Tanta University

Faculty of Engineering

SCARA Robot
VIA Machine Vision

GRADUATION PROJECT TANTA UNIVERSITY

MECHATRONICS DEPARTMENT January 2025

AUTHORS:

Eslam Samy Mohamed

Amr Ali Sakr

Zeyad Mohamed Galal

Mohamed Samir ELbatal

Mahmoud Maged El Aidy

Ahmed Mohamed Shalash

Elsayed Kandil Shalaby

Mohamed Mamon Elassy

Mohamed Adel Elkomy

Supervised By:

Dr. Bikheet Mohamed Sayed

ABSTRACT

The Selective Compliance Assembly Robot Arm (SCARA) is a high-performance robotic system widely used in industrial automation for its speed, precision, and efficiency. This project focuses on the design, development, and implementation of a SCARA robot specifically tailored for pick-and-place operations, a critical task in manufacturing, packaging, and material handling industries.

The SCARA robot's unique kinematic design, which combines selective compliance in the horizontal plane with rigidity in the vertical axis, enables it to perform repetitive pick-and-place tasks with exceptional accuracy and reliability. This study explores the mechanical structure, control systems, and motion planning algorithms that optimize the robot's performance. Advanced features such as vision-based object detection and dynamic trajectory planning are integrated to enhance adaptability to varying operational conditions.

Through simulations and practical experiments, the SCARA robot's performance is evaluated in terms of speed, accuracy, and repeatability. Results demonstrate its ability to significantly reduce cycle times while maintaining high precision, making it an ideal solution for automation in industries requiring rapid and consistent material handling.

This work provides a comprehensive analysis of SCARA robots in pick-and-place applications, offering valuable insights for researchers, engineers, and industry professionals seeking to implement efficient robotic solutions in modern industrial environments.

Overview of SCARA Robot for Pick-and-Place Operations

The Selective Compliance Assembly Robot Arm (SCARA) is a specialized robotic system designed for high-speed and precise operations, particularly excelling in pick-and-place tasks. Its unique mechanical design provides selective compliance in the horizontal plane while maintaining rigidity in the vertical axis, making it ideal for repetitive and precise material handling tasks in manufacturing, packaging, and assembly industries.

Summary

The Selective Compliance Assembly Robot Arm (SCARA) is a specialized robotic system designed for high-speed and precise pick-and-place tasks. Its kinematic design allows selective compliance in the horizontal plane, enabling smooth and rapid lateral movements, while maintaining rigidity in the vertical axis for stable lifting and placement.

Key Features:

- **Speed and Efficiency:** SCARA robots are optimized for fast cycle times, making them ideal for repetitive pick-and-place operations.
- **Precision and Repeatability:** Advanced control systems ensure accurate handling and placement of objects.
- **Compact Design:** The small footprint allows operation in limited spaces, maximizing workspace efficiency.

Applications:

- **Electronics Manufacturing:** Handling components such as circuit boards and small electronic parts with precision.
- **Food and Beverage Packaging:** Sorting and placing items into packaging containers at high speeds.
- **Pharmaceuticals:** Precise handling of delicate products such as vials, syringes, or blister packs.
- **Automotive:** Assembling small components like screws, washers, and other parts.

Advantages:

- **Cost-Effective:** Combines high performance with affordability.
- **Flexibility:** Adaptable to various tasks with minimal reprogramming.
- **Reliability:** Consistent performance with low maintenance requirements.

SCARA robots are essential in industries requiring fast and accurate material handling. Their ability to enhance productivity and efficiency makes them a cornerstone of modern automation in pick-and-place operations.

Impact:

The implementation of SCARA robots in pick-and-place operations significantly improves productivity by automating repetitive tasks with high speed and precision. These robots reduce human errors, ensuring consistent quality and accuracy. Their compact design saves valuable floor space, enabling efficient integration into existing workflows. SCARA robots minimize downtime with easy programming and quick cycle times, enhancing overall operational efficiency. Additionally, they lower labor costs while increasing throughput, providing a fast return on investment. Their reliability and adaptability make them ideal for various industries, including packaging, electronics, and manufacturing.

Organization of the thesis

The thesis compromises five chapters as follows:

Chapter 1 shows the different designs of previous SCARA robots.

Chapter 2 shows the design of the robot using “SOLIDWORKS”, “MOTOR SIZING” and performing the “STRESS ANALYSIS” of each part.

Chapter 3 Electrical Components and Circuits.

Chapter 4 Robot control.

Chapter 5 Machine vision and its task.

Contents

Chapter 1
INTRODUCTION AND SURVEY

Chapter 1
INTRODUCTION AND SURVEY

1.1 Introduction

The Selective Compliance Assembly Robot Arm (SCARA) has emerged as a pivotal technology in industrial automation, renowned for its speed, precision, and reliability in repetitive tasks. Among its many applications, pick-and-place operations stand out as a critical use case, where efficiency and accuracy are paramount. By integrating advanced object detection technologies, SCARA robots are further enhanced, enabling them to identify, locate, and manipulate objects dynamically, even in unstructured or variable environments.

SCARA robots are designed with a unique kinematic structure that combines selective compliance in the horizontal plane with rigidity in the vertical axis. This design allows for rapid and precise lateral movements, making them ideal for tasks such as sorting, packaging, and assembly. The addition of object detection systems, such as vision-based cameras and sensors, provides the robot with the ability to adapt to diverse operational conditions, ensuring high performance in dynamic workflows.

This study explores the integration of SCARA robotics with object detection for pick-and-place applications, focusing on the design, control, and optimization of the system. Through advanced algorithms and real-time data processing, the robot achieves superior adaptability and efficiency, addressing the challenges of modern manufacturing and logistics. The findings highlight the transformative potential of combining SCARA robots with intelligent detection technologies, offering a scalable and cost-effective solution for industries seeking to enhance automation and productivity.

- **Why This Project?**

We chose the SCARA robot project because of its potential to improve industrial automation systems, especially in manufacturing processes that require speed, accuracy, and versatility. SCARA robots are well known for their high precision and efficiency in repetitive tasks.
Incorporating AI and a camera system into the robot adds another layer of intelligence, allowing for more adaptive and flexible behavior. The project will provide valuable experience in robotics, AI, and automation, preparing me for a career in advanced robotics and automation systems.

1.2 Survey

- **Serial Mechanism:**

In a serial mechanism, the robotic joints or links are connected in a linear arrangement, where each link affects the position and orientation of the next one in sequence. The movements occur one after another, requiring each part to reach its position before the next one begins moving. This can lead to slower performance but is simpler to implement and control. Serial mechanisms are generally used in tasks where precision is required, and the movements can be carried out step-by-step.

Key tasks in serial systems include:

- Gradual movement from one position to another
- Tasks that require specific sequencing of movements

- **Parallel Mechanism:**

In a parallel mechanism, multiple joints or links work simultaneously to achieve the desired position or orientation. This setup enables faster and more synchronized movements as all parts of the robot work in parallel. Parallel mechanisms are ideal for high-speed operations and applications requiring high precision and quick responses.

Key tasks in parallel systems include:

- Simultaneous movements of multiple joints
- High-speed tasks such as assembly lines or material handling

- **Comparison of Serial and Parallel Mechanisms:**

The main difference between serial and parallel mechanisms lies in how the robotic joints and links are configured and how they move. Serial mechanisms are simple to control and are ideal for precise, step-by-step operations. However, they tend to be slower. In contrast, parallel mechanisms offer faster and more efficient performance due to the simultaneous movements of multiple joints. Parallel systems are best suited for high-speed tasks and environments that require fast responses.

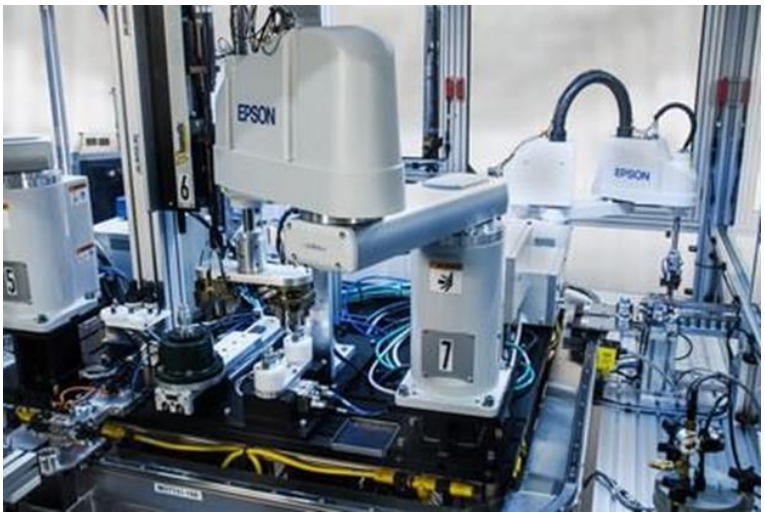
Feature	Serial Mechanism	Parallel Mechanism
Movement Type	Step-by-step, one joint moves at a time	Simultaneous movement of multiple joints
Speed	Slower, due to sequential movement	Faster, simultaneous movements
Complexity	Simpler to design and implement	More complex to design and implement
Use Case	Precise tasks, sequential movement	High-speed tasks, efficient simultaneous movements
Work Space	Limited workspace due to the sequential nature of movements. Best suited for compact, controlled environments.	Maximized workspace due to simultaneous joint actions. Ideal for wide, open, and dynamic environments.

1.2.1Similar SCARA Robot Projects:

- **Project 1: SCARA Robot for Automated Assembly:**

The robot's movements are primarily in the X and Y axes for precise placement, with Z-axis adjustments for height control. In this project, the SCARA robot was used for automated assembly of small electronic components. The robot's high precision allowed for rapid assembly of small parts on a production line.

- Problems faced: Lack of flexibility in changing tools for different components.
- Advantages: Increased production speed reduced human error.

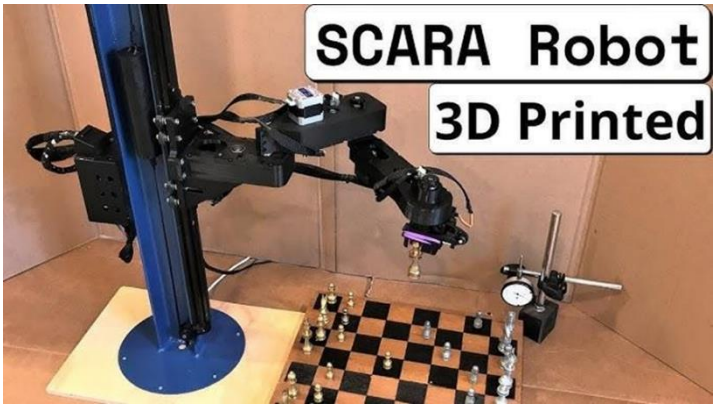


- **Project 2: SCARA Robot for 3D Printing:**

The robot operates in the X, Y, and Z axes. X and Y handle the path trajectory, while Z is used for layering during printing.

A SCARA robot was designed to assist in 3D printing, where its precision and flexibility enabled high-quality printing. The robot could move in precise paths and adjust the speed of printing based on the complexity of the model.

- Problems faced: Calibration issues when switching between print materials.
- Advantages: Enhanced printing speed and accuracy.



- **Project 3: SCARA Robot for Material Handling:**

  Movements involve X and Y for horizontal transportation, and Z for picking or placing items.

  This SCARA robot was used for material handling tasks, transporting items along assembly lines. The robot's speed and precision reduced operational downtime.

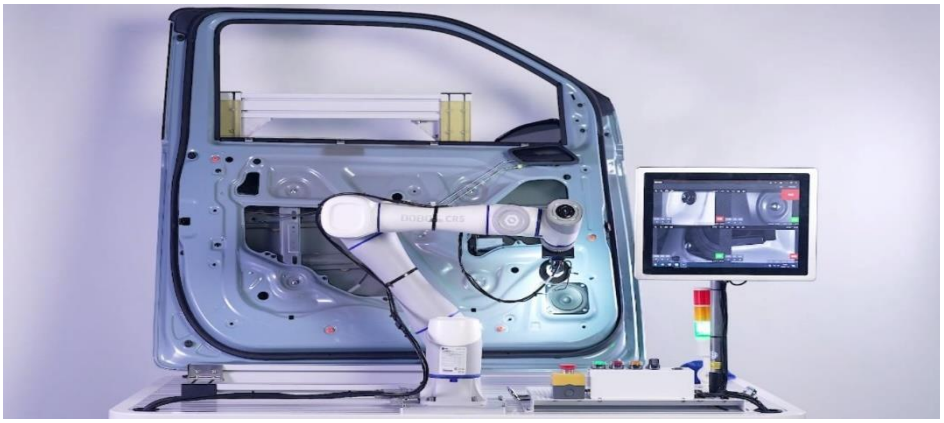
  - Problems faced: Difficulty in handling larger or heavier objects.
  - Advantages: Increased speed and efficiency in handling smaller materials

- **Project 4: SCARA Robot for Inspection and Quality Control:**

  The robot moves along the X and Y axes for positioning and uses the Z-axis for height variation. Additional adjustments include camera tilting.

  This project involved using a SCARA robot equipped with a camera to perform automated visual inspections of parts. The robot was programmed to detect defects in manufactured parts and sort them accordingly.

  - Problems faced: Integration issues with the AI software for defect detection.
  - Advantages: Reduced human inspection errors, faster processing time.



- **Project 5: SCARA Robot for Packaging:**

  Movements include X and Y for item sorting and placement, with Z-axis motion for depth control.

  In this project, the SCARA robot was used for packaging applications, where it would pick, pack, and seal products into boxes for shipment. The robot's ability to perform multiple tasks simultaneously made it ideal for packaging.

  - Problems faced: Limited ability to handle various types of packages.
  - Advantages: Higher packaging efficiency, reduced labor costs.

- **Project 6: SCARA Robot for Medical Equipment Assembly:**

  The robot uses X and Y for accurate positioning, Z for vertical assembly tasks, and rotational movements for alignment.

  This SCARA robot was used in the assembly of medical equipment, where precision and clean environments were paramount. The robot's flexibility allowed for assembly of delicate components with minimal errors.

  - Problems faced: Need for high cleanliness standards and strict calibration.
  - Advantages: Increased assembly speed, reduced contamination risk.

Chapter 2
Mechanical Design

Chapter 2
Mechanical Design

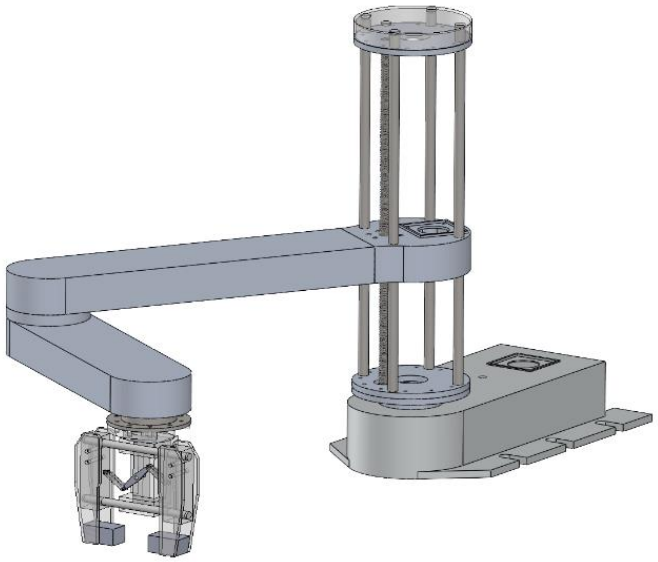
2.1 Introduction

In this chapter, we present the mechanical design of our 4 DOF Scara Robot, which is intended for pick and place and small assembly via machine vision, we describe the main components and features of the robot and explain the design choices and trade-offs made during the development process. We also show the results of the simulation and analysis of the robot's performance, focusing on its capabilities in locomotion and movement, as well as its perception and navigation functionalities. Additionally, we discuss the overall performance and effectiveness of the robot in successfully completing its tasks.

2.2 Appearance and structure:

Our SCARA robot features a modern and practical design, consisting of four major parts that define the robot for the mechanical approach.

- **BASE**
- **PRISMATIC MECHANISM**
- **THE ARM**
- **END EFFECTOR**



SCARA ROBOT

2.3 Design and Workspace:

Our Scara robot is equipped with various sensors and actuators to perceive and interact with its environment. The main motion actuators are four motors for the rotation, one servo motor for the gripper assigned as the end effector, one motor for the base, another for power screw system for lifting, two motors for the arm, and lastly a motor for the rotation for the end effector, The robot consists of two links in its main arm allowing it to work seamlessly with the applied kinematics for its movement to achieve the required target .

2.3.1 MATERIALS:

The main parts in the robot face a high torque and forces that require material that offer more endurance and durability that able to neglect the deflection caused by the weight and forces on the scara robot such as **aluminum 6061** that offers:

Property	Value	Units
Elastic Modulus	6.9e+10	N/m^2
Poisson's Ratio	0.33	N/A
Shear Modulus	2.6e+10	N/m^2
Mass Density	2700	kg/m^3
Tensile Strength	124084000	N/m^2
Compressive Strength		N/m^2
Yield Strength	55148500	N/m^2
Thermal Expansion Coefficient	2.4e-05	/K
Thermal Conductivity	170	W/(m·K)
Specific Heat	1300	J/(kg·K)
Material Damping Ratio		N/A

1-High Strength-to-Weight Ratio:

Lightweight yet strong, ideal for structural and aerospace applications.

2-Corrosion Resistance: Excellent resistance to corrosion due to its protective oxide layer, especially in mild environments.

3-Good Machinability: Easy to machine with standard tools, making it versatile for manufacturing.

4-Weldability: Readily weldable using various techniques like TIG and MIG welding, with proper post-weld treatments to avoid weakening.

5-Versatility: It can be used in a wide range of applications, including automotive, aerospace, construction, and marine industries.

6-Heat Treatability: It can be strengthened through heat treatments like T6 tempering.

7-Excellent Surface Finish: Good for anodizing and painting to enhance aesthetics and durability.

8-Cost-Effective: Widely available and relatively affordable compared to specialized alloys.

-As for the parts that do not deal with higher torque or load, we use lighter materials that do not have to have high durability and endurance such as material used in 3D printing such as

Pros of PLA (Polylactic Acid)

Eco-Friendly: Made from renewable resources (e.g., corn starch), biodegradable, and environmentally sustainable.

Easy to Print: Low printing temperatures (~190-220°C) and minimal warping make it beginner-friendly.

Pros of ABS (Acrylonitrile Butadiene Styrene)

Higher Strength and Durability: More impact-resistant and tougher than PLA, ideal for functional prototypes and durable products.

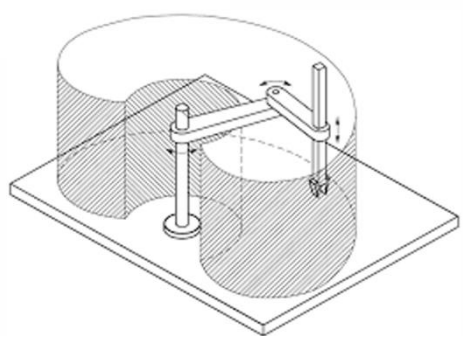
Better Heat Resistance: Handles higher temperatures (~100°C) without deforming, suitable for more demanding applications.

2.3.2 WORKSPACE:

The robot's workspace depends on the two links that are considered as the main arm for the robot.

As stated in the equation that achieves the 2D workspace area:

$$A_{workspace} = (\frac{\theta_1 + \theta_2}{360} * ((l1 + l2)^2) * \pi)$$



-$l1$ >> the first link's length

-$l2$ >> the second link's length

-θ_1 >> the rotation angle for the first link

-θ_2 >>the rotation angle for the second link

To make this a 3D workspace, consider the Z-axis motion added by SCARA's vertical prismatic joint. This joint allows the robot to move up and down within a certain range (H).

3D Workspace Volume:

The 3D workspace is formed by extruding the 2D workspace area along the Z-axis (height). The volume is given by:

$$V_{workspace} = A_{workspace}\ x\ H$$

-H >> is the vertical range of motion of the prismatic joint.

2.4 Main Parts of Scara Robot:

2.4.1 THE BASE:

- **Purpose**:
  - The base serves as the foundation of the robot. It provides stability and houses the components for the first rotational degree of freedom (DOF 1).
- **Movement**:
  - Typically, rotational (theta1).
  - Enables the arm to sweep across the workspace horizontally.
- **Key Features**:
  - Must be rigid to avoid vibrations.
  - Usually holds the motor or actuator for the first joint.
  - Designed to allow maximum workspace utilization.

