İZMİR KÂTİP ÇELEBİ ÜNİVERSİTESİ

2010

FACULTY OF ENGINEERING AND ARCHITECTURE MECHATRONIC ENGINEERING

Graduation Project

Design and Control of a Serial Robot Arm using ROS (Robot Operating System)

SUPERVISOR: Asst. Prof. Dr. Fatih Cemal Can

Project Members: Elif Özdemir 190412043
Kaan Çelik 180412010
Mehmet Akif Demirlek 180412057
Saad Abo Ali 190412005

June 2024

ABSTRACT

This project focuses on the design and development of a 4 degrees of freedom (DoF) robotic arm manipulator integrated with the Robot Operating System (ROS). The aim of our project is to learn robot control systems in depth and transform the industry by adapting open source robot control systems to industrial applications. In this direction, we aim to develop algorithms that optimize the movement control of robots. GAZEBO simulation environment will be used to simulate the movements of the robot arm, and plug-ins such as Rviz and Motion Planning in Rviz will be used for control. Packets broadcast wirelessly to the ROS control node running on ESP32 will provide control of the robot. Data from the servos used in the robot arm will be broadcast from the ESP32 node to be fed to our PID controller using pre-calculated kinematic calculations in the ROS environment. This integration will enable remote planning, monitoring and control of the robot arm's movements, increasing diagnostic and intervention capabilities.

ÖZET

Bu proje, Robot İşletim Sistemi (ROS) ile entegre edilmiş 4 serbestlik dereceli (DoF) bir robot kol manipülatörünün tasarım ve geliştirilmesine odaklanmaktadır. Projemizin amacı, robot kontrol sistemlerini derinlemesine öğrenmek ve açık kaynak robot kontrol sistemlerini endüstriyel uygulamalara uyarlayarak sektörde dönüşüm sağlamaktır. Bu doğrultuda, robotların hareket kontrolünü optimize eden algoritmalar geliştirmeyi hedefliyoruz. Robot kolunun hareketlerinin simülasyonu için GAZEBO simülasyon ortamı kullanılacak olup, kontrol için Rviz ve Rviz içindeki Hareket Planlama gibi eklentiler kullanılacaktır. ESP32 üzerinde çalışan ROS kontrol düğümüne kablosuz olarak yayınlanan paketler, robotun kontrolünü sağlayacaktır. Robot kolunda kullanılan servolardan gelen veriler, ROS ortamındaki önceden hesaplanmış kinematik hesaplamaları kullanan PID kontrolcümüze beslenmek üzere ESP32 düğümünden yayınlanacaktır. Bu entegrasyon, robot kolunun hareketlerinin uzaktan planlanmasını, izlenmesini ve kontrol edilmesini sağlayarak tanılama ve müdahale yeteneklerini artıracaktır.

ACKNOWLEDGMENTS

We would like to thank our teacher Fatih Cemal Can for guiding us in the progress of our project every week. Finally, we would like to thank our other professors, Erkin Gezgin, Özgün Başer and Mertcan Koçak, who helped us with some problems we encountered in our project.

CONTENTS

1. INTRODUCTION

The field of robotics is inherently an interdisciplinary study. From underwater exploration to space missions on Mars and beyond. Many engineers are interested in the evolving field of robotics. Industrial robot arms, in use for over fifty-five years, have been successful in high structured environments. [1] However, recent technological advancements have allowed industrial robots to operate in less structured environments. In our project, we conducted kinematics analysis to learn fundamental robotics concepts, we utilized the Open-Source Robotics Operating System (ROS) and Moveit software to develop the robot for real-world applications. [2] [3]

Figure 1. Ros Operating System (ROS)

1.1. Robot Manipulators

An industrial robot's arm-like structure, commonly referred to as a robot manipulator, plays an important role in executing programmed tasks. Also known as a robot arm, this manipulator attaches to the robot body and comprises multiple links and joints. Robots of this type are commonly used for repetitive tasks, including pick-and-place operations on production lines and precision welding of metal parts, where high accuracy and reliability are essential. However, most of the robot arms currently used in the industry use highly specialized control software, require the production line to be stopped at every issue on motion planning of the robot, and sometimes even require an operator from the manufacturer’s company. This lack of transparency in controlling and troubleshooting of the robot causes prolonged downtimes during maintenance and repairs of existing robot arms adversely affecting both production efficiency and operational costs.

1.2 Aim of the Project

The primary aim of our project is to develop a robot arm manipulator that uses the open-source ROS (Robot Operating System) environment. Our project seeks to address this problem by enhancing industrial robot arm maintenance through open-source control software. The project objectives include utilizing the open-source platform ROS, the system controls, and creates motion planning for its tasks through accessible software open to everyone, establishing a modular and flexible framework for developing robotic applications. Accelerating maintenance and fault detection processes through remote access, thereby preventing production line stoppages during potential malfunctions. Providing a real-time simulation environment for testing robot arm movements during both the design phase and subsequent stages using GAZEBO and Rviz simulation environments.

By achieving these objectives, our project aims to surpass the limitations of current robot arms, supporting Industry 4.0's vision for enhanced, flexible, and dynamic production processes.

1.3 Research and Development

The project's Research and Development (R&D) focuses on delivering innovative solutions in the field of industrial robotics, addressing challenges such as enhancing robot arm manipulator performance, optimizing maintenance processes, and ensuring production continuity. This involves designing industry-capable custom parts and achieving rapid, cost-effective production using 3D printing technology. Furthermore, unlike closed-loop systems commonly found in existing robot arm systems, our project, utilizing the open-source ROS platform, allows for the modification and improvement of robot arm systems with support from a large developer community. The integration of open-source robotic control systems into the industry, along with automation, and simulation technologies, contributes to the project's overall goal.

2. METHODOLOGY PROPOSED

To design a robot arm manipulator, we first need to determine our workspace and the structure of the work environment to find how many degrees of freedom we need to utilize the best of the workspace we are working with and the task we need to accomplish. Our robot arm will be used for pick-and-place tasks around obstacles, and it should be able to rotate its picked object around its own axis to accomplish our pre-determined task. After these considerations we decided to design a 4-DoF robot arm with a gripper at the end, the design also has to be 3D printable, this means our links has height limitations. In order to work around this limitation, we split the longest two links by half so that each that each part can be scaled as big as printing volume of the printer. With this design parameters, using Fusion 360 we finalized a design that can be 3D printed and can be used on our determined tasks.

2.1. Structural Design

A practical 4 degrees of freedom (4-DoF) robot arm is designed for efficient pick and place tasks, allowing precise positioning along the X, Y and Z axes. A gear wheel system is used in Joint 4 to facilitate the movements of the bus servo motors. LX-16A Serial Bus Servo Motor will be used to provide all movements of the robot, such as rotation and translation. Modular end effector design allows angles to be easily adjusted for effective pick and place operations.

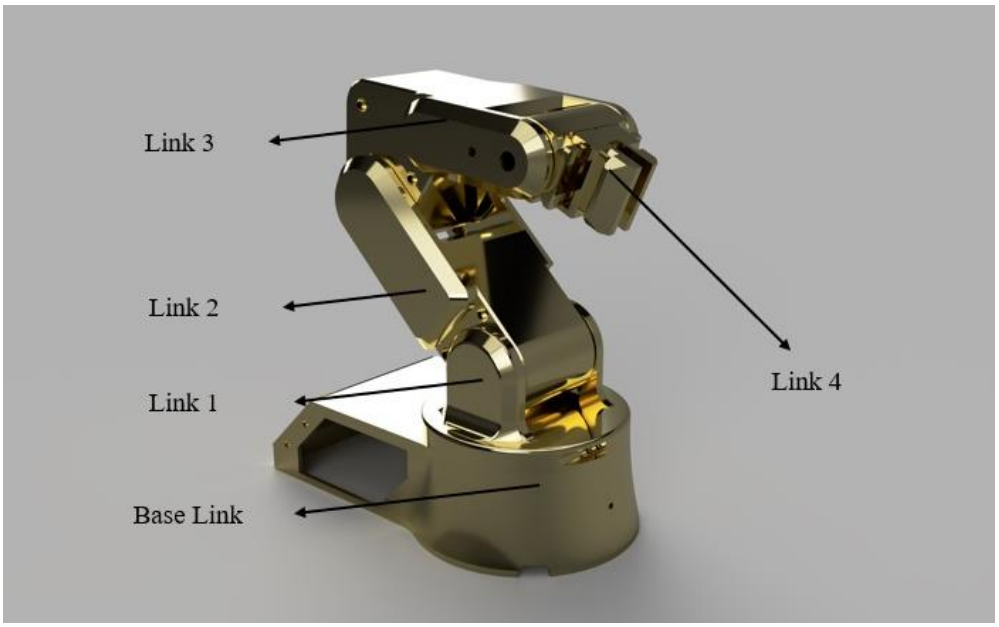


Figure 2. 4-DoF Manipulator CAD Design

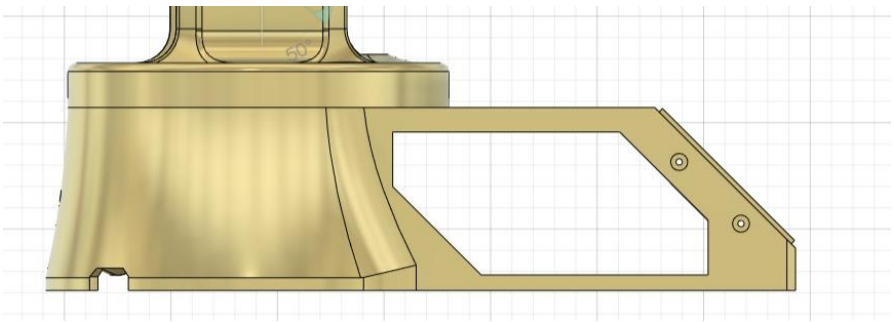


Figure 3.Base Link location

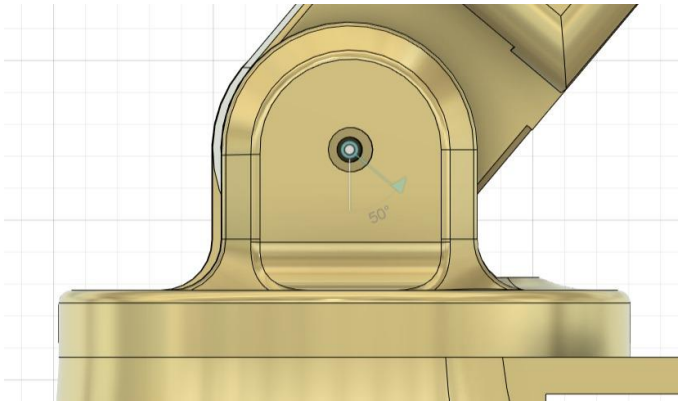


Figure 4. Link 1 location with Joint 1

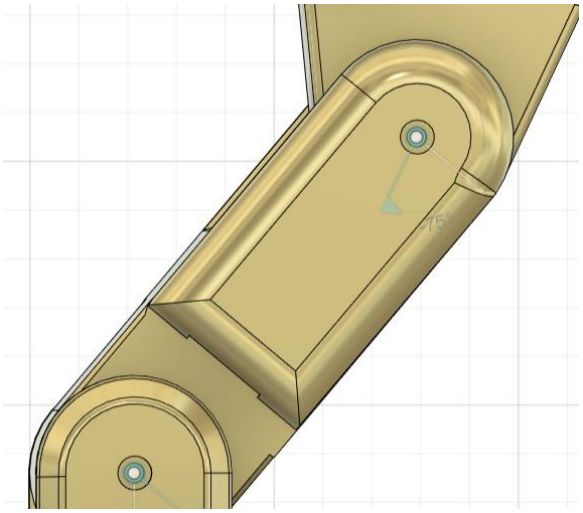


Figure 5.Link 2 location with Joint 1 and Joint 2

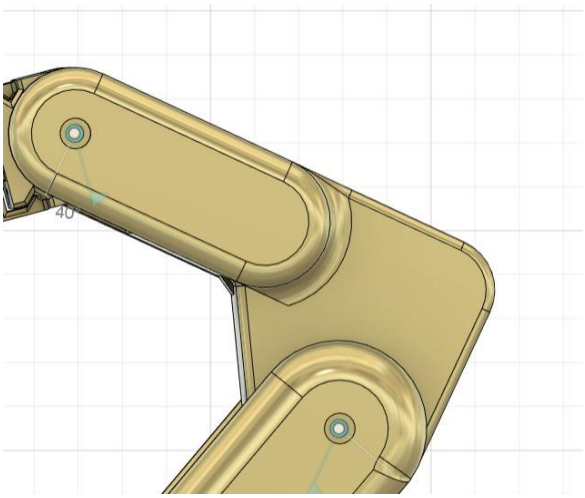


Figure 6. Link 3 location located between Joint 2 and Joint 3

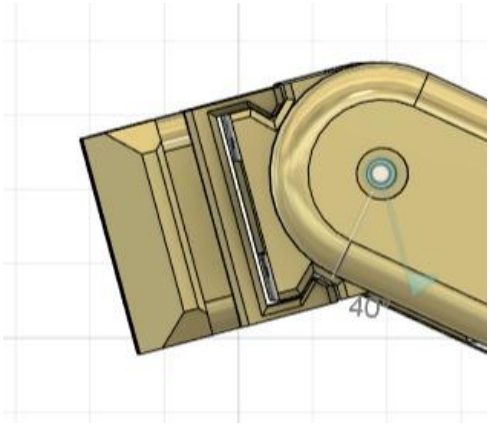


Figure 7.Link 4 location is located between Joint 4 and End Effector

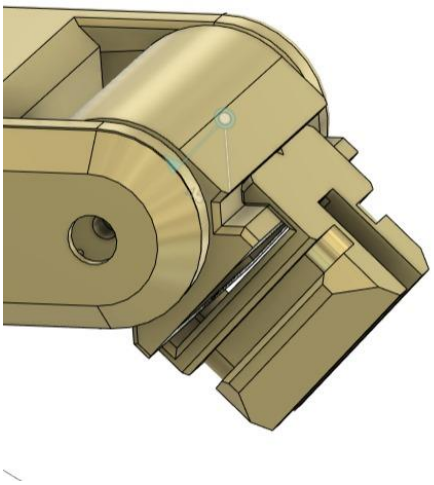


Figure 8. End Effector location

2.2. Electronic Circuit Design

For the electronic circuit selection, we carefully choosed materials that fits to our robot's needs. After checking the relevant literature and many products for Robot Arm electronics, we decided on the ESP32 as the first electronic circuit element.

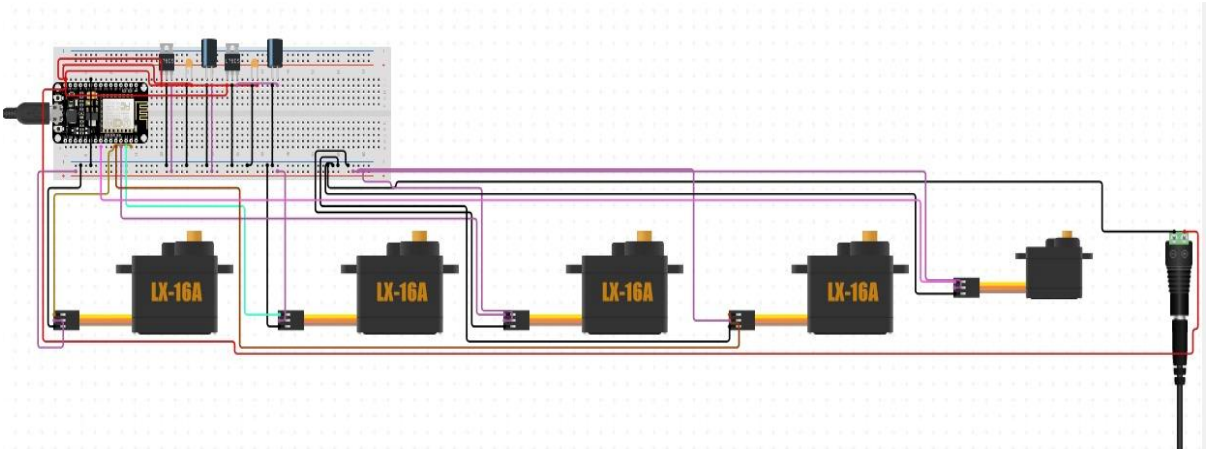


Figure 9.Electronic Circuit Design

Figure 10.ESP32 ESP-32S WiFi+Bluetooth Dual-Mode Development Board ESP32 [7]

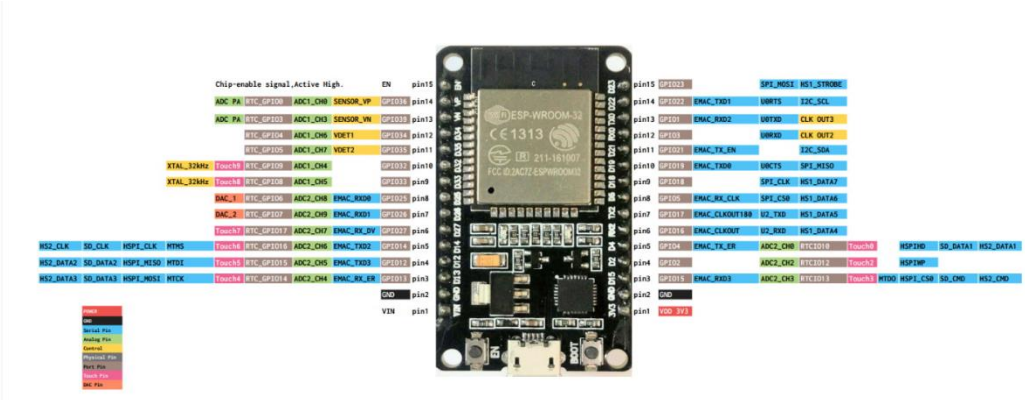


Figure 11.ESP32 Datasheet

After reviewing Robot Dynamics, we selected the LX-16A Serial Bus Servo Motor for our application. These motors are widely used in robotics. Because the most important difference between bus servo motors and other motors is that they have an integrated position feedback system and digital control capability. Thanks to these features, they can provide precise

positioning and motion control when used in robotics fields. This means that bus servo motors are ideal for precisely moving to desired positions in robot arms, joint movements and other similar applications.

