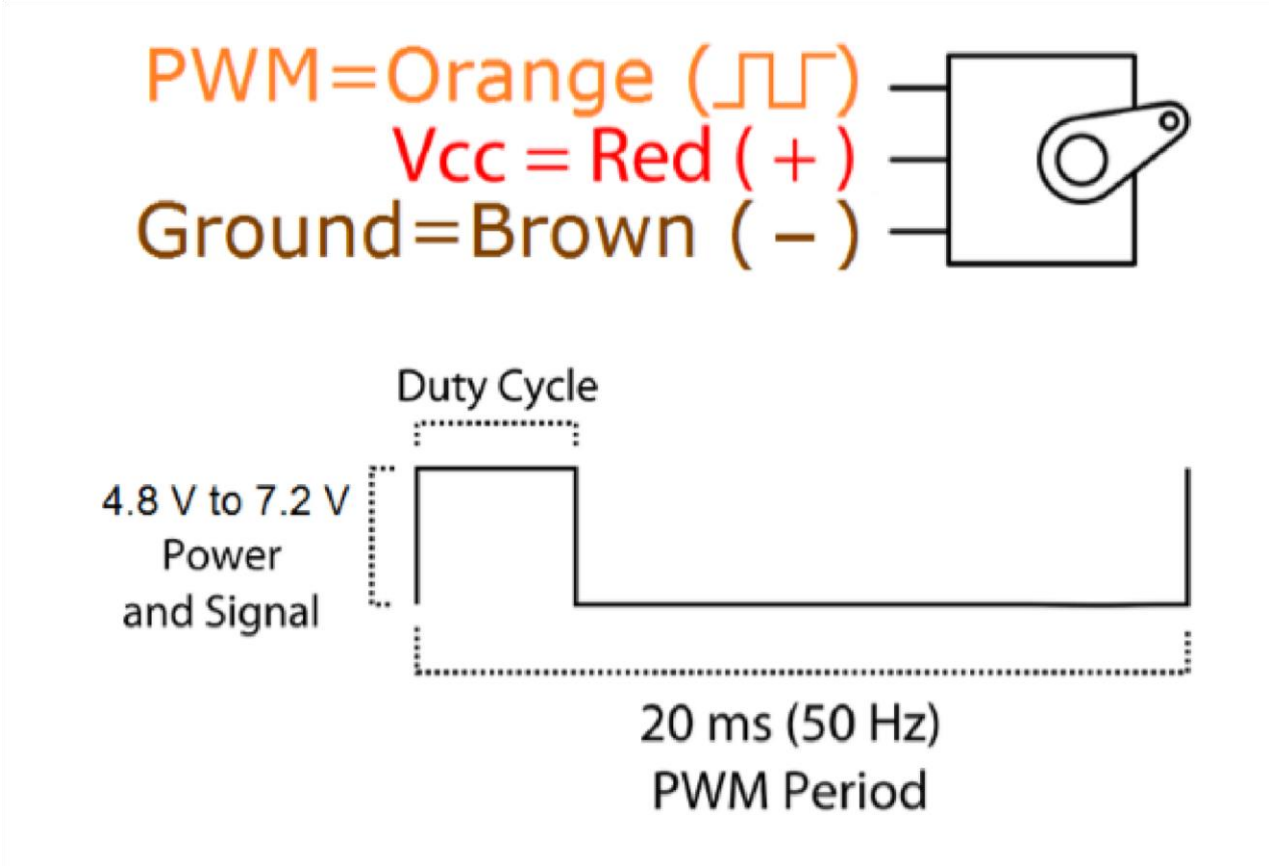
SG90 MICRO 9G

3.10.2 MG-996R SERVO MOTORS:



The metal gearing of this High-Torque MG996R Digital Servo results in an extremely high 10kg stalling torque in a small package. The MG996R is an enhanced version of the well-known MG995 servo, with improved shock-proofing and a revised PCB and IC control system that makes it much more accurate than its predecessor. To increase dead bandwith and centering, the gearing and motor have been modified. Most receivers, including Futaba, JR, GWS, Cirrus, Blue Bird, Blue Arrow, Corona, Berg, Spektrum, and Hitec, are compatible with the unit's 30cm cable and 3 pin 'S' type female header connector.

This basic servo with high torque can rotate about 120 degrees (60 in each direction). You can operate these servos using any servo code, hardware, or library, making it ideal for beginners who want to make things move without having to create a motor controller with feedback and gear box, especially since it fits in small spaces. The MG996R Metal Gear Servo also includes a variety of arms and accessories to help you get started quickly.



MG-996R SERVO MOTOR

3.11 WIRE CONFIGURATIONS

Wire Number	Wire Colour	Description
1	**Brown**	**Ground wire connected to the ground of system**
2	**Red**	**Powers the motor typically +5V is used**
3	**Orange**	**PWM signal is given in through this wire to drive the motor**

3.11.1 Applications

- Actuators are found in a variety of robotics, including the biped robot, the hexapod, and the robotic arm.
- Commonly used for steering mechanism in RC toys.
- Robots that require position control without feedback
- As a result, it is used in multi-DOF robotics such as humanoid robots.

A jumper wire is an electrical wire (or a group of them in a cable) with a connector or pin at each end (or sometimes without – simply "tinned") that is used to connect the components of a breadboard or other prototype or test circuit internally or with other equipment or components without soldering.
Individual jump wires are connected by slipping their "end connectors" into slots on a breadboard, a circuit board's header connector, or a piece of test equipment.

3.11.2 TYPES OF WIRES

Jumper wires come in a variety of shapes and sizes. Some electrical connectors are the same on both ends, whereas others are different. The following are some common connectors:

- Solid tips – are used to attach to a breadboard or a female header connection. The elements' layout and ease of insertion on a breadboard allow for increased component and jump wire mounting density without concern of short circuits. To distinguish the various functioning signals, the jump wires vary in size and colour.

- Crocodile clips – are used to temporarily link sensors, buttons, and other prototype parts with components or equipment that have random connectors, cables, screw terminals, and so on.

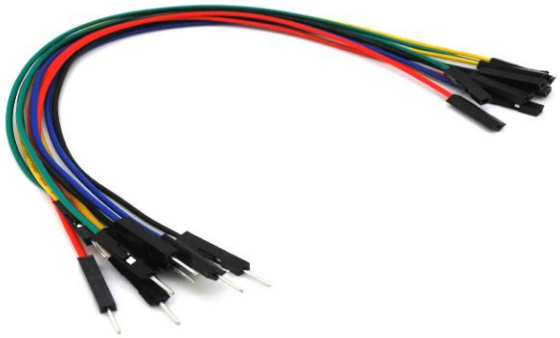
- Banana connectors – For DC and low-frequency AC transmissions, are often utilised on test equipment.

- Registered jack (RJnn) – are widely utilised in telephone (RJ11) and computer networking applications (RJ45).

- RCA connectors – Audio, low-resolution composite video signals, and other low-frequency applications that require a shielded connection are frequently employed.

- RF connectors – Radio frequency signals are carried between circuits, test equipment, and antennas via these cables.

- RF jumper cables - Jumper cables are corrugated cables that are smaller and more pliable and are used to connect antennas and other components to network wiring. In base stations, jumpers are also used to connect antennas to radio devices. The diameter of the most bendable jumper cable is usually 1/2".



3.12 KINEMATICS

Kinematics is a part of classical mechanics that deals with the motion of points, bodies (objects), and systems of bodies (groups of objects) without taking the forces into account. Kinematics is sometimes referred to as the "geometry of motion" and is sometimes considered a branch of mathematics. The initial conditions of any known values of position, velocity, and/or acceleration of points inside the system are declared in a kinematics issue.

In astrophysics, kinematics is used to describe the motion of celestial bodies and groups of celestial bodies. Kinematics is a term used in mechanical engineering, robotics, and biomechanics to explain the motion of multi-link systems, such as an engine, a robotic arm, or the human skeleton.

Geometric transformations, also known as stiff transformations, are used to simplify the derivation of the equations of motion by describing the movement of components in a mechanical system. They're also important in dynamic analysis.

The technique of measuring the kinematic quantities used to characterise motion is known as kinematic analysis. In engineering, for example, kinematic analysis can be used to determine a mechanism's range of motion, and kinematic synthesis can be used to create a mechanism with a desired range of motion.

3.12.1 Forward Kinematics

Forward kinematics is the use of a robot's kinematic equations to calculate the position of the end-effector given provided joint parameter values. The graphical approach can be used to construct a simple forward kinematics from its space. The graphical approach for solving forward kinematics will be discussed in this part using a three-link planar robot as seen in fig (as shown in fig8)

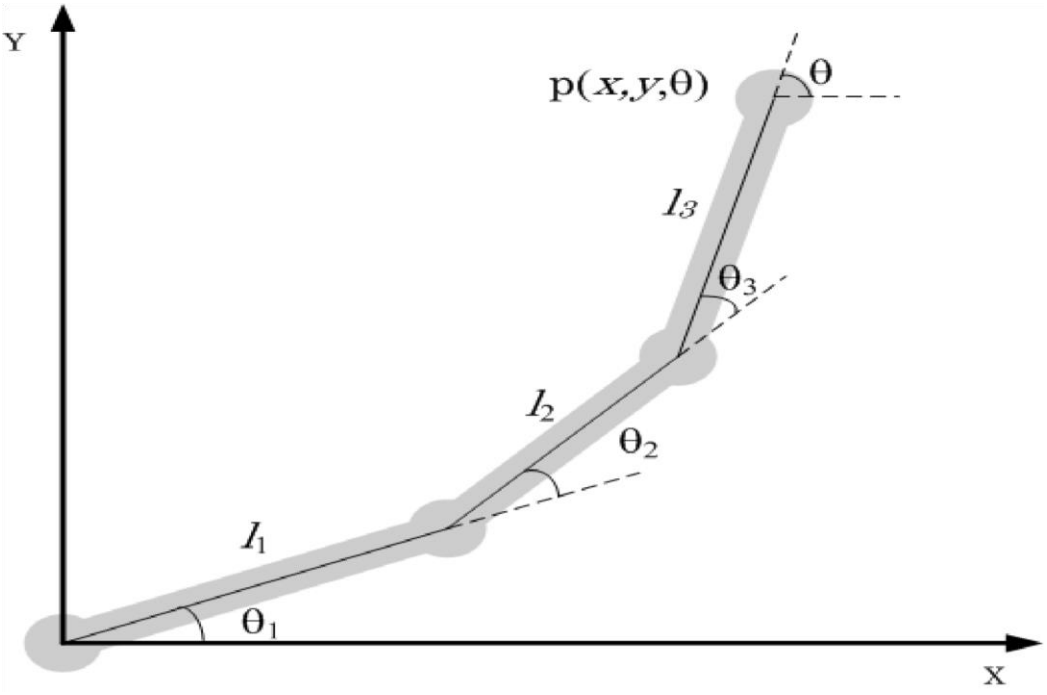


Figure 10: Forward Kinematics

The coordinates of the robot end-effector can be solved using the vector algebra solution to analyse the graph as follows:

x=l1cos(θ1)+l2cos(θ1+θ2)+l3cos(θ1+θ2+θ3)

y=l1sin(θ1)+l2sin(θ1+θ2)+l3sin(θ1+θ2+θ3) θ=θ1+θ2+θ3

3.12.2 Inverse Kinematics

Robotics, computer gaming, and animation all employ the robot's kinematics equations. Inverse kinematics is the process of computing joint parameters in order to obtain a specific position of the end-effector.
Inverse kinematics is the mathematical process of reconstructing an object's movements in the real world from other data, such as a video of those movements or a film of the world as viewed by a camera that is producing those movements. This is beneficial in robotics and animation.