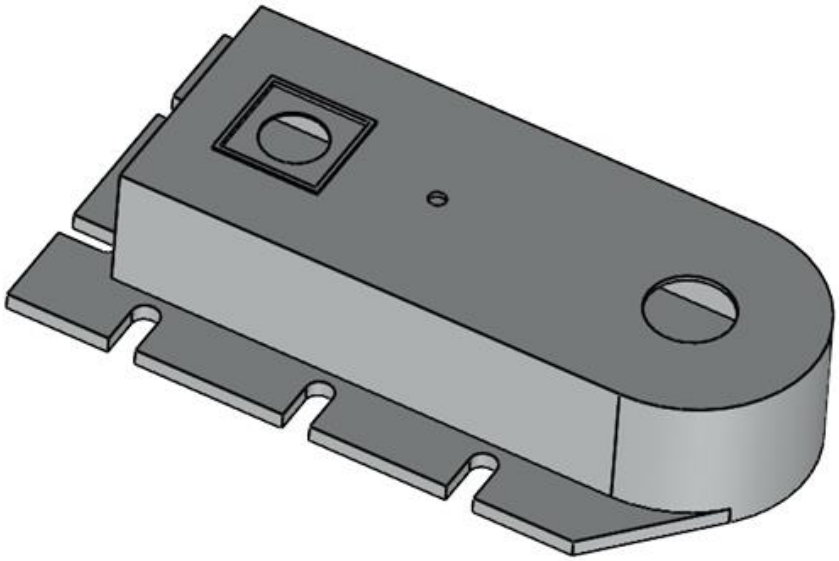


FIGURE 1

- **The base has two parts.**
  1- The base itself which holds the components, made of aluminum 6061 with 150x375x75 with 3mm thickness and hold the motor as the first degree of freedom (revolute the prismatic system)
  2- The fixing plate that fixes the base to ground with 230x415 and 5mm thickness

COMPONANTS OF THE BASE:

Components of the Base:

1. **Stepper Motor:**
   - The stepper motor at the top is likely responsible for the rotational motion of the base or for driving a mechanism inside the base.
   - **Purpose:** Provides precise rotational control for the robot's first degree of freedom (DOF 1).
2. **Belt and Pulley System:**
   - The belt and pulley mechanism shown appears to transfer motion or torque from the stepper motor to other components in the base.
   - **Purpose:**
     - Reduces the motor speed while increasing torque using a gear ratio (depending on the pulley sizes).
     - Ensures smooth and efficient motion transfer.
3. **Rotary Shaft or Bearing Assembly:**
   - The central shaft (connected to the larger pulley) likely serves as the primary rotational axis of the robot's base.
   - **Purpose:**
     - Transmits the rotational motion to the arm or other connected components.
     - Supported by bearings for smooth rotation.
4. **Base Structure:**
   - The large solid component at the bottom houses all the mechanisms and provides rigidity.
   - **Purpose:**
     - Protects the internal components.
     - Offers stability and mounting points for the entire robot.

Mechanism Inside the Base:

- **Power Transmission:**
  - The stepper motor drives the smaller pulley, which in turn rotates the belt. The belt then drives the larger pulley, transferring the rotational motion to the central shaft.
  - The use of a belt and pulley system allows for a compact design while achieving the desired speed and torque requirements.
- **Rotational Axis:**
  - The central shaft or bearing assembly ensures that the base rotates around a fixed axis, allowing the robot arm to move horizontally across the workspace.

- **Precision and Control:**
  - The stepper motor provides precise angular control. The gear ratio from the pulleys can improve accuracy by reducing the step size at the output.

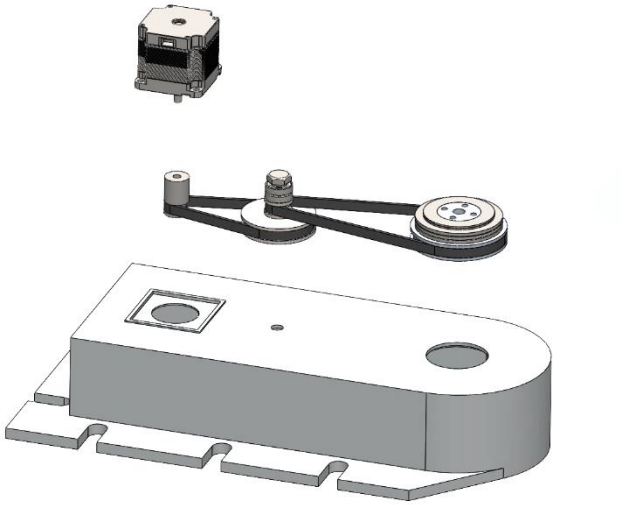


FIGURE 2 (EXPLODE VIEW OF BASE STRUCTURE)

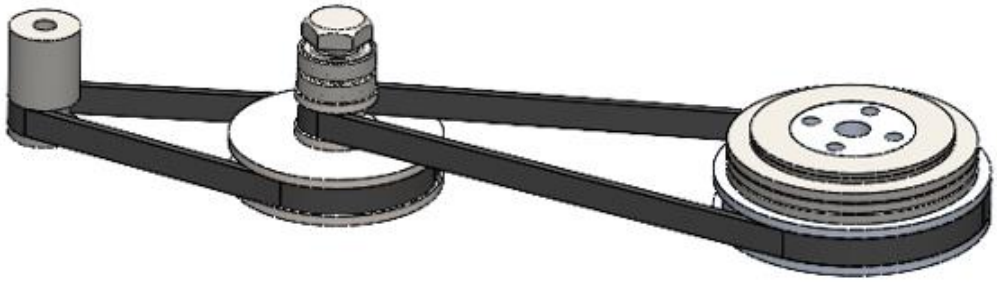


FIGURE 3 (POWER TRANSMISSION MECHANISM)

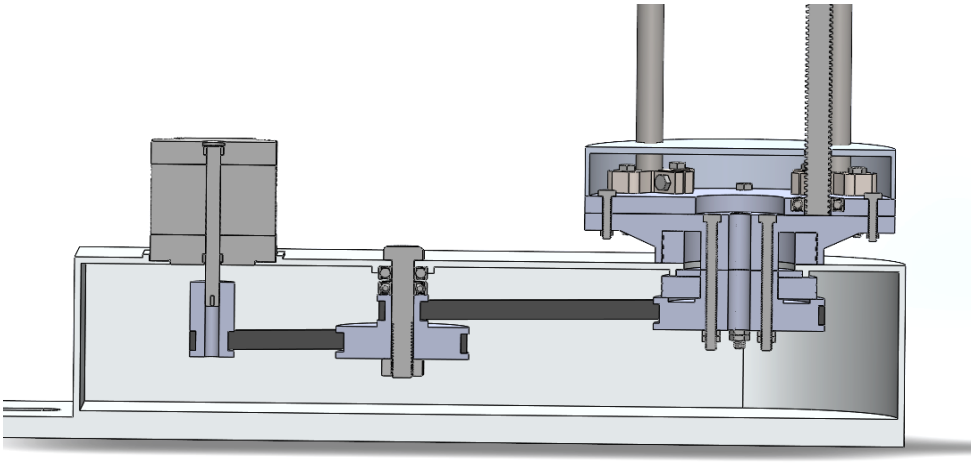
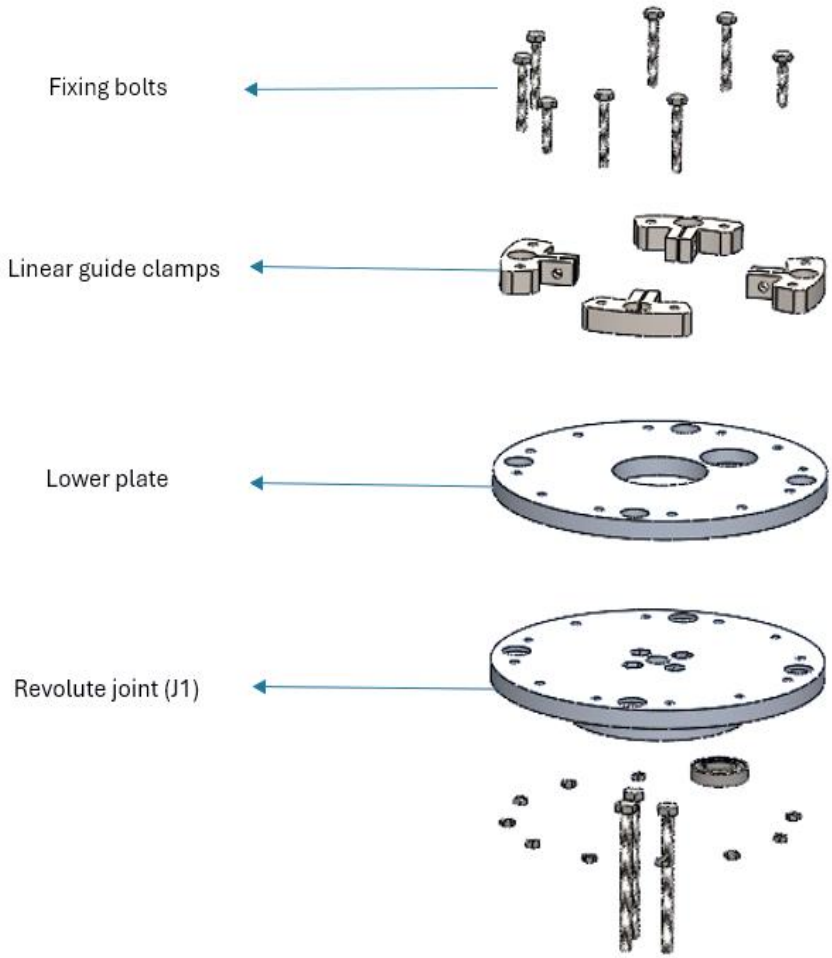
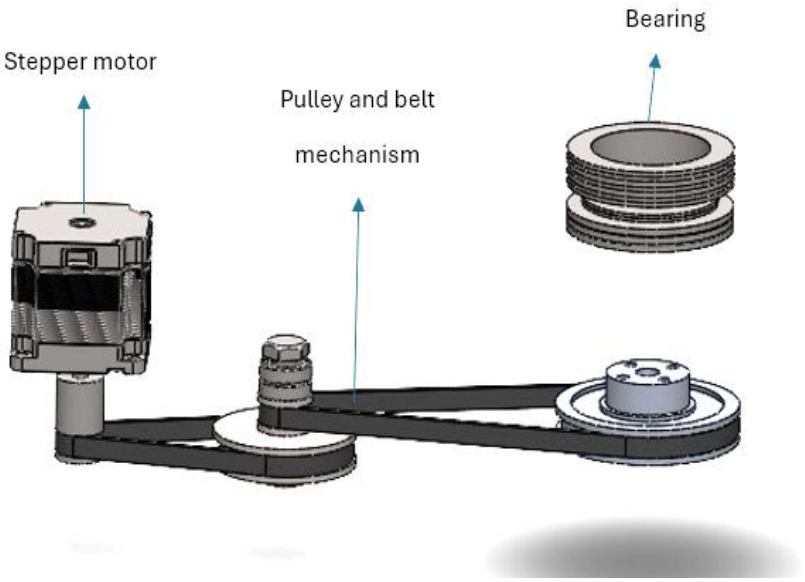


FIGURE 4 (CROSS SECTION VIEW OF BASE INNER STRUCTURE)





BASE STRUCTURE AND REVOLUTE SYSTEM

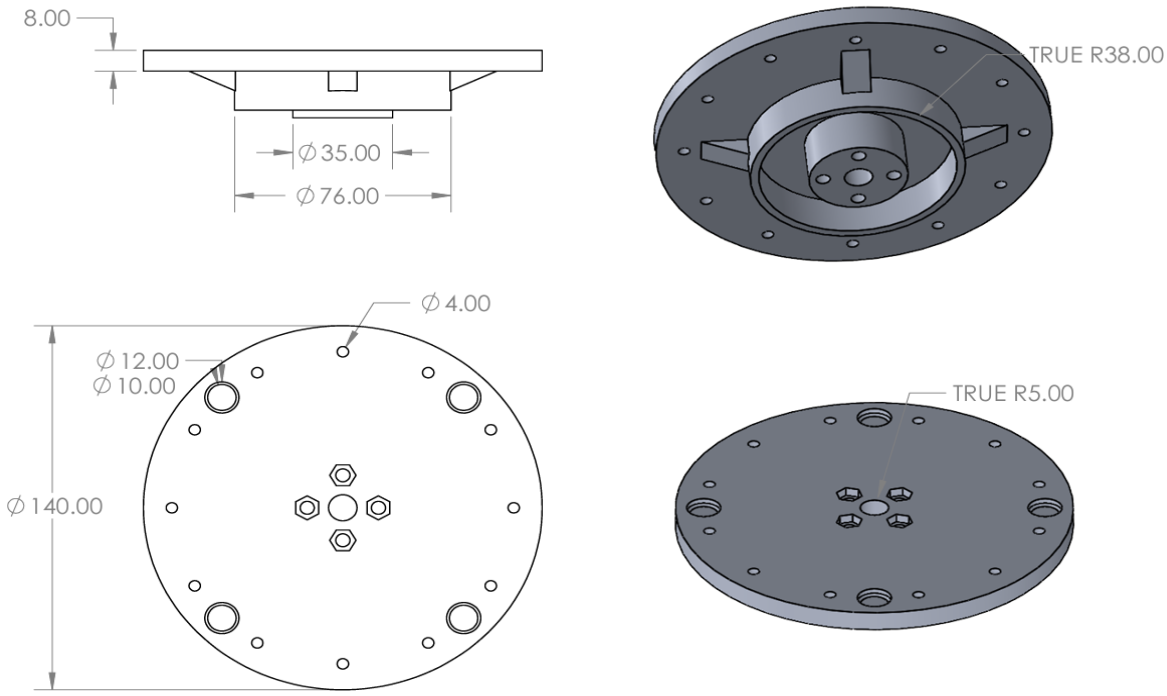


FIGURE 5 (REVOLUTE JOINT J1)

A revolute joint is a one-degree-of freedom kinematic pair used frequently in mechanisms and machines. Joint J1 is a disk with 140mm diameter with 35 mm diameter rod.

2.4.2 PRISMATIC MECHANISM:

Prismatic Joint

- **Purpose:**
  Provides linear motion along a specific axis, adding flexibility for tasks requiring vertical adjustments (DOF 2).
- **Movement:**
  - Linear (Z-axis, or sometimes radial depending on the design).
  - This motion can be achieved using pneumatic cylinders, ball screws, or other linear actuators.
- **Key Features:**
  - Supports precise vertical adjustments.
  - Critical for height control and object handling.
  - Contributes to the Z-axis motion range of the robot.

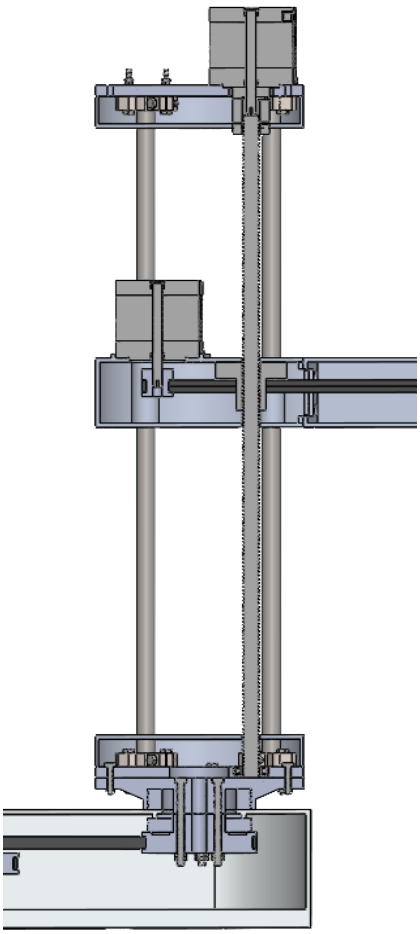


FIGURE 6

- A prismatic joint provides a linear sliding movement between two bodies, and is often called a slider, as in the slider-crank linkage. A prismatic pair is also called a sliding pair. A prismatic joint can be formed with a polygonal cross-section to resist rotation. See for example the dovetail joint and linear bearings.

- We have four linear guides as shown in **Fig 7.** With diameter 10mm and height 45cm and fixed between two discs each of them with diameter 140mm, span of prismatic joint is 35cm.

- the slide part that connected to the power screw is the first part in the first link in the arm system

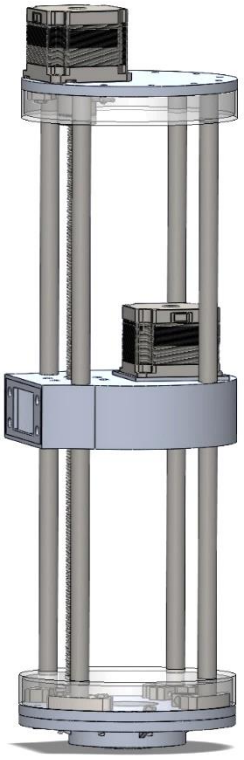


FIGURE 7

- As shown in **Fig 8 and Fig 9,** the prismatic mechanism provides the second degree of freedom from the motor that is fixed in the upper plate.

- the motor is connected the power screw which left the screw nut that is fixed in the very first part of the arm system to provides a linear motion along z-axis.

- the linear guide and the power screw length about 45 cm, but due to the arm part height and the arm motor that is mounted in the part we have around 30 cm of free movement length along z -axis.

- the lower part of the power screw and the linear guide is connected to the base via the revolute system (1).

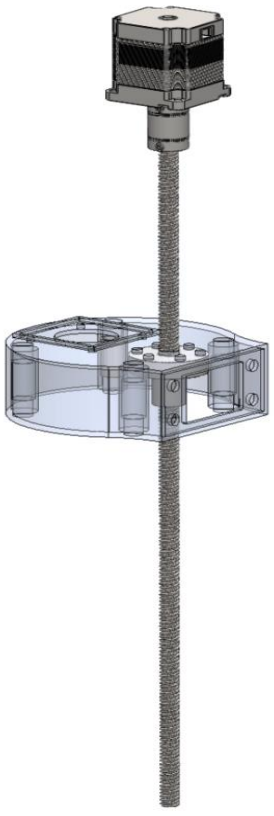


FIGURE 8

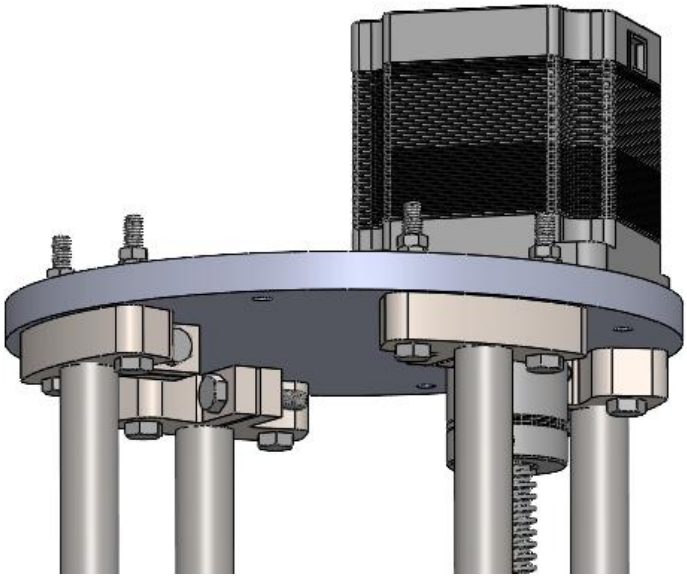


FIGURE 9

(THE UPPER PART OF THE PRISMATIC MECHANISM)

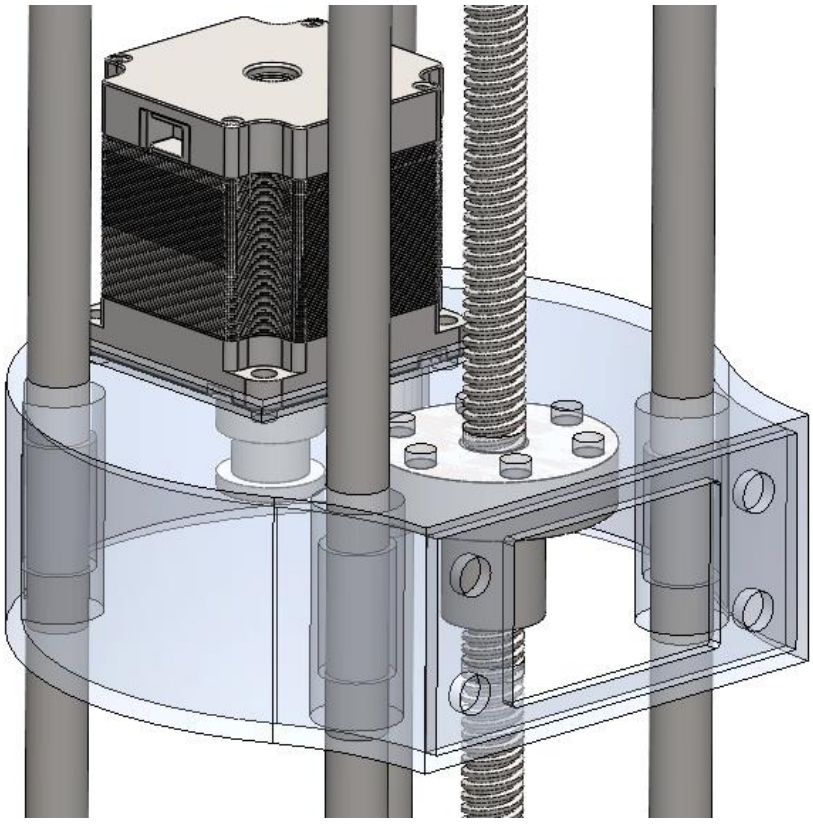
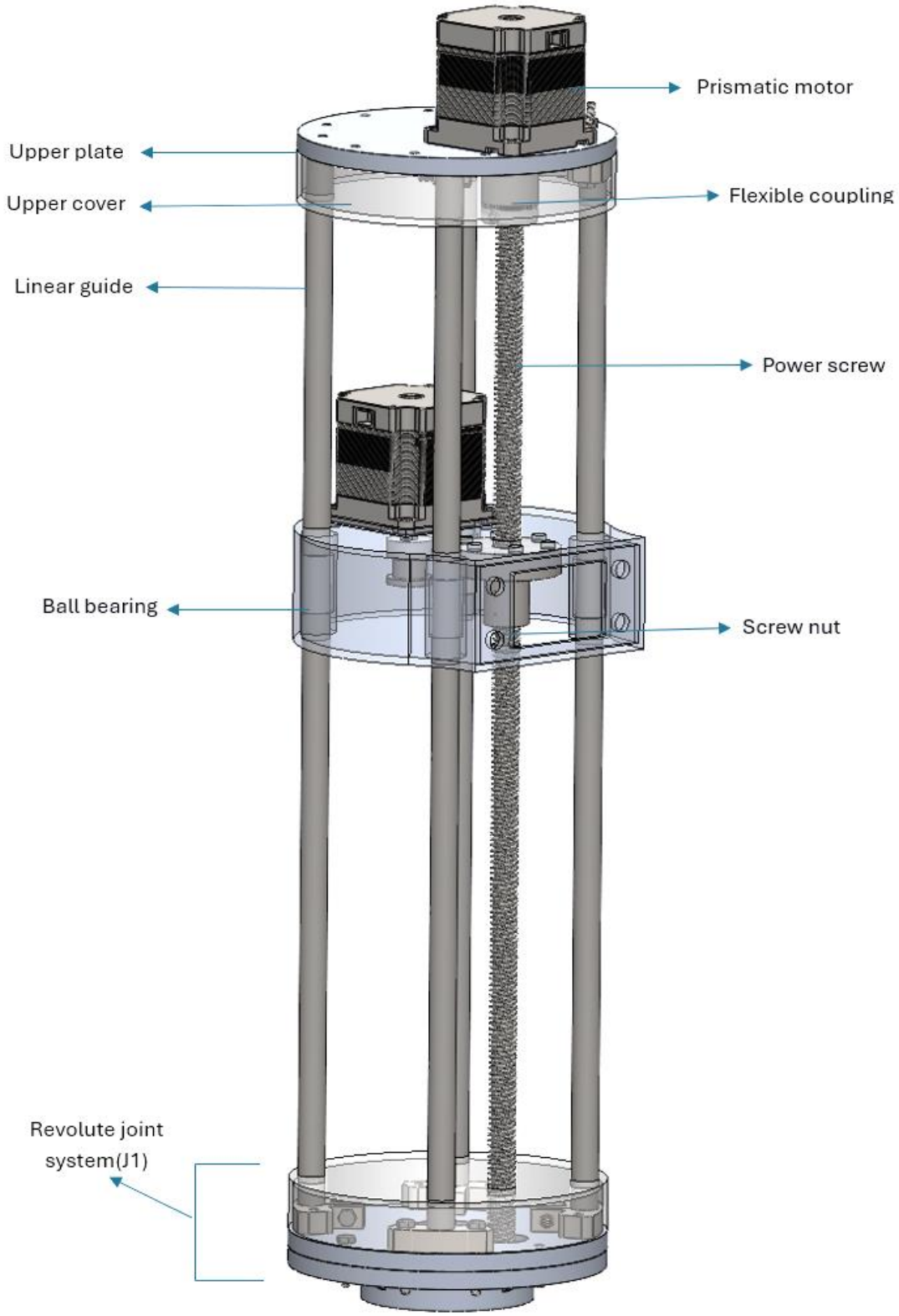


FIGURE 10

(THE MIDDLE PART AND THE SCREW NUT)


Prismatic motor
Upper plate
Upper cover
Flexible coupling
Linear guide
Power screw
Ball bearing
Screw nut
Revolute joint
system(J1)

2.4.3 THE ARM:

- **Purpose:**
  - The arm is responsible for extending the robot's reach and positioning the end effector accurately in the XY plane (DOF 3).