This makes them a solid choice for our robotic arm, which needs to move stably along different axes over specific time intervals. We used a total of 4 bus servo motors in our project. We chose these models because they can effectively handle the varying loads that the motors must carry on different axes.

Figure 12.LX-16A Serial Bus Servo Motors

Weight	54g	**Rotation**	0-240°
Dimension	45.22 x 24.72 x 36.3 mm	**No-load current**	100mA
Speed	0.19sec/60°7.4V	**Control method**	UART serial command
Servo accuracy	0.3°	**Communication baud rate**	115200
Torque	17kg.cm 6V; 19.5kg.cm 7.4V	**Protection**	Avoid stalling and overheat
Stall Current	2.4~3A	**Data feedback**	Temperature, Voltage, Position
Working voltage	6-8.4V	**Indicator**	LED
Servo ID	0~253	**Working mode**	Servo mode and deceleration motor mode
Readback function	Support angle readback	**Storage**	Save data when the power off
Wire length	200mm	**Gear**	Metal

Figure 13. Specifications of LX-16A Serial Bus Servo Motor

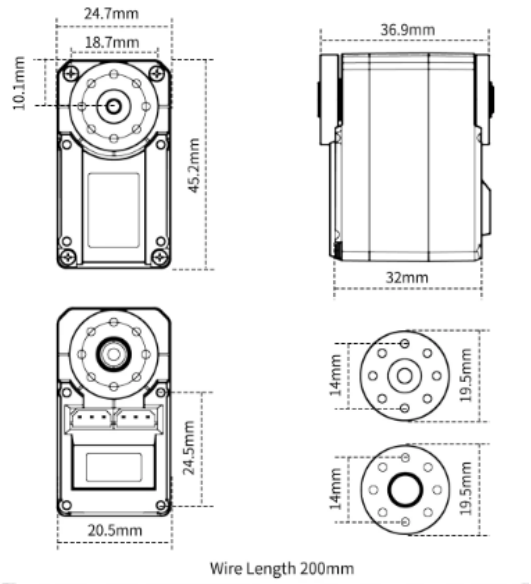


Figure 14.LX-16A Serial Bus Servo Motor Datasheet

After researching the bus servo motors and driver circuits that meet our criteria on the market, we determined the most suitable bus servo motor and driver pairs. We decided to control four of the LX-16A Serial Bus Servo Motors with one TTL\USB Debugging Board.

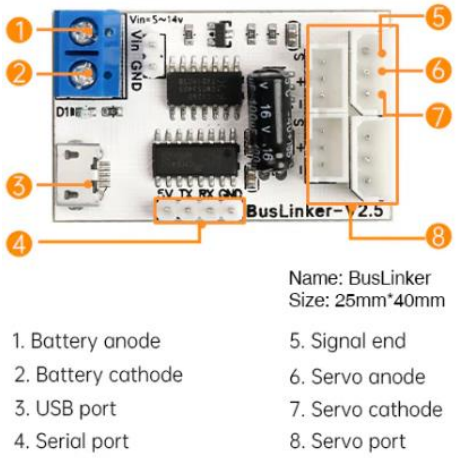


Figure 15. TTL\USB Debugging Board

After extensive literature review and market research for connections from microcontroller to servos, we also determined to use a bus servo controller if needed.

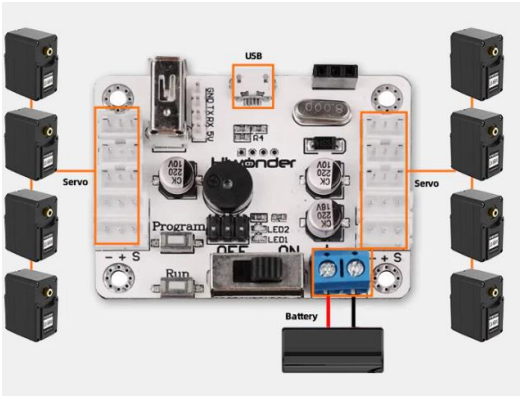


Figure 16.Servo Controller

To power the entire robot arm and its controller, we settled on a 12 2A Power Supply.

Figure 17.12V 2A Power Supply

2.3. Gripper System

The most important part of the robotic arm is the end effector, because the end effector is selected according to the task that the robot will perform and plays a critical role in its ability to perform this task. Industrial robots often use grippers to grasp and move objects. Grippers provide a reliable grip, allowing the robot to properly perform material picking and placing operations. In this way, robots can work with high efficiency and precision in industrial processes.

In this section, a small circuit was created for the gripper of the robot arm. A servo motor was used in this circuit as it provides precise control and mobility. The servo motor plays an important role in the gripper grasping and releasing objects as it has the ability to rotate at certain angles. All circuit elements will be brought together to create an electronic system and this system will be integrated into the gripper of the robot arm. This integration will provide the movement and control capabilities necessary for the gripper to perform a variety of tasks. [8]

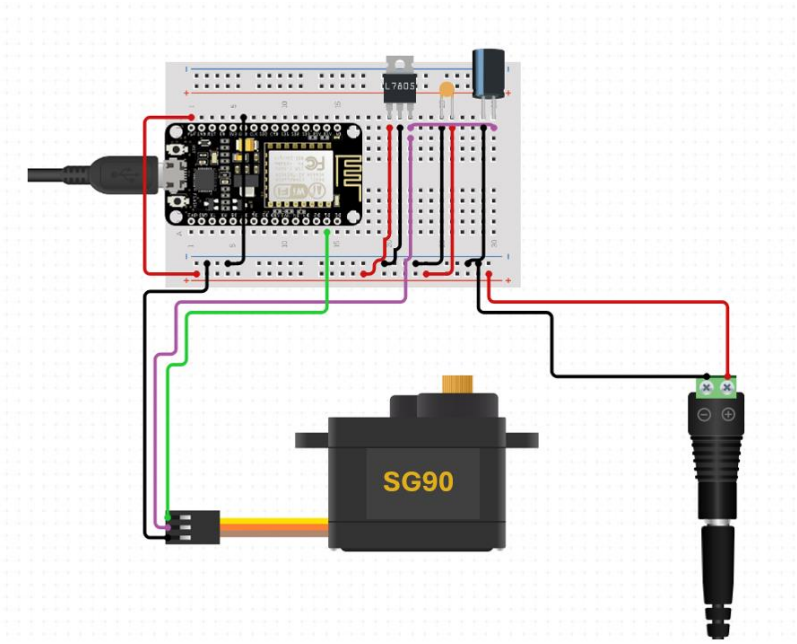
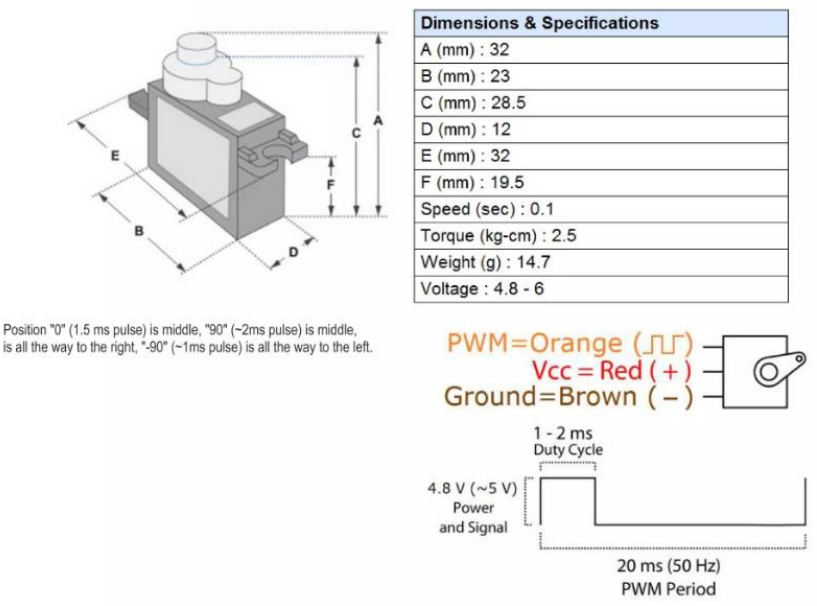


Figure 18.Gripper Circuit Design



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

Figure 19.SG90 Micro Servo Datasheet

2.4. Electronic Control System

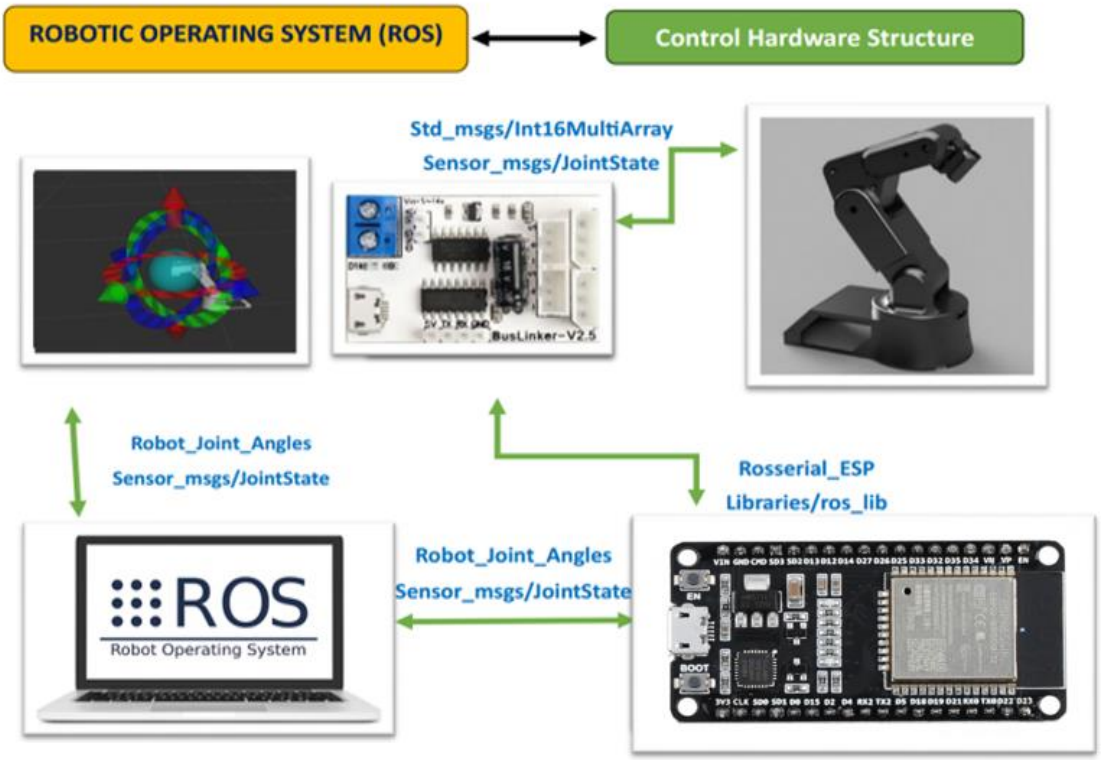


Figure 20.Control Hardware Structure connection diagram with Ros

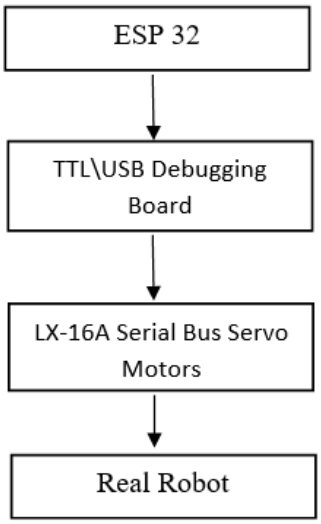


Figure 21.Motor Robot flow diagram

The schematic diagram shows how a robot arm's controller circuit is integrated with the Robotic Operating System (ROS). The ROS communicates with the control hardware structure, which consists of an ESP 32 microcontroller and five servo motors that power the robot arm's joints. The Microcontroller receives data from ROS as the desired joint angles and sensor states and sends PWM signals to the servos accordingly. The ROS also displays a 3D model of the robot arm on a computer screen, which can be used to visualize and monitor the robot arm's movements and operations. The diagram uses different colors and labels to indicate the data flow and the components involved in the system.

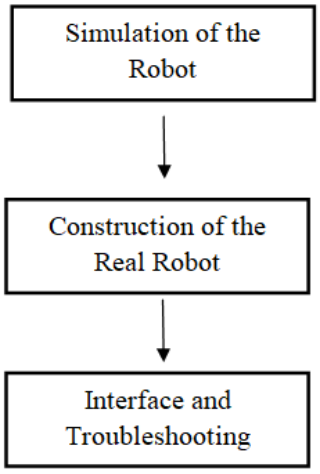


Figure 22.Ros Internal Control Algorithm of the Robot Arm

3. KINEMATICS AND CALCULATIONS

3.1. Forward Kinematic

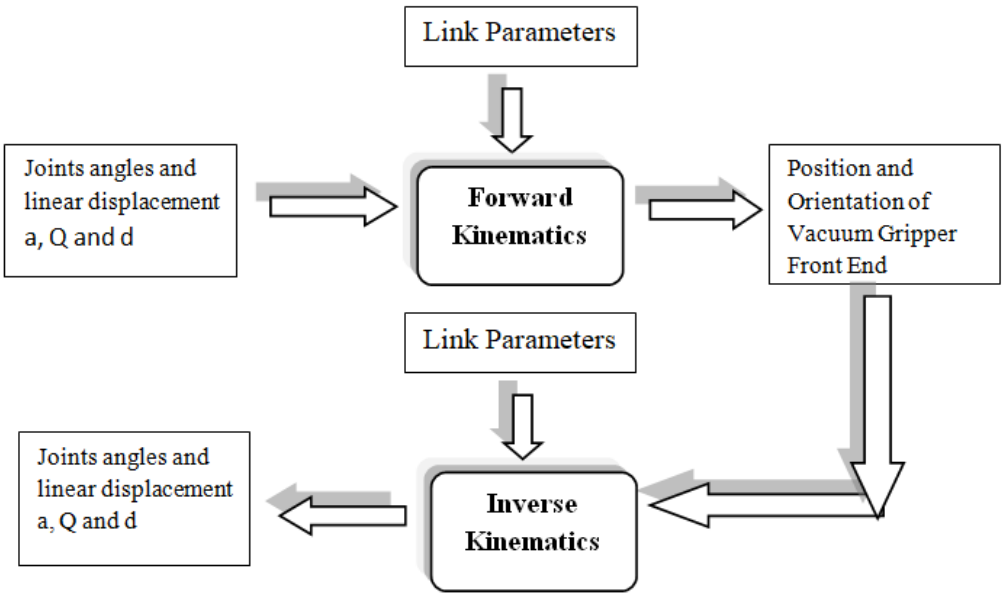


Figure 23.Forward and Inverse Kinematic diagram

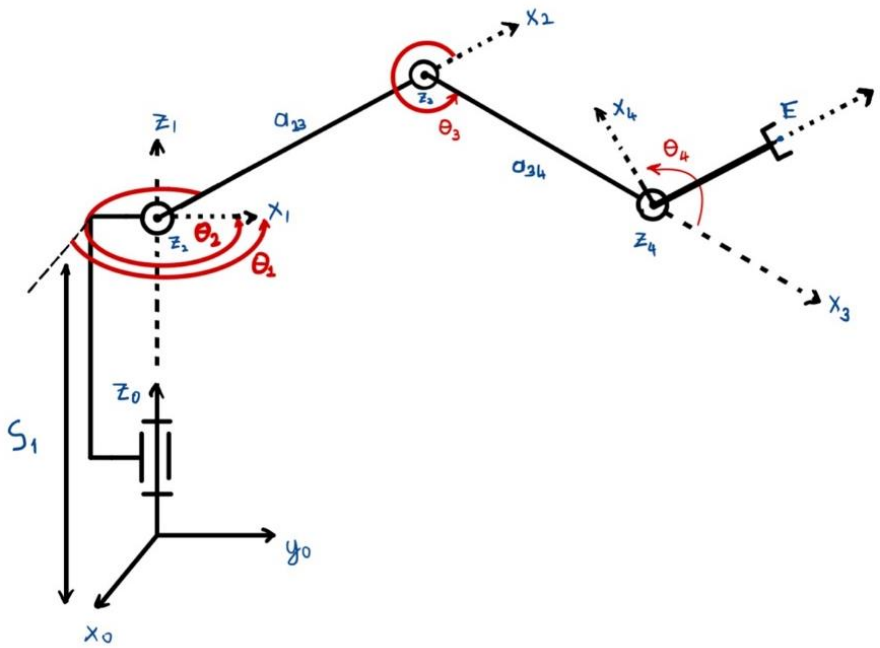


Figure 24.Robot Kinematic Diagram

Kinematics is the science of motion that treats motion without regard to the forces which cause it. Within the science of kinematics, one studies position, velocity, acceleration, and all higher order derivatives of the position variables (with respect to time or any other variables. A manipulator is composed of serial links which are connected to each other by revolute or prismatic joints from the base frame through the end-effector. Calculating the position and orientation of the end-effector in terms of the joint variables is called as forward kinematics.