In robotics, inverse kinematics uses the kinematics equations to find the joint parameters that give each of the robot's end-effectors the required position. Motion planning is the specification of a robot's movement in order for its end-effectors to perform the desired tasks. Inverse kinematics converts the robot's motion plan into joint actuator trajectories.

The kinematics equations of a kinematic chain, whether it be a robot or an animated figure, model the movement of the chain. These equations describe the chain's configuration in terms of joint parameters. Forward kinematics calculates the chain's configuration using joint parameters, while inverse kinematics reverses the calculation to find the joint parameters that produce the desired configuration.

One of the initial steps in the design of most industrial robots is kinematic analysis. The designer can use kinematic analysis to find out where each component in the mechanical system is located. This data, as well as control routes, are required for further dynamic analysis.

The kinematic analysis of a constrained system of rigid bodies, or kinematic chain, is an example of inverse kinematics. The loop equations of a complicated articulated system can be defined using the kinematic equations of a robot. These loop equations impose nonlinear limitations on the system's setup parameters. The degrees of freedom of the system are the independent parameters in these equations.

While analytical solutions to the inverse kinematics problem exist for a wide range of kinematic chains, Newton's approach is frequently used to solve nonlinear kinematics equations in computer modelling and animation software. Interactive manipulation, animation control, and collision avoidance are some of the other applications of inverse kinematic algorithms.

3.12.2.1 Approximating Solutions to IK Systems

Inverse kinematics problems can be modelled and solved using a variety of techniques. Due to the difficulties of inverting the forward kinematics equation and the potential of an empty solution space, the most flexible of these systems often use iterative optimization to find an approximation solution. The basic idea underlying several of these solutions is to use a Taylor series expansion to describe the forward kinematics equation, which is easier to invert and solve than the original system.

3.12.2.2 The Jacobian Inverse Technique

Inverse kinematics can be implemented using the Jacobian inverse technique, which is a basic yet effective method. Let the forward-kinematics equation, i.e. the position function, be governed by variables. Joint angles, lengths, and other arbitrary real numbers can be used as variables. The position function can be considered as a mapping if the IK system exists in three dimensions. Let be the system's beginning position, and let be the system's final position. The Jacobian inverse technique calculates an estimate that minimises the error iteratively.

3.12.2.3 Heuristic Methods

Heuristic approaches can also be used to approximate the Inverse Kinematics problem. These methods use simple iterative procedures to eventually arrive at a solution approximation. Heuristic methods are computationally efficient (they return the final pose rapidly) and usually handle joint constraints. Cyclic Coordinate Descent (CCD) and Forward and Backward Reaching Inverse Kinematics (FABRIK) are two of the most used heuristic methods.

3.12.3 The relationship between forward and inverse kinematics is below (in fig.9)

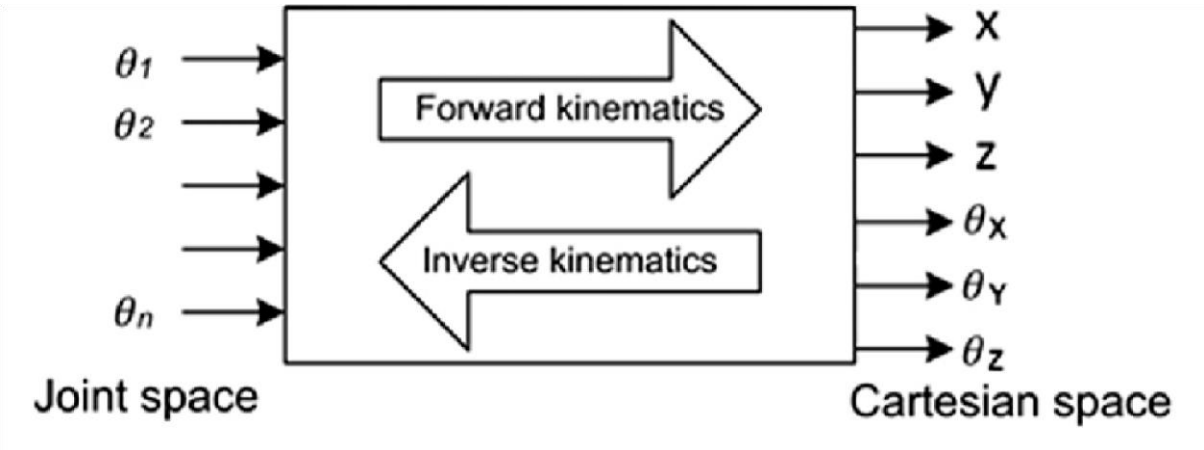


Figure: 11

KFast solves robot inverse kinematics equations analytically and produces optimised C++ files. The inverse kinematics equations are the result of attempting to locate the robot end effector coordinate system in the real environment while keeping joint and user-specified limitations in mind. Many different IK Types are made up of user-defined constraints, each with its own set of benefits based on the task.

The Robot Manipulator determines the number of joints in a chain that IKFast will work with. When a chain has more degrees of freedom (DOF) than the IK type requires, the user can assign arbitrary values for a subset of the joints until the total number of unknown joints equals the IK type's degrees of freedom.

Although creating hand-optimized inverse kinematics solutions for arms that can capture all degenerate instances is difficult, having closed-form IK speeds up numerous activities, including planning algorithms, and is therefore a must for most robotics researchers.

For motion planning, closed-form solutions are required for two reasons:

- Closed form solutions will always be faster than numerical inverse kinematics solvers. The ability to process hundreds of configurations per second is required of planners. Ikfast's closed-form code can generate solutions in the order of 41 microseconds! In comparison, most numerical solutions take 10 milliseconds or less (assuming good convergence).
- Because all solutions are computed, the null space of the solution set can be examined.
- Robots with arbitrary joint complexity, such as non-intersecting axes, can be handled.

Features

- Calculated all possible discrete solutions (can be up to 16).
- Generated C++ code independent of OpenRAVE or any other library.
- Automatically detects degenerate cases where 2 or more axes align and cause infinite solutions.
- Invalid solutions are detected by checking if square roots are given negative values or are sines and arc cosines are given inputs exceeding the [-1,1] range.
- All divide by zero conditions are automatically checked and handled.

3.12.4 IK Types

Inverse kinematics of the following categories are supported:

- **Transform6D** - 6D transformation achieved through end effector.
- **Rotation3D** - end effector reaches desired 3D rotation.
- **Translation3D** - The 3D translation of the end effector origin is achieved.
- **Direction3D** - The end effector coordinate system moves in the desired direction.
- **Ray4D** - ray on end effector coordinate system reaches desired global ray
- **Lookat3D** - direction on end effector coordinate system points to desired 3D position
- **Translation Direction5D** - The origin and direction of the end effector reaches the desired 3D translation and direction. It's similar to Ray IK in that the ray's origin must be the same.
- **Translation XY2D** - end effector origin reaches desired XY translation position, Z is ignored. The coordinate system with relative to the base link.
- **Translation LocalGlobal6D** - local point on end effector origin reaches desired 3D global point. Because both local point and global point can be specified, there are 6 values.

- **Translation XAxisAngle4D, Translation YAxisAngle4D, Translation ZAxisAngle4D** - end effector origin reaches desired 3D translation, manipulator direction makes a specific angle with x/y/z-axis (defined in the manipulator base link's coordinate system)
- **Translation XAxisAngleZNorm4D, Translation YAxisAngleXNorm4D, Translation ZAxisAngleYNorm4D** - end effector origin reaches desired 3D translation, manipulator direction needs to be orthogonal to z, x, or y axis and be rotated at a certain angle starting from the x, y, or z axis (defined in the manipulator base link's coordinate system)

CHAPTER 4: EXPERIMENTAL EXECUTION

4.1 Robotic Operating system

- **Installations**

The version is ROS 2 Foxy Fitzroy

Prerequisites: Ubuntu Linux - Focal Fossa (20.04)

Debian packages

- **Configure your Ubuntu repositories**

Allow "limited," "universe," and "multiverse" in your Ubuntu repositories.