- **Movement:**
  - Rotational (theta2).
  - Provides flexibility for reaching different points in the workspace.

- **Key Features:**
  - May consist of one or more links connected by joints.
  - Lightweight yet strong materials are typically used to minimize inertia.
  - The arm's length defines the robot's effective reach.

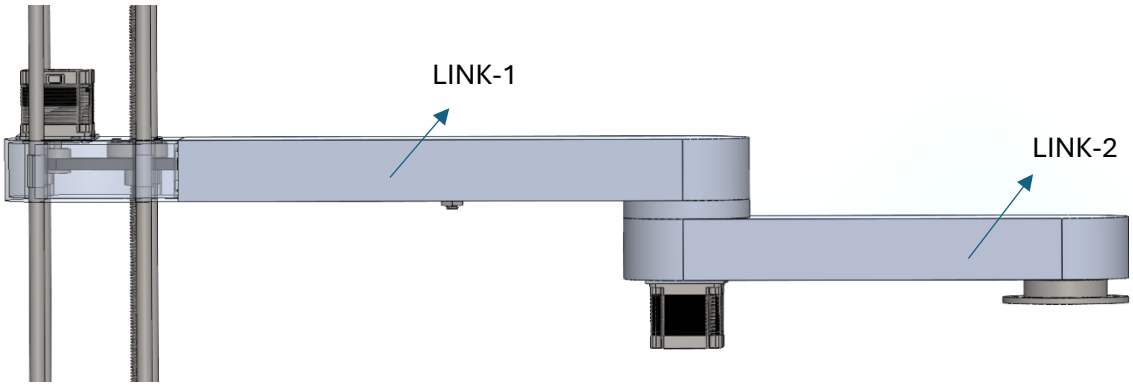


FIGURE 11

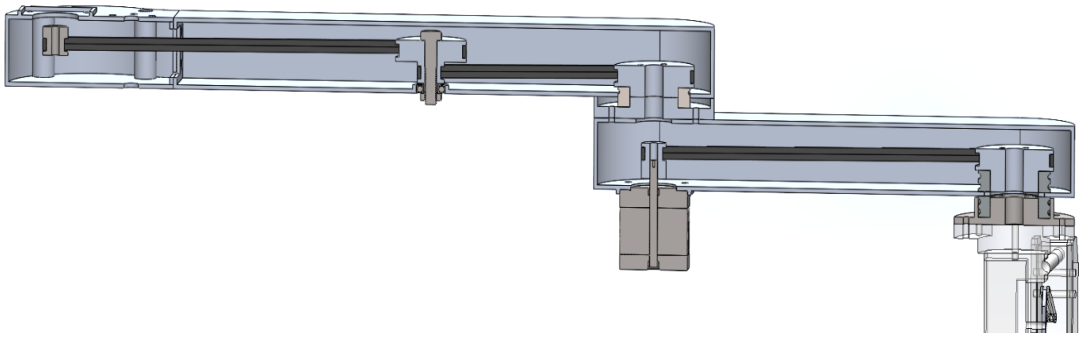


FIGURE 12

- **Arm Part:**

1. **Two Links:**
    - The arm consists of **two links** connected by rotational joints. These links work together to provide motion in the XY plane.
    - **Purpose**: Extends the robot's reach and positions the end effector accurately.

2. **Rotational Joints**:
    - Each joint is driven by a motor, allowing independent control of each link's rotation (DOF 2 and DOF 3).

3. **Material Consideration**:
    - The links are typically made of lightweight yet strong materials to minimize inertia and improve efficiency.

The arm structure ensures a balance of reach, flexibility, and precision for the SCARA robot's operations.

link 1:

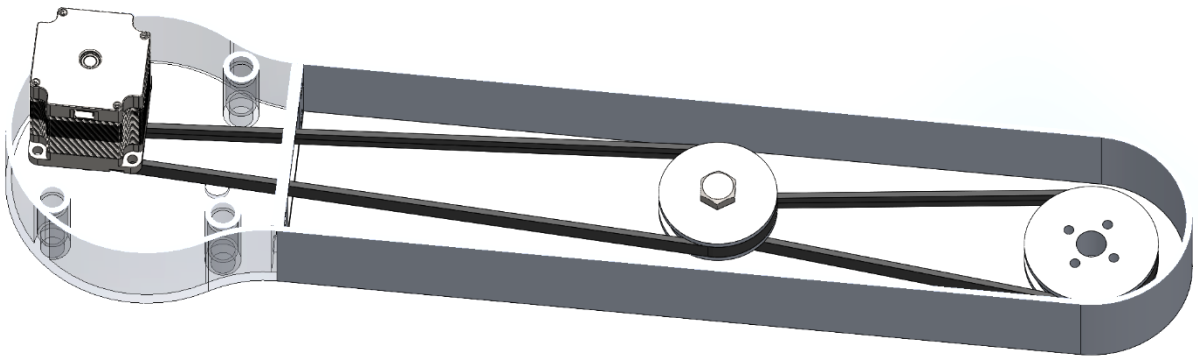


FIGURE 13 (TOP VIEW OF THE FIRST LINK)

-Components of the link-1

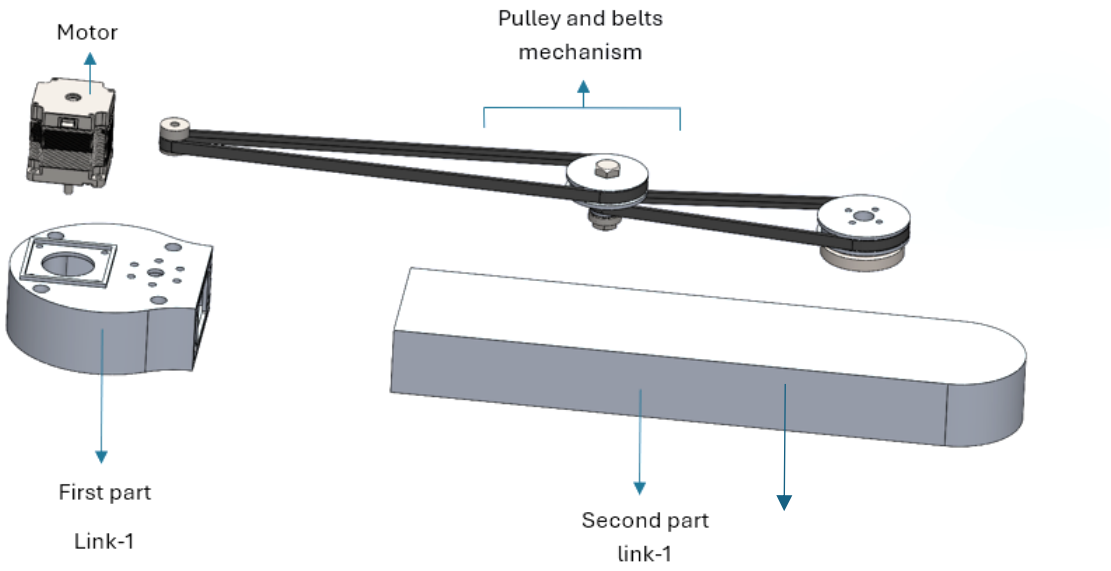


FIGURE 14 (COMPONENTS OF THE LINK-1)

-the first part of link-1 is connected to the prismatic system, housing the bearing of the four linear guides, the screw nut for the power screw

-the motor providing the third degree of freedom is mounted on the top of the first part of link-2

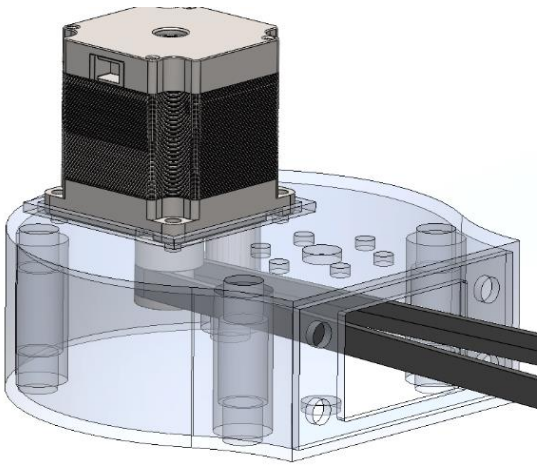


FIGURE 15

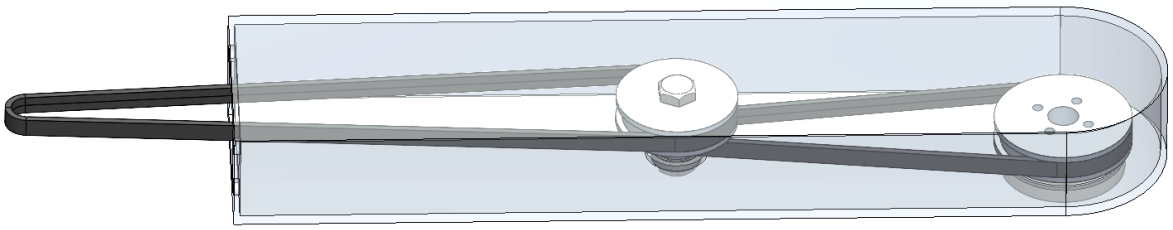


FIGURE 16

- the second part of link-1 housing the pulley and belt system
- Its connected to the the revolute joint (j2)

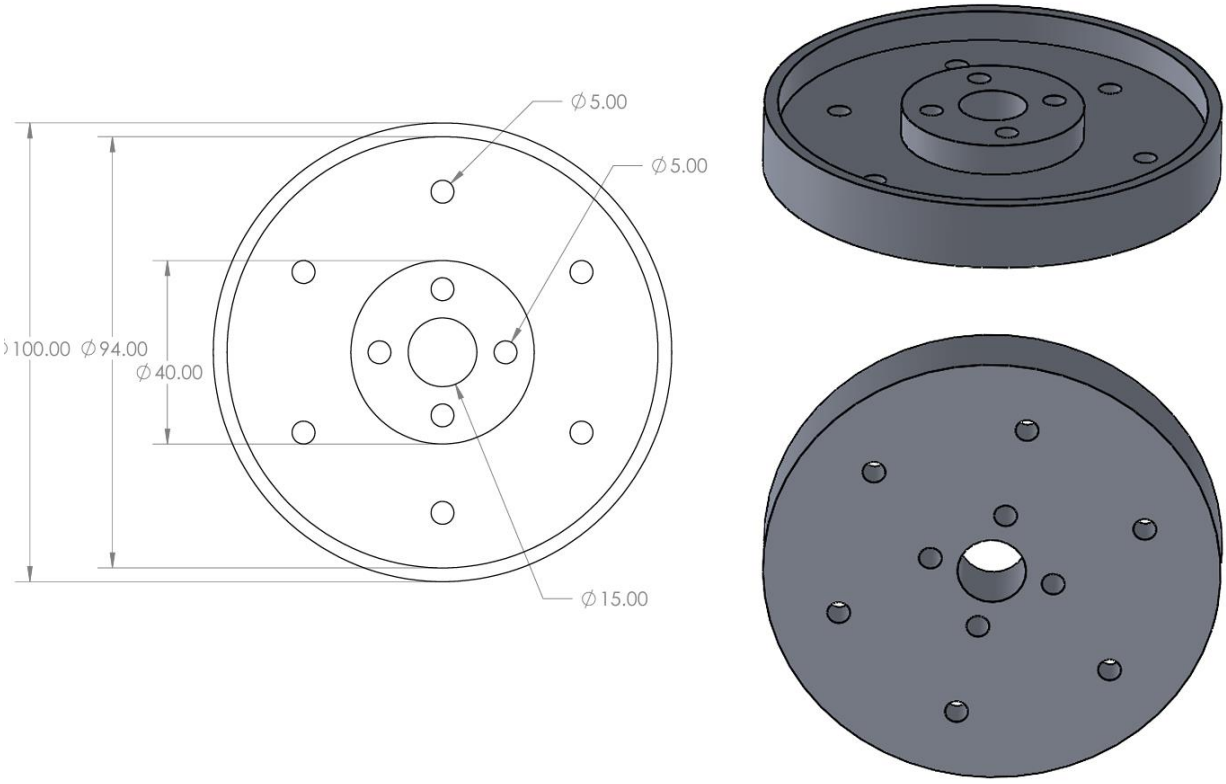


FIGURE 17 (REVOLUTE JOINT (J2))

A revolute joint is a one-degree-of freedom kinematic pair used frequently in mechanisms and machines. The joint J2 is a disk with 100 mm diameter and 40 mm diameter rod.

link-2

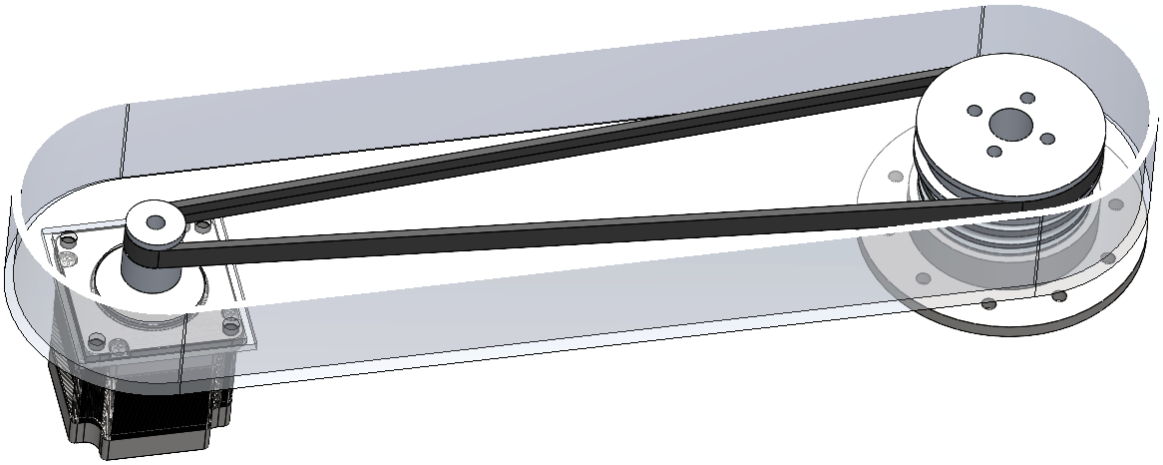


FIGURE 18 (TOP VIEW OF THE SECOND LINK)

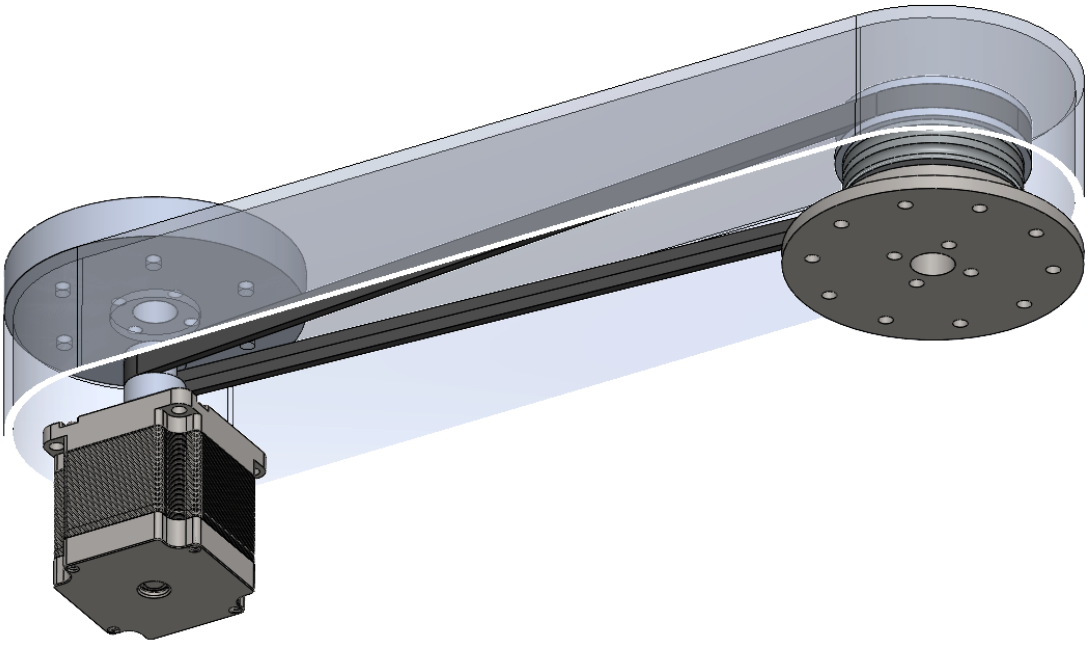


FIGURE 19 (BOTTOM VIEW OF THE SECOND LINK)

-component of the link-2

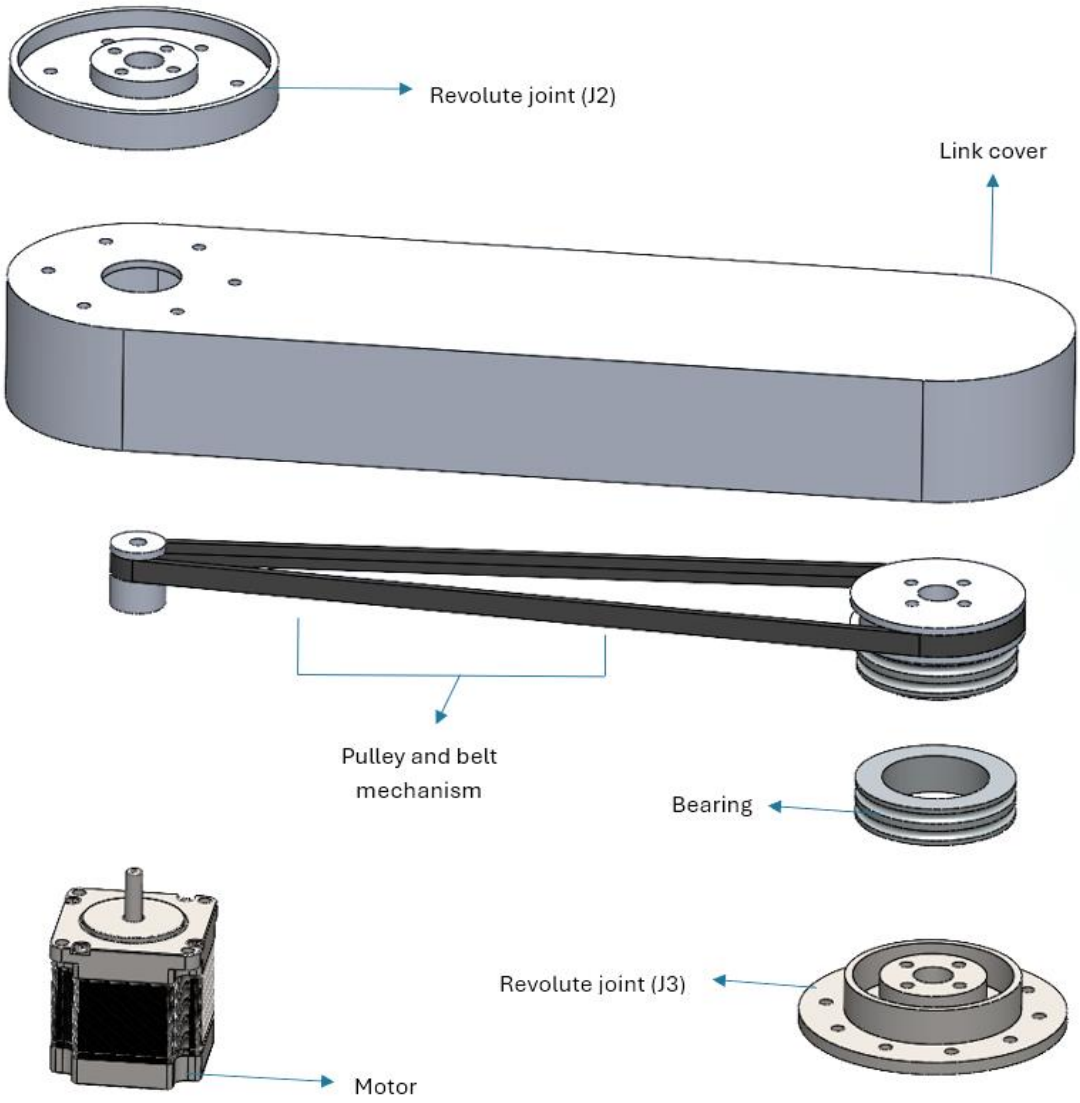


FIGURE 20(COMPONENT OF THE LINK-2)

-link2 receives movement from the motor mounted in the link-1 through revolute joint (J2)

- link-2 is housing the pulley and belt mechanism from the motor directly to the revolute joint (J3)

- link-2 connect with the link 1 via revolute joint (J2) and connect with the end effector (last main part in the SCARA robot) via revolute joint (J3).