3.2 Denavit–Hartenberg (D-H) Table

I	$\mathbf{a_{(i-1,\,i)}}$	$\boldsymbol{\alpha_{(i-1,\,i)}}$	**Si**	$\boldsymbol{\Theta_i}$
1	0	0	S1	Θ_1
2	0	$\pi/2$	0	Θ_2
3	a_{23}	0	0	Θ_3
4	a_{34}	0	0	Θ_4

Table 1. D-H Table

The Denavit-Hartenberg (DH) kinematic model is the most widely used method to describe robot kinematics. There are four parameters in this model: $a_{i-1,i}$, $\alpha_{i-1,i}$, Si and θi. These parameters are called connection length, connection twist, connection offset and connection angle, respectively. A coordinate frame is added for each joint to determine the DH parameters. The distance from Zi-1 to Zi is assigned by measuring $a_{i-1,i}$ along Xi-1. Likewise, the angle between Zi-1 and Zi is measured along $\alpha_{i-1,i}$,Xi, the angle measured along Zi is assigned as Si, and the angle between Xi-1 and Xi is assigned as θi.

3.3 Transformation Matrix

A transformation matrix is a mathematical tool used to express the relationship of one coordinate system to another coordinate system. This matrix is used to express the position of an object in one coordinate system and its position in another coordinate system. So, this matrix is used to convert the position and orientation of an object to the target coordinate system, with the initial coordinate system as reference. The transformation matrix is represented using homogeneous coordinates. It is generally a 4x4 dimensional matrix and contains translation and rotation components in a three-dimensional space.

A 4x4 dimensional homogeneous transformation matrix is expressed in the form:

$$T = \begin{bmatrix} R & P \\ 000 & 1 \end{bmatrix} \quad (1)$$

Here, R represents a 3x3-dimensional rotation matrix and P represents a 3x1-dimensional translation vector.

3.4 Calculation of Transformation Matrices

The matrix ${}^{i-1}_{i}T$ is known as a D-H convention matrix. In the matrix ${}^{i-1}_{i}T$, the quantities of $a_{i-1,i}, \alpha_{i-1,i}$, are constant for a given link while the parameter θ_i for a revolute joint is variable.

$$ {}^{i-1}_{i}T = D_{x_{i-1}}(a_{i-1,i}) \times R_{x_{i-1}}(\alpha_{i-1,i}) \times D_{z_i}(Si) \times R_{z_i}(\theta_i) \tag{2} $$

Using ${}^{i-1}_{i}T$, we compute each of the link transformations as follows:

$$ {}^{0}_{1}T = D_{x_0}(a_{01}) \times R_{x_0}(\alpha_{01}) \times D_{z_1}(S1) \times R_{z_1}(\theta_1); \tag{3} $$

$$ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & S1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} C\theta_1 & -S\theta_1 & 0 & 0 \\ S\theta_1 & C\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} C\theta_1 & -S\theta_1 & 0 & 0 \\ S\theta_1 & C\theta_1 & 0 & 0 \\ 0 & 0 & 1 & S1 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{0}_{1}T = \begin{bmatrix} C\theta_1 & -S\theta_1 & 0 & 0 \\ S\theta_1 & C\theta_1 & 0 & 0 \\ 0 & 0 & 1 & S1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1} $$

$$ {}^{1}_{2}T = D_{x_1}(a_{12}) \times R_{x_1}(\alpha_{12}) \times D_{z_2}(S2) \times R_{z_2}(\theta_2); \tag{3.2} $$

$$ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\frac{\pi}{2} & -S\frac{\pi}{2} & 0 \\ 0 & S\frac{\pi}{2} & C\frac{\pi}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} C\theta_2 & -S\theta_2 & 0 & 0 \\ S\theta_2 & C\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} C\theta_2 & -S\theta_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ S\theta_2 & C\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{1}_{2}T = \begin{bmatrix} C\theta_2 & -S\theta_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ S\theta_2 & C\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3} $$

$$ {}^{2}_{3}T = D_{x_2}(a_{23}) \times R_{x_2}(\alpha_{23}) \times D_{z_3}(S3) \times R_{z_3}(\theta_3); \tag{3.4} $$

$$ \begin{bmatrix} 1 & 0 & 0 & a_{23} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} C\theta_3 & -S\theta_3 & 0 & 0 \\ S\theta_3 & C\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} C\theta_3 & -S\theta_3 & 0 & a_{23} \\ S\theta_3 & C\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{2}_{3}T = \begin{bmatrix} C\theta_3 & -S\theta_3 & 0 & a_{23} \\ S\theta_3 & C\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.5) $$

$$ {}^{3}_{4}T = D_{x_3}(a_{34}) \times R_{x_3}(\alpha_{34}) \times D_{z_4}(S4) \times R_{z_3}(\theta_4); \quad (3.6) $$

$$ \begin{bmatrix} 1 & 0 & 0 & a_{34} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} C\theta_4 & -S\theta_4 & 0 & 0 \\ S\theta_4 & C\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} C\theta_4 & -S\theta_4 & 0 & a_{34} \\ S\theta_4 & C\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{3}_{4}T = \begin{bmatrix} C\theta_4 & -S\theta_4 & 0 & a_{34} \\ S\theta_4 & C\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.7) $$

We form ${}^{0}_{4}T$ by multiplication of the above individual link matrices (from equation 3.1 to equation 3.7). Using this ${}^{0}_{4}T$ matrix, we will perform forward kinematics and inverse kinematics calculations for the robot arm. The ${}^{0}_{4}T$ matrix is as follows;

$$ {}^{0}_{4}T = {}^{0}_{1}T \times {}^{1}_{2}T \times {}^{2}_{3}T \times {}^{3}_{4}T \quad (4) $$

$$ {}^{0}_{4}T = \begin{bmatrix} C\theta_1 C\theta_{234} & -C\theta_1 S\theta_{234} & S\theta_1 & C\theta_1(a_{23}C\theta_2 + a_{34}C\theta_{23}) \\ S\theta_1 C\theta_{234} & -S\theta_1 S\theta_{234} & -C\theta_1 & S\theta_1(a_{23}C\theta_2 + a_{34}C\theta_{23}) \\ S\theta_{234} & C\theta_{234} & 0 & S1 + a_{23}S\theta_2 + a_{34}S\theta_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Using the ${}^{0}_{4}T$ matrix, it is possible to calculate the position and orientation of frame 5 relative to frame 0 of the robot. These are the basic equations for the entire kinematic analysis of this manipulator.

$$ {}^{0}_{4}T = \begin{bmatrix} u_x & v_x & w_x & p_x \\ u_y & v_y & w_y & p_y \\ u_z & v_z & w_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5) $$

3.5 Forward (Direct) Kinematics Calculations

${}^{0}_{5}T$ matrix specify how to compute the position and orientation of frame 6 relative to frame 0 of the robot as follows;

$$u_x = C\theta_1 C\theta_{234} \quad (6)$$

$$u_y = S\theta_1 C\theta_{234} \quad (6.1)$$

$$u_z = S\theta_{234} \quad (6.2)$$

$$v_x = -C\theta_1 S\theta_{234} \quad (6.3)$$

$$v_y = -S\theta_1 S\theta_{234} \quad (6.4)$$

$$v_z = C\theta_{234} \quad (6.5)$$

$$w_x = S\theta_1 \quad (6.6)$$

$$w_y = -C\theta_1 \quad (6.7)$$

$$w_z = 0 \quad (6.8)$$

$$p_x = C\theta_1(a_{23}C\theta_2 + a_{34}C\theta_{23}) \quad (6.9)$$

$$p_y = S\theta_1(a_{23}C\theta_2 + a_{34}C\theta_{23}) \quad (6.10)$$

$$p_z = S1 + a_{23}S\theta_2 + a_{34}S\theta_{23} \quad (6.11)$$

3.6 Inverse Kinematics

The tasks that a manipulator performs are usually described in Cartesian space, but actuators often operate in joint space. The Cartesian space contains the position vector and the orientation matrix. However, joint angles are often used to express the movement of manipulators, which belongs to the joint space.The process obtained by transforming the position and orientation of the manipulator's end effector in Cartesian space into joint angles in joint space is called the "inverse kinematics problem".

3.7 Inverse Kinematics Calculations

Using equations (6.6) and (6.7), the angle θ_1 is calculated by creating an atan2 function with $C\theta_1$ and $S\theta_1$.

- $w_x = S\theta_1$ (7)
- $w_y = -C\theta_1$ (7.1)
- $\boldsymbol{\theta_1 = Atan2}(w_x, -w_y)$ (7.2)

Using equations (6.2) and (6.5), the angle θ_{234} is calculated by creating an atan2 function with $C\theta_{234}$ and $S\theta_{234}$.

- $u_z = S\theta_{234}$ (7.3)
- $v_z = C\theta_{234}$ (7.4)
- $\boldsymbol{\theta_{234} = Atan2}(u_z, v_z)$ (7.5)

After taking the squares of equations (6.9) and (6.11), the results obtained are added together and $\boldsymbol{C\theta_{234}}$ is obtained

- $p_x = C\theta_1(a_{23}C\theta_2 + a_{34}C\theta_{23})$ (7.5)
- $p_z = S1 + a_{23}S\theta_2 + a_{34}S\theta_{23})$ (7.6)
- $\frac{p_x}{C\theta_1} = a_{23}C\theta_2 + a_{34}C\theta_{23}$ $\frac{p_x}{C\theta_1} = K_1$ (7.7)
- $p_z - S1 = a_{23}S\theta_2 + a_{34}S\theta_{23}$ $p_z - S1 = K_2$ (7.8)
- $(K_1)^2 + (K_2)^2 = a_{23}^2C\theta_2^2 + a_{34}^2C\theta_{23}^2 + 2a_{23}a_{34}C\theta_2C\theta_{23} + a_{23}^2S\theta_2^2 + a_{34}^2S\theta_{23}^2 + 2a_{23}a_{34}S\theta_2S_{23}$ $= a_{23}^2 + a_{34}^2 + 2a_{23}a_{34}(C\theta_2C_{23} + S\theta_2S_{23})$ (7.9)

$S\theta_3$ is $\sqrt{1 - C\theta_3{}^2}$. Then, θ_3 is calculated by creating atan2 function with $C\theta_3$ and $S\theta_3$

- $\boldsymbol{C\theta_3 = \frac{(K_1)^2 + (K_2)^2 - a_{23}^2 - a_{34}^2}{2a_{23}a_{34}}}$ (8)
- $\boldsymbol{S\theta_3 = \sqrt{1 - \left(\frac{(K_1)^2 + (K_2)^2 - a_{23}^2 - a_{34}^2}{2a_{23}a_{34}}\right)^2}}$ (8.1)

- $$\theta_3 = \mathbf{Atan2}\left(\sqrt{1-\left(\frac{(K_1)^2+(K_2)^2-a_{23}^2-a_{34}^2}{2a_{23}a_{34}}\right)^2}, \frac{(K_1)^2+(K_2)^2-a_{23}^2-a_{34}^2}{2a_{23}a_{34}}\right) \quad (8.2)$$

In order to obtain θ_2, (6.10) is used;

- $$p_Y = S\theta_1(a_{23}C\theta_2 + a_{34}C\theta_{23}) \quad (8.3)$$
$$\frac{p_y}{S\theta_1} = a_{23}C\theta_2 + a_{34}C\theta_{23} = a_{23}C\theta_2 + a_{34}C\theta_2C\theta_3 - a_{34}S\theta_2S\theta_3$$
- $$= C\theta_2(a_{23} + a_{34}C\theta_3) - a_{34}S\theta_2S\theta_3 \quad (8.4)$$

- $$K_3 = a_{23} + a_{34}C\theta_3 \quad (8.5)$$
- $$K_4 = a_{34}S\theta_3 \quad (8.6)$$
- $$K_5 = \frac{p_y}{S\theta_1} \quad (8.7)$$
- $$K_5 = K_3C\theta_2 - K_4S\theta_2 \quad (8.8)$$

$S\theta_2$ and $C\theta_2$ are obtained in terms of tangent with half angle formulas. After, θ_2 calculated with arctan function;

- $$S\theta_2 = \frac{2\tan\frac{\theta_2}{2}}{1+\left(\tan\frac{\theta_2}{2}\right)^2} = \frac{2t_2}{1+t_2^2} \qquad C\theta_2 = \frac{1-\left(\tan\frac{\theta_2}{2}\right)^2}{1+\left(\tan\frac{\theta_2}{2}\right)^2} = \frac{1-t_2^2}{1+t_2^2}$$
(9)
- $$K_3\frac{1-t_2^2}{1+t_2^2} - K_4\frac{2t_2}{1+t_2^2} = K_5 \quad (9.1)$$
$$K_3 - K_3t_2^2 - 2K_4t_2 = K_5 + K_5t_2^2$$
$$K_3 - K_3t_2^2 - 2K_4t_2 - K_5 - K_5t_2^2 = 0$$
$$t_2^2(-K_3 - K_5) - 2K_4t_2 + K_3 - K_5 = 0 \quad (9.2)$$
$$x = \frac{-b\pm\sqrt{b^2-4ac}}{2a} \quad (9.3)$$

$$\tan\frac{\theta_2}{2} = t_2 = 2K_4 \pm \sqrt{\frac{(-2K_4)^2-4(-K_3-K_5)(K_3-K_5)}{2(-K_3-K_5)}}$$

$$\boldsymbol{\theta_2 = 2\arctan\left(2K_4 \pm \sqrt{\frac{(-2K_4)^2-4(-K_3-K_5)(K_3-K_5)}{2(-K_3-K_5)}}\right)} \quad (9.4)$$

$$\theta_{234} = \theta_2 + \theta_3 + \theta_4$$

- $$\boldsymbol{\theta_4 = \theta_{234} - \theta_3 - \theta_4} \quad (9.5)$$

4. 3D WORKSPACES OF 4-DOF ROBOT ARM

The 3D Collaborative Workspace of the Robot Arm is the workspace created by spot scanning all the movements that the robot arm can make while working. Figure. 21 shows the 3D working area of the robot arm.

By examining the graphs, the values adopted by the joint angles (θ1, θ2, θ3 and θ4) were obtained and the resulting value ranges for the working area of the joints (defined in Table 2) were taken. A code was created in Matlab by taking the value ranges of these angles and the Px, Py and Pz values found in inverse kinematics. Thanks to this code, the 3D working area of the robotic arm was found.

This 3D workspace should be taken into account to prevent unwanted movements by detecting and preventing sudden changes when operating the robot arm.