- Make sure your Ubuntu package index is up-to-date:

sudo apt-get update

Figure 12

4.1.1 Installing ROS 2 via Debian Packages

- Set locale
- Setup Sources
- Install ROS 2 packages
- Environment setup
- Sourcing the setup script

Set locale

locale # check for UTF-8

sudo apt update && sudo apt install locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8

locale # verify settings

Setup Sources

The ROS 2 apt repositories must be added to your system. To accomplish so, use apt to authorise our GPG key as follows:

sudo apt update && sudo apt install curl gnupg2 lsb-release
sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg

And then add the repository to your sources list:

echo "deb [arch=**$(**dpkg --print-architecture**)** signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu **$(**source /etc/os-release && echo $UBUNTU_CODENAME**)** main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null

4.2 Install ROS 2 packages

Update your apt repository caches after setting up the repositories.

sudo apt update

Desktop Install (Recommended): ROS, RViz, demos, tutorials.

sudo apt install ros-foxy-desktop

ROS-Base Install (Bare Bones): Communication libraries, message packages, command line tools. No GUI tools.

sudo apt install ros-foxy-ros-base

Environment setup

Sourcing the setup script:

❖ Set up your environment by sourcing the following file

source /opt/ros/foxy/setup.bash

4.3 ROS2 Docmentation : Foxy

- Installations
- Core ROS2 Tutorials
- Tutorials for other ROS2 libraries
- About ROS2 interafces
- Using ROS2 on your custom robot

4.3.1 Core ROS2 Tutorials

- Configuring your ROS 2 environment
- Introducing turtlesim and rqt
- Understanding ROS 2 nodes Understanding ROS 2 topics
- Understanding ROS 2 services
- Understanding ROS 2 parameters

- Understanding ROS 2 actions
- Using rqt_console
- Introducing ROS 2 launch
- Recording and playing back data
- Creating a workspace
- Creating your first ROS 2 package
- Writing a simple publisher and subscriber (C++)
- Writing a simple publisher and subscriber (Python)
- Writing a simple service and client (C++)
- Writing a simple service and client (Python)
- Creating custom ROS 2 msg and srv files
- Expanding on ROS 2 interfaces
- Using parameters in a class (C++)
- Using parameters in a class (Python)
- Getting started with ros2doctor
- Creating and Using Plugins (C++)
- Creating an action
- Writing an action server and client
- (C++) Writing an action server and client (Python)
- Composing multiple nodes in a single process
- Using colcon to build packages

4.3.2 Create ROS2 Workspace

- mkdir –p robotic_arm_ws/src
- cd robotic_arm_ws
- colcon build colcon build creates build, log, install folders

4.4 Creating your first ROS 2 package

What is a ROS 2 package?

A package can be considered a container for your ROS 2 code. If you want to be able to install your code or share it with others, then you'll need it organized in a package. With packages, you can release your ROS 2 work and allow others to build and use it easily.

The build mechanism for ROS 2 package creation is ament, and the build tool is colcon. Although additional build types exist, you can construct a package using either CMake or Python, which are officially supported.

The command syntax for creating a new package in ROS 2 is:

- cd robotic_arm_ws/src
- ros2 pkg create --build-type ament_python big_bazu
- colcon build

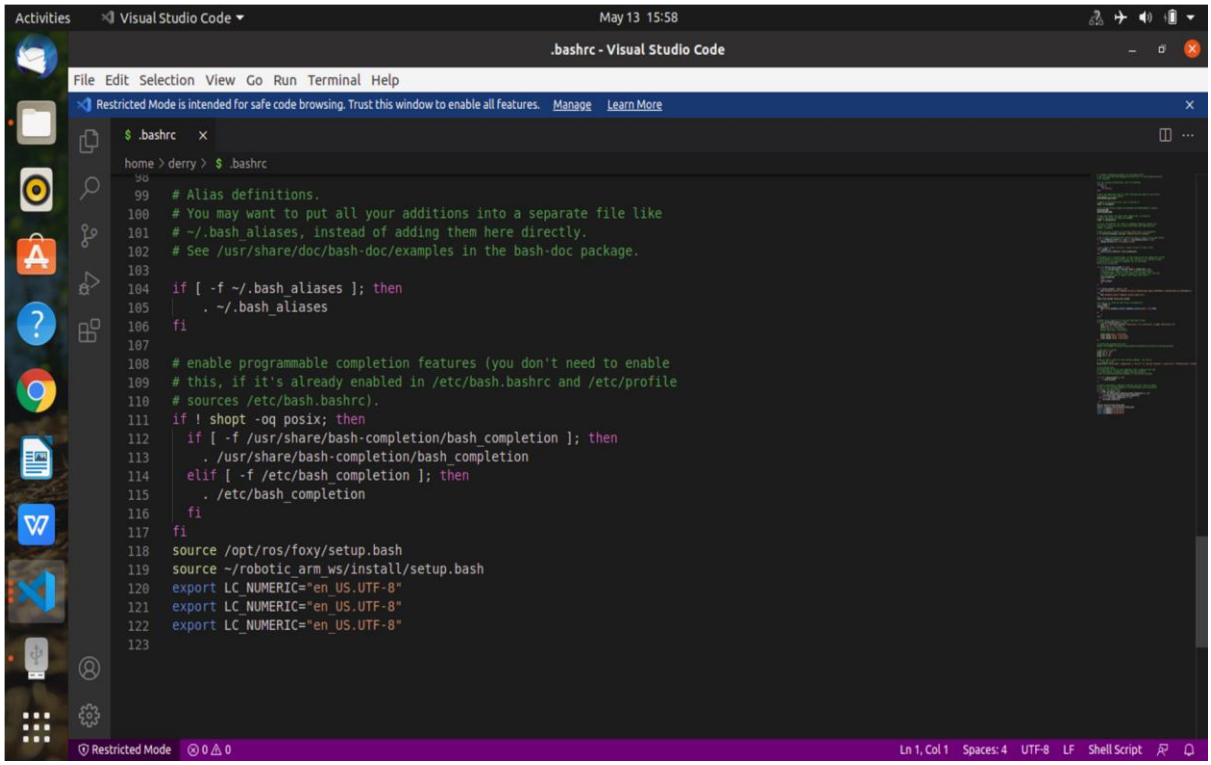
Building a workspace and sourcing the setup bash file inside install folder

- cd robotic_arm_ws/src
- source install/setup.bash

Now if we perform

- run ros2 run big_bazu

It automatically completely knows where this big_bazu package is.
So we are going to go the VScode and press ctrl ‘’O’’ and go to the home folder and pressing ctrl “H” to show the hidden files as well. Here we have the ‘’bashrc” file as shown below figure:



bashrc file

We also have the sourcing command line pasted in this file itself shown above.

4.5 Unified Robotics Description Format (URDF)

While designing our robotic arm we need to specify the basic structures and how each joint and link will be connected.

Transforms, states publishing-Robot State Publisher and joint state publisher.

There are two major things when we design a robot that is :

1) Joints
2) The links

In our case joints are the points where the motion happens. Links are the bodies that connects the joints

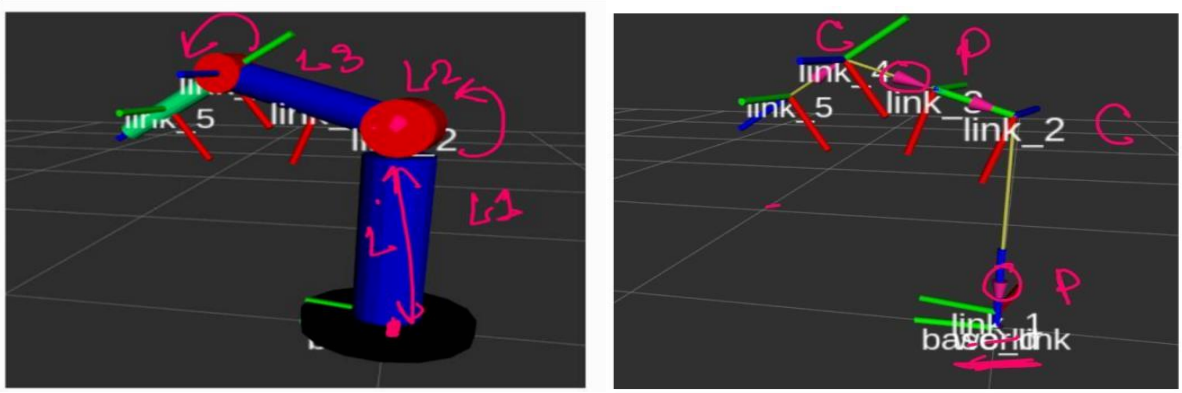


Figure 13: Links and Joints

The transform is the concept which is used to make the simulation understand the structure of the robot. For example we have the circular base on which we have this bar which is the link. The simulation doesn't recognise this, the transform is used to produce these structures for the robot. There is going to be a difference in X, Y, Z from base to the link number 1. And that is going to be defined through a transform. The base is on the origin i.e (0,0,0) on the X,Y,Z axis. And the base is at a value say (0,0,0.1) on the x, y, z. So this point a world to base we have a transform whenever we move the world the base is also going to move, with an offset of 0.1 on the z axis. This is known as the transform between the two things. Whole of the robot structure is produced with the strategy. The link number 1 is on the base and there is no difference in the origins. And we have link number 2, which is a joint and it can rotate along a certain axis. It is located at a point 0 in x axis, 0 in y axis and 1.2 in z axis. The 1.2 in z axis is from base. This a transform between link number 1 and link number 2. To move this world the base is also going to move. And the link number 2 is also going to be moved. Because we have a specific transform structure in between these links and joints.

So this is the basic concept of transforms which is utilized to keep the robot in a structure now coming to the point of actuated in non actuated joints. We have certain joints which are fixed, they are not moving. Those are known as non actuated. For example this base. Base is never going to be, move is never going to move. It is fixed with the world. You will see it is directly written. This joint is fixed. This link is fixed, so word and base are fixed. Although there is an offset but they are not going to move. In our case, this link number two here, this red circular. Joint is a rotational joint. It is rotating along this axis. Same goes for this one link number four. It is going to rotate this link #3 and link number one are fixed joints. They are not going to be actuated, actuated, being producing. So link two and link four are going to be rotating, so they are actuated and those blue bars are non actuated, they are fixed. Coming to the last thing, parent and child. Parent and child. In links and joints, these are going to create a structure for the robot to make the simulation understand what a robot looks like. Although this words are messed up, we'll see in our ways what actually they are saying. You can see there is some sort of an arrow coming from each of a point to another now. You can see from link #4 to link #3 this arrow is going outside of four and going inside or two a point in link #3 this is representing that link #3 is actually the parent and link number 4 is the child. So we have to define the structure of the robot in a parent child relation. Similarly, we have a lot of things at this point, Base link world and link number one all are written at the same point because there is very small offset. We'll look into link number 2, link #2 is child because this error is coming out of link number 2 to link number one. So link number one. Is base and for link number one base is parent. And for base world is parent. And so that's how a structure has been created and which gives a meaning to your robots definition for Weather simulation.

Link number one has been created in link number one we have a specified geometry and this geometry is of length and radius distribute this bar has been created. rgba It is a blue and is for a parent, but it is a blue bar. Of this length and this radius. Origin and collision values are same as the visual values of each geometric. Joint1 it is a continues joint, parent is base link, child is link number one and there is a. Offset in the XYZ. In link number two, it is again a cylinder of this length and this radius it is of red color. Link #2. Inertia values are given. Visual cylinder, radius, color red. And the length of the cylinder and radius are provided. Collision is the same as of the visual tag geometric. Joint is again a continues. Parent is link number one, child is link number 2.