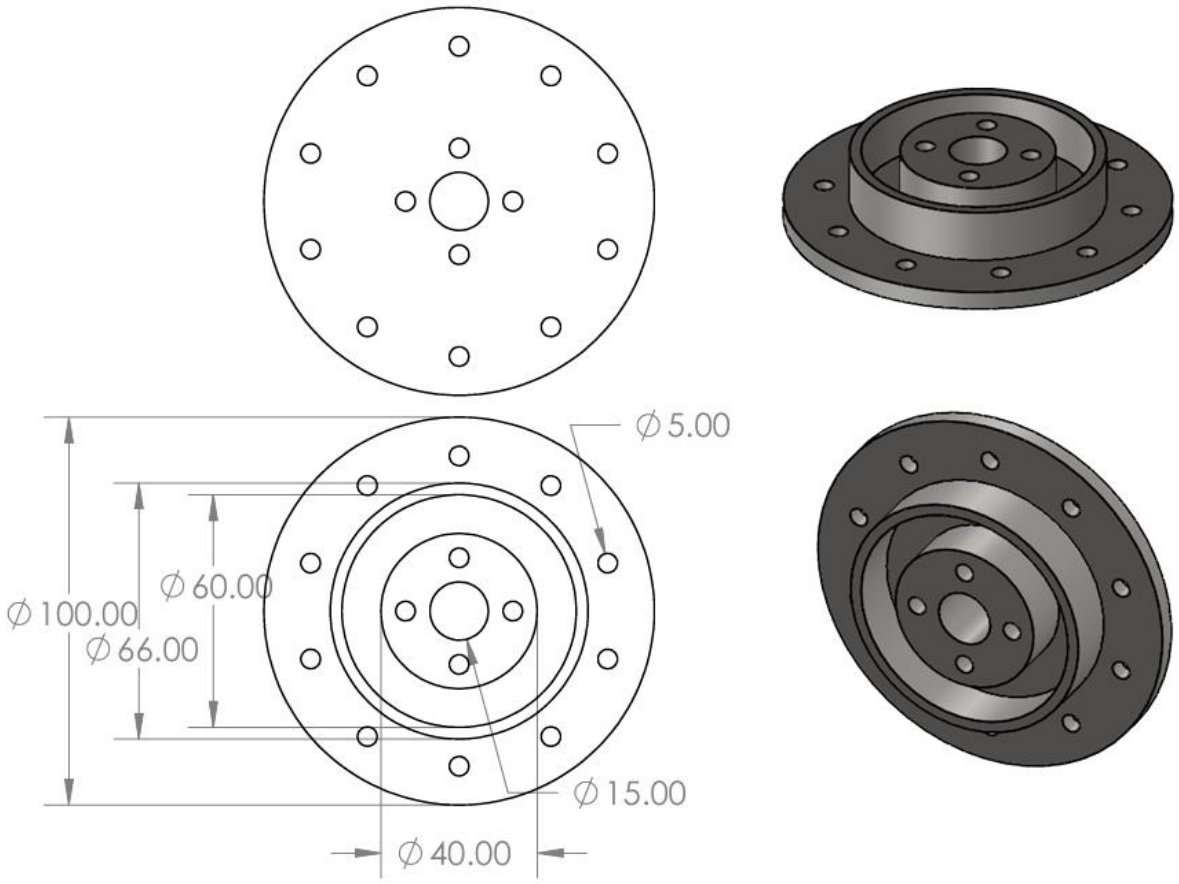


FIGURE 21 REVOLUTE JOINT (J3)

A revolute joint is a one-degree-of freedom kinematic pair used frequently in mechanisms and machines. The joint J3 is a disk with 78mm diameter with 30 mm diameter rod.

2.4.4 THE END EFFECTOR:

- **Purpose:**
  - The end effector interacts with objects in the workspace. It could be a gripper, a tool, or another specialized attachment (DOF 4).

- **Movement:**
  - Often rotational (theta4) to orient the tool or gripper.
  - Alternatively, it could involve additional functions like gripping or suction.

- **Key Features:**
  - Adaptable to various applications (e.g., pick-and-place, assembly, welding).
  - Its design depends on the specific task.
  - May include sensors for feedback (e.g., force or proximity sensors).

- The gripper mechanism is the end effector of the robot arm, it's used to hold the product to move it from position to another. The actuator of the gripper is servo motor, the servo connected with links by revolute joint and links connected with slides. The left and right hands are fixed on the sliders and the gripper end fixed on these hands. When the servo rotates the slider will make a linear motion on the gripper rail which will cause the same linear motion for the gripper end to hold or let the product.

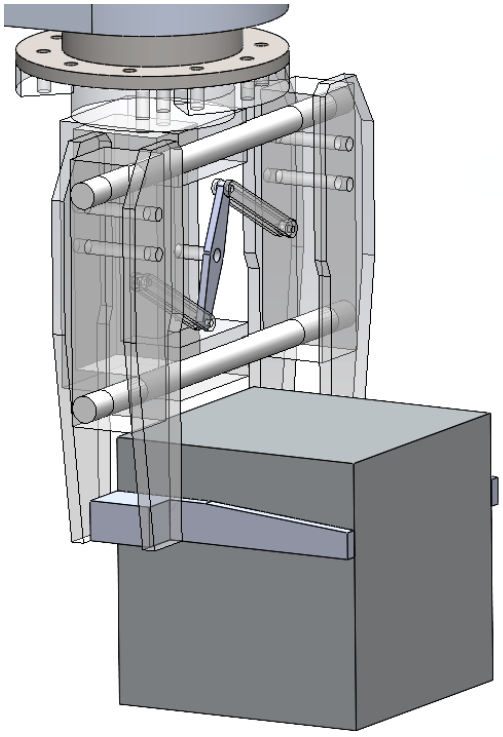


FIGURE 22

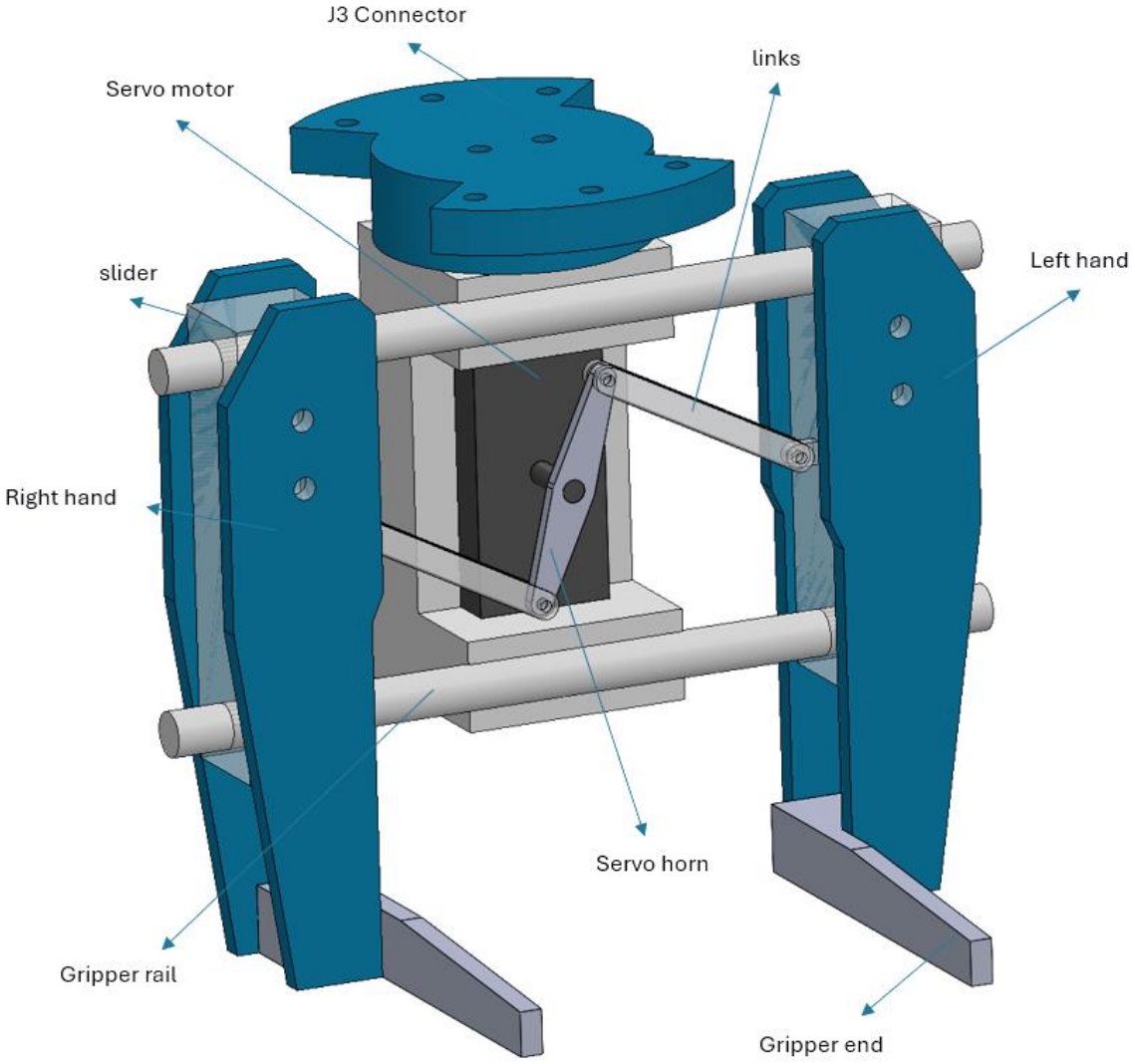

J3 Connector
links
Servo motor
slider
Left hand
Right hand
Servo horn
Gripper rail
Gripper end

FIGURE 23

2.5 Stress Analysis

INTRODUCTION

Stress analysis is a method used in engineering to determine the stresses and strains in materials and structures subjected to forces, stress analysis is crucial for several reasons:

1. **Safety:** it ensures that the robot can withstand the forces it will encounter during operation.

2. **Durability:** it helps in predicting the robot's lifespan by analyzing how the materials will behave under different stress conditions.

3. **Optimization:** it aids in material selection and design optimization, ensuring the robot is neither over-engineered (leading to unnecessary costs) nor under-engineered (which could cause premature failure).

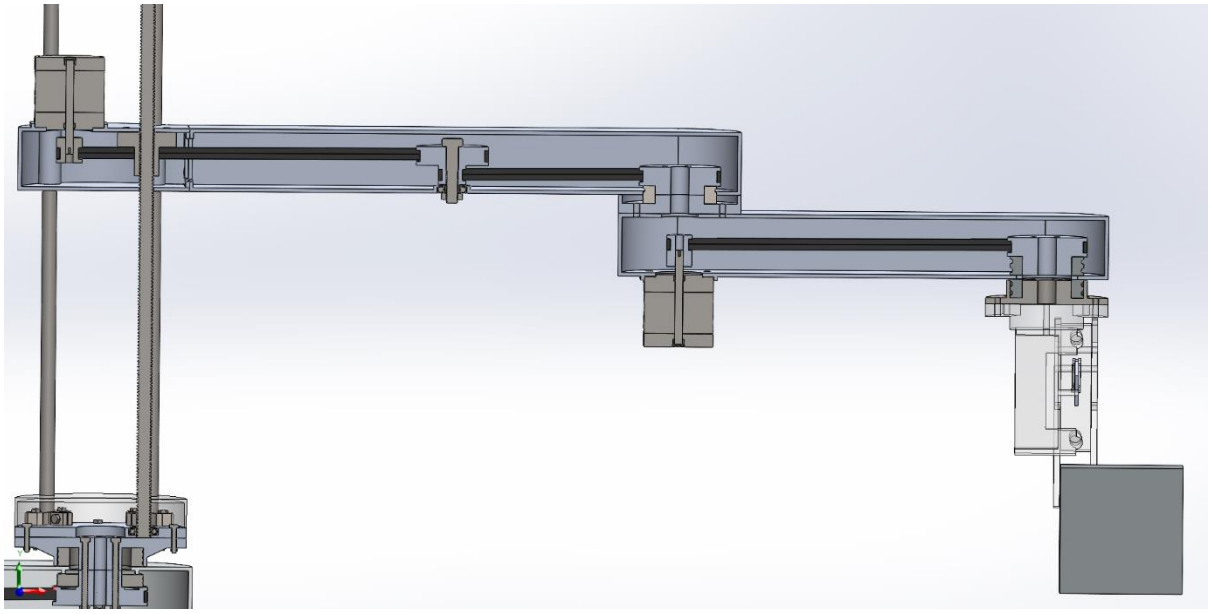


FIGURE 24

- **We divided this process into two sections:**
  1. outer link analysis (link (2)).
  2. inner link analysis (link (1)).

2.5.1 OUTER LINK:

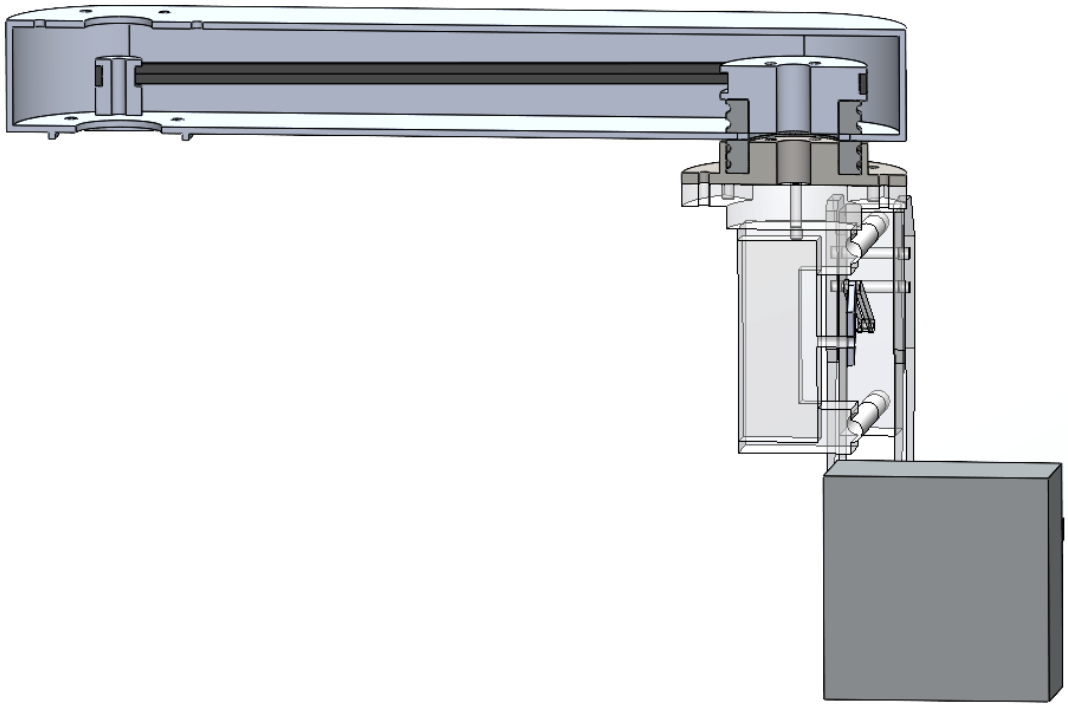


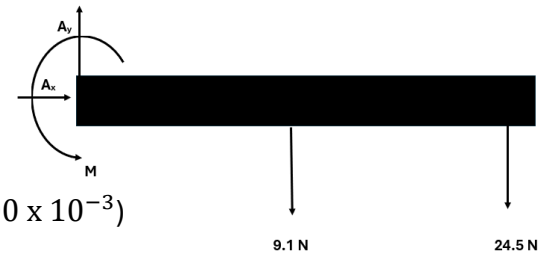
FIGURE 25(OUTER LINK)

Determining (forces & moments):

$\because \Sigma F_x = 0 \quad \therefore \ A_x = 0$

$\because \Sigma F_y = 0 \ \therefore \ A_y = 9.1 + 24.5 = 33.6 \text{ N}$

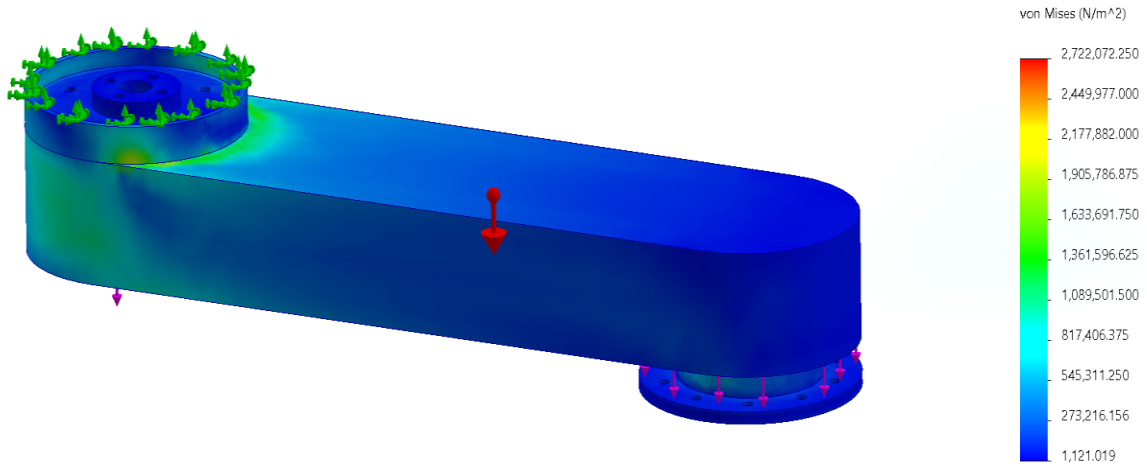
$\because \Sigma M_A = 0 \ \therefore$ M = (9.1 x 150 x 10^{-3}) + (24.5 x 300 x 10^{-3})



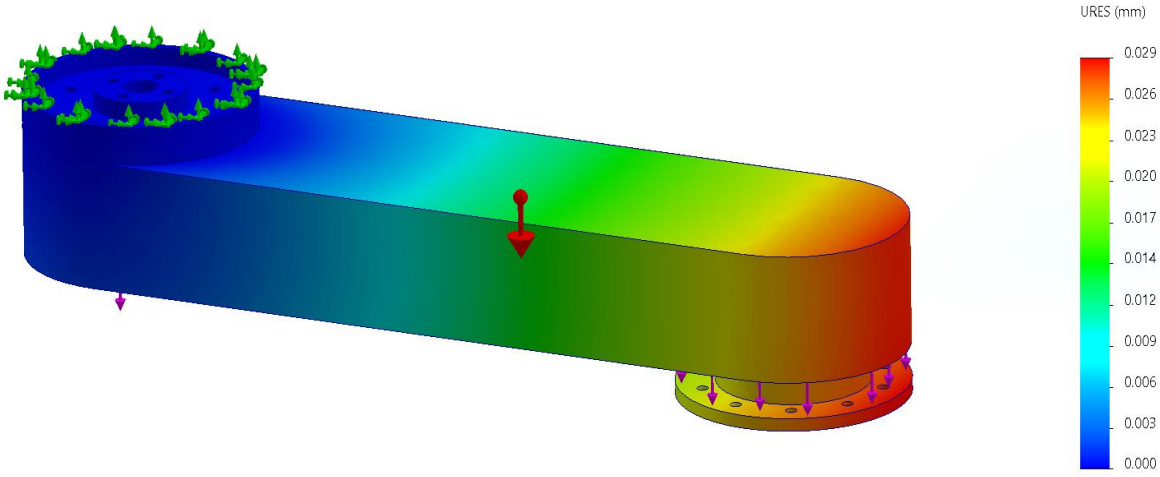
∴ M = 8.715 Nm

SOLIDWORKS SIMULATION:

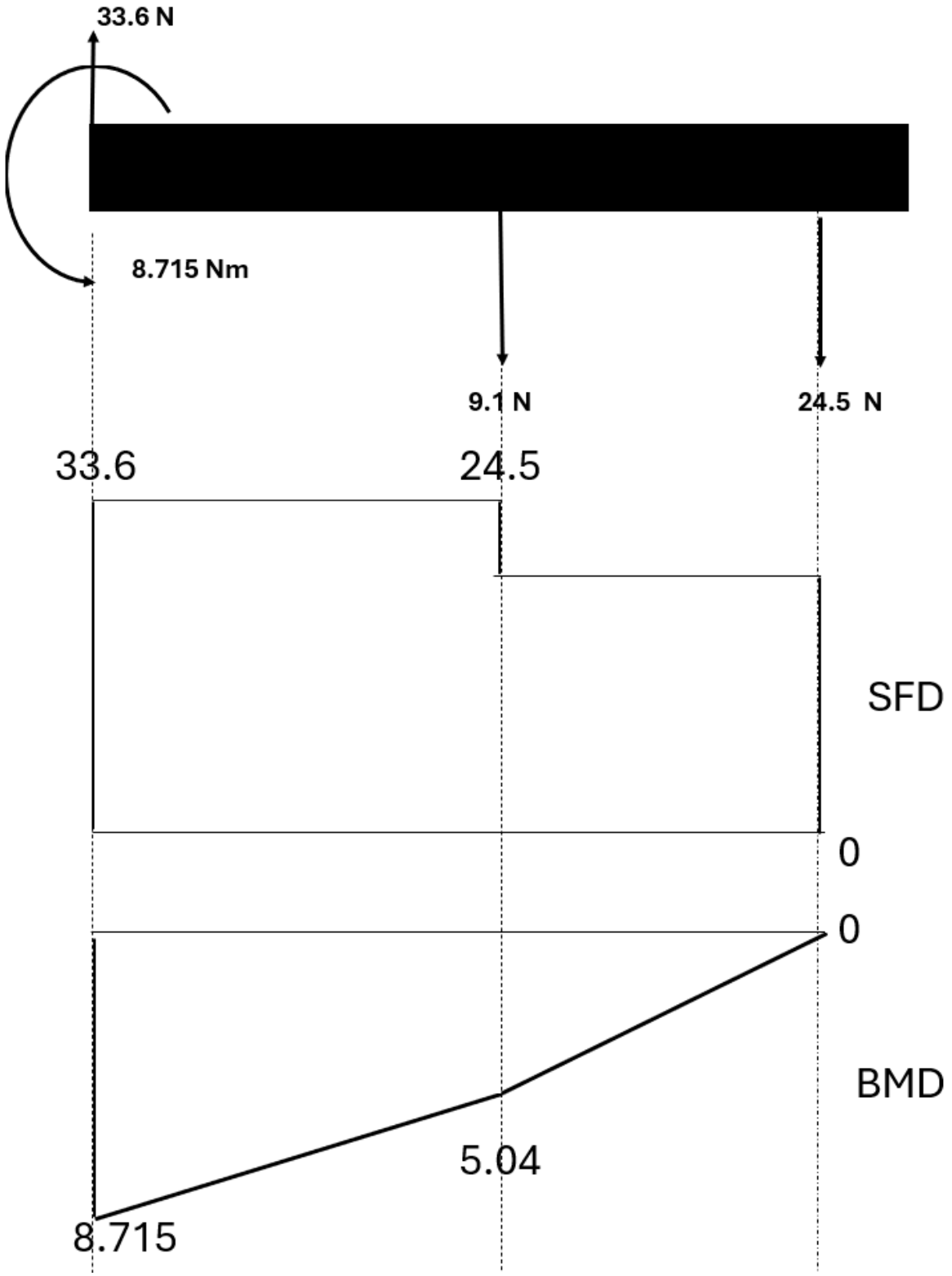
- **stress**



- **deflection**



BMD & SFD:

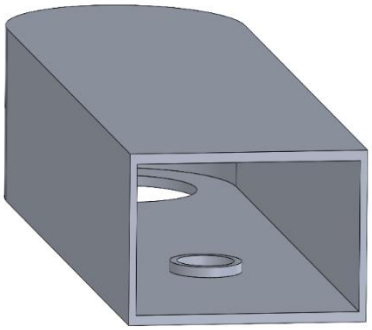


Bending Stress:

$$\sigma_b = \frac{M \times y}{I}$$

Where:

- σ_b is the bending stress.
- M is the bending moment applied to the section.
- Y is the distance from the neutral axis to the outermost fiber.
- I is the moment of inertia of the cross-section about the neutral axis.

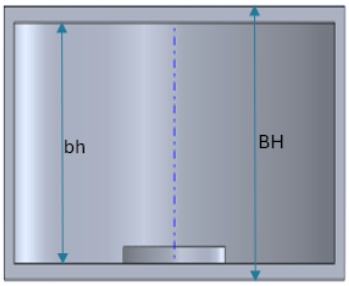


$\because \boldsymbol{\sigma_b} = \frac{M \times y}{I}$ $\because$ **M** = 8.715 N.m $\because$ **Y** = H/2

$\because$ **I** = $\frac{1}{12}$ $(BH^3 - bh^3)$

$\therefore \boldsymbol{\sigma_b} = \frac{M \times \left(\frac{H}{2}\right)}{\frac{1}{12} \times (BH^3 - bh^3)} = \frac{6M\ x\ H}{BH^3 - bh^3}$

$\therefore \boldsymbol{\sigma_b} = \frac{6 \times 8.715 \times 0.05}{(0.1\ x\ (0.05)^3) - (0.094\ x\ (0.044)^3)}$ = **0.5819435 Mpa**



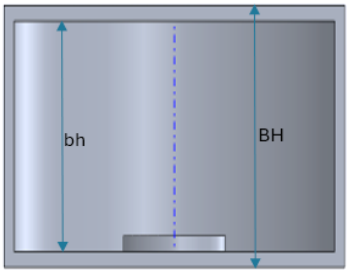
Shear Stress:

$$\tau = \frac{v \times Q}{I \times t}$$

Where:

- V is the shear force applied to the section.
- Q is the first moment of area about the neutral axis.
- I is the moment of inertia of the cross-section about the neutral axis.
- t is the width of the section where the shear stress is being calculated.

$\because \boldsymbol{\tau} = \frac{v \times Q}{I \times t}$ $\because \boldsymbol{Q} = A \times \bar{y}$ $\because \mathbf{A} = \left(\frac{H}{2} \times B\right) - \left(\frac{h}{2} \times b\right)$



$\mathbf{A} = (25 \times 100) - (22 \times 94) = 432\ mm^2$

$\because \bar{\boldsymbol{y}} = \frac{A_1 y_1 - A_2 y_2}{A_1 - A_2} = \frac{2500 \times 12.5 - 2068 \times 11}{2500 - 2068} = 19.68$ mm

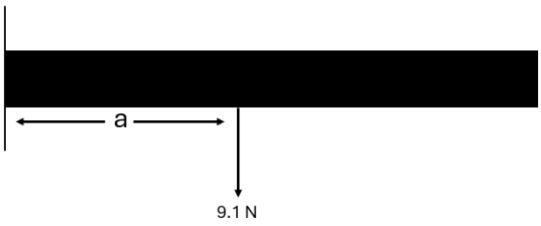
$\therefore \boldsymbol{Q} = 432 \times 10^{-6} \times 19.6 \times 10^{-3} = 8.4672 \times 10^{-6}\ mm^3$

$\therefore \boldsymbol{\tau} = \frac{33.6 \times 8.46 \times 10^{-6}}{\frac{3}{12} \times 10^{-3} \times ((0.1\ x\ (0.05)^3) - (0.094\ x\ (0.044)^3))}$ = **0.25329772 Mpa**

Deflection:

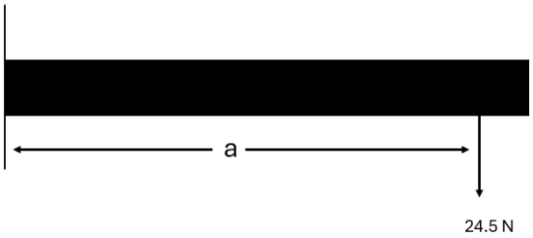
we used the superposition method to determine the deflection

$\because \boldsymbol{\delta_1} = \frac{Fa^2(3L - a)}{6EI} = \frac{Fa^2(3L - a)}{E \times \frac{1}{2}\left(BH^3 - bh^3\right)}$



$\therefore \boldsymbol{\delta_1} = \frac{9.1 \times (0.15)^2 \times (0.9)}{2.246 \times 10^{-6} \times 68.9 \times 10^9}$ = **1.1906 × 10^{-6} m**

$\because \boldsymbol{\delta_2} = \frac{Fa^2(3L - a)}{6EI} = \frac{Fa^2\ (3L - a)}{E \times \frac{1}{2}\left(BH^3 - bh^3\right)}$



$\therefore \boldsymbol{\delta_2} = \frac{24.5 \times (0.3)^2 \times (0.75)}{2.246 \times 10^{-6} \times 68.9 \times 10^9}$ = **1.06849 × 10^{-5} m**

$\because \boldsymbol{\delta_{total}} = \delta_1 + \delta_2 = 1.1906 \times 10^{-6} + 1.06849 \times 10^{-5}$

$\therefore \boldsymbol{\delta_{total}} = 1.18755 \times 10^{-5}$ m = **0.0118755 mm**

2.5.2 INNER LINK:

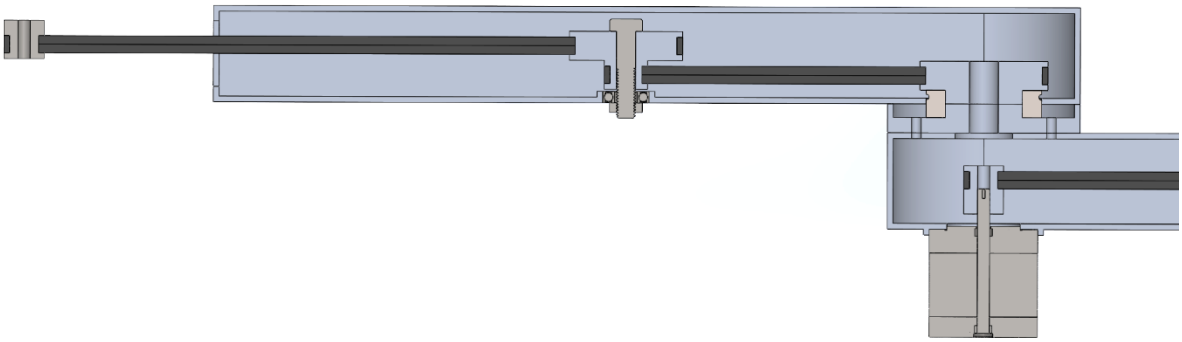


FIGURE 26(INNER LINK)

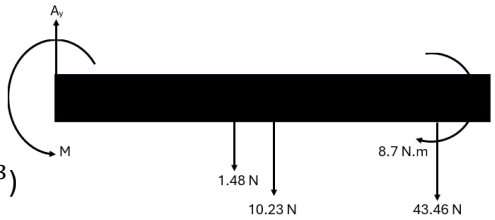
Determining (forces & moments):

$\because \Sigma F_y = 0 \quad A_y = 1.48 + 10.23 + 43.46 =$ **55. 17** $\boldsymbol{N}$

$\because \Sigma M_A = 0$

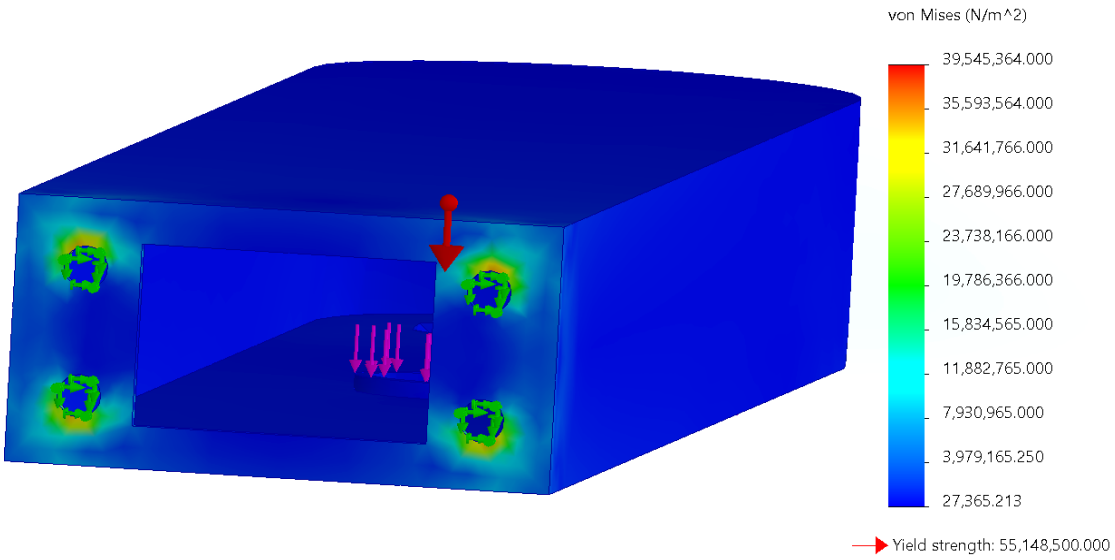
$\therefore$ **M** = $(1.48 \times 214.21 \times 10^{-3}$ + $(10.23 \times 225 \times 10^{-3})$

+ $(43.46 \times 400 \times 10^{-3})$ + 8.7 = **28.7 N.m**

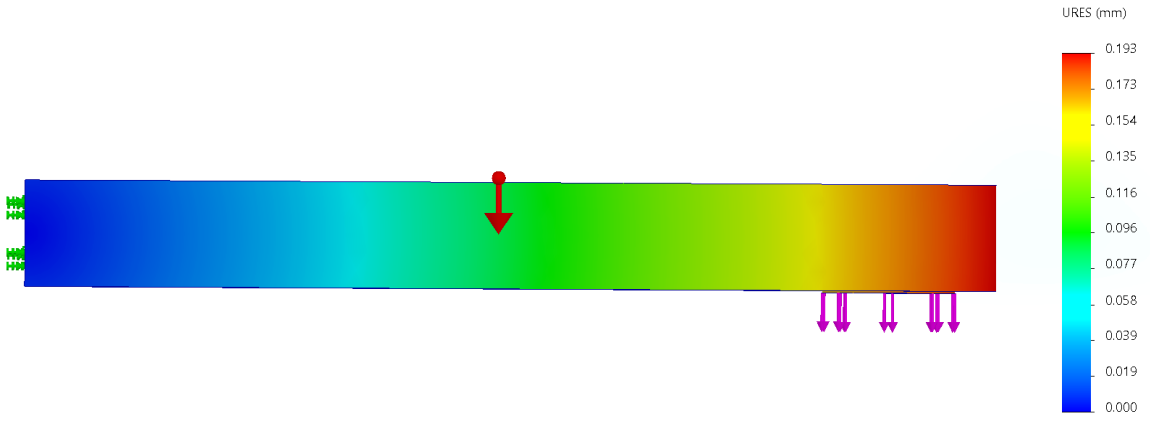


SOLIDWORKS SIMULATION:

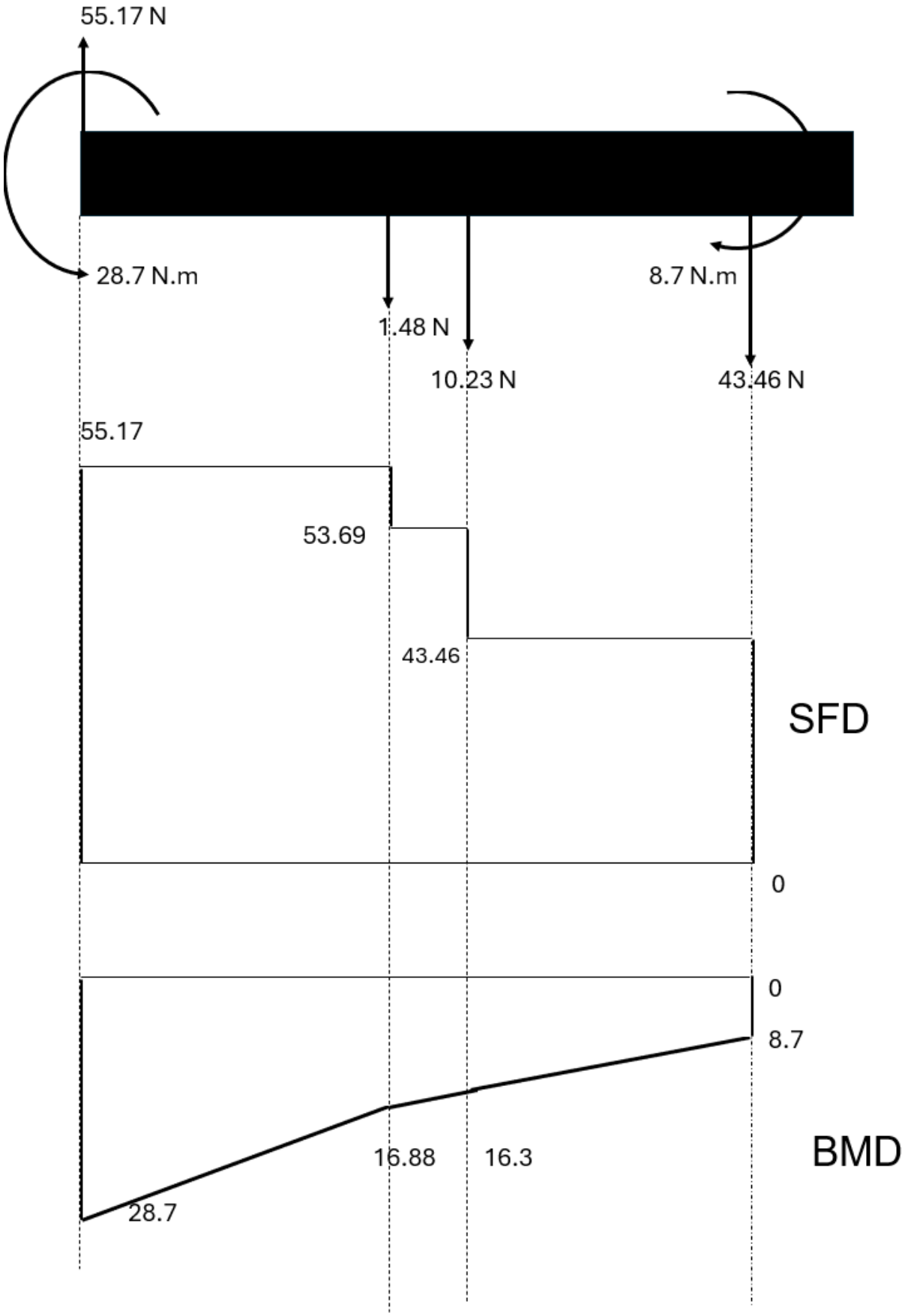
- **stress**



- **deflection**



BMD & SFD:


55.17 N
28.7 N.m
1.48 N
10.23 N
8.7 N.m
43.46 N
55.17
53.69
43.46
SFD
0
0
8.7
16.88
16.3
28.7
BMD

Bending Stress:

$$\because \sigma_b = \frac{M \times y}{I} = \frac{M \times \left(\frac{H}{2}\right)}{\frac{1}{12} \times (BH^3 - bh^3)} = \frac{6M \times H}{BH^3 - bh^3}$$

$$\therefore \sigma_b = \frac{6 \times 28.7 \times 0.05}{(0.1\ x\ (0.05)^3) - (0.094\ x\ (0.044)^3)} = \textbf{1.9164405 Mpa}$$

Shear Stress:

$$\because \tau = \frac{v \times Q}{I \times t} \quad \because Q = A \times \bar{y} = 8.4672 \times 10^{-6}\ mm3$$

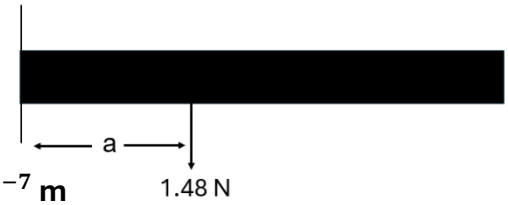
$$\therefore \tau = \frac{55.17 \times 8.46 \times 10^{-6}}{\frac{3}{12} \times 10^{-3} \times ((0.1\ x\ (0.05)^3) - (0.094\ x\ (0.044)^3))}$$

$$\therefore \tau = \textbf{0.41590581 Mpa}$$

Deflection:

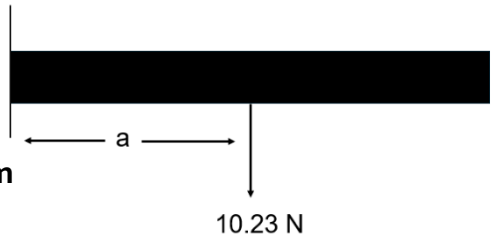
we used the superposition method to determine the deflection

$$\because \delta_1 = \frac{Fa^2(3L - a)}{6EI} = \frac{Fa^2(3L - a)}{E \times \frac{1}{2}\left(BH^3 - bh^3\right)}$$



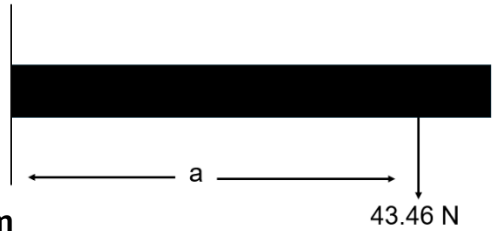
$$\therefore \delta_1 = \frac{1.48 \times (0.21421)^2 \times (1.13579)}{2.246 \times 10^{-6} \times 68.9 \times 10^9} = \textbf{4.984369899} \times \textbf{10}^{-7}\ \textbf{m}$$

$$\because \delta_2 = \frac{Fa^2(3L - a)}{6EI} = \frac{Fa^2(3L - a)}{E \times \frac{1}{2}\left(BH^3 - bh^3\right)}$$



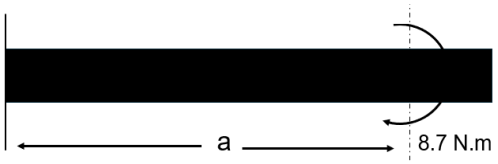
$$\therefore \delta_2 = \frac{10.23 \times (0.225)^2 \times (1.125)}{2.246 \times 10^{-6} \times 68.9 \times 10^9} = \textbf{3.764993394} \times \textbf{10}^{-6}\ \textbf{m}$$

$$\because \delta_3 = \frac{Fa^2(3L - a)}{6EI} = \frac{Fa^2(3L - a)}{E \times \frac{1}{2}\left(BH^3 - bh^3\right)}$$



$$\therefore \delta_3 = \frac{43.46 \times (0.4)^2 \times (0.95)}{2.246 \times 10^{-6} \times 68.9 \times 10^9} = \textbf{4.268785533} \times \textbf{10}^{-5}\ \textbf{m}$$

$$\because \boldsymbol{\delta_4} = \frac{Fa^2(3L-a)}{6EI} = \frac{Fa^2(3L-a)}{E \times \frac{1}{2}\left(BH^3 - bh^3\right)}$$



$$\therefore \boldsymbol{\delta_4} = \frac{8.7 \times (0.4)^2 \times 0.5}{68.9 \times 10^9 \times 7{\cdot}487 \times 10^{-7}} = \mathbf{1.349218 \times 10^{-5}\ m}$$

$$\because \boldsymbol{\delta_{total}} = \delta_1 + \delta_2 + \delta_3 + \delta_4$$

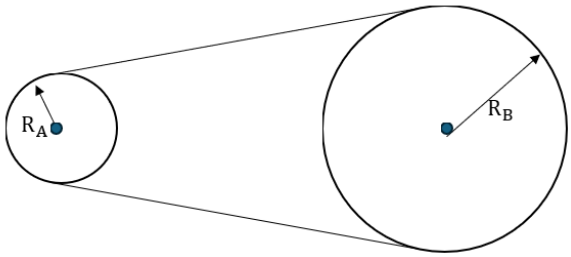
$$\therefore \boldsymbol{\delta_{total}} = 4.984369899 \times 10^{-7} + 3.764993394 \times 10^{-6} + 4.268785533 \times 10^{-5} + 1.349218 \times 10^{-5} = \mathbf{0.060443\ mm}$$

2.6 Motor Sizing and Motion Profiles

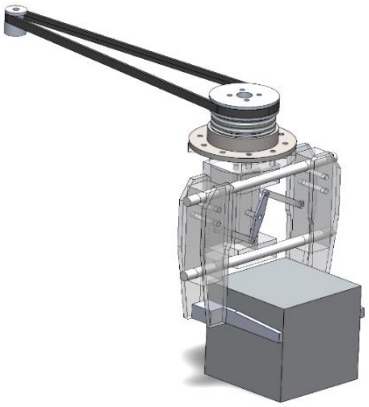
INTRODUCTION

Choosing the right motor is an essential part of designing the SCARA robot. The motor needs to be powerful enough to handle the job but not so large that it wastes energy or increases costs. Motion profiles define how the motor starts, moves, and stops. One of the most common and practical motion profiles is the trapezoidal curve. This profile has three parts: a smooth increase in speed (acceleration), a steady speed (constant velocity), and a smooth decrease in speed (deceleration). In this part we will calculate the torque required by each motor and its Motion Profile.