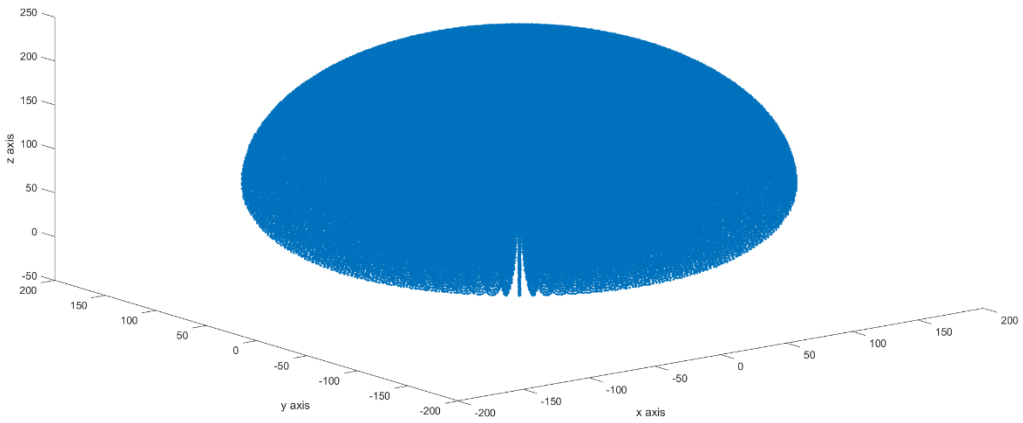


Figure 25. 3D Workspace of Robot Arm Iso View

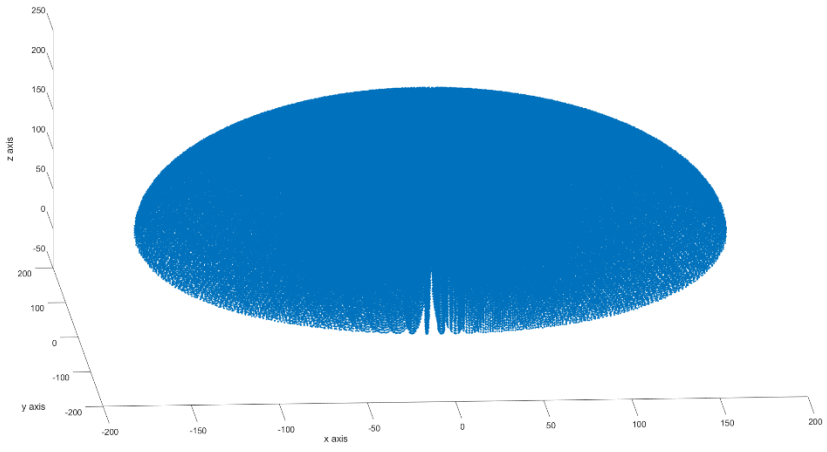


Figure 26. 3D Workspace of Robot Arm Front View

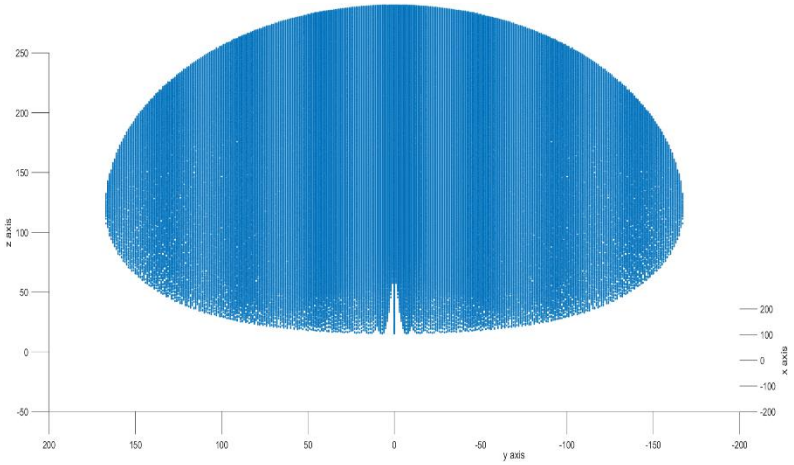


Figure 27. 3D Workspace of Robot Arm Left Side View

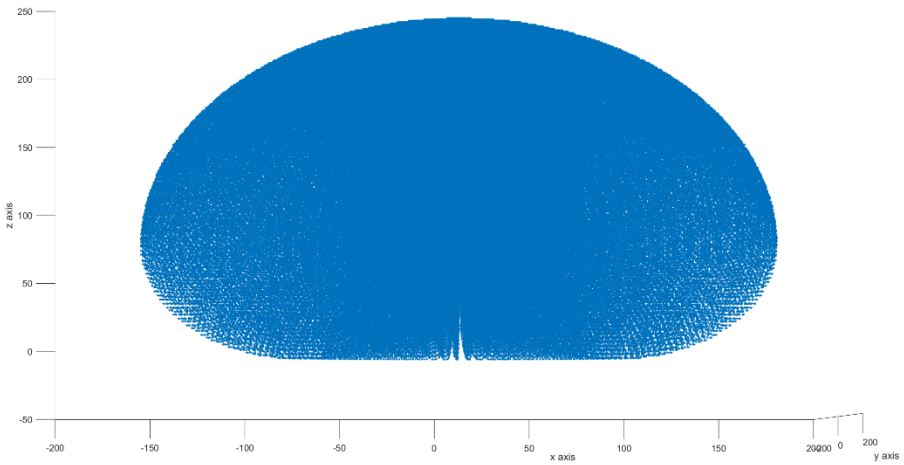


Figure 28. 3D Workspace of Robot Arm Right Side View

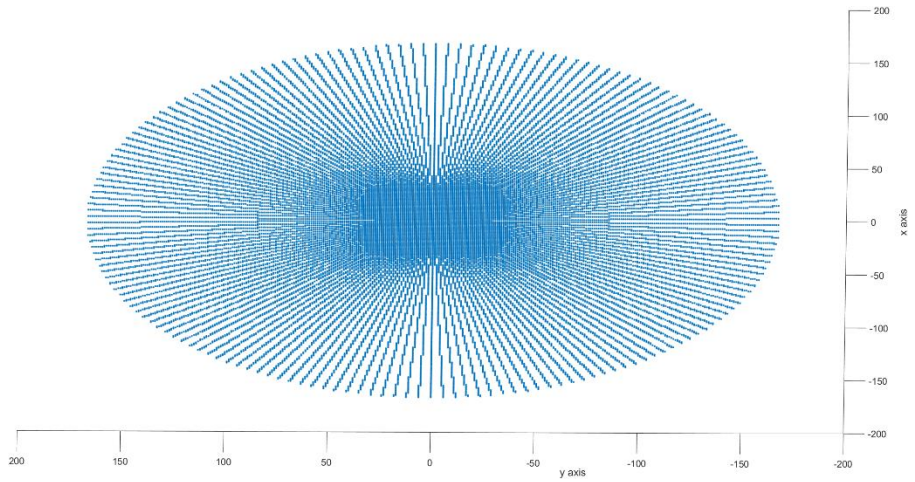


Figure 29. 3D Workspace of Robot Arm Top View

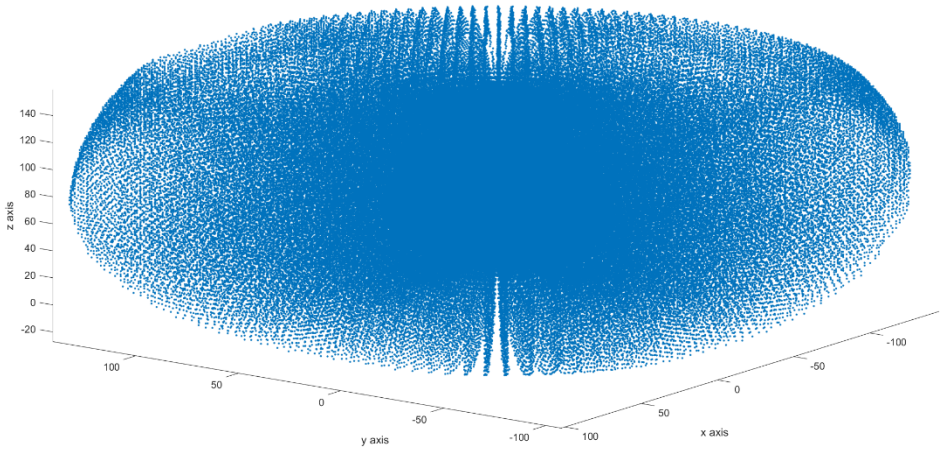


Figure 30.Iso View of the 3D Working Area of the Robot Arm with 5X zoom

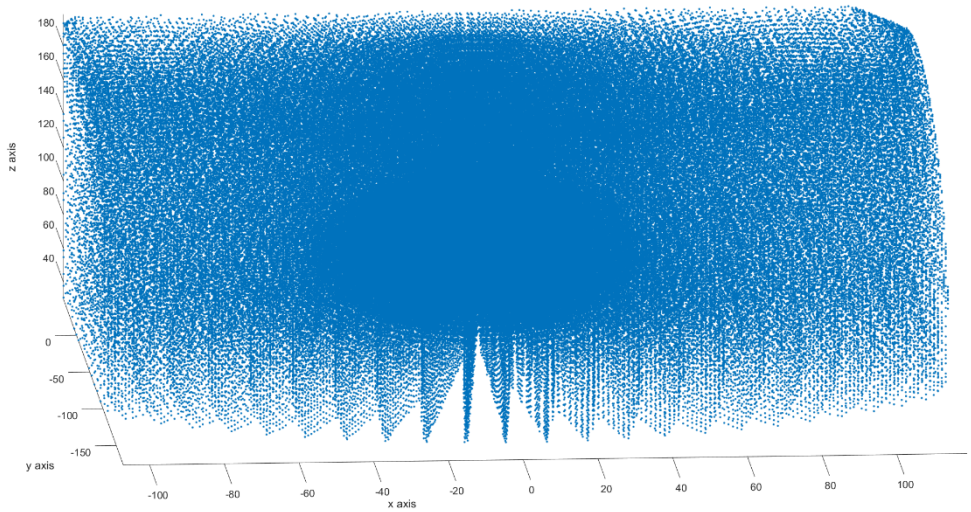


Figure 31.Front View of the 3D Working Area of the Robot Arm with 5X zoom

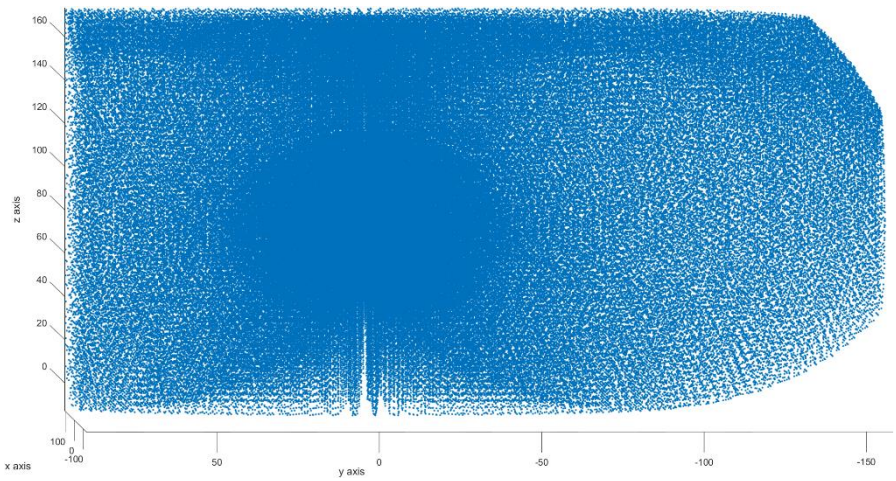


Figure 32.Left Side View of the 3D Working Area of the Robot Arm with 5X zoom

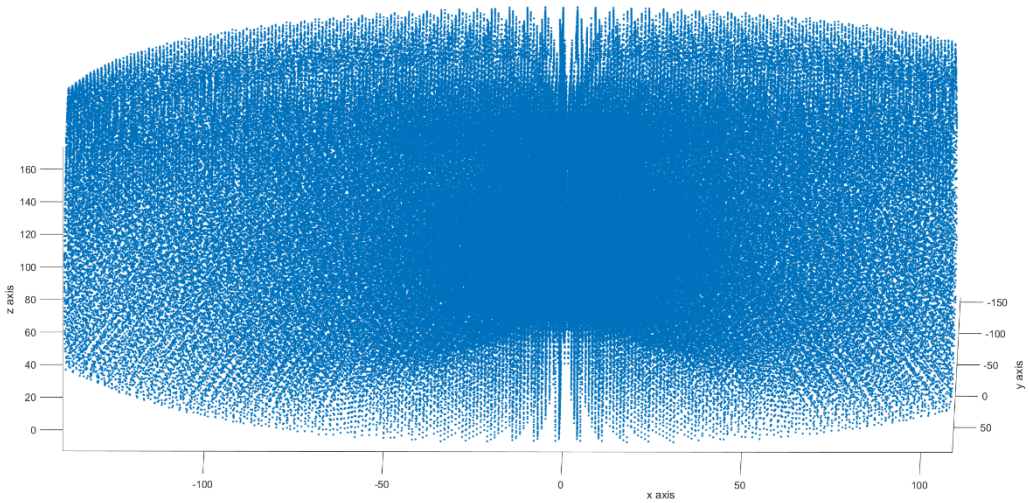


Figure 33.Right Side View of the 3D Working Area of the Robot Arm with 5X zoom

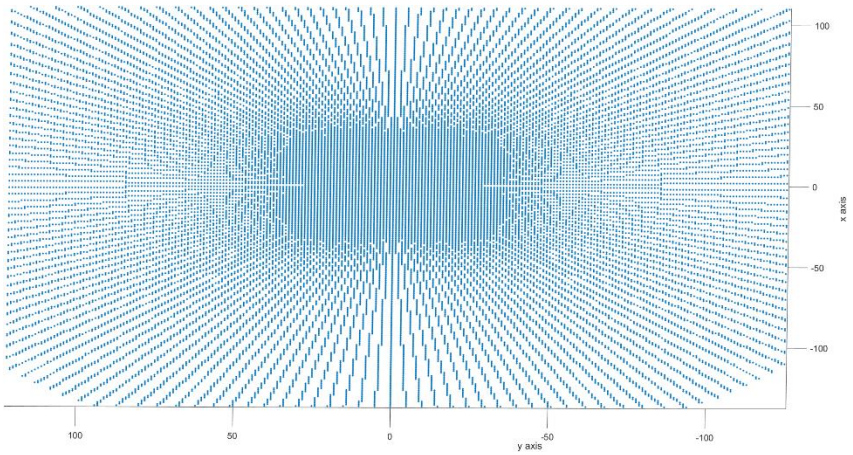


Figure 34.Top View of the 3D Working Area of the Robot Arm with 5X zoom

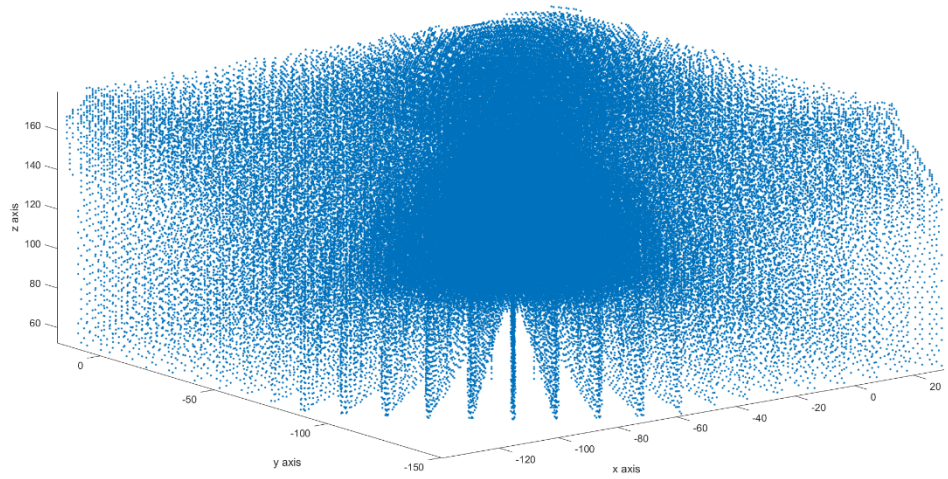


Figure 35.Iso View of the 3D Working Area of the Robot Arm with 15X zoom

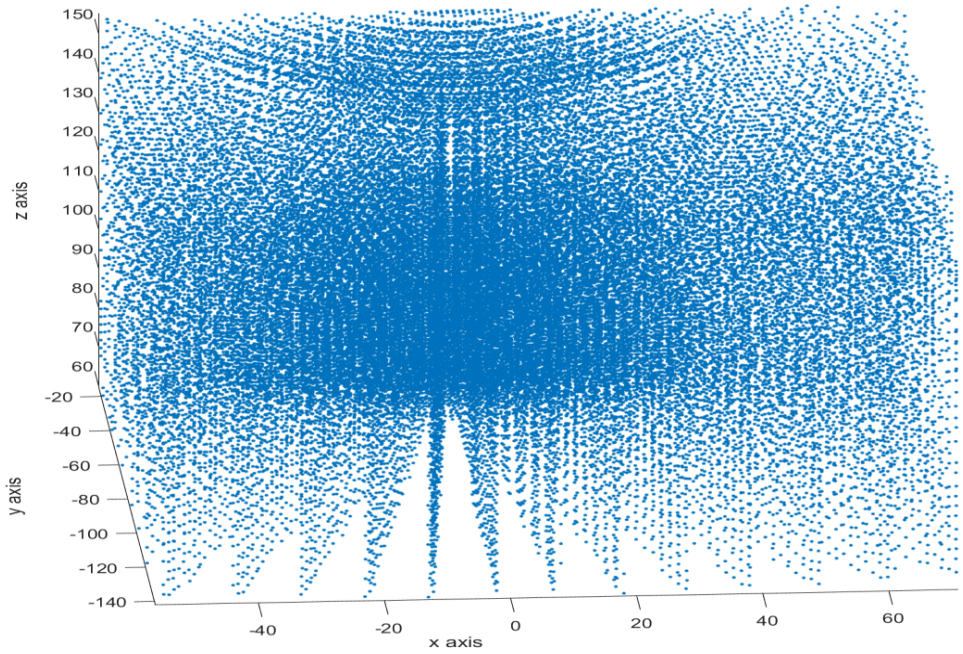


Figure 36.Front View of the 3D Working Area of the Robot Arm with 15X zoom

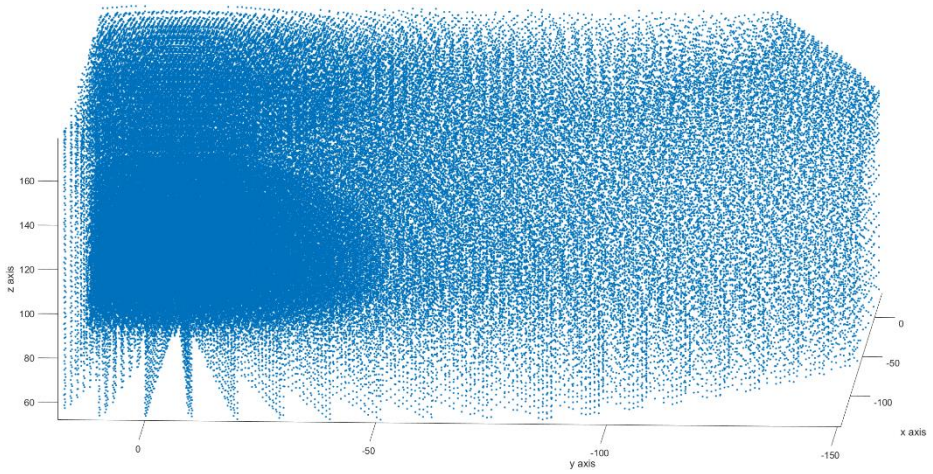


Figure 37.Left Side View of the 3D Working Area of the Robot Arm with 15X zoom

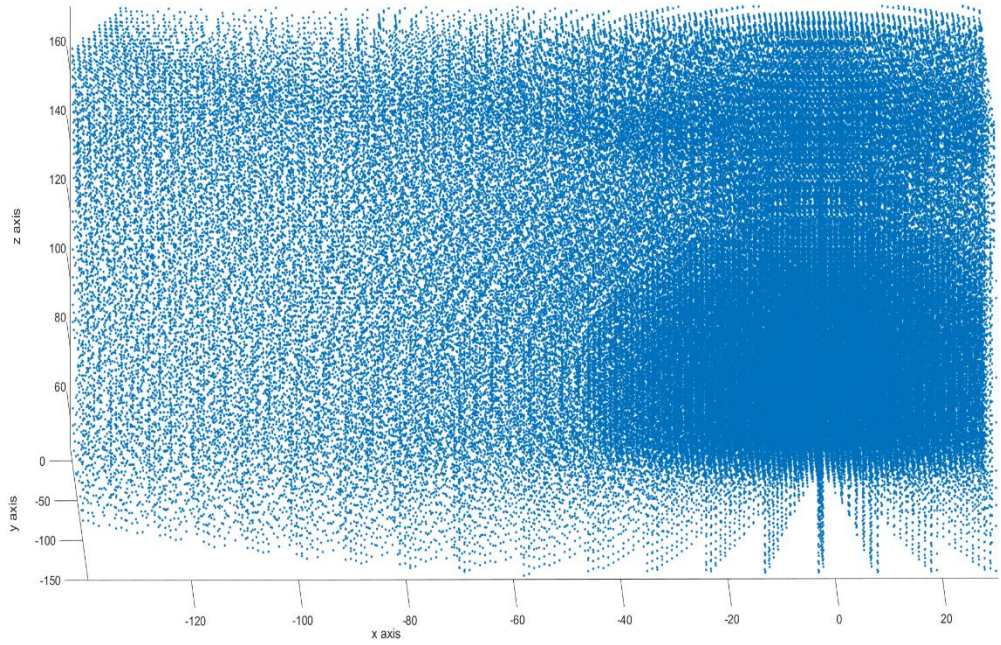


Figure 38.Right Side View of the 3D Working Area of the Robot Arm with 15X zoom

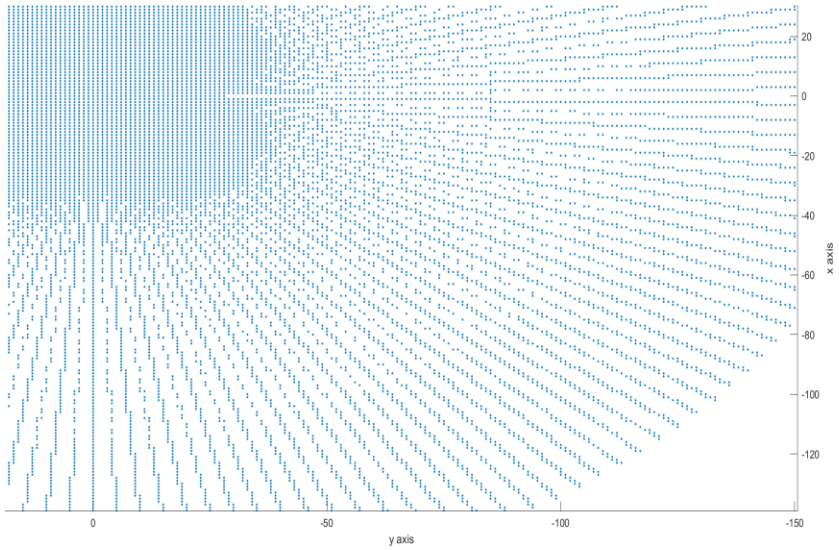


Figure 39.Top View of the 3D Working Area of the Robot Arm with 15X zoom

In addition, a 2D working area of the robot arm was created using nested for loops in MATLAB.

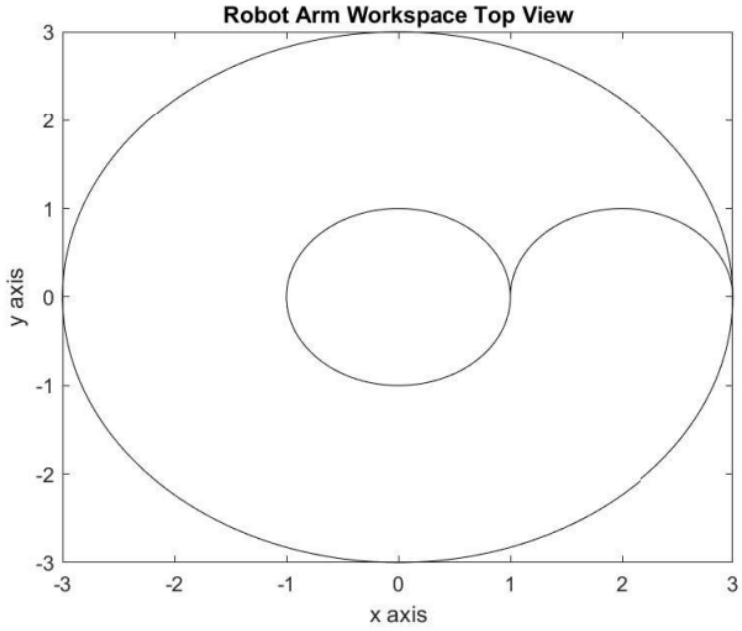


Figure 40.2D Workspace of Robot Arm Top View

Angle	Minimum Angle	Maximum Angle
Θ_1	0	360
Θ_2	0	180
Θ_3	0	210
Θ_4	-90	90

Table 2.Minimum-Maximum angles table

5. SETTING UP CONTROLLER, SIMULATION AND ROS

Figure 41.Finalized design in Fusion 360

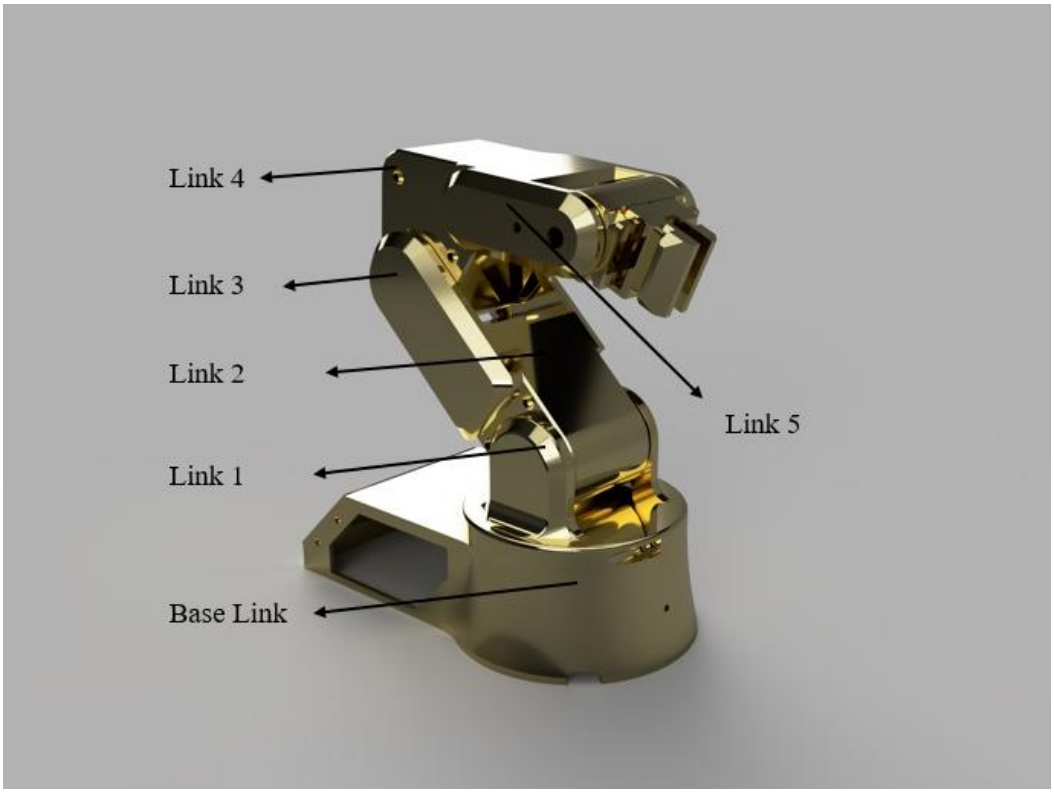
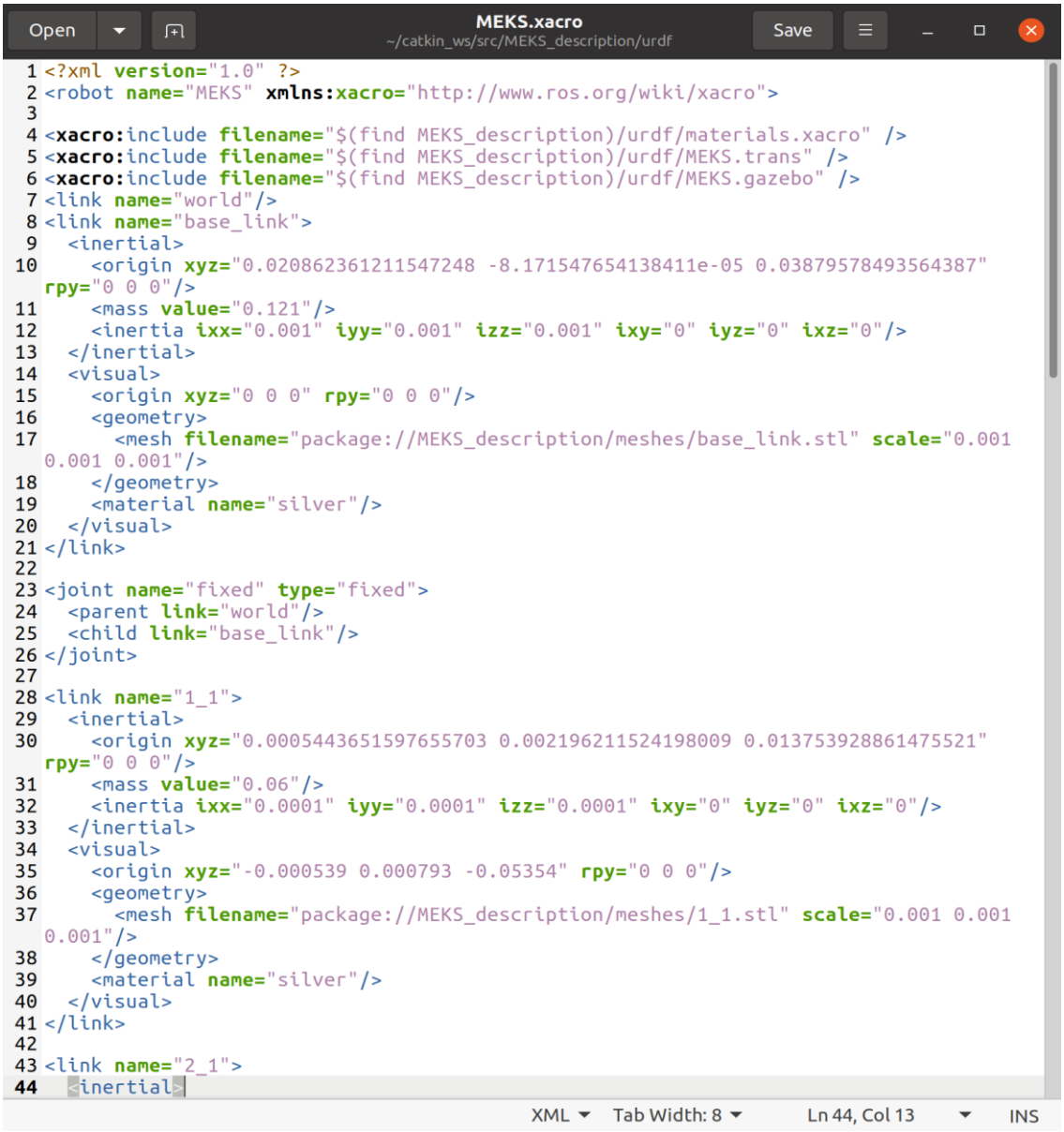


Figure 42.4-DoF Manipulator CAD Design