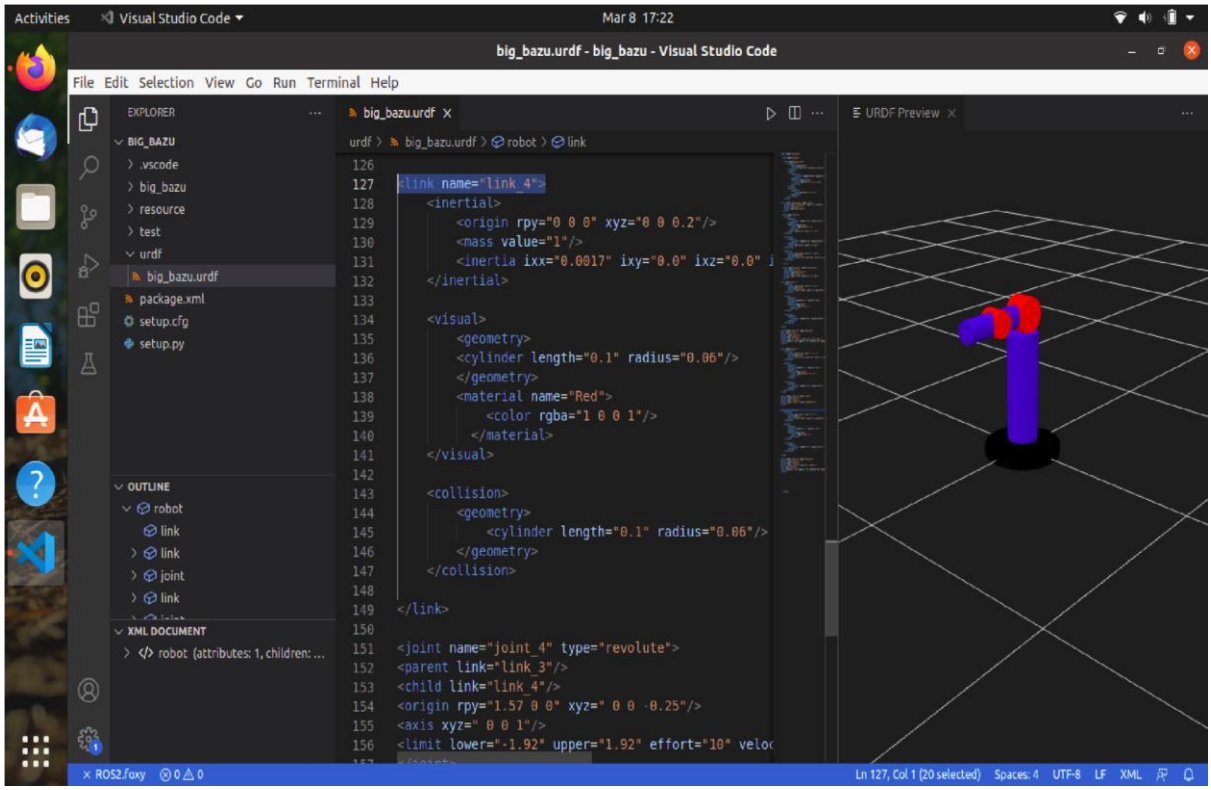


Figure 14: URDF Preview

One thing to note here is this is a link. This is a link. This blue or is also a link and we have this huge bar as well as a link. These two red dots are red circles cylinders points with rotation is going to happen. Similarily all the joints are described in urdf file.

4.6 RViz simulation for joints test

To launch our robot it in rviz. create a folder Launch and in launch folder create a new file named as rviz.launch.py. We have a basic structure in which some things are imported and there is python syntax followed a function and it contains some lines and then returns these nodes to run another variable in this we have share directory in which we have a our big_bazu package .we access all of the package resources through the share folder in a workspace where packages are residing.

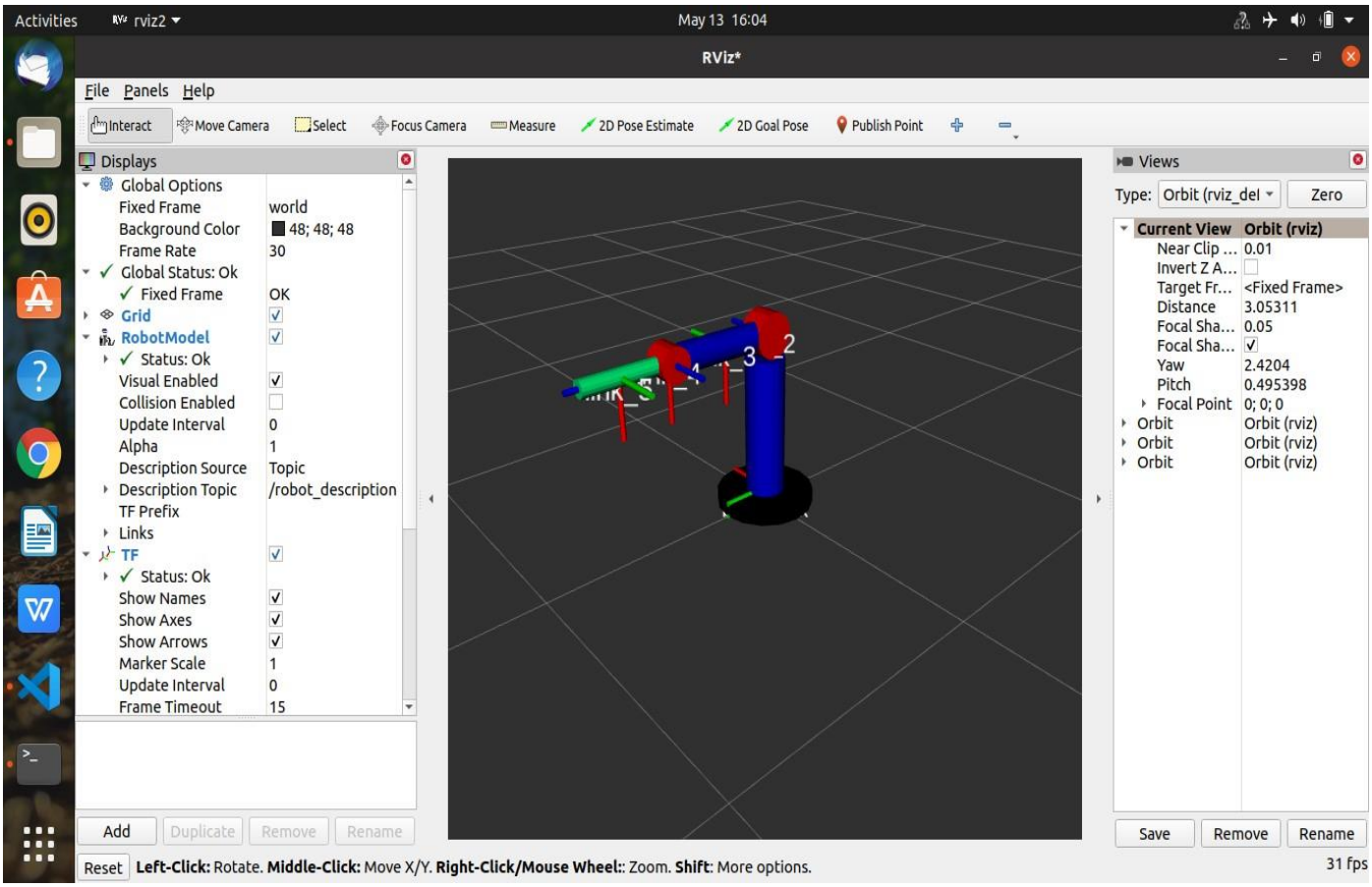


Figure 15: RViz Simulation

4.7 Setup.py file

The setup file is going to have all of the essential paths to which the file should be copied in the share folder here we import some libraries, import OS and from globe import globe These are going to be required to copy the folders and files both from this package into the shared folder of this big bazu.

Here we give paths of launch and urdf folder. We copy all the contents from this launch folder in a share folder and we do the same for the urdf folder. The transforms joint_state_publisher and robot _state _publisher these are basically the basic nodes of packages.

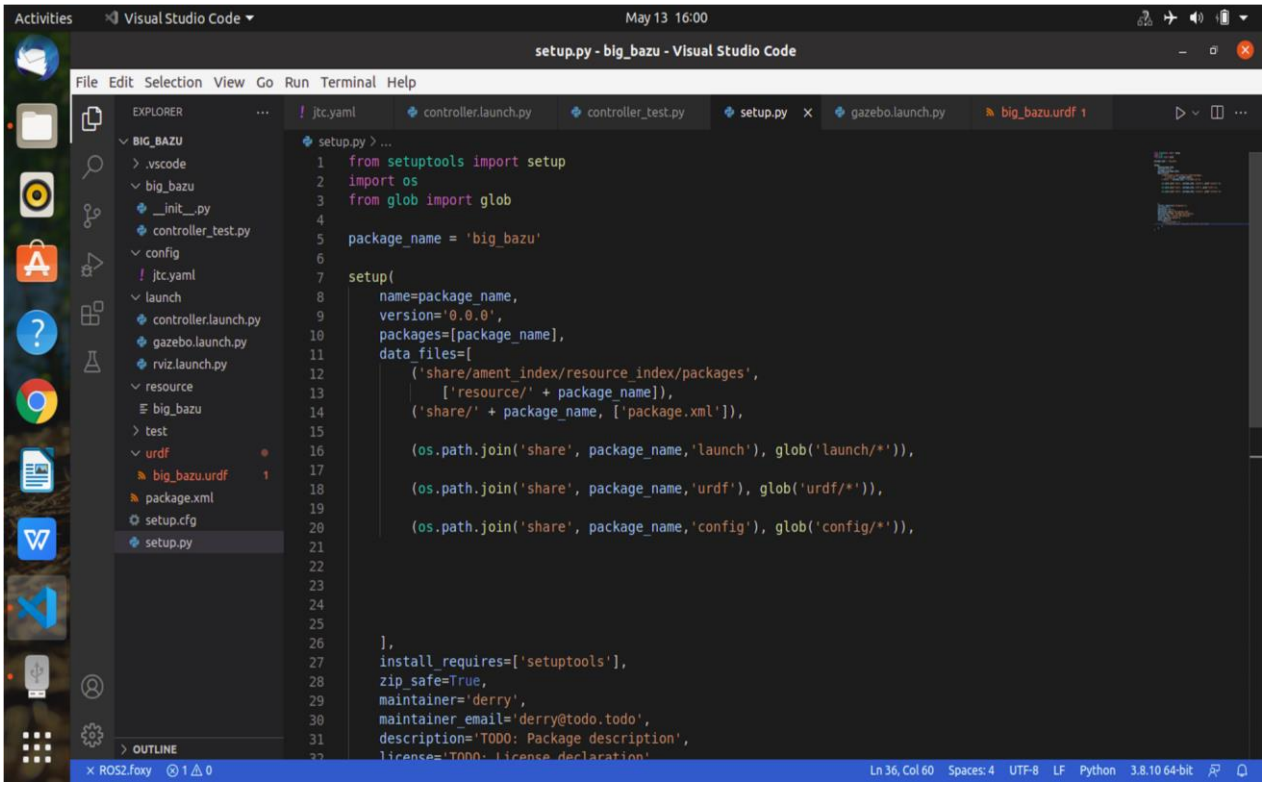


Figure 16: setup.py

Command to launch rviz.launch.py for joints test

Prerequisites:

- sudo apt-get install ros-foxy-joint-state –publisher-gui
- sudo apt-get install ros-foxy-robot-state –publisher-gui

ros2 launch big_bazu rviz.launch.py

4.8 Gazebo and Physical properties

- Gazebo <>ROS
- Spawning a model process
- Forces action on Robot

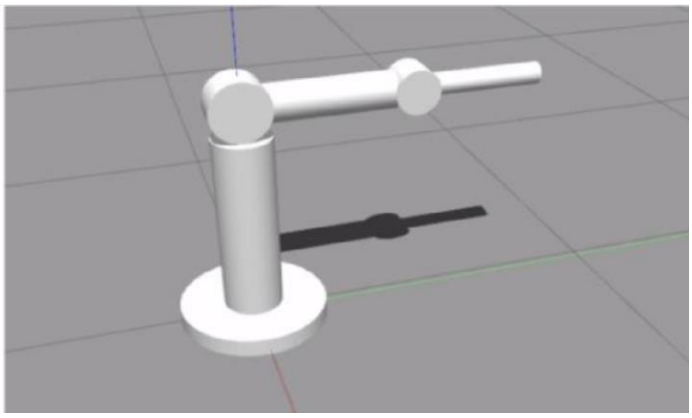


Figure 17: Robotic Arm in Gazebo

Gazebo is known to be very near to real-world simulation for one small reason. Because it produces simulation for the forces as well. To bring the robot in gazebo simulation. We need to go through certain steps. The first step is Gazebo and ROS talking with each other. This is the most essential one because we want all of our systems, all of our simulation, talking with ROS, Gazebo is independent Software, we make Gazebo and Ross talk to each other. So we can have a communication with nodes, services, launch files all of the ROS implementations. So we need make them talk with gazebo First, we performed this thing. The second thing is we make this robot model to be spawned. Inside of gazebo simulation.

4.8.1 Pipeline of launching Gazebo

We created a new file in launch folder gazebo.launch.py here we have some important and required imports and libraries. In generate_launch_description file we obtained the package and URDF file from the shared folder of our package.

ExecuteProcess: It actually executes a command **gazebo","-s","libgazebo_ros_factory.so** this basically deals with the communication between gazebo and ROS so it initializes that communication that we need for this ecosystem and gazebo ecosystem and we need to make them communicate.

executable="spawn_entity.py": This is a spawn entity ,which spawns a robot into the gazebo

executable command: **ros2 run gazebo_ros spawn_entity.py -entity big_bazu file/home/derry/robotic_arm_ws/src/big_bazu/urdf/big_bazu.urdf**