2.6.1 Motor (4):



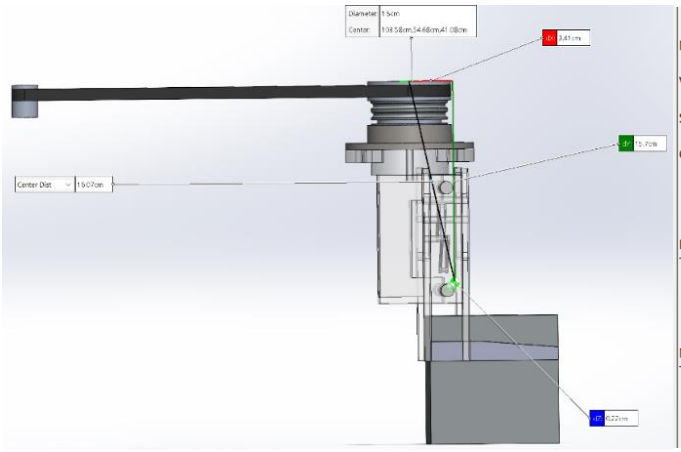
GEAR MECHANISM DESIGN



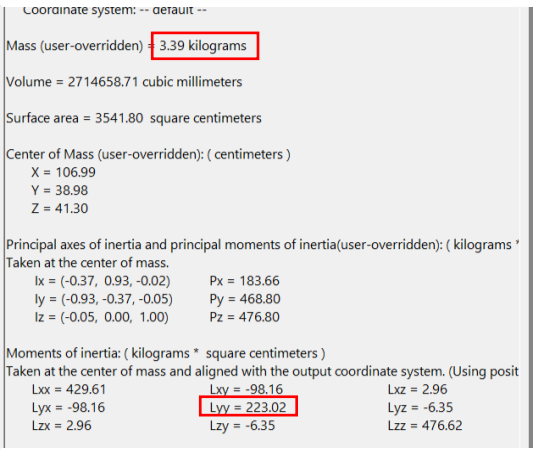
MECHANICAL PARTS LOADED ON MOTOR (4)

- **Reduction ratio between Pulley A and Pulley B:**

$$N = \frac{R_B}{R_A} = \frac{29.15}{7.31} = 3.987$$



CENTER OF MASS OF THE LOAD



MASS MOMENT OF INERTIA OF THE LOAD

- **Mass moment of inertia on pulley B (Joint):**

By applying parallel axis theorem mass moment of inertia:

$$\therefore I = I_{yy} + M\,d^2$$

$$\therefore I = 223.02 + 3.39 \text{ x } (3 \cdot 41)^2$$

$$\therefore I = \boxed{262.439 \text{ Kg.cm}^2}$$

- **Torque on Pulley B (Joint):**

Torque = Mass moment of inertia x Angular Acceleration

$$\tau_{joint} = I \text{ x } \alpha$$

$$\therefore \tau_{joint} = 262.439 \text{ x } 0.9$$

$$\therefore \tau_{joint} = \boxed{236.195 \text{ N.cm}}$$

- **Torque of Motor (4):**

$$\tau_{Motor} = \frac{\tau_{joint}}{N} \qquad \therefore \tau_{Motor} = \frac{236.195}{3.987} = \boxed{59.241 \text{ N.cm}}$$

- **Friction Torque of Motor (4):**

$$\tau_{Friction} = \mu\,\tau_{Motor} \qquad \therefore \tau_{Friction} = 0.2 \text{ x } 59.241 = \boxed{11.848 \text{ N.cm}}$$

- **Accelerating Torque of Motor (4):**

$$T_{acc} = \tau_{Motor} + \tau_{Friction} = 59.241 + 11.848 = \boxed{71.089 \text{ N.cm}}$$

- **decelerating Torque of Motor (4):**

$$T_{dec} = -\tau_{Motor} + \tau_{Friction} = -59.241 + 11.848 = \boxed{-47.393 \text{ N.cm}}$$

- **Running Torque of Motor (4):**

$$T_{run} = \tau_{Friction} = \boxed{11.848 \text{ N.cm}}$$

- **Trapezoidal Motion Profile of Motor (4):**
  - The End Effector can rotate 360^0 = 2π rad
  - The End Effector takes 9 seconds to rotate 360^0
  - Angular acceleration = 0.9 rad/sec2 = 51.5 degree/sec2

$$t_t = t_a + t_d + t_m \longrightarrow t_m = t_t - 2t_a = 9 - 2t_a$$

$$\omega_m = \alpha_m \times t_a = 0.9\, t_a$$

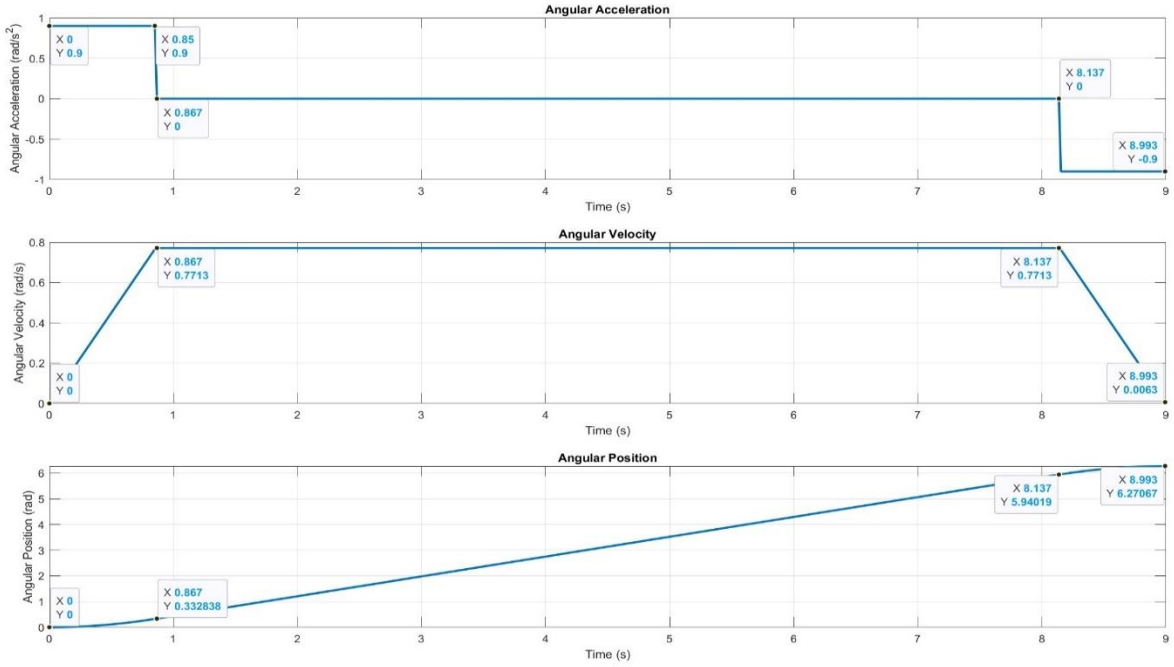
$$\because \theta = (\tfrac{1}{2} \times \alpha_m \times t_a^2) + (\omega_m \times t_m) + (\tfrac{1}{2} \times \alpha_m \times t_d^2)$$

$$\therefore 2\pi = \left(\tfrac{1}{2} \times 0.9 \times t_a^2\right) + 0.9 t_a \times (9 - 2t_a) + (\tfrac{1}{2} \times 0.9 \times t_a^2)$$

$$\therefore t_a = t_d = \boxed{0.857 \text{ second}} \qquad \therefore t_m = \boxed{7.286 \text{ seconds}}$$

$$\therefore \omega_m = 0.9 \times 0.857 = \boxed{0.772 \text{ rad/sec}}$$

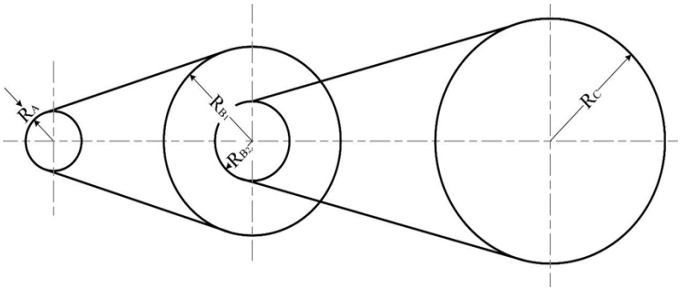
- **Plot of the Trapezoidal Motion Profile of Motor (4) in MATLAB:**



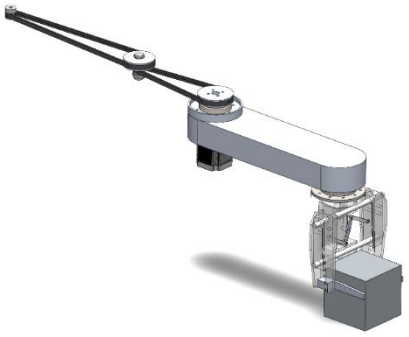
- **Root Mean Square Torque of Motor (4):**

$$T_{rms} = \sqrt{\frac{T_{acc}^2 \times t_a + T_{dec}^2 \times t_d + T_{run}^2 \times t_m}{t_t}} \quad \therefore T_{rms} = \boxed{28.438 \text{ N.cm}}$$

2.6.2 Motor (3):



GEAR MECHANISM DESIGN



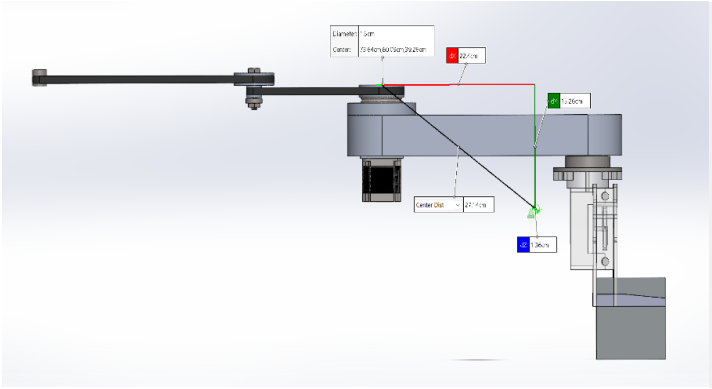
MECHANICAL PARTS LOADED ON MOTOR (3)

- **Reduction ratio between Pulley B and Pulley C:**

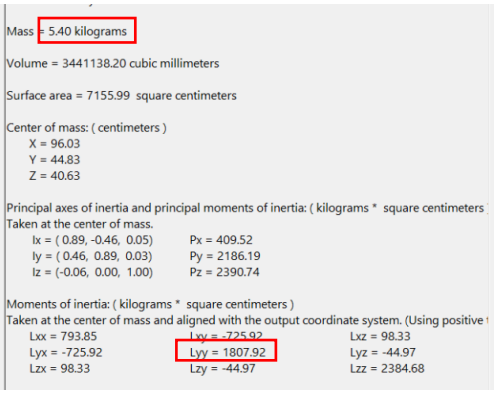
$$N_2 = \frac{R_C}{R_{B2}} = \frac{30.23}{8.265} = 3.658$$

- **Reduction ratio between Pulley A and Pulley B:**

$$N_1 = \frac{R_{B1}}{R_A} = \frac{26.36}{7.5} = 3.515$$



CENTER OF MASS OF THE LOAD



MASS MOMENT OF INERTIA OF THE LOAD

- **Mass moment of inertia on pulley C (Joint):**

By applying parallel axis theorem mass moment of inertia:

$$\therefore I = I_{yy} + M\,d^2$$

$$\therefore I = 1807.92 + 5.4 \times (22.4)^2$$

$$\therefore I = \boxed{4517.424\ Kg.cm^2}$$

- **Torque on Pulley C (Joint):**

Torque = Mass moment of inertia x Angular Acceleration

τ_{joint} = I x α

$\therefore \tau_{joint}$ = 4517.424 x 0.5

$\therefore \tau_{joint}$ = 2258.712 N.cm

- **Torque of Motor (3):**

$$\tau_{Motor} = \frac{\tau_{joint}}{N_1 x\, N_2} \qquad \therefore \tau_{Motor} = \frac{2258.712}{3.515 \text{ x } 3.658} = 175.667 \text{ N.cm}$$

- **Friction Torque of Motor (3):**

$$\tau_{Friction} = \mu\, \tau_{Motor} \qquad \therefore \tau_{Friction} = 0.2 \text{ x } 175.667 = 35.13 \text{ N. cm}$$

- **Accelerating Torque of Motor (3):**

$$T_{acc} = \tau_{Motor} + \tau_{Friction} = 175.667 + 35.13 = 210.797 \text{ N. cm}$$

- **decelerating Torque of Motor (3):**

$$T_{dec} = -\tau_{Motor} + \tau_{Friction} = -175.667 + 35.13 = -140.537 \text{ N. cm}$$

- **Running Torque of Motor (3):**

$$T_{run} = \tau_{Friction} = 35.13 \text{ N. cm}$$

- **Trapezoidal Motion Profile of Motor (3):**
  - Link (2) can rotate $360^0 = 2\pi$ rad
  - Link (2) takes 10 seconds to rotate 360^0
  - Angular acceleration = 0.5 rad/sec2 = 28.65 degree/sec2

$$t_t = t_a + t_d + t_m \longrightarrow t_m = t_t - 2t_a = 10 - 2t_a$$

$$\omega_m = \alpha_m \times t_a = 0.5\, t_a$$

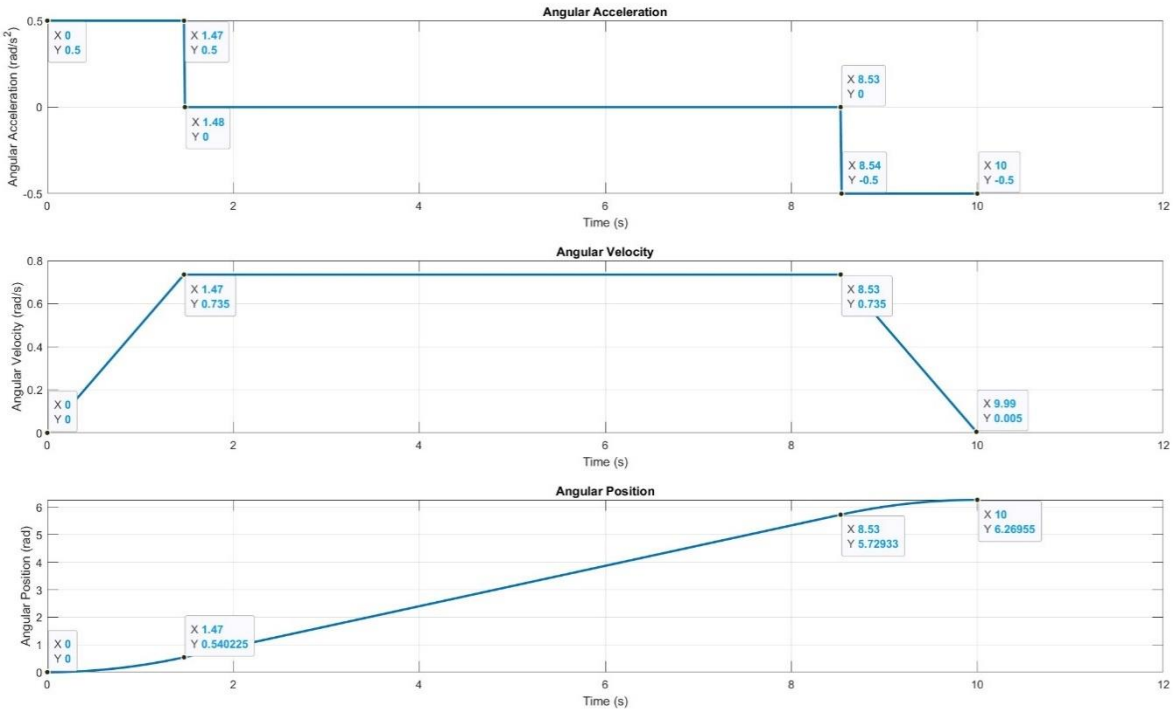
$$\because \theta = (\tfrac{1}{2} \times \alpha_m \times t_a^2) + (\omega_m \times t_m) + (\tfrac{1}{2} \times \alpha_m \times t_d^2)$$

$$\therefore 2\pi = \left(\tfrac{1}{2} \times 0.5 \times t_a^2\right) + 0.5t_a \times (10 - 2t_a) + (\tfrac{1}{2} \times 0.5 \times t_a^2)$$

$$\therefore t_a = t_d = \boxed{1.47\ seconds} \qquad \therefore t_m = \boxed{7.06\ seconds}$$

$$\therefore \omega_m = 0.5 \times 1.47 = \boxed{0.735\ rad/sec}$$

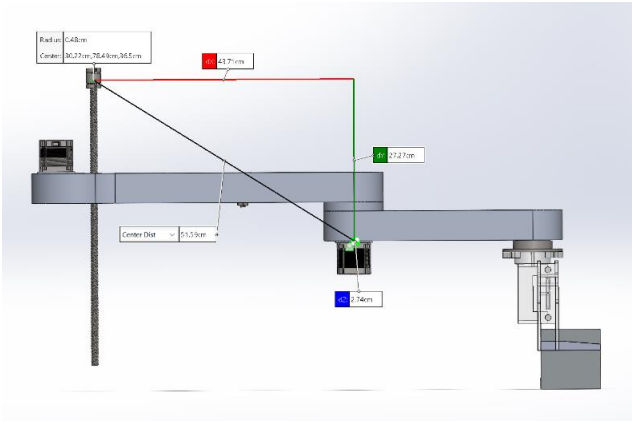
- **Plot of the Trapezoidal Motion Profile of Motor (3) in MATLAB:**



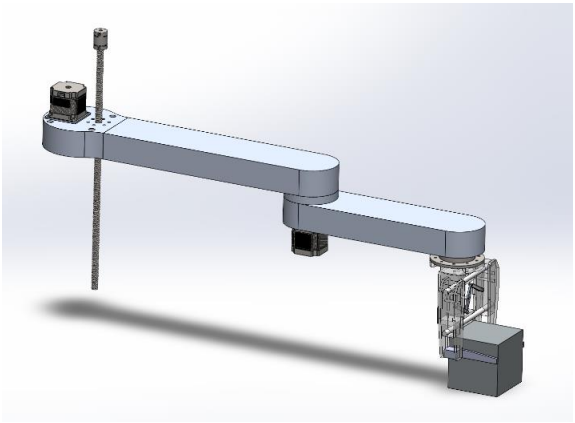
- **Root Mean Square Torque of Motor (3):**

$$T_{rms} = \sqrt{\frac{T_{acc}^2 \times t_a + T_{dec}^2 \times t_d + T_{run}^2 \times t_m}{t_t}} \quad \therefore T_{rms} = \boxed{101.52\ N.cm}$$

2.6.3 Motor (2):



CENTER OF MASS OF THE LOAD



MECHANICAL PARTS LOADED ON MOTOR (2)

Mass = 8.55 kilograms

Volume = 4233437.14 cubic millimeters

Surface area = 11792.38 square centimeters

Center of mass: (centimeters)
X = 73.93
Y = 51.22
Z = 39.24

Principal axes of inertia and principal moments of inertia: (kilograms * square centimet
Taken at the center of mass.