Upon completing the design of our robot arm, taking into account both the constraints and design parameters, we have successfully developed a 3D printable, 4-degree-of-freedom (4-DoF) robot arm manipulator.

All of this designing process has been done in Fusion 360 but in order to control and simulate our robot in ROS environment [4], we need to transfer the design into an URDF (Universal Robot Description File) and to do that we used an open-source script called “fusion2urdf” by syuntoku14.




```
<?xml version="1.0" ?>
<robot name="MEKS" xmlns:xacro="http://www.ros.org/wiki/xacro">

<xacro:include filename="$(find MEKS_description)/urdf/materials.xacro" />
<xacro:include filename="$(find MEKS_description)/urdf/MEKS.trans" />
<xacro:include filename="$(find MEKS_description)/urdf/MEKS.gazebo" />
<link name="world"/>
<link name="base_link">
  <inertial>
    <origin xyz="0.020862361211547248 -8.171547654138411e-05 0.03879578493564387"
rpy="0 0 0"/>
    <mass value="0.121"/>
    <inertia ixx="0.001" iyy="0.001" izz="0.001" ixy="0" iyz="0" ixz="0"/>
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://MEKS_description/meshes/base_link.stl" scale="0.001
0.001 0.001"/>
    </geometry>
    <material name="silver"/>
  </visual>
</link>

<joint name="fixed" type="fixed">
  <parent link="world"/>
  <child link="base_link"/>
</joint>

<link name="1_1">
  <inertial>
    <origin xyz="0.0005443651597655703 0.002196211524198009 0.013753928861475521"
rpy="0 0 0"/>
    <mass value="0.06"/>
    <inertia ixx="0.0001" iyy="0.0001" izz="0.0001" ixy="0" iyz="0" ixz="0"/>
  </inertial>
  <visual>
    <origin xyz="-0.000539 0.000793 -0.05354" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://MEKS_description/meshes/1_1.stl" scale="0.001 0.001
0.001"/>
    </geometry>
    <material name="silver"/>
  </visual>
</link>

<link name="2_1">
  <inertial>
```

Figure 43.URDF Description of the robot.

This script automatically created the URDF, TRANSMISSIONS, CONTROLLER and GAZEBO codes for us and then all we needed to do was to make some corrections with controller and .launch file.

Open *controllers.yaml ~/catkin_ws/src/MEKS_description/yaml Save

```yaml
arm_controller:
  type: "position_controllers/JointTrajectoryController"
  joints:
    - j1
    - j2
    - j3
    - j4
    - j5
```

Figure 44.Controllers defined for each joint.