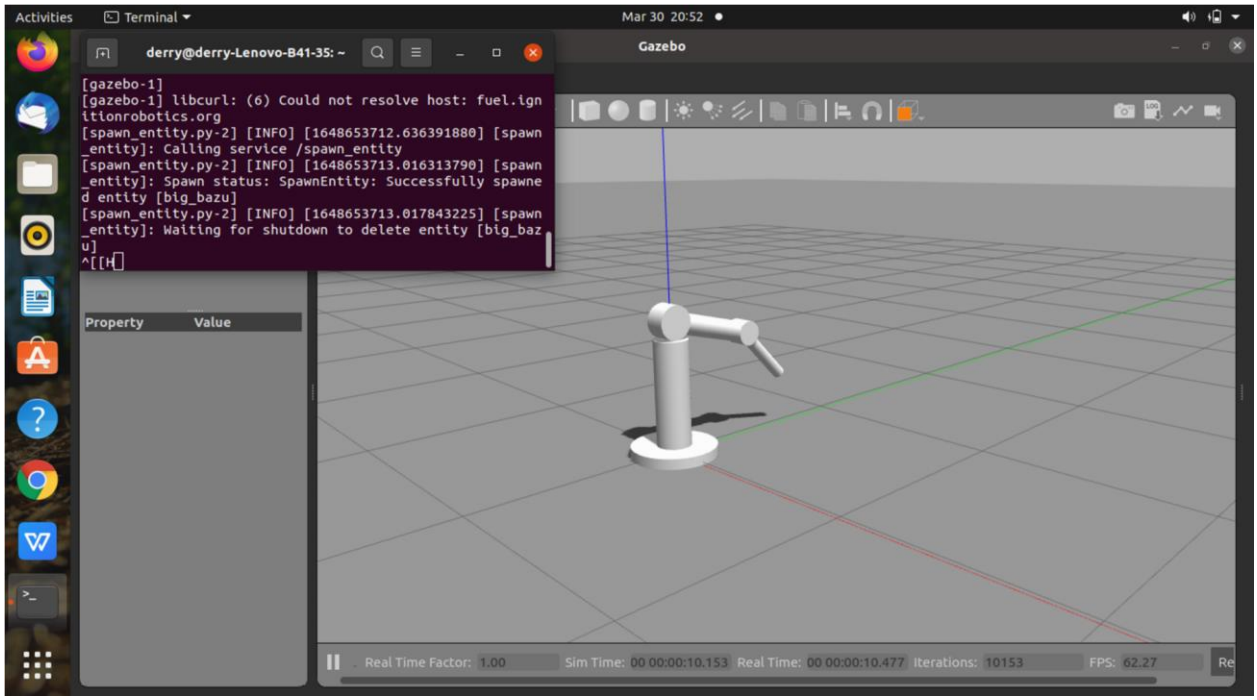


Figure 18: Gazebo Simulaton

4.9 Calculating Inertia for robots link

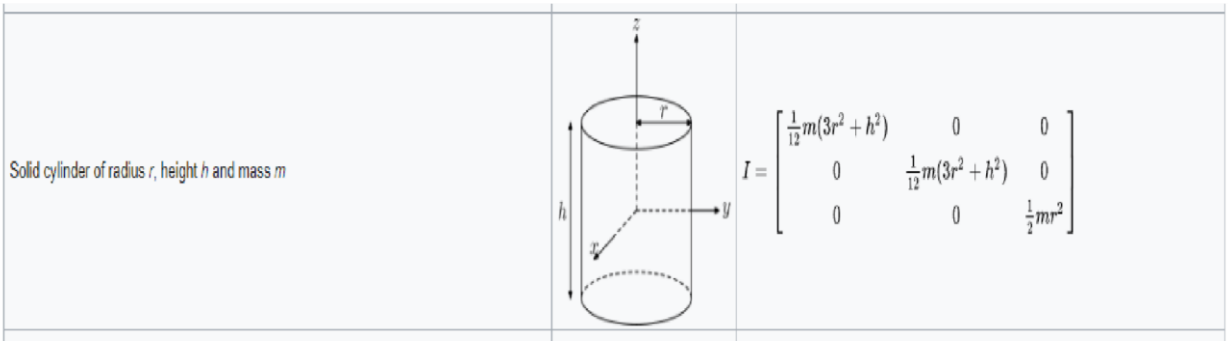


Figure 19: Formula for calculating Inertia

1/12 * m (3r^2 + h^2) .... 1/12*mr^2

executable command: **ros2 launch big_bazu gazebo.launch.py**

4.10 Introduction to ROS Control Stack

- ROS2 Control

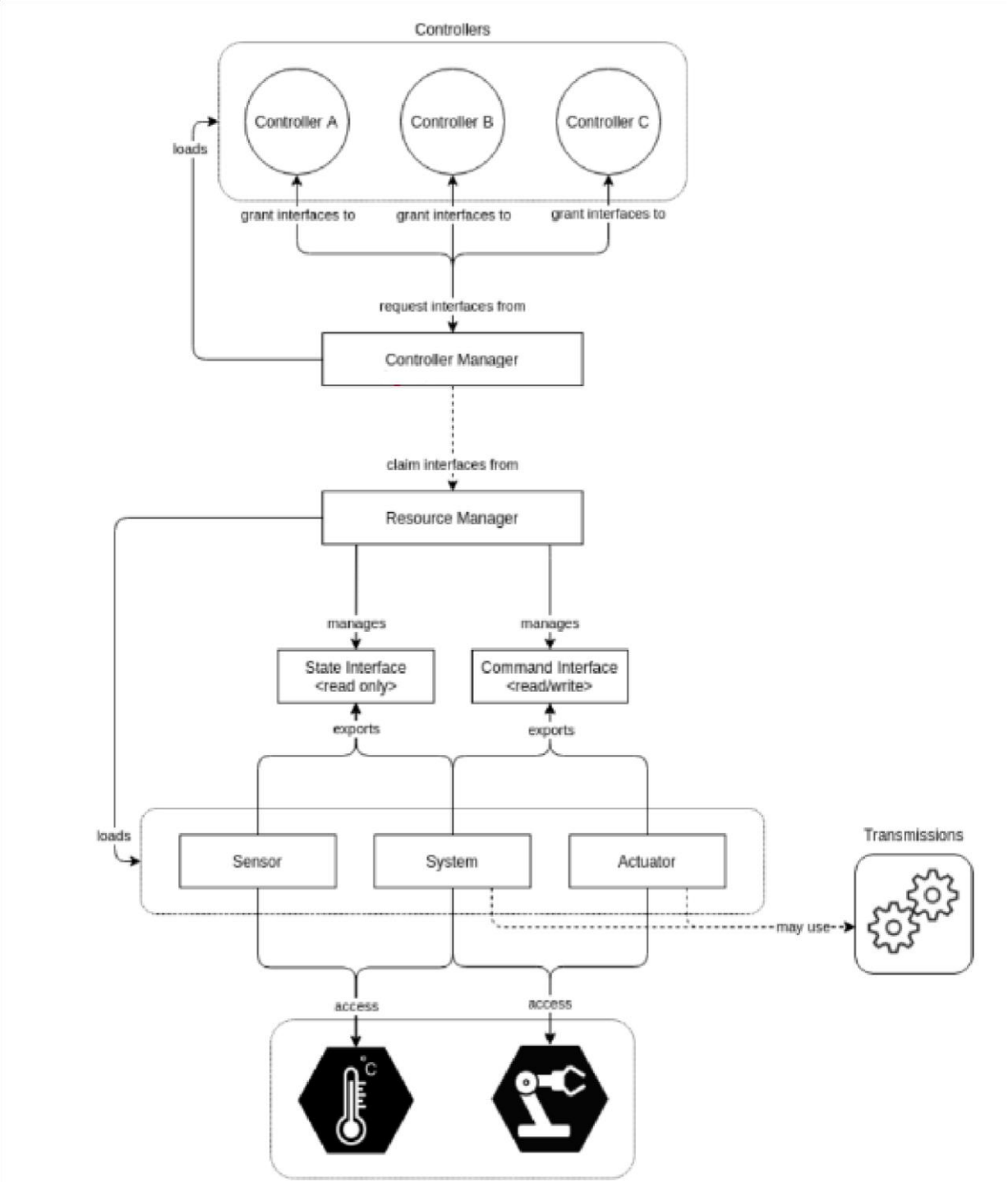


Figure 20: ROS2 control

To make a robot stand still and make it move accordingly with our requirements in gazebo, we need to have control. ROS2 control comes into play when we are dealing with controllers in our robot. We need to first understand and look into what the basic architect of ROS2 control is. First we have the controllers there can be multiple types of controllers. First we have controllers that talks with controller manager. So controller manager is in the middle talking from a software script which is a controller with the robot. Resource manager comes after controller manager. Controller manager was the main deal.

Resource manager loads of all of the components using plugins, libraries and manages their lifecycle. Basic definition of resource manager. It is not going to come up. Explicitly as controller manager, but on the back end it is going to be doing a lot of stuff. Resource manager now deals with two important things. Command interfaces and straight interfaces. Command interfaces are dependent on the controller type here utilizing, state interfacing are read only we can get information in whatever type view point.

4.10.1 Types of Controllers

1. Positon controller
2. Acceleration controller
3. Velocity controller
4. Joint Trajectory controller

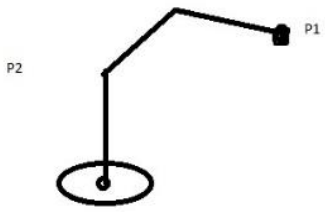


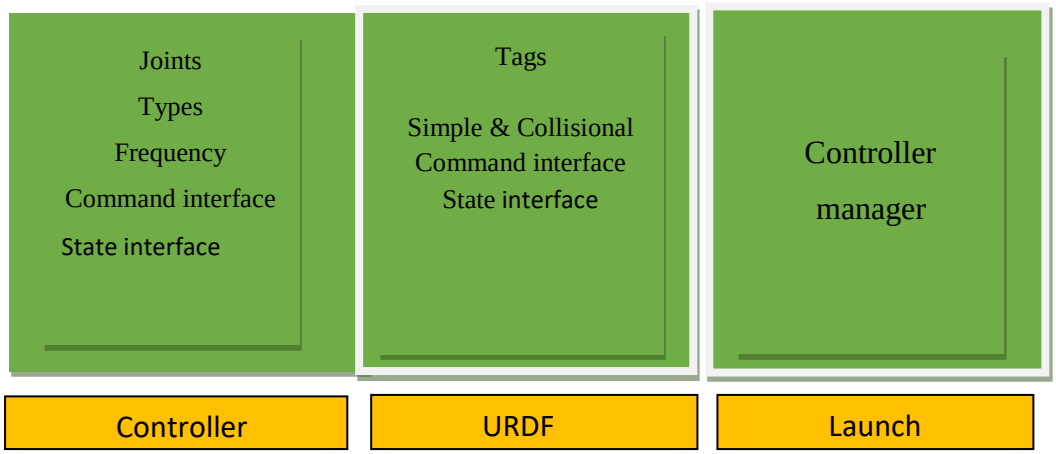
Figure 21

In ROS, we need to control the robot we need controllers they're simply one single script. The position controller. As the name suggests, we need to distinguish between them. Position controller is going to be dealing with just the positions if we will tell 30 degree of joint one it will make the joint one to move to 30 degree. Acceleration and velocity are same and will be more useful for differential type.

This point P1, this end effector point, we want it to come to point number P2. so this motion all of these joints which are included in the robot are going to be included into the motion and make the robot go from point P1 to P2 this done with very easy controller, with joint trajectory controller that will just tell the

position and hold the robot joints and adjust themselves to move to the point number P2. And everything of such motion is done by joint trajectory controller

4.10.2 How to install a controller?



We described it in three terms. The first part of this installation is writing the controller and that controller is written in a file that has a very unique extension YAML. Controller is defined inside of this file in which we define the joints of our robot, then the type and frequency at which you want to publish and command interfaces and the state interfaces. These are the things that we dealt with when we defined a controller. It is not very complex, it's just one script.

The second thing is to be done inside of a URDF. ROS control tags are used like simple inertia and Collision tags that we used while building the URDF. These control tags are going to contain a lot of properties for our controllers, include the Yaml file to define the joints. And command interfaces and state interfaces are

defined inside of these control tag and that is implemented. The last part is most important one it is of launch file, launch file contains very essential things we have controller manager So all of these three things are very essential and necessity to make controller work on our robot .

4.10.3 Installing Joint Trajectory Controller

Prerequisites:

- sudo apt-get install ros-foxy-controller-manager
- sudo apt-get install ros-foxy-joint-trajactory-controller
- sudo apt-get install ros-foxy-controller-manager

Executable Command: **ros2 launch big_bazu controller.launch.py**

CHAPTER 5: RESULT AND CONCLUSION

Result:

After successfully assembling the 3D printed robotic arm parts. We started working on the programming of the servo motors for each joint. We used the Arduino mega 2560 board and connected an external power supply for the servos.

Than we successfully designed a program in Arduino Ide to move each servo for a fixed angle so that the robotic arm can move and place an object and return to its original location. Rviz and Gazebo tools were used for simulation and path planning. The similar concept can be applied to arms with greater degrees of freedom.

Conclusion:

The purpose of the study was to design a 5DOF robotic arm that has the capability to pick and place a lightweight object from one location to another location using specific coding methods and by using the URDF file to define the movements of the robotic arm to get accurate position on where to lift the object and place it.

From this project we conclude that we can use the robotic arm not just for games but we have used it for controlling different types of machines. During the period of the software of the project we found that the processing language (python) have made a revolution in the technology domain. This project is a prototype of an immense number of projects that we can use it in a lot of purposes like military, medical, social, educational, and industrial and we can use it for people with special needs.

We also conclude that we merge more than programming language to achieve some processes to run the project as we have done in our project, we have merged the processing program (python) and the output from gazebo with Arduino program (C) by using serial port .

Additionally, the forward and inverse kinematics analysis were carried out. Simulation and path planning were executed using Rviz software. The simplified graphical approach enables the controlling of the arm using Arduino mega board, since complex matrix operation involved in finding out the inverse kinetics are being eliminated. The method can be used to arms with higher degrees of freedom.

Chapter 6:
Future Work

- Robotic Arms has a wide scope of development. In the near future the arms will be able to perform every task as humans and in much better way. Imagination is the limit for its future applications.
- It can be a real boon for handicapped people, who are paralyzed or lost their hands in some accident. The arm can be trained to listen to the command from a human and perform that task. A Precise gesture controlled system is also possible. Wearable devices can be used to send the command and control the movements of the arm.
- Brain Computer Interface (BCI) is an immerging field of research. BCI can be used to acquire signals from the human brain and control the arm. The system can work in the same way as human arm. A person who may have lost his hand in any accident can resume his life like previous by such artificial arms. Robotic arms are versatile and have enormous ways of implementations.
- The benefits of using a robotic arm, system or device during surgery will drastically change and impact how surgeons run their operating rooms in the future.
- "By extending human surgeons' ability to plan and carry out surgical interventions more accurately and less invasively, surgical robotic systems can address a vital international need to greatly reduce costs, improve clinical outcomes, and improve the efficiency of health care delivery," notes the Robotics and Automation Society.
- Essentially, robotic arms are capable of assisting in surgeries that were once seen as risky or complicated. Robots are beneficial because they are consistently more accurate and steady as compared to a human hand.

Chapter 7:

Bibliography

Paper Referred:

[1] Dynamic Modeling And Simulation Of Articulated Robotic Arm With MATLAB Robotics System Toolbox

Khin Soe New , Wai Phyomaung, Ei Eihtwe Student, Lecture, Professor Department Of Mechanical Engineering, Mandalay Technology University, Mandalay,Myanmar

[2] Intuitive And Adaptive Robotic Arm Manipulation Using The Leap Motion Controller

D. Bassily, C. Georgoulas, J. Güttler, T. Linner, T. Bock, Tu München, Germany

[3] Survey On Robotic Arm Controlling Technique

V.Priyanka Dept Of ECE, PSNA College Of Engineering And Technology, Dindigul, Tamil Nadu, India. E.Thangaselvi Dept Of ECE, PSNA College Of Engineering And Technology, Dindigul, Tamil Nadu, India

[4] Design And Development Of Inverse Kinematics Based 6 Dof Robotic Arm Using Ros

Rajesh Kannan Megalingam, Vinu Sivanantham, K Sai Kumar, Sriharsha Ghanta, , Kerala, India

[5] Solving Kinematics Problems Of A 6-Dof Robot Manipulator

Alireza Khatamian Computer Science Department, The University Of Georgia, Athens, Ga, U.S.A

[6] Robotic Arm Control Through Human Armmovement Using Accelerometers Department Of Electronics And Communication Engineering, National Institute Of Technology, Rourkela

[7] Design And Development Of 6-Dof Robotic Arm Controlled By Man Machine Interface Sulabh Kumra1 , Rajat Saxena , Shilpa Mehta Department Of Electronics And Instrumentation Engineering, Itm University, Gurgaon, India

[8] Neural Network Based Inverse Kinematics Solution For Trajectory Tracking Of A Robotic Arm
Petru Maior University Of Tg. Mureş, No. 1 N.Iorga St., Tg.Mureş, 540088, Romania