Ix = (0.95, -0.30, 0.06)	Px = 647.06
Iy = (0.30, 0.95, 0.02)	Py = 10722.58
Iz = (-0.06, 0.00, 1.00)	Pz = 11115.73

Moments of inertia: (kilograms * square centimeters)
Taken at the center of mass and aligned with the output coordinate system. (Using posit

Lxx = 1580.88	Lxy = -2862.49	Lxz = 599.43
Lyx = -2862.49	Lyy = 9826.44	Lyz = -179.44
Lzx = 599.43	Lzy = -179.44	Lzz = 11078.04

MASS MOMENT OF INERTIA OF THE LOAD

- **Mass moment of inertia on Power Screw (Joint):**

By applying parallel axis theorem mass moment of inertia:

$$\therefore I = I_{yy} + M\,d^2$$

$$\therefore I = 9826.44 + 8.55 \times (43.71)^2$$

$$\therefore I = \boxed{26161.76\ Kg.cm^2}$$

- **Torque of Motor (2):**

Torque = Mass moment of inertia x Angular Acceleration

$$\tau_{Motor} = \frac{I \times \alpha}{2\pi\, p_{screw}}$$

$$\therefore \tau_{Motor} = \frac{26161.76 \times 0.007}{2\pi \times 0.25}$$

$$\therefore \tau_{Motor} = \boxed{116.585\ N.cm}$$

- **Friction Force:**

$$F_{Friction} = \mu Weight \quad \therefore F_{Friction} = 0.2 \times 8.55 \times 9.806 = \boxed{16.768\ N}$$

- **Load Force:**

$$F_{Load} = Weight = 8.55 \times 9.806 = \boxed{83.84\ N}$$

- **Gravity Force:**

  - **Ascending**

$$F_G = +Weight = 8.55 \times 9.806 = \boxed{83.84\ N}$$

  - **Descending**

$$F_G = -Weight = -8.55 \times 9.806 = \boxed{-83.84\ N}$$

- **Ascending Accelerating Torque of Motor (2):**

$$T_{acc} = \tau_{Motor} + \frac{F_L + F_F + F_G}{2\pi\, p_{screw}} = 116.585 + \frac{83.84 + 16.768 + 83.84}{2\pi\ 0.25} = \boxed{234.01\ N.cm}$$

- **Ascending decelerating Torque of Motor (2):**

$$T_{dec} = -\tau_{Motor} + \frac{F_L+F_F+F_G}{2\pi\, p_{screw}} = -116.585 + \frac{83.84+16.768+83.84}{2\pi\ 0.25} = \boxed{0.84\ N.cm}$$

- **Ascending Running Torque of Motor (2):**

$$T_{run} = \frac{F_L+F_F+F_G}{2\pi\, p_{screw}} = \frac{83.84+16.768+83.84}{2\pi\ 0.25} = \boxed{117.423\ N.cm}$$

- **Descending Accelerating Torque of Motor (2):**

$$T_{acc} = \tau_{Motor} + \frac{F_L+F_F-F_G}{2\pi\, p_{screw}} = 116.585 + \frac{83.84+16.768-83.84}{2\pi\ 0.25} = \boxed{127.26\ N.cm}$$

- **Descending decelerating Torque of Motor (2):**

$$T_{dec} = -\tau_{Motor} + \frac{F_L+F_F-F_G}{2\pi\, p_{screw}} = -116.585 + \frac{83.84+16.768-83.84}{2\pi\ 0.25} = \boxed{-105.91\ N.cm}$$

- **Descending Running Torque of Motor (2):**

$$T_{run} = \frac{F_L+F_F-F_G}{2\pi\, p_{screw}} = \frac{83.84+16.768-83.84}{2\pi\ 0.25} = \boxed{10.675\ N.cm}$$

- **Trapezoidal Motion Profile of Motor (2):**
  - Prismatic joint can travel 30 cm = 0.3 m
  - The Pitch of the ball screw is 2.5mm/revolution
  - Total time to travel 30cm equals 12 seconds

$$\therefore\ t_a = t_d = t_m = \frac{12}{3} = \boxed{4\ seconds}$$

$$\because s = \left(\frac{1}{2} v_{max} \times t_a\right) + (v_{max} \times t_m) + \left(\frac{1}{2} v_{max} \times t_d\right)$$

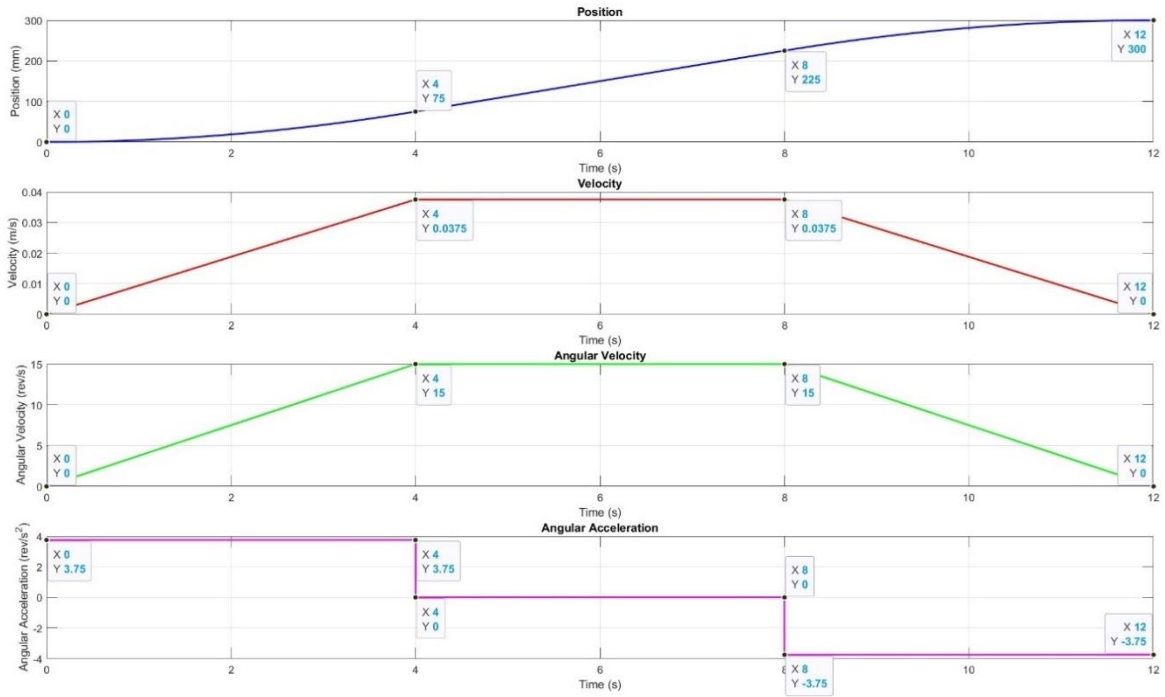
$$\therefore 0.3 = \left(\frac{1}{2} v_{max} \times 4\right) + (v_{max} \times 4) + \left(\frac{1}{2} v_{max} \times 4\right)$$

$$\therefore\ v_{max} = \frac{0.3}{8} = 0.0375\ m/\sec = 37.5\ mm/sec$$

$$\because\ \omega_m = \frac{v_{max}}{Pitch} \qquad \therefore \omega_m = \frac{37.5}{2.5} = \boxed{15\ rev/sec}$$

$$\because \mathrm{a} = \frac{v_{max}}{t_a} \qquad \therefore \mathrm{a} = \frac{37.5}{4} = \boxed{9.375\ mm\ /\ \sec 2}$$

- **Plot of the Trapezoidal Motion Profile of Motor (2) in MATLAB:**



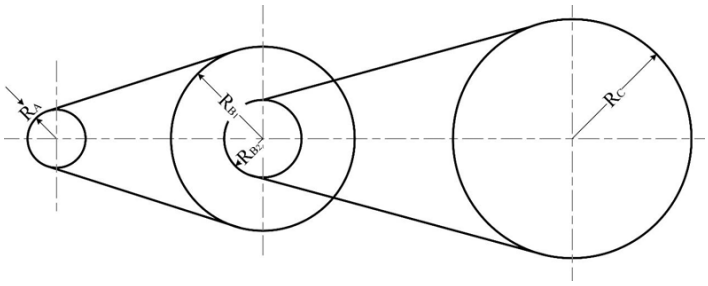
- **Ascending Root Mean Square Torque of Motor (2):**

$$T_{rms} = \sqrt{\frac{T_{acc}^2 \times t_a + T_{dec}^2 \times t_d + T_{run}^2 \times t_m}{t_t}} \quad \therefore T_{rms} = \boxed{151.16 \text{ N.cm}}$$

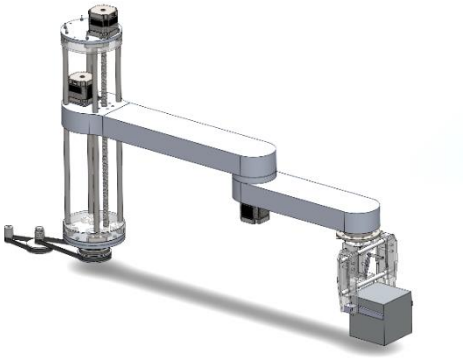
- **Descending Root Mean Square Torque of Motor (2):**

$$T_{rms} = \sqrt{\frac{T_{acc}^2 \times t_a + T_{dec}^2 \times t_d + T_{run}^2 \times t_m}{t_t}} \quad \therefore T_{rms} = \boxed{95.788 \text{ N.cm}}$$

2.6.4 Motor (1):



GEAR MECHANISM DESIGN



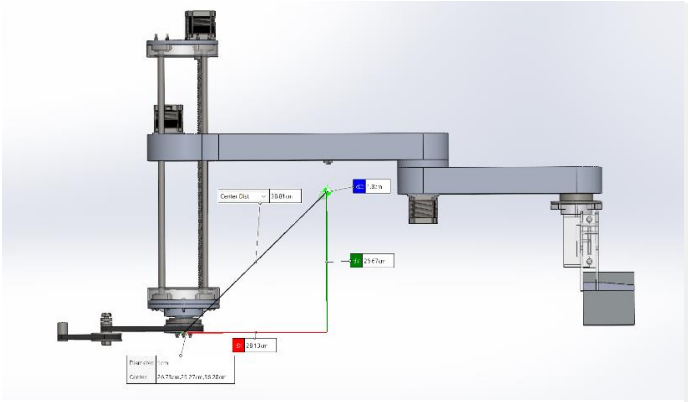
MECHANICAL PARTS LOADED ON MOTOR (1)

- **Reduction ratio between Pulley B and Pulley C:**

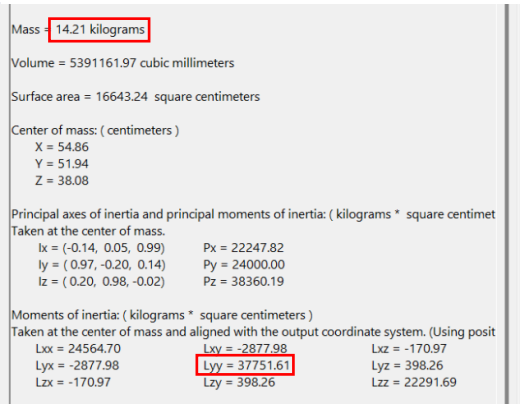
$$N_2 = \frac{R_c}{R_{B2}} = \frac{35.96}{7.95} = 4.52$$

- **Reduction ratio between Pulley A and Pulley B:**

$$N_1 = \frac{R_{B1}}{R_A} = \frac{26.41}{7.31} = 3.61$$



CENTER OF MASS OF THE LOAD



MASS MOMENT OF INERTIA OF THE LOAD

- **Mass moment of inertia on pulley C (Joint):**

By applying parallel axis theorem mass moment of inertia:

$$\therefore I = I_{yy} + M\,d^2$$

$$\therefore I = 37751.61 + 14.21 \text{ x } (28.13)^2$$

$$\therefore \boxed{I = 48995.94 \text{ Kg.cm}^2}$$

- **Torque on Pulley C (Joint):**

Torque = Mass moment of inertia x Angular Acceleration

$$\tau_{joint} = I \text{ x } \alpha$$

$$\therefore \tau_{joint} = 48995.94 \text{ x } 0.068$$

$$\therefore \tau_{joint} = \boxed{3331.72 \text{ N.cm}}$$

- **Torque of Motor (1):**

$$\tau_{Motor} = \frac{\tau_{joint}}{N_1 \text{x } N_2} \qquad \therefore \tau_{Motor} = \frac{3331.72}{4.52 \text{ x } 3.61} = \boxed{204.18 \text{ N.cm}}$$

- **Friction Torque of Motor (1):**

$$\tau_{Friction} = \mu\,\tau_{Motor} \qquad \therefore \tau_{Friction} = 0.2 \text{ x } 204.18 = \boxed{40.836 \text{ N.cm}}$$

- **Accelerating Torque of Motor (1):**

$$T_{acc} = \tau_{Motor} + \tau_{Friction} = 204.18 + 40.836 = \boxed{245.016 \text{ N.cm}}$$

- **decelerating Torque of Motor (1):**

$$T_{dec} = -\tau_{Motor} + \tau_{Friction} = -204.18 + 40.836 = \boxed{-163.344 \text{ N.cm}}$$

- **Running Torque of Motor (1):**

$$T_{run} = \tau_{Friction} = \boxed{40.836 \text{ N.cm}}$$

- **Trapezoidal Motion Profile of Motor (1):**
  - Robot body can rotate $240^0 = \frac{4}{3}\pi$ rad
  - Robot body takes 16 seconds to rotate 240^0
  - Angular acceleration = 0.068 rad/sec2 = 3.896 degree/sec2

$$t_t = t_a + t_d + t_m \longrightarrow t_m = t_t - 2t_a = 16 - 2t_a$$

$$\omega_m = \alpha_m \times t_a = 0.068\, t_a$$

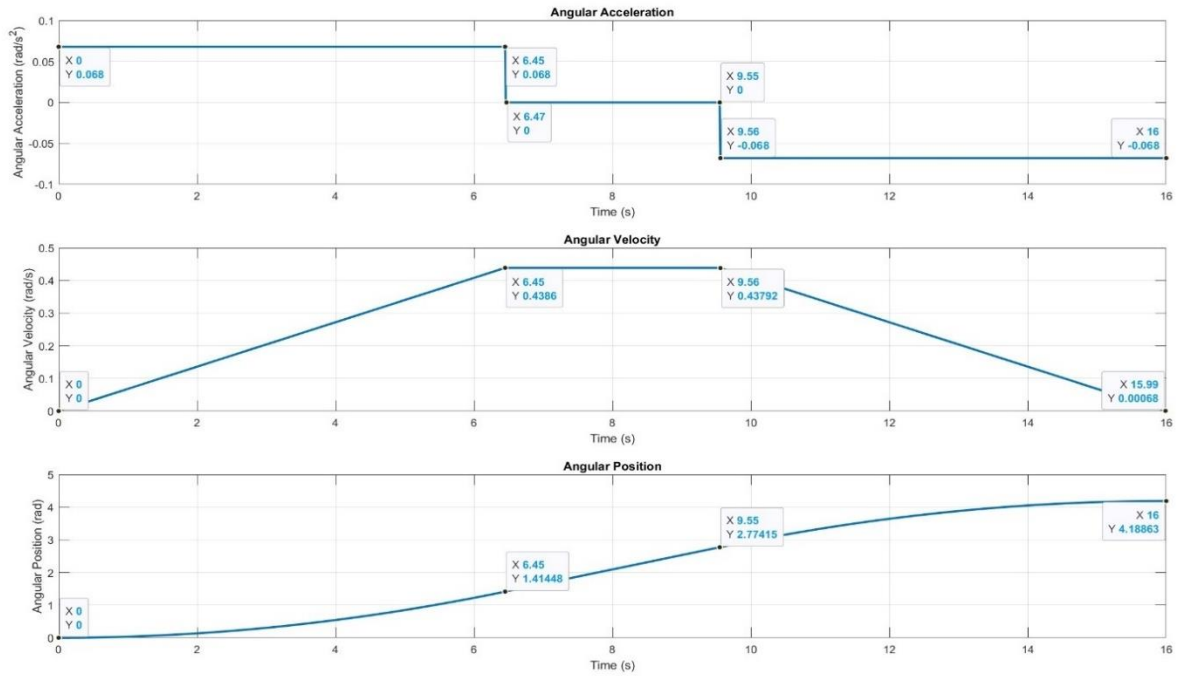
$$\because \theta = (\tfrac{1}{2} \times \alpha_m \times t_a^2) + (\omega_m \times t_m) + (\tfrac{1}{2} \times \alpha_m \times t_d^2)$$

$$\therefore \frac{4}{3}\pi = \left(\frac{1}{2} \times 0.068 \times t_a^2\right) + 0.068 t_a \times (16 - 2t_a) + (\tfrac{1}{2} \times 0.068 \times t_a^2)$$

$$\therefore t_a = t_d = \boxed{6.45\ seconds} \qquad \therefore t_m = \boxed{3.1\ seconds}$$

$$\therefore \omega_m = 0.068 \times 6.45 = \boxed{0.44\ rad/sec}$$

- **Plot of the Trapezoidal Motion Profile of Motor (1) in MATLAB:**



- **Root Mean Square Torque of Motor (1):**

$$T_{rms} = \sqrt{\frac{T_{acc}^2 \times t_a + T_{dec}^2 \times t_d + T_{run}^2 \times t_m}{t_t}} \quad \therefore T_{rms} = \boxed{187.839\ \text{N.cm}}$$

2.7 Power screw

2.7.1 Introduction

A power screw is a mechanical mechanism used to convert rotational motion into linear motion while being able to withstand large loads. It is mainly used in applications that require lifting or adjusting heavy loads, such as in jacks and mechanical presses.

A power screw consists of a screw with threaded grooves (such as square or acme threads) and a nut that matches these threads. When rotational motion is applied to the screw, the nut moves linearly along the screw.

Power screws are known for their ability to handle large forces due to the high thread angle. However, their efficiency is generally lower compared to other systems like gears, due to the friction in the threads that generates energy loss. Efficiency can be improved by using low-friction materials or lubricants.

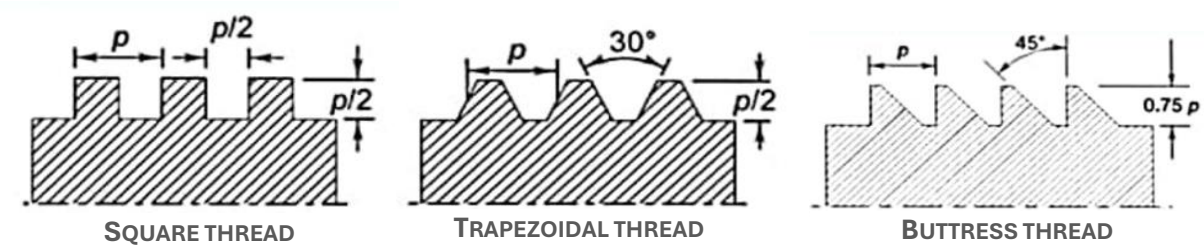
Power screws come in several types depending on the shape of the threads used, and these types affect the load-bearing capacity, efficiency, and speed. The main types of power screws are:

1. **Square Thread Power Screw:**
   - Features square-shaped threads, providing high load-bearing capacity and efficiency.
   - Primarily used in applications that require heavy loads, such as jacks and heavy machinery.
   - It is more efficient at converting rotational motion into linear motion.

2. **Acme Thread Power Screw:**
   - The threads have two sharp angles with a wider base, making it more durable and less prone to wear.
   - Used in applications requiring faster and more effective power transmission with improved efficiency.
   - It offers more flexibility than square threads and is commonly used in light to medium industries.

3. **Buttress Thread Power Screw:**
   - Features threads shaped like the letter "V" and is primarily used to withstand large forces in one direction.
   - It is designed to efficiently handle heavy loads in one direction, making it ideal for applications like heavy lifting or machines requiring high force in one direction.

4. **Lead Screw:**
   - It has circular threads that provide linear motion transmission with moderate efficiency.
   - Used in applications that require motion conversion between parallel surfaces, such as in precise machinery and tools.
   - Each of these types is used according to the application's requirements in terms of force, efficiency, and speed.

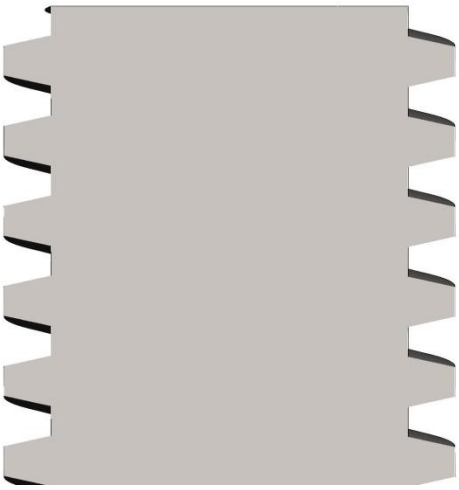


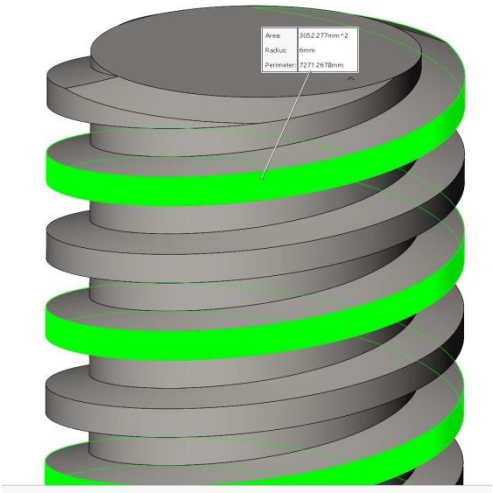
SQUARE THREAD TRAPEZOIDAL THREAD BUTTRESS THREAD

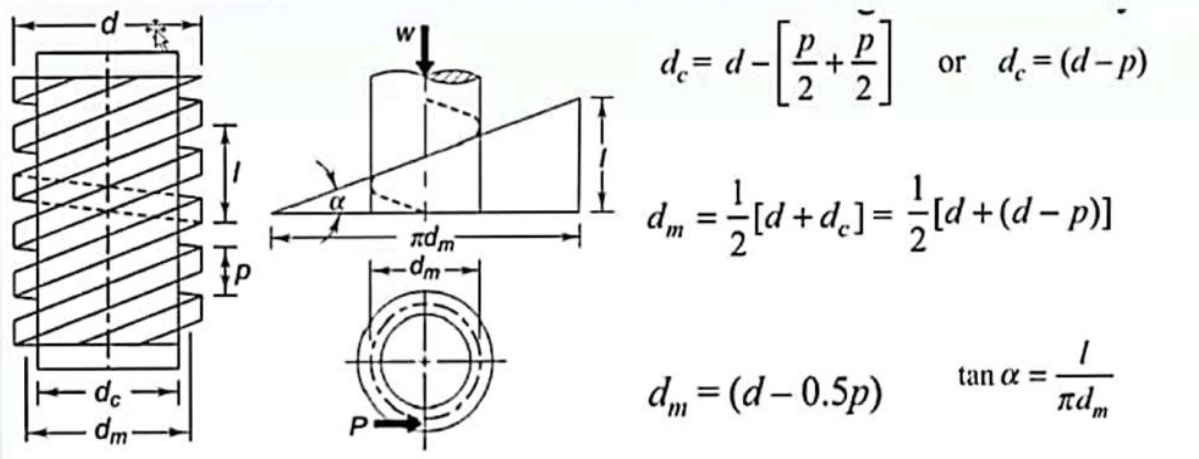
2.7.2 Advantages of Acme Trapezoidal Power Screw:

1) **High Efficiency in Motion Transfer:**
   - Acme screws offer good efficiency in converting rotational motion into linear motion, compared to other types like square threads. This makes them suitable for applications requiring smooth movement and good performance.
2) **High Load-Bearing Capacity:**
   - The threads of the Acme screw are designed with a steeper angle, which provides a higher load-bearing capacity and better resistance to pressure.
3) **Wear Resistance:**
   - Due to the specific angled design of the Acme threads, they have good resistance to wear compared to some other types of threads.
4) **Flexibility in Use:**
   - Acme threads can be adjusted to suit different needs in terms of force and speed, making them adaptable for a wide variety of industrial applications.
5) **Ease of Manufacturing:**
   - The design of Acme threads makes them easier to manufacture compared to square threads, which reduces the overall production cost.
6) **Ability to Operate in Both Directions:**
   - Acme screws work efficiently in both directions (lifting and lowering), making them suitable for devices requiring reciprocating movement.
7) **Resistance to Friction:**
   - Acme threads have lower friction compared to other types of screws, enhancing their efficiency and reducing the need for lubricants.

- **Due to these advantages, Acme screws are widely used in industrial applications requiring motion transfer with the ability to handle heavy loads, such as jacks, hydraulic systems, and mechanical adjustment devices.**







$$d_c = d - \left[\frac{p}{2} + \frac{p}{2}\right] \quad \text{or} \quad d_c = (d - p)$$

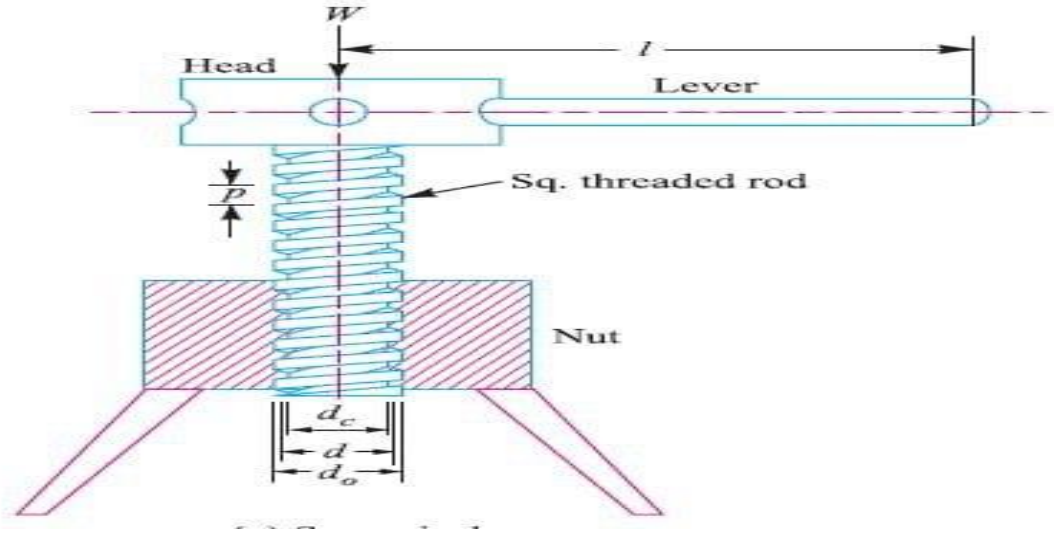
$$d_m = \frac{1}{2}[d + d_c] = \frac{1}{2}[d + (d - p)]$$

$$d_m = (d - 0.5p) \qquad \tan\alpha = \frac{l}{\pi d_m}$$

- **Lifting Torque**
  - $\mathbf{T = \frac{W.d_m}{2}.(\tan\varphi + \alpha)}$ $\qquad \boldsymbol{tan\,\alpha = \frac{l}{\pi.d_m}}$ $\qquad \boldsymbol{\tan\varphi = \mu}$
- **Lowering Torque**
  - $\mathbf{T = \frac{W.d_m}{2}.(\tan\varphi - \alpha)}$
- **Collar friction torque**
  - $\mathbf{T_c = \frac{\mu_c.W}{4}(D_o - D_i)}$
- **Total torque**
  - $\mathbf{T_T = T + T_c}$
- **Lifting force, Lowering force**
  - $\mathbf{F = \frac{T_T}{radius}}$
- **Efficiency**
  - $\boldsymbol{\eta = \frac{W.L}{2\pi.T_T}}$

2.7.3 Mechanical stress equations (Design of screw, Nut):

The body of screw is subjected to an axial force and torsional moment T_t



tensile stress $\sigma = \frac{W}{\frac{\pi}{4}{d_c}^2}$ **, shear stress** $\tau = \frac{16*T_t}{\pi.(d_c)^3}$, $\tau_{max} = \sqrt{\left(\frac{\sigma}{2}\right)^2 + \tau^2}$

- **the teeth of Nut that connected to screw is subjected to direct shear:**
  - $\tau_s = \frac{w}{\pi d_i t\, z}$ **t** the thickness of teeth , **z** is the number of teeth
- **Shear stress** $\tau_n = \frac{w}{\pi d_o t\, z}$
- **The bearing pressure on the thread in the nut:**
  - $S_b = \frac{4W}{\pi z(d^2 - {d_c}^2)}$ $z = \frac{length\ of\ nut}{p}$
- **Safety factor**
  - $\mathbf{SF} = \frac{\textbf{Yield strength}}{\textbf{Maximum Applied stress}}$ if SF is less than 2 or 3 the power is not safe to use
- **Friction and efficiency test**
  - If efficiency(η) is too low (less than 30%), this means significant energy loss and the screw is not perfect.

The weight of end effector and two arm is 8.5 Kg ,power screw made of aluminum $\sigma_y = 240MPa$, $\mu = 0.15$, $D_o = 12mm$ $D_i = 10mm$ $step\ L = 2.5mm$

Solution

F=W*.9.81=8. 5*9.81 =83. 385N $d_m = \frac{d_o + d_i}{2} = \frac{10+12}{2} = 11mm$

Helix angle $\tan\alpha = \frac{l}{\pi.d_m} = \frac{2.5}{11*\pi} = 0.0723$, $\alpha = 4.14°$

$\tan\varphi = 0.15$, $\varphi = 8.53°$ $\eta = \frac{\tan\alpha}{\tan(\alpha+\varphi)} = \frac{0.072}{0.2246} = 0.322\ (32.2\%)$

$T_{up} = \frac{W.d_m}{2}.(\tan\varphi + \alpha) = 83.385 * 11 * \frac{\tan 12.67}{2} = 103\ Nmm\ \ 0.103Nm$

$T_{down} = \frac{W.d_m}{2}.(\tan\varphi - \alpha) = 83.385 * 11 * \frac{\tan(8.53-4.14)}{2} = 35.2\ Nmm$

0.0352Nm

$If\ \varphi > \alpha$ the screw is self_locking

tensile stress $\sigma_a = \frac{F}{A} = \frac{83.385}{78.54} = 1.06MPa$ $A = \frac{\pi}{4}{d_i}^2 = \frac{\pi}{4}.10^2 = 78.5mm^2$

shear stress $\tau = \frac{16*103}{\pi.(11)^3} = 0.394\frac{N}{mm^2}$ $\tau_{al} = 0.6 * \sigma_y = 0.6 * 240 = 144\ N/mm^2$

The resulting shear stress is very small compared to the permissible stress, which means that the design is safe in terms of shear stress.

Chapter 3
Electrical Components and Circuits

Chapter 3
Electrical Components and Circuits

3.1 Introduction

Electrical components and circuits are essential for making a SCARA robot work. They control the motors, read sensors, and power the entire system. Without these elements, the robot wouldn't be able to move, sense objects, or complete tasks. In this chapter, we will introduce the basic electrical parts needed for a SCARA robot, like motors, drivers, sensors, microcontrollers, and power supplies. We'll also look at simple circuits that connect these components, such as motor control circuits and circuits for reading sensor data.

3.2 Electrical Components and Circuits

3.2.1 Stepper Motor NEMA 23 (Actuator):

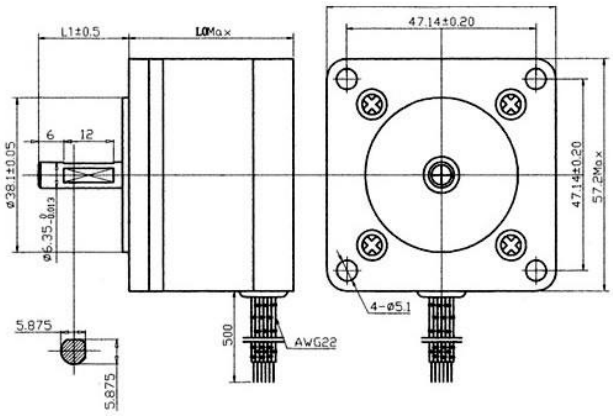
The NEMA 23 stepper motor is a widely used component in various applications due to its precise control and reliability.

NEMA 23 STEPPER MOTOR

- **Motor Specifications:**
  - Step Angle: 1.8° per step, resulting in 200 steps per revolution.
  - Holding Torque: Varies by model common values include 1.9 Nm to 2.4 Nm.
  - Current Rating: Typically around 2.8 A per phase.
  - Voltage Rating: Approximately 3.2 V.
  - Number of Phases: 2-phase bipolar.
  - Number of Leads: Commonly 4 leads for bipolar configuration.
  - Mass: 1.05 Kg.
  - Temperature rise: 80°C Max.
  - Insulation Class: Class B.

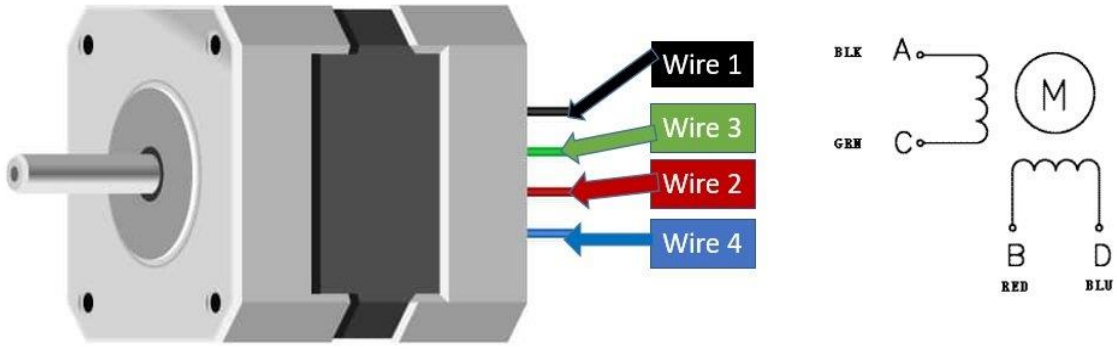
- **Mechanical Dimensions:**

(Dimensions in mm)



NEMA 23 STEPPER MOTOR MECHANICAL

- **Connections:**



NEMA 23 STEPPER MOTOR CONNECTIONS

3.2.2 Stepper Motor NEMA 17 (Actuator):

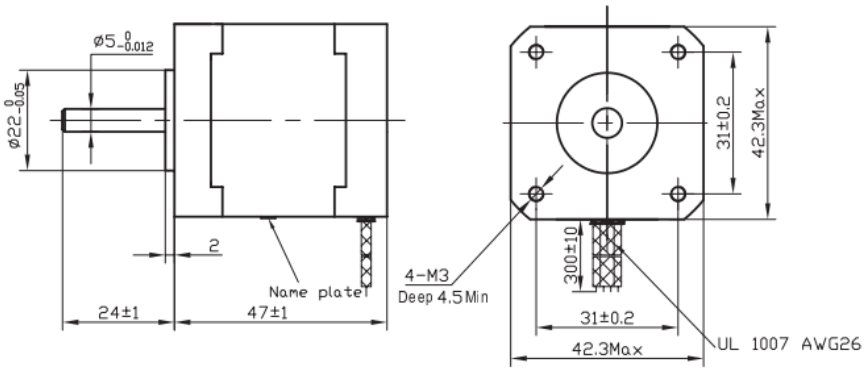
NEMA 17 STEPPER

- **Motor Specifications:**

  - Step Angle: 1.8° per step, resulting in 200 steps per revolution.
  - Holding Torque: Varies by model common values include 0.28 Nm to 0.6 Nm.
  - Current Rating: Typically around 1.2 A per phase.
  - Voltage Rating: Approximately 4 V.
  - Number of Phases: 2-phase bipolar.
  - Number of Leads: Commonly 6 leads for bipolar configuration.
  - Mass: 0.365 Kg.
  - Temperature rise: 80°C Max.
  - Insulation Class: Class B.

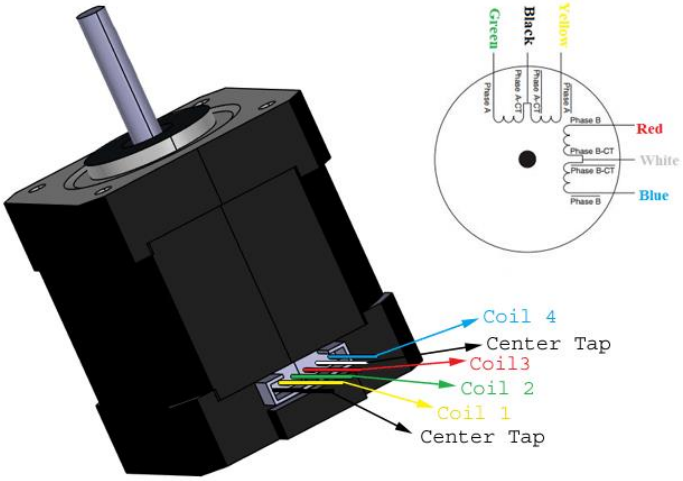
- **Mechanical Dimensions:**

(Dimensions in mm)



NEMA 17 STEPPER MOTOR MECHANICAL

Connections:



NEMA 17 STEPPER MOTOR CONNECTIONS

3.2.3 Stepper Motor Driver A4988 (Driver):

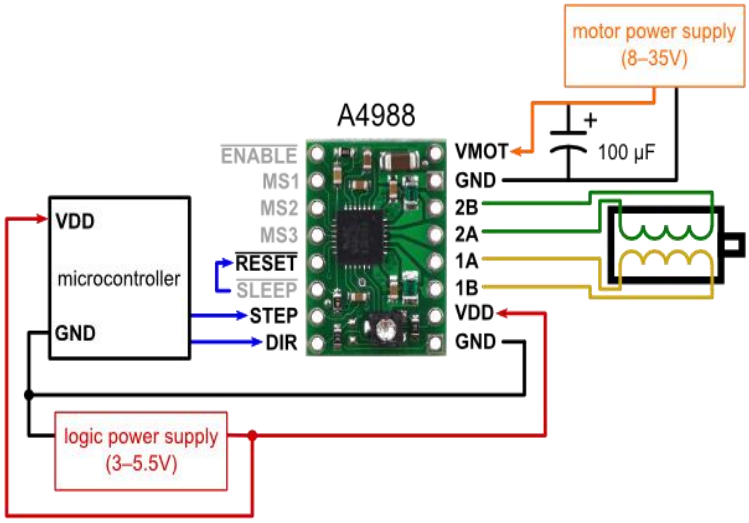
The A4988 is a micro stepping driver for controlling bipolar stepper motors which has built-in translator for easy operation. This means that we can control the stepper motor with just 2 pins from our controller, one for controlling the rotation direction and the other for controlling the steps.

STEPPER MOTOR DRIVER A4988

- **Specifications:**

  - Max. Operating Voltage: 35V.
  - Min. Operating Voltage: 8V.
  - Max. Current Per Phase: 2A.
  - Micro step resolution: Full step, $\frac{1}{2}$ step, $\frac{1}{4}$ step, $\frac{1}{8}$ step and $\frac{1}{16}$ step.
  - No Reverse voltage protection.
  - Dimensions: 15.5 × 20.5 mm.
  - Short-to-ground and shorted-load protection.

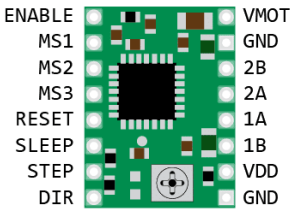
- **Connection:**



STEPPER MOTOR DRIVER A4988 CONNECTIONS

- **Pin Configuration:**

Pin Name	Description
VDD & GND	Connected to 5V and GND of Controller
VMOT & GND	Used to power the motor
1A, 1B, 2A, 2B	Connected to the 4 coils of motor
DIRECTION	Motor Direction Control pin
STEP	Steps Control Pin
MS1, MS2, MS3	Micro step Selection Pins
SLEEP	Pins For Controlling Power States
RESET	
ENABLE	



STEPPER MOTOR DRIVER A4988 PINS

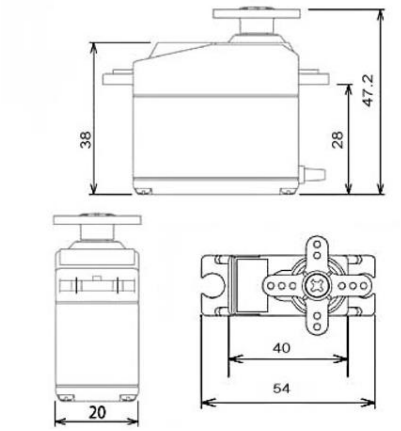
3.2.4 MG995 Servo Motor (Actuator):

MG995 servo is a simple, commonly used standard servo for your mechanical needs such as robotic head, robotic arm. It comes with a standard 3-pin power and control cable for easy use and metal gears for high torque.

MG995 SERVO MOTOR

- **Mechanical Dimensions:**

(Dimensions in mm)

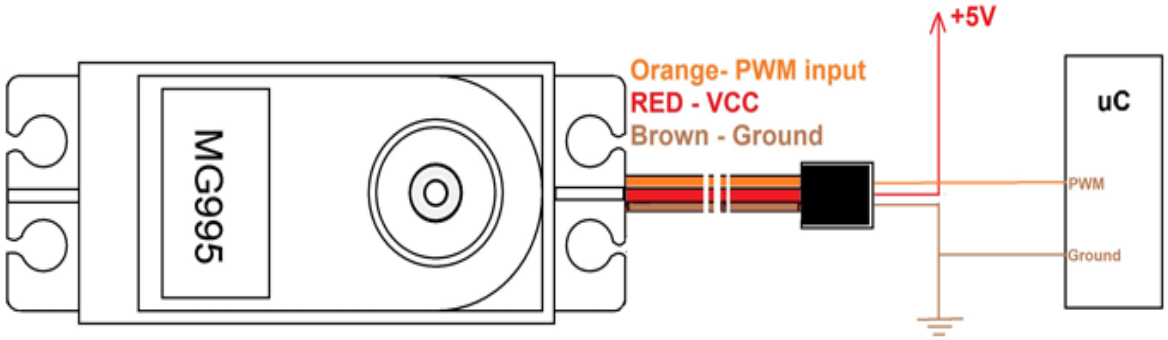


MG995 SERVO MOTOR
MECHANICAL DIMENSIONS

- **Specifications:**

- Metal geared servo for more life.
- Stable and shock proof double ball bearing design.
- High speed rotation for quick response.
- Stall torque: 9.4kg/cm (4.8v); 11kg/cm (6v).
- Operating speed: 0.2 s/60º (4.8 V), 0.16 s/60º (6 V).
- Rotational degree: 180º.
- Current draw at idle: 10mA.
- No load operating current draw: 170mA.
- Current at maximum load: 1200mA.
- Weight: 55g.

- **Connections:**



MG995 SERVO MOTOR CONNECTIONS