Open MEKS.trans ~/catkin_ws/src/MEKS_description/urdf Save

```xml
<?xml version="1.0" ?>
<robot name="MEKS" xmlns:xacro="http://www.ros.org/wiki/xacro" >

<transmission name="j1_tran">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="j1">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="j1_actr">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>

<transmission name="j2_tran">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="j2">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="j2_actr">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>

<transmission name="j3_tran">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="j3">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="j3_actr">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>

<transmission name="j4_tran">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="j4">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="j4_actr">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

Figure 45.Transmission defined for each joint.

Open *gazebo.launch ~/catkin_ws/src/MEKS_description/launch Save

```
<launch>
  <param name="robot_description" command="$(find xacro)/xacro $(find
MEKS_description)/urdf/MEKS.xacro"/>
  <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" args="-param
robot_description -urdf -model MEKS"/>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>
  <rosparam file="$(find MEKS_description)/yaml/controllers.yaml" command="load"/>
  <node name="controller_spawner" pkg="controller_manager" type="spawner"
args="arm_controller"/>
  <node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher"/>
</launch>
```

Figure 46.Simulation configuration with various parameters.

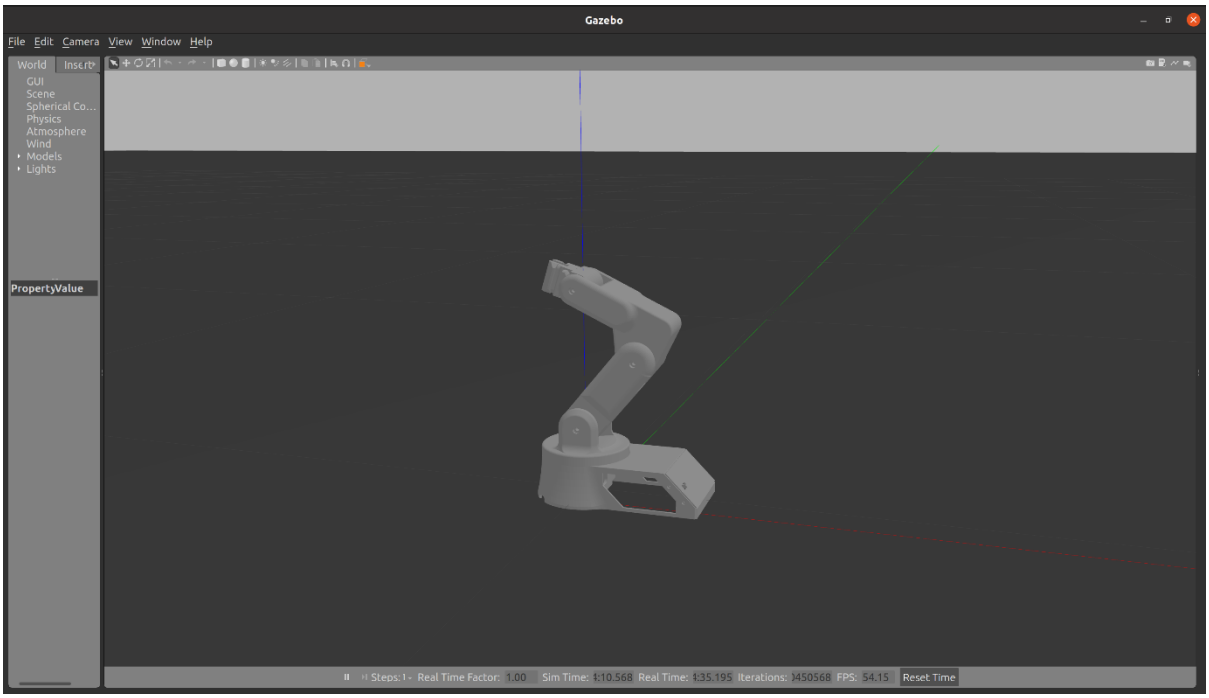


Figure 47.Robot in Gazebo simulation.

As we can see in the Figure 47, it's a success. Since now we have the simulation environment and control parameters ready, we can go and start to implement the control system by creating a Moveit[6] Configuration file from Moveit Setup Assistant. After we import our URDF file, we can start customizing our control settings like show in the figures.

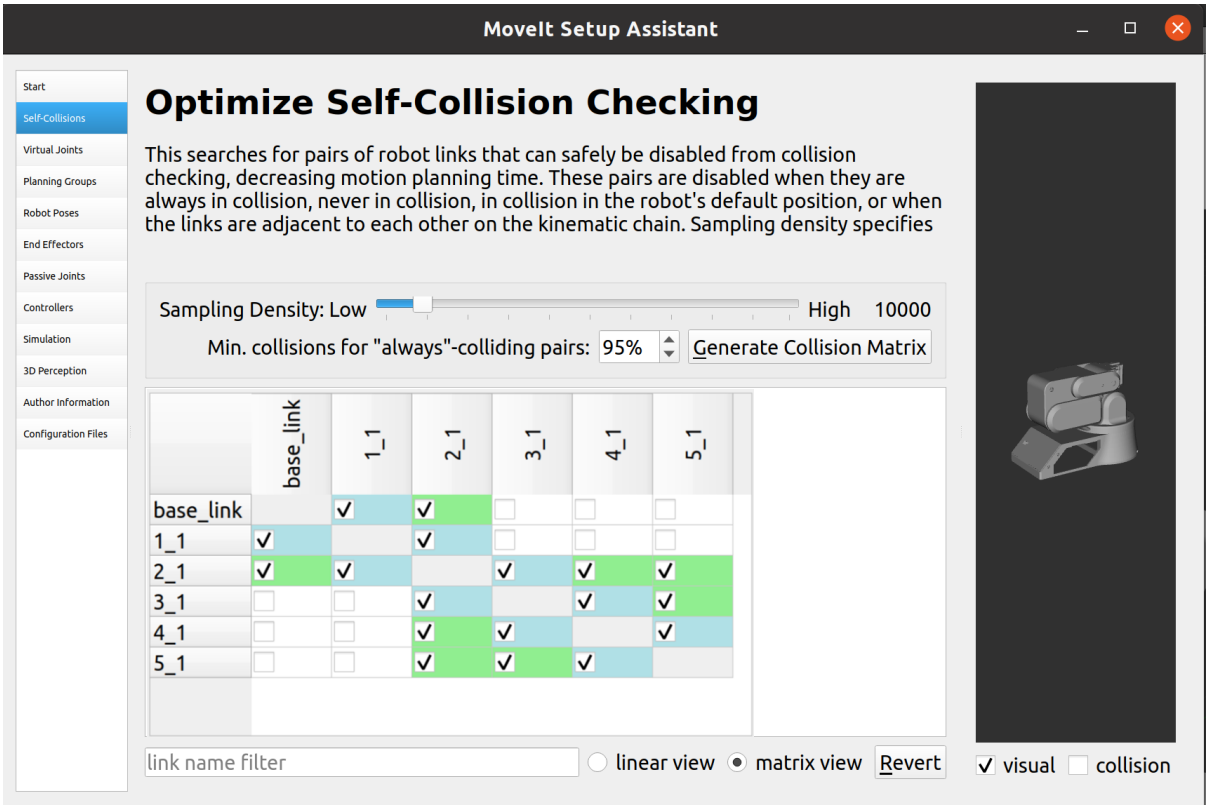


Figure 48.Auto-generated self-collision matrix.

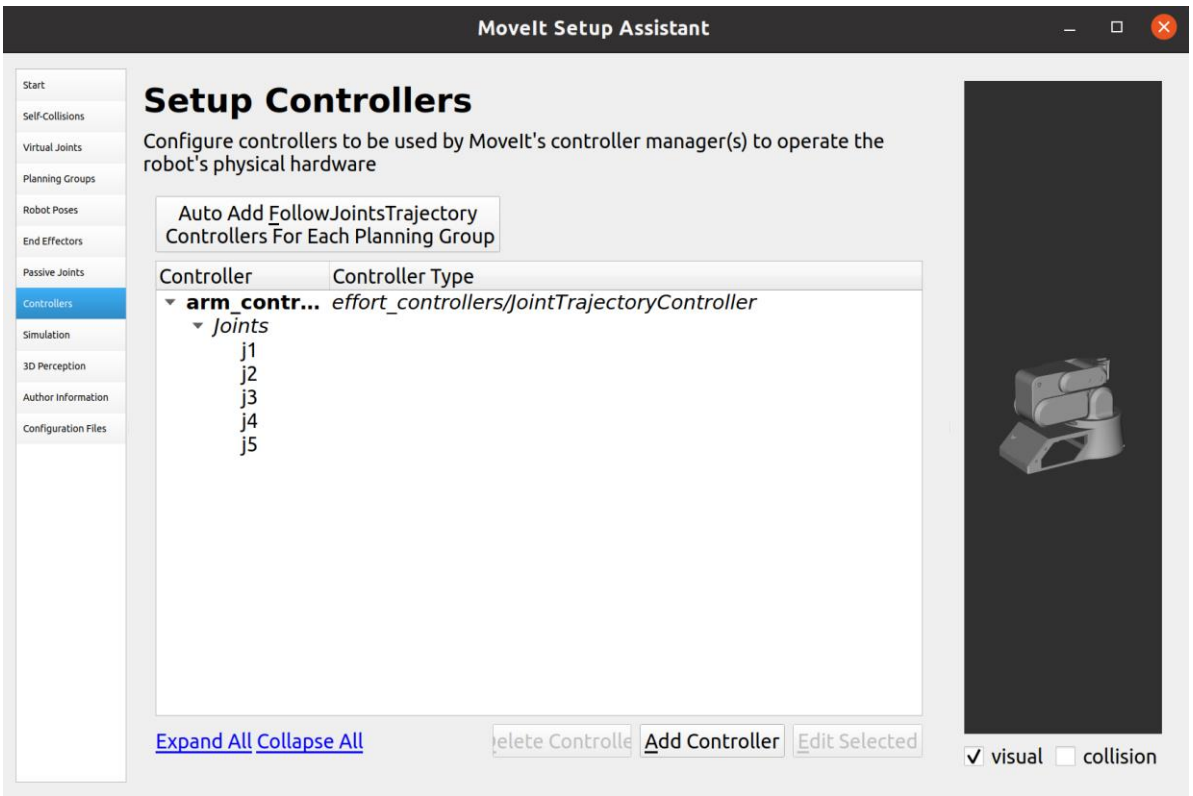


Figure 49.Controller configuration for each joint.

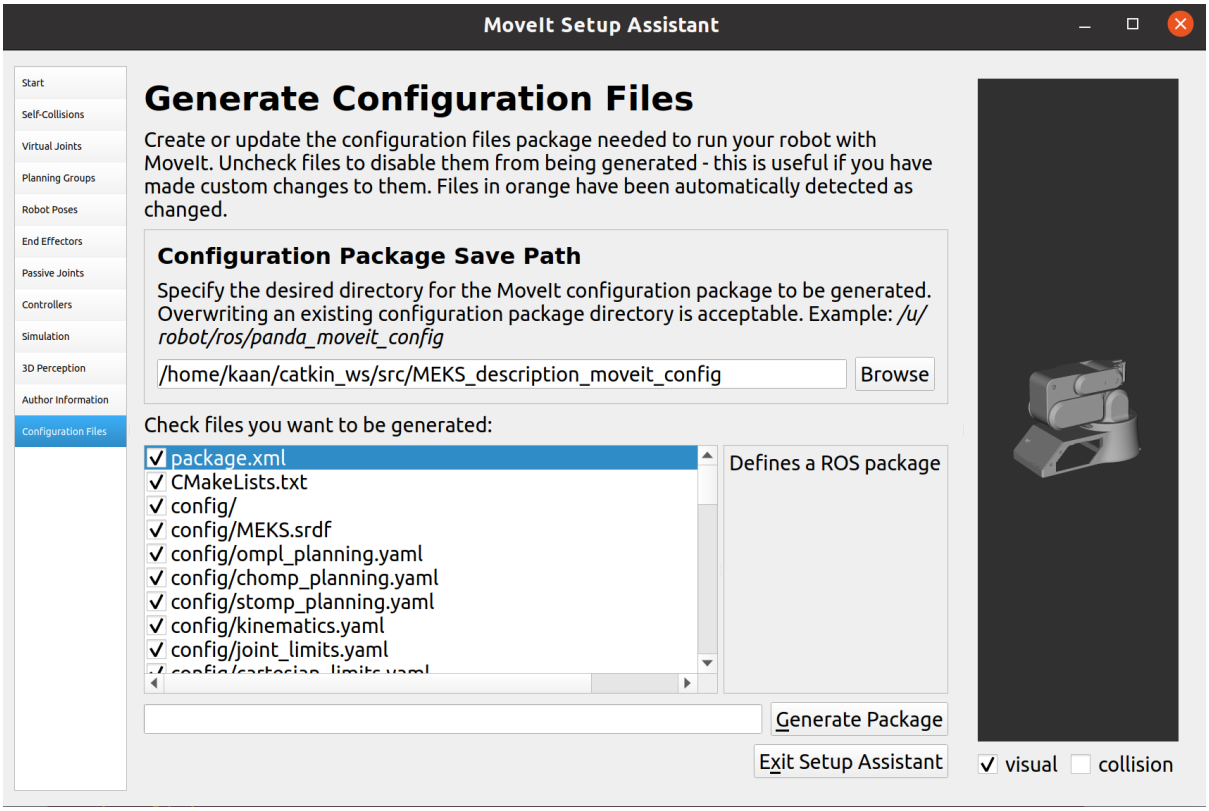


Figure 50.All packages included in Moveit along with save location.

With this configuration, we've established a secondary folder named 'EIKR_description_moveit_config' to store all the necessary configurations. These configuration files include 'moveit_group.launch,' which acts as an intermediary between Rviz and the GAZEBO simulation environment. Additionally, the 'moveit_rviz.launch' launch file ensures the consistent launch of Rviz with the correct control parameters every time.

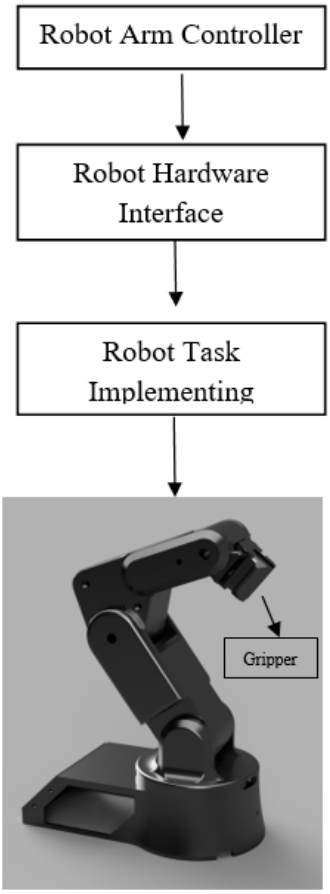


Figure 51.Flow chart from Robot Arm control part to vacuum gripper.

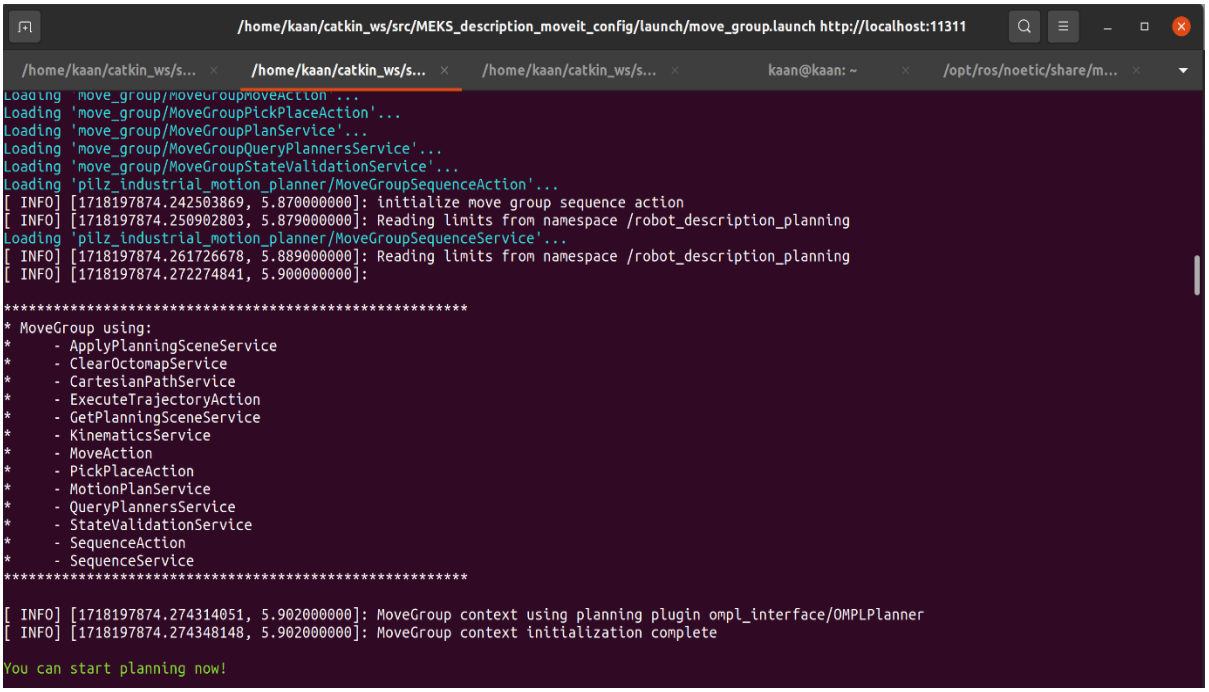


Figure 52. move_group node in action.

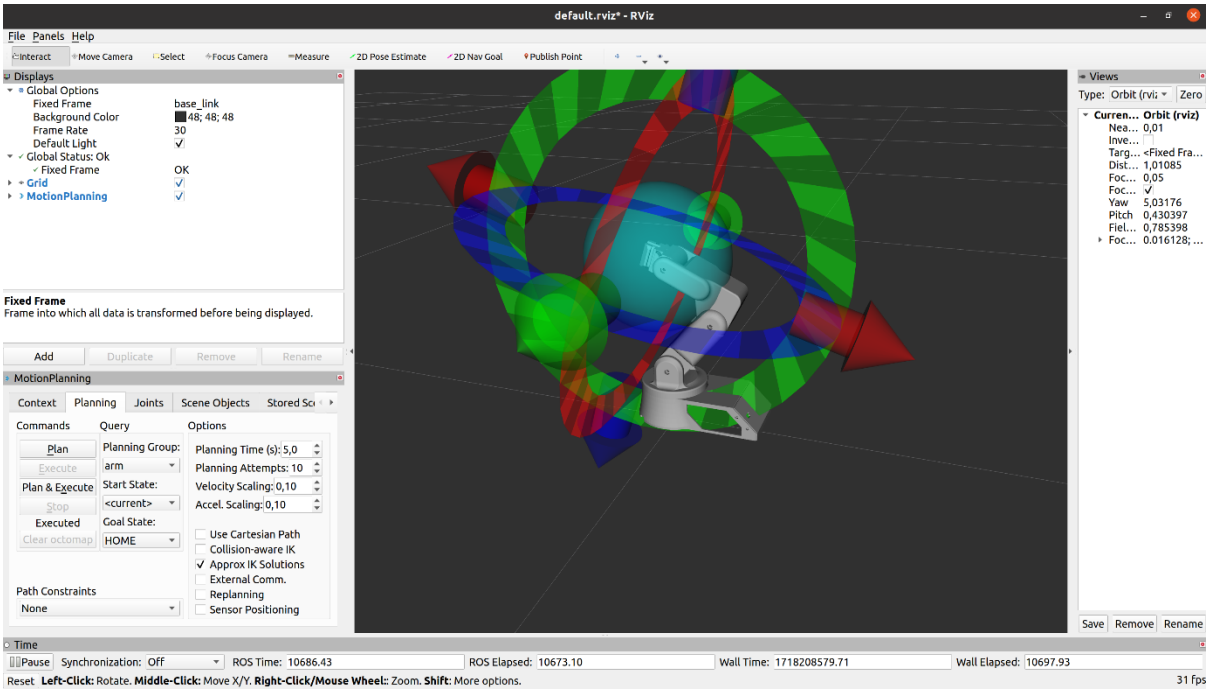


Figure 53. Rviz with MotionPlanning plugin working.

As we can see in the Figure 53 shown above, now we have GAZEBO simulation, Rviz [5] to use Motion Planing [6] and an intermediary between them called “moveit_group.launch” in terminal.

Utilizing the Motion Planning plugin, we can effortlessly position the end effector at any desired location. Simply by clicking 'plan and execute,' we can witness the execution of the desired motion in the GAZEBO simulation environment.

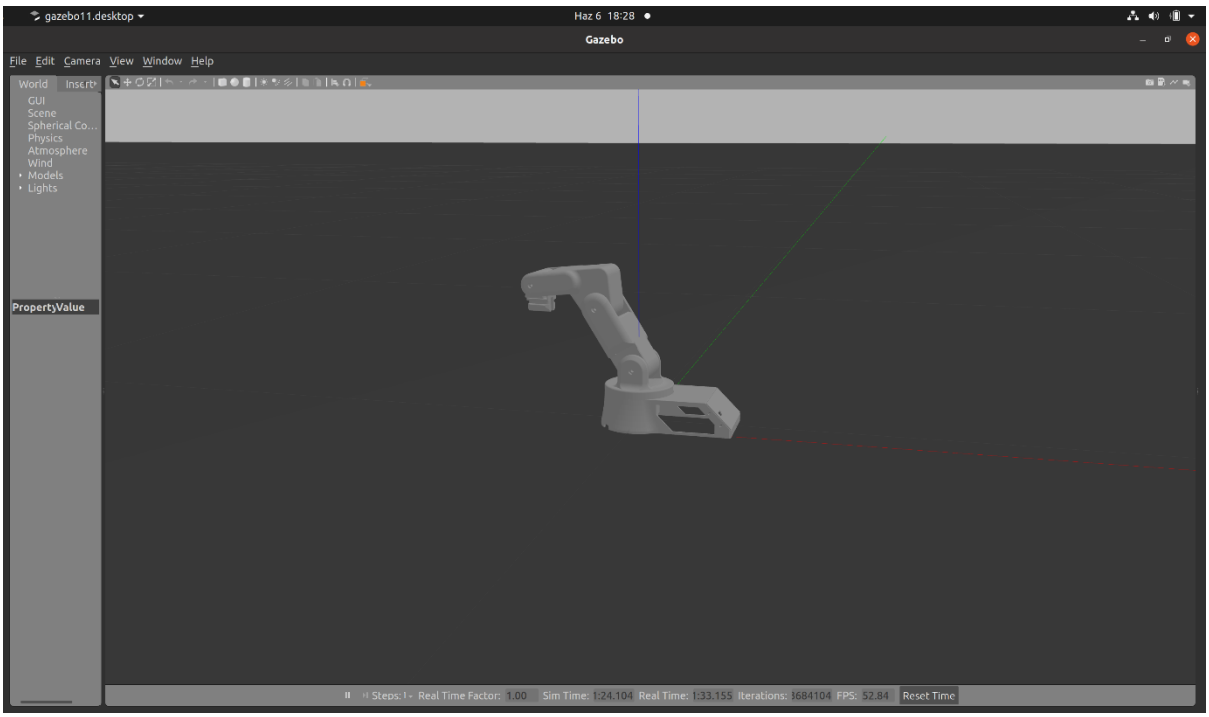


Figure 54. Simulation after the new trajectory for to the robot applied.

Here are some schematics to better understand which nodes and topics are used while the control system is working as a whole.

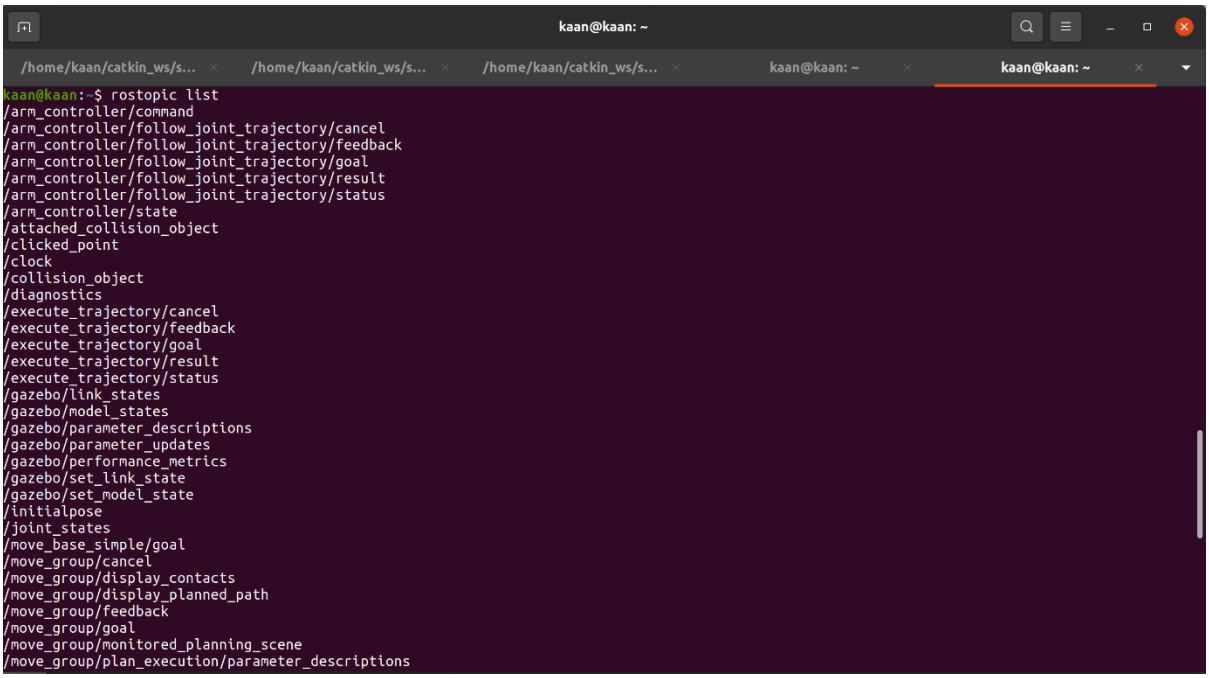


Figure 55. List of topics that are interacting with each other.

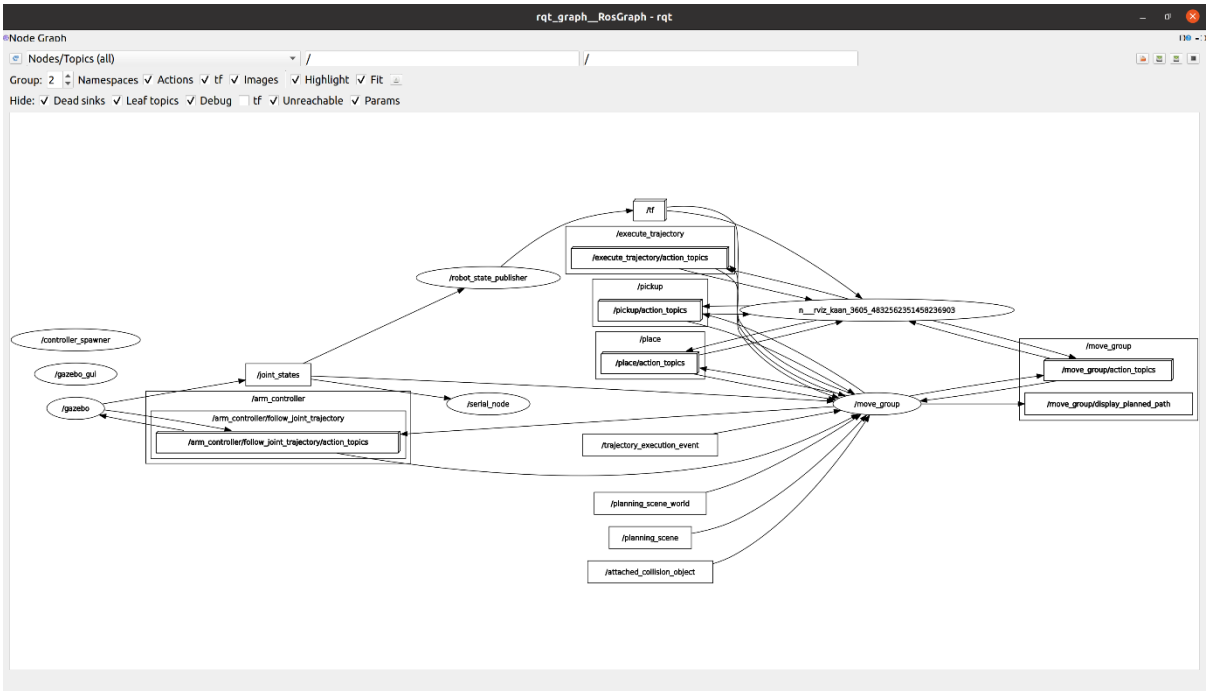


Figure 56. All nodes in the system that are interacting with each other via topics.

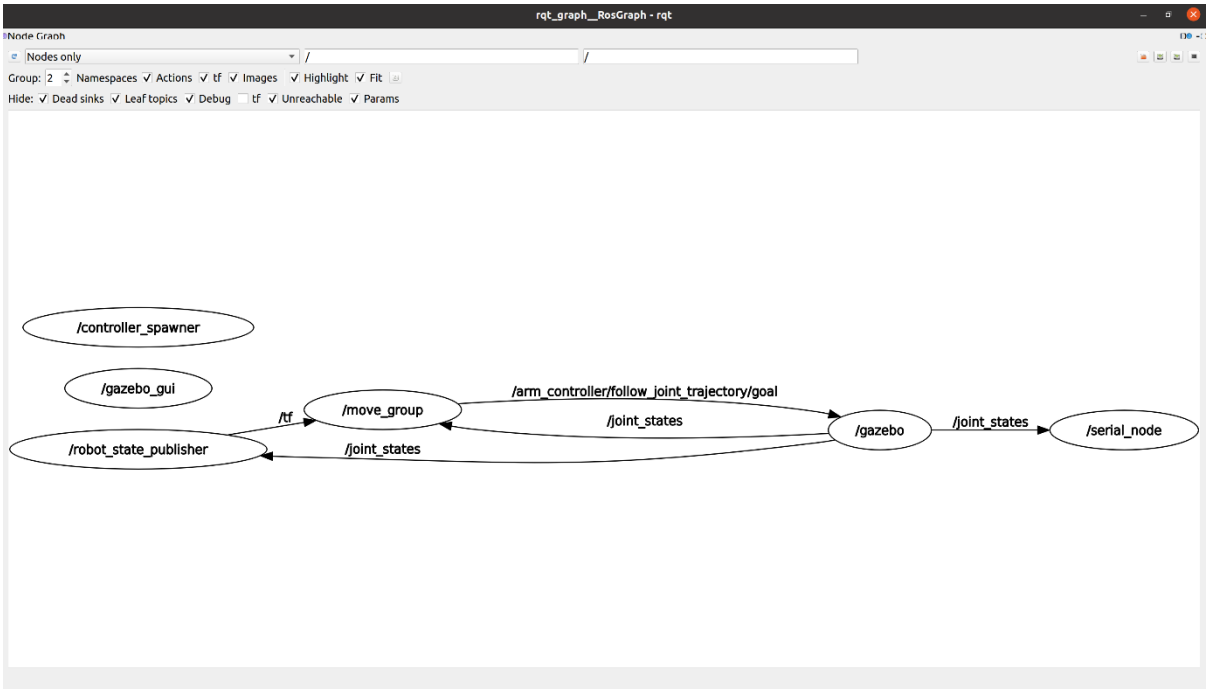


Figure 57. All relevant nodes of the control system.

6. BILL OF MATERIALS

#	Part Name	Price	Number	Total
1	ESP32 ESP-32S WiFi + Bluetooth Dual-Mode Development Board (30 Pin)	302,83 ₺	1	302,82 ₺
2	Hiwonder LX-16A Full Metal Gear Serial Bus Servo with Real-Time Feedback Function for RC Robot(Control Angle 240)	16.99 $	4	67.96 $
3	Hiwonder TTL / USB Debugging Board	12.99 $	1	12.99 $
4	Tower Pro MG90S Micro Servo Motor	117 ₺	1	117 ₺