[9] 3d Modelling And Simulation Of A Crawler Robot In Ros/Gazebo Sokolov M., Lavrenov R., Gabdullin A., Afanasyev I., Magid E. Kazan Federal University, 420008, Kremlevskaya 18, Kazan, Russia

[10] Design Analysis Of A Remote Controlled “Pick And Place” Robotic Vehicle B.O.Omijeh,R.Uhunmwangho,M.Ehikhamenle Department Of Electrical/Electronic Engineering,University Of Port Harcourt, Port Harcourt

[11] Modeling And Analysis Of A 6 DOF Robotic Arm Manipulator Canadian Journal on Electrical and Electronics Engineering Vol. 3, No. 6, July 2012
Jamshed Iqbal, Raza ul Islam, and Hamza Khan

[12] Forward Kinematics Of A Robotic Arm Using ROS And Rviz With Moveit
Shubham Tiwari1 Prof. M. Shakebuddin
Department of Mechanical Engineering Anjuman College of Engineering and Technology, India

[13] Design And Simulation Of Vision Based Robotic Arm Using Ros
Vishwa Priya V, Srinivasan V, Dakshineshkar N R, Thamesh N R, Sanjayakumaran S, Vikram R
Robotics and Automation Engineering PSG College of Technology Coimbatore, India

[14] Development And Control Of Modular 5 Dof robotic ARM Using ROS
Saksham Sangwan Department of Electrical Engineering Delhi Technological University New Delhi, India

[15] Control And Benchmarking Of A 7-DOF Robotic Arm Using Gazebo And ROS
Bowei Zhang, Pengcheng Liu March 23, 2021

[16] Keyboard-Based Control And Simulation Of 6-DOF Robotic Arm Using ROS
Conference: 2018 4th International Conference On Computing Communication And Automation (ICCCA)

[17] Integrating Soft Robotics With ROS: A Hybrid Pick And Place Arm*
Ross M. Mckenzie,
Thomas W. Barraclough, And Adam A. Stokes, Member, IEEE

[18] A ROS/Gazebo Based Method In Developing Virtual Training Scene For Upper Limb Rehabilitation Zhijiang Du, Yixuan Sun, Yanyu Su,

[19] Design And Implementation Of An ROS Based Autonomous Navigation System
Proceedings of 2015 IEEE International Conference On Mechatronics And Automation.

[20] Introduction to Robotics Mechanics and Control
Third Edition John J. Craig.

Website reffered

https://docs.ros.org/en/foxy/index.html

https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html

https://docs.ros.org/en/foxy/Tutorials/Configuring-ROS2-Environment.html

https://docs.ros.org/en/foxy/Tutorials/Understanding-ROS2-Nodes.html

https://docs.ros.org/en/foxy/Tutorials/Launch/CLI-Intro.html

https://docs.ros.org/en/foxy/Tutorials/Workspace/Creating-A-Workspace.html

https://docs.ros.org/en/foxy/Tutorials/Creating-Your-First-ROS2-Package.html

https://docs.ros.org/en/foxy/Tutorials/Launch/Creating-Launch-Files.html

https://docs.ros.org/en/foxy/Tutorials/URDF/Building-a-Visual-Robot-Model-with-URDF-from-Scratch.html

https://docs.ros.org/en/foxy/Tutorials/URDF/Adding-Physical-and-Collision-Properties-to-a-URDF-Model.html

https://docs.ros.org/en/foxy/How-To-Guides/Launch-file-different-formats.html

Chapter 8: Appendix

8.1 Arduino

8.1.1 Arduino Mega Specifications

The specifications of Arduino Mega include the following:

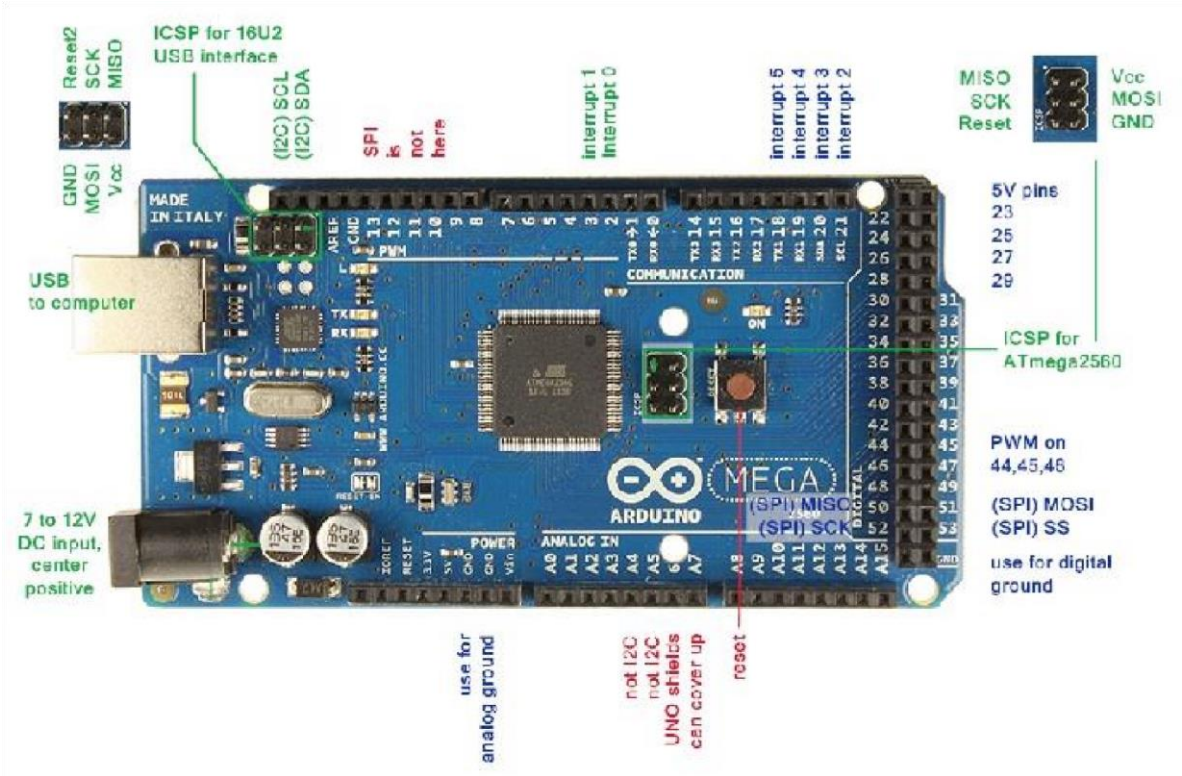
- The ATmega2560 is a Microcontroller
- The operating voltage of this microcontroller is 5volts
- The recommended Input Voltage will range from 7volts to 12volts
- The input voltage will range from 6volts to 20volts
- The digital input/output pins are 54 where 15 of these pins will supply PWM o/p.
- Analog Input Pins are 16
- DC Current for each input/output pin is 40 mA
- DC Current used for 3.3V Pin is 50 mA
- Flash Memory like 256 KB where 8 KB of flash memory is used with the help of bootloader
- The static random access memory (SRAM) is 8 KB
- The electrically erasable programmable read-only memory (EEPROM) is 4 KB
- The clock (CLK) speed is 16 MHz
- The USB host chip used in this is MAX3421E
- The length of this board is 101.52 mm
- The width of this board is 53.3 mm
- The weight of this board is 36 g

ARDUINO SPECIFICATIONS

Microcontroller	**ATmega 2560**
Operating Voltage	**5 V**
Input Voltage (recommended)	**7-12 V**
Input Voltage (limit)	**6-20 V**
Digital I/O Pins	**54 (of which 15 provide PWD output)**
Analog Input pins	**16**
DC Current per I/O Pin	**20 mA**
DC current for 3.3V Pin	**50 mA**
Flash Memory	**256 KB of which 8 KB used by bootloader**
SRAM	**8 KB**
EEPROM	**4 KB**
Clock Speed	**16 MHz**
LED_BUILTIN	**13**
Length	**101.52 mm**
Width	**53.3 mm**
Weight	**37 g**

8.1.2 Arduino Mega Pin Configuration

The pin configuration of this Arduino mega 2560 board is shown below. Every pin of this board comes by a particular function which is allied with it. All analog pins of this board can be used as digital I/O pins. By using this board, the Arduino mega projected can be designed. These boards offer flexible work memory space is the more & processing power that permits to work with different types of sensors without delay. When we compare with other types of Arduino boards, these boards are physically superior.



Pin configuration

1) Pin 3.3V & 5V

These pins are used for providing o/p regulated voltage approximately 5V. This RPS (regulated power supply) provides the power to the microcontroller as well as other components which are used over the Arduino mega board. It can be attained from Vin-pin of the board or one more regulated voltage supply-5V otherwise USB cable, whereas another voltage regulation can be offered by 3.3V0-pin. The max power can be drawn by this is 50mA

2) GND Pin

The Arduino mega board includes 5-GND pins where one of these pins can be used whenever the project requires.

3) Reset (RST) Pin

The RST pin of this board can be used for rearranging the board. The board can be rearranged by setting this pin to low.

4) Vin Pin

The range of supplied input voltage to the board ranges from 7volts to 20volts. The voltage provided by the power jack can be accessed through this pin. However, the output voltage through this pin to the board will be automatically set up to 5V.

5) Serial Communication

The serial pins of this board like TXD and RXD are used to transmit & receive the serial data. Tx indicates the transmission of information whereas the RX indicates receive data. The serial pins of this board have four combinations. For serial 0, it includes Tx(1) and Rx (0), for serial 1, it includes Tx(18) & Rx(19), for serial 2 it includes Tx(16) & Rx(17), and finally for serial 3, it includes Tx(14) & Rx(15).

6) External Interrupts

The external interrupts can be formed by using 6-pins like interrupt 0(0), interrupt 1(3), interrupt 2(21), interrupt 3(20), interrupt 4(19), interrupt 5(18). These pins produce interrupts by a number of ways i.e. providing LOW value, rising or falling edge or changing the value to the interrupt pins.

7) LED

This Arduino board includes a LED and that is allied to pin-13 which is named as digital pin 13. This LED can be operated based on the high and low values of the pin. This will give you to modify the programming skills in real time.