5	Single Core Assembly Cable 50 m and Some Electronic Component (Switch etc.)	220 ₺	1	220 ₺
6	Microzey 1.75 mm Pla Pro Filament 1 kg	480,99 ₺	1	480.99 ₺
7	Mechanical Fasteners (Bolts, Nuts, Screws, etc.)	350 ₺	1	350 ₺
8	**Total**			**4308.55 ₺**

Table 3. Bill of Materials

7. PARTS PRINTED FROM 3D PRINTER

Developing a cost-effective robot has been a key highlight of our prototyping process. To achieve this, all parts of the robot arm, particularly the end effector mechanism, were manufactured using a 3D printer.

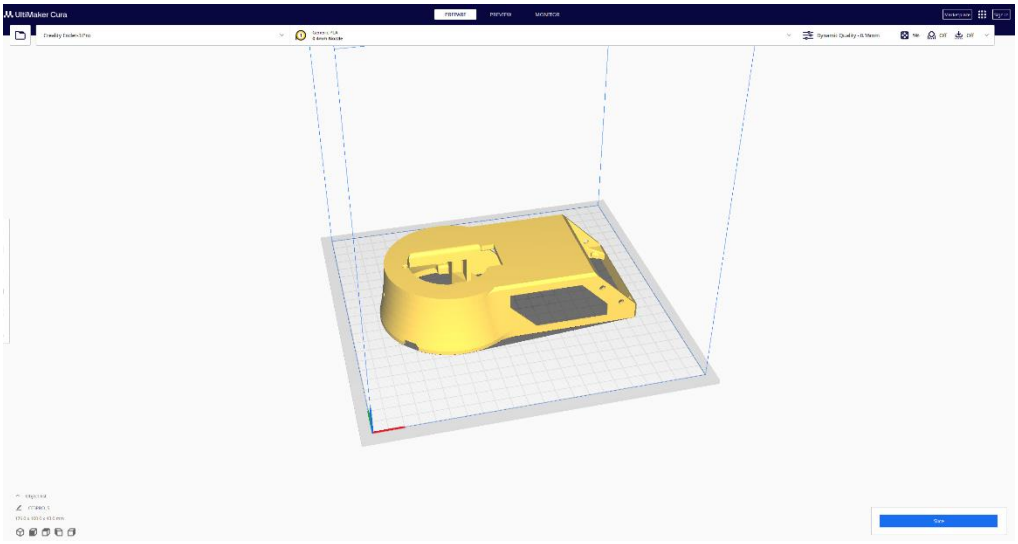


Figure 58. The base part of the robot from 3D printable parts in the slicing software

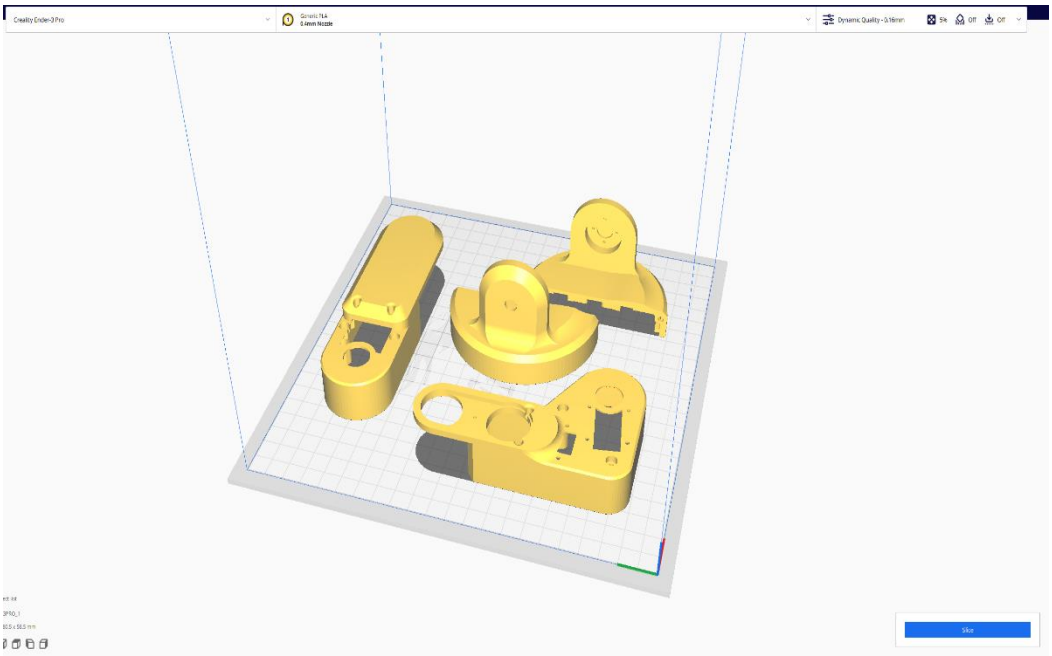


Figure 59. 3D Printable parts of the robot in the slicing software

8. FINAL VIEW OF FUSION 360 DESIGN

Figure 60. Final Design of Robot Arm

Figure 61. 4-DoF Manipulator Real World Design

Figure 62. Motor assembly in Link 3

Figure 63. 3D printed parts of the robot

9. RESULT AND CONCLUSION

In conclusion, this graduation project aimed to design and control a 4-DoF robot arm manipulator using the Robot Operating System (ROS). The primary objective is to enhance industrial maintenance processes by eliminating the need for on-site operators and enabling remote troubleshooting capabilities. To achieve this

Firstly, the mechanical design of our robot was carried out with our design needs, then kinematic calculations of our robot were made to ensure that further implementation of ROS would be accurate. Later, the working area of our robot was determined on Matlab by using our previous direct kinematic calculations. With all the calculations and initial design parameters ready. Through the utilization of GAZEBO, Rviz, and Motion Planning-like plugins within Rviz, along with the integration of an accelerometer for real-time feedback, the robot arm demonstrates efficient and precise movements.

The project successfully addresses the limitations of existing robot arms by implementing an open-source approach with ROS, allowing for transparency in control and troubleshooting. The developed control system, based on PID controllers and kinematic calculations, facilitated remote planning, monitoring, and control of the robot arm's movements. The methodology employed in the project encompassed various stages, from the design and 3D printing of the robot arm to the development of the control system using ROS. The use of Fusion 360 for design, along with the fusion2urdf script, simplified the transition from design to simulation in ROS.

Acknowledgments goes to Etkin Medical Devices company for sponsoring the project and to the project supervisor, Fatih Cemal Can, for continuous guidance. The support received from professors Erkin Gezgin, Özgün Başer, and Mertcan Koçak in overcoming challenges during the project is also greatly appreciated.

In summary, this project not only met its objectives but also contributed to the advancement of open-source robotics and automation. The developed robot arm showcases the potential for enhanced, flexible, and dynamic production processes, aligning with the vision of Industry 4.0. The knowledge gained and the solutions devised throughout this project lay the foundation for future innovations in the realm of industrial robotics and automation.

10. REFERENCES

[1] Corke, Peter (2013) “Robotics, Vision and Control” (pp. 1-6). Springer-Verlag Berlin Heidelberg

[2] “Why teach robotics using ROS”. Retrieved January 30, 2016 from https://www.researchgate.net/publication/274267022_Why_teach_robotics_using_ROS

[3] Asad Yousuf, Savannah State University; William Lehman, Bill's Robotic Solutions; Mir Hayder, Savannah State University; Mohamad Mustafa, Savannah State University, “Introducing Kinematics into Robotic Operating Systems”, pg. 39-47, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH AND INNOVATION | V7, N2, FALL/WINTER 2015

[4]"Robust Control Platform for Robotic Arm Using ROS "
https://www.academia.edu/44251986/Controlling_Platform_for_Robotic_Arm_Using_ROS

[5] "Rviz" http://wiki.ros.org/rviz

[6] Deng, H., Xiong, J., and Xia, Z. Mobile manipulation task simulation using ROS with MoveIt. IEEE International Conference on Real-time Computing and Robotics (RCAR) IEEE.2017, ;pp. 612-616. doi: 10.1109/rcar.2017.8311930.

[7] "ESP32 "ESP32 ESP-32S WiFi + Bluetooth Dual-Mode Geliştirme Kartı (30 Pin) Satın Al | Robotistan

[8]"Robot Grippers and End Effectors: Uses, Benefits, and Cost Analysis" Robot Grippers and End Effectors: Uses, Benefits, and Cost Analysis | HowToRobot

11. APPENDICES

11.1 MEKS.xacro:

```
<?xml version="1.0" ?>
 <robot name="MEKS" xmlns:xacro="http://www.ros.org/wiki/xacro">

<xacro:include filename="$(find
MEKS_description)/urdf/materials.xacro" />
 <xacro:include filename="$(find MEKS_description)/urdf/MEKS.trans"
/>
 <xacro:include filename="$(find MEKS_description)/urdf/MEKS.gazebo"
/>
 <link name="world"/>
 <link name="base_link">
   <inertial>
     <origin xyz="0.020862361211547248 -8.171547654138411e-05
0.03879578493564387" rpy="0 0 0"/>
     <mass value="0.121"/>
     <inertia ixx="0.001" iyy="0.001" izz="0.001" ixy="0" iyz="0"
ixz="0"/>
   </inertial>
   <visual>
     <origin xyz="0 0 0" rpy="0 0 0"/>
     <geometry>
       <mesh
filename="package://MEKS_description/meshes/base_link.stl"
scale="0.001 0.001 0.001"/>
     </geometry>
     <material name="silver"/>
   </visual>
 </link>

<joint name="fixed" type="fixed">
   <parent link="world"/>
   <child link="base_link"/>
```

```
 </joint>

<link name="1_1">
   <inertial>
     <origin xyz="0.0005443651597655703 0.002196211524198009
0.013753928861475521" rpy="0 0 0"/>
     <mass value="0.06"/>
     <inertia ixx="0.0001" iyy="0.0001" izz="0.0001" ixy="0" iyz="0"
ixz="0"/>
   </inertial>
   <visual>
     <origin xyz="-0.000539 0.000793 -0.05354" rpy="0 0 0"/>
     <geometry>
       <mesh filename="package://MEKS_description/meshes/1_1.stl"
scale="0.001 0.001 0.001"/>
     </geometry>
     <material name="silver"/>
   </visual>
 </link>

<link name="2_1">
   <inertial>
     <origin xyz="0.04520803367712563 0.03256019909616021 -
0.0004035076284029243" rpy="0 0 0"/>
     <mass value="0.08"/>
     <inertia ixx="0.0002" iyy="0.0002" izz="0.0002" ixy="0" iyz="0"
ixz="0"/>
   </inertial>
   <visual>
     <origin xyz="-0.001342 0.028639 -0.08604" rpy="0 0 0"/>
     <geometry>
       <mesh filename="package://MEKS_description/meshes/2_1.stl"
scale="0.001 0.001 0.001"/>
     </geometry>
     <material name="silver"/>
   </visual>
 </link>

<link name="3_1">
   <inertial>
     <origin xyz="-0.02570348224599696 0.026637375836800703
0.02642025643075306" rpy="0 0 0"/>
     <mass value="0.07"/>
     <inertia ixx="0.0002" iyy="0.0002" izz="0.0002" ixy="0" iyz="0"
ixz="0"/>
   </inertial>
   <visual>
     <origin xyz="-0.091284 0.025327 -0.08604" rpy="0 0 0"/>
     <geometry>
       <mesh filename="package://MEKS_description/meshes/3_1.stl"
scale="0.001 0.001 0.001"/>
     </geometry>
     <material name="silver"/>
   </visual>
 </link>
```

```
<link name="4_1">
   <inertial>
     <origin xyz="-0.00901843254806367 0.024299437151006624 -
5.749457711363981e-05" rpy="0 0 0"/>
     <mass value="0.04"/>
     <inertia ixx="0.0001" iyy="0.0001" izz="0.0001" ixy="0" iyz="0"
ixz="0"/>
   </inertial>
   <visual>
     <origin xyz="-0.00855 0.025489 -0.12154" rpy="0 0 0"/>
     <geometry>
       <mesh filename="package://MEKS_description/meshes/4_1.stl"
scale="0.001 0.001 0.001"/>
     </geometry>
     <material name="silver"/>
   </visual>
 </link>

<link name="5_1">
   <inertial>
     <origin xyz="-0.00025020454249763055 -6.97824175790074e-06 -
2.414043697579471e-07" rpy="0 0 0"/>
     <mass value="0.001"/>
     <inertia ixx="0.00001" iyy="0.00001" izz="0.00001" ixy="0"
iyz="0" ixz="0"/>
   </inertial>
   <visual>
     <origin xyz="0.009678 0.00076 -0.12154" rpy="0 0 0"/>
     <geometry>
       <mesh filename="package://MEKS_description/meshes/5_1.stl"
scale="0.001 0.001 0.001"/>
     </geometry>
     <material name="silver"/>
   </visual>
 </link>

<joint name="j1" type="continuous">
   <origin xyz="0.000539 -0.000793 0.05354" rpy="0 0 0"/>
   <parent link="base_link"/>
   <child link="1_1"/>
   <axis xyz="-0.0 -0.0 1.0"/>
   <limit upper="1.31" lower="-1.31" effort="100" velocity="100"/>
 </joint>

<joint name="j2" type="continuous">
   <origin xyz="0.000803 -0.027846 0.0325" rpy="0 0 0"/>
   <parent link="1_1"/>
   <child link="2_1"/>
   <axis xyz="0.029154 -0.999575 0.0"/>
   <limit upper="2.6" lower="0" effort="100" velocity="100"/>
 </joint>

<joint name="j3" type="continuous">
   <origin xyz="0.089942 0.003312 0.0" rpy="0 0 0"/>
```

```
   <parent link="2_1"/>
   <child link="3_1"/>
   <axis xyz="0.029154 -0.999575 0.0"/>
   <limit upper="0" lower="-2" effort="100" velocity="100"/>
 </joint>

<joint name="j4" type="continuous">
   <origin xyz="-0.082734 -0.000162 0.0355" rpy="0 0 0"/>
   <parent link="3_1"/>
   <child link="4_1"/>
   <axis xyz="0.029154 -0.999575 -0.0"/>
   <limit upper="1.5" lower="-1.5" effort="100" velocity="100"/>
 </joint>

<joint name="j5" type="continuous">
   <origin xyz="-0.018228 0.024729 0.0" rpy="0 0 0"/>
   <parent link="4_1"/>
   <child link="5_1"/>
   <axis xyz="-0.999575 -0.029154 -0.0"/>
 </joint>

</robot>
```

11.2 MEKS.trans

```
<?xml version="1.0" ?>
 <robot name="MEKS" xmlns:xacro="http://www.ros.org/wiki/xacro" >

<transmission name="j1_tran">
   <type>transmission_interface/SimpleTransmission</type>
   <joint name="j1">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwa
reInterface>
   </joint>
   <actuator name="j1_actr">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwa
reInterface>
     <mechanicalReduction>1</mechanicalReduction>
   </actuator>
 </transmission>

<transmission name="j2_tran">
   <type>transmission_interface/SimpleTransmission</type>
   <joint name="j2">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwa
reInterface>
```

```
    </joint>
    <actuator name="j2_actr">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwa
reInterface>
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
 </transmission>

<transmission name="j3_tran">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="j3">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwa
reInterface>
    </joint>
    <actuator name="j3_actr">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwa
reInterface>
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
 </transmission>

<transmission name="j4_tran">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="j4">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwa
reInterface>
    </joint>
    <actuator name="j4_actr">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwa
reInterface>
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
 </transmission>

<transmission name="j5_tran">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="j5">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwa
reInterface>
    </joint>
    <actuator name="j5_actr">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwa
reInterface>
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
 </transmission>

</robot>
```

11.3 Controllers.yaml

```
arm_controller:
  type: "position_controllers/JointTrajectoryController"
  joints:
    - J1
    - J2
    - J3
    - J4
    - J5
```

11.4 gazebo.launch

```
<launch>
  <param name="robot_description" command="$(find xacro)/xacro
$(find MEKS_description)/urdf/MEKS.xacro"/>
  <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
args="-param robot_description -urdf -model EIKR"/>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="paused" value="true"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>
</launch>
```

11.5 3D Workspaces of 4-Dof Robot Arm

```
%l3=a23 l4=a34 S1 Px=xM Py=yM Pz=zM
l3=84.5; l4=82.70; S1=75.5;
Q1=linspace(0,360,90)*pi/180;
Q2=linspace(0,180,90)*pi/180;
Q3=linspace(0,210,90)*pi/180;
Q4=linspace(-90,90,90)*pi/180;
[Q1,Q2,Q3]=ndgrid(Q1,Q2,Q3);
% This will create matrices of 90x90x90 for each variable
xM = round(cos(Q1).*(l3*cos(Q2)+l4.*cos(Q2+Q3))); %Px l3=a23 l4=a34
yM = round(sin(Q1).*(l3.*cos(Q2)+l4.*cos(Q2+Q3))); %Py
zM = round(S1+(l3.*sin(Q2))+(l4.*sin(Q2+Q3))); %Pz
plot3(xM(:),yM(:),zM(:),'.')
% This is the plot type you should be using.
% With a '.' as an argument to show only locations and not lines
% Also, (:) converts any matrix into a list of its elements in one single column.
```

11.6 2D Workspaces of 4-Dof Robot Arm

```
theta1_min=0;
theta1_max=360;
theta2_min=0;
theta2_max=180;
theta3_min=0;
theta3_max=210;
theta4_min=-90;
theta4_max=90;
% Workspace function is used to plot the boundaries of
% a 4 Dof Serial Robot Arm where:
% workspace(theta1_min, theta1_max, theta2_min, theta2_max, theta3_min,
theta3_max, theta4_min, theta4_max)
samples = 200;
a1=2;
a2=1.0;
theta1_start_end= pi*[theta1_min,theta1_max]/180;
theta2_start_end= pi*[theta2_min,theta2_max]/180;
theta3_start_end= pi*[theta3_min,theta3_max]/180;
theta4_start_end= pi*[theta4_min,theta4_max]/180;
theta1=pi*linspace(theta1_min,theta1_max,samples)/180;
theta2=pi*linspace(theta2_min,theta2_max,samples)/180;
theta3=pi*linspace(theta3_min,theta3_max,samples)/180;
theta4=pi*linspace(theta4_min,theta4_max,samples)/180;
x= zeros(2*length(theta1_start_end),length(theta2));
y= zeros(2-length(theta1_start_end),length(theta2));
for t=1:2
for ith1=1:length(theta1)
x(t,ith1)= a1*cos(theta1(ith1))+a2*cos(theta1(ith1)+theta2_start_end(t));
y(t,ith1)= a1*sin(theta1(ith1))+a2*sin(theta1(ith1)+theta2_start_end(t));
end
for ith2=1:length(theta1)
x(t+2,ith2)= a1*cos(theta1_start_end(t))+a2*cos(theta1_start_end(t)+theta2(ith2));
y(t+2,ith2)= a1*sin(theta1_start_end(t))+a2*sin(theta1_start_end(t)+theta2(ith2));
end
end
x=x';
y=y';
plot(x(:,1),y(:,1),'k')
hold on
plot(x(:,2),y(:,2),'k')
plot(x(:,3),y(:,3),'k')
plot(x(:,4),y(:,4),'k')
```