8) AREF

The term AREF stands for Analog Reference Voltage which is a reference voltage for analog inputs

9) Analog Pins

There are 16-analog pins included on the board which is marked as A0-A15. It is very important to know that all the analog pins on this board can be utilized like digital I/O pins. Every analog pin is accessible with the 10-bit resolution which can gauge from GND to 5 volts. But, the higher value can be altered using AREF pin as well as the function of analog Reference ().

10) I2C

The I2C communication can be supported by two pins namely 20 & 21 where 20-pin signifies Serial Data Line (SDA) which is used for holding the data & 21-pin signifies Serial Clock Line (SCL) mostly utilized for offering data synchronization among the devices

11) SPI Communication

The term SPI is a serial peripheral interface which is used to transmit the data among the controller & other components. Four pins like MISO (50), MOSI (51), SCK (52), and SS (53) are utilized for the communication of SPI.

12) Dimensions

The dimension of Arduino Mega 2560 board mainly includes the length as well as widths like 101.6mm or 4 inch X 53.34 mm or 2.1 inches. It is comparatively superior to other types of boards which are accessible in the marketplace. But, the power jack and USB port are somewhat expanded from the specified measurements.

13) Shield Compatibility

Arduino Mega is well-suited for most of the guards used in other Arduino boards. Before you propose to utilize a guard, confirm the operating voltage of the guard is well-suited with the voltage of the board. The operating voltage of most of the guards will be 3.3V otherwise 5V. But, guards with high operating voltage can injure the board. In addition, the distribution header of the shield should vibrate with the distribution pin of the Arduino board. For that, one can connect the shield simply with the Arduino board & make it within a running state.

14) Memory Specifications:

The ATmega2560 includes 256 KB of flash memory, 8 KB of SRAM, and 4 KB of EEPROM for storing code (of which 8 KB is utilised for the bootloader) (which can be read and written with the EEPROM library).

8.2 Rotary Encoder (KY-040)

8.2.1 KY-040 Pin Outs

The pin outs for this rotary encoder are identified in the illustration below

The module is designed so that a low is output when the switches are closed and a high when the switches are open.

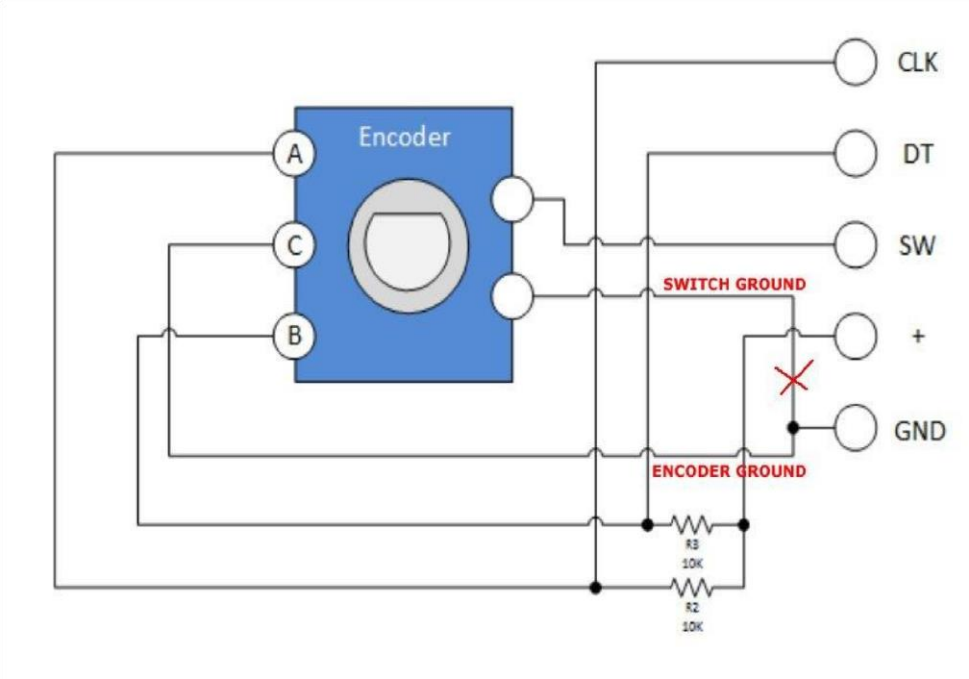
The low is generated by placing a ground at Pin C and passing it to the CLK and DT pins when switches are closed.

The high is generated with a 5V supply input and pull-up resistors, such that CLK and DT are both high when switches are open.

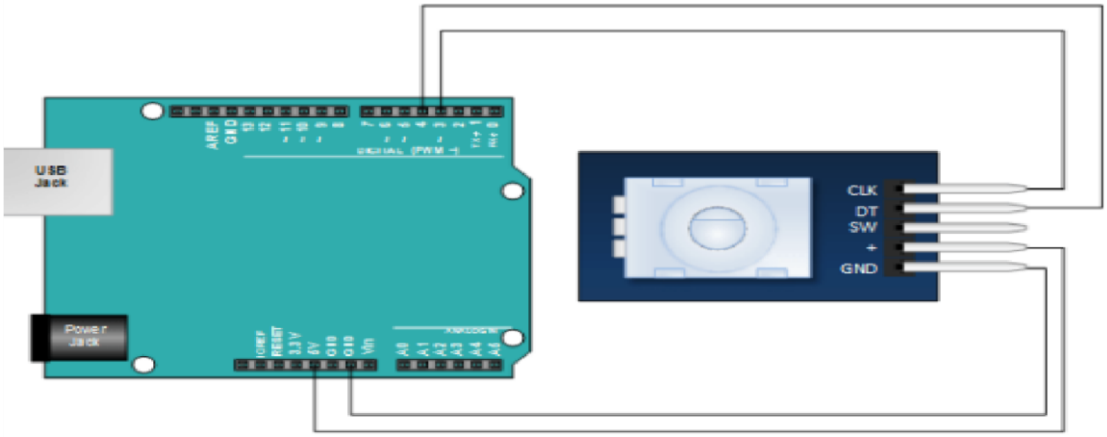
Not previously mentioned is the existence of push button switch that is integral to the encoder. If you push on the shaft, a normally open switch will close. The feature is useful if you want to change switch function. For example, you may wish to have the ability to adjust between coarse and fine adjustments.

8.2.2 Keyes Rotary Encoder Schematic:

A schematic for this module is provided below. R2 and R3 in the schematic are pull up resistors.



8.2.3 Module Connection to the Arduino:



8.3 Motor

8.3.1 SG90 MICRO 9G:

Specifications:

- Weight: 9 g
- Dimension: 22.2 x 11.8 x 31 mm approx.
- Stall torque: 1.8 kgf·cm
- Operating speed: 0.1 s/60 degree
- Operating voltage: 4.8 V (~5V)
- Dead band width: 10 μs
- Temperature range: 0 ºC – 55 ºC

Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is all the way to the left. ms pulse) is all the way to the right, ""-90" (~1ms pulse) is all the way to the left.

Basic Information

Modulation: Analog
Torque: 4.8V: 25.0 oz.-in (1.80 kg-cm)
Speed: 4.8V: 0.10 sec/60°
Weight: 0.32 oz. (9.0 g)

Dimensions:
Length: 0.91 in (23.1 mm)
Width: 0.48 in (12.2 mm)
Height: 1.14 in (29.0 mm)

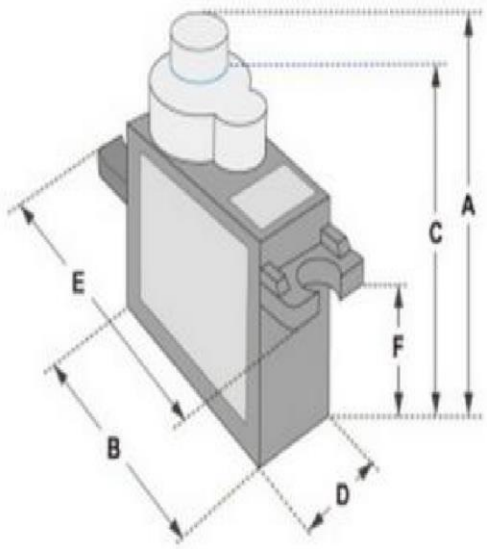
Motor Type: 3-pole

Gear Type: Plastic

Rotational Range: 180°

Pulse Cycle: ca. 20 Ms

Pulse Width: 500-2400 µs

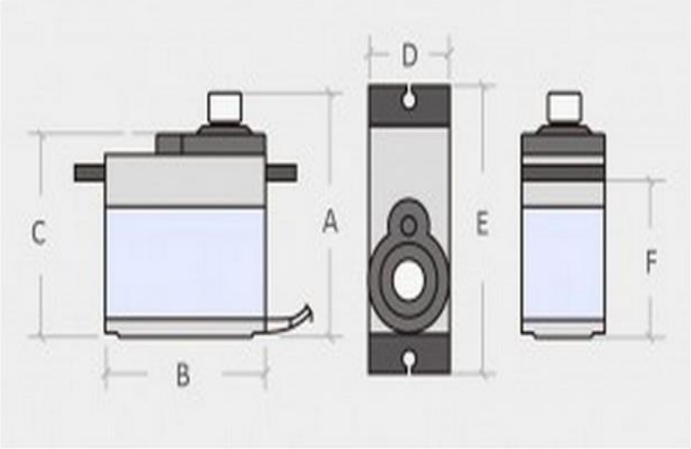


Dimensions & Specifications
A (mm) : 32
B (mm) : 23
C (mm) : 28.5
D (mm) : 12
E (mm) : 32
F (mm) : 19.5
Speed (sec) : 0.1
Torque (kg-cm) : 2.5
Weight (g) : 14.7
Voltage : 4.8 - 6

8.3.2 MG-996R SERVO MOTOR:

Specifications:

- Weight: 55 g
- Dimension: 40.7 x 19.7 x 42.9 mm approx.
- Stall torque: 9.4 kgf·cm (4.8 V), 11 kgf·cm (6 V)
- Operating speed: 0.17 s/60º (4.8 V), 0.14 s/60º (6 V)
- Operating voltage: 4.8 V a 7.2 V
- Running Current 500 mA –900 mA (6V)
- Stall Current 2.5 A (6V)
- Dead band width: 5 µs
- Stable and shock proof double ball bearing design
- Temperature range: 0 ºC –55^C



PRODUCT CONFIGURE TABLE

Weight(g)	55
Torque(kg)(4.8v)	9.4
Speed(sec/60deg)	0.17
A(mm)	42.7
B(mm)	40.9
C(mm)	37
D(mm)	20
E(mm)	54
F(mm)	26.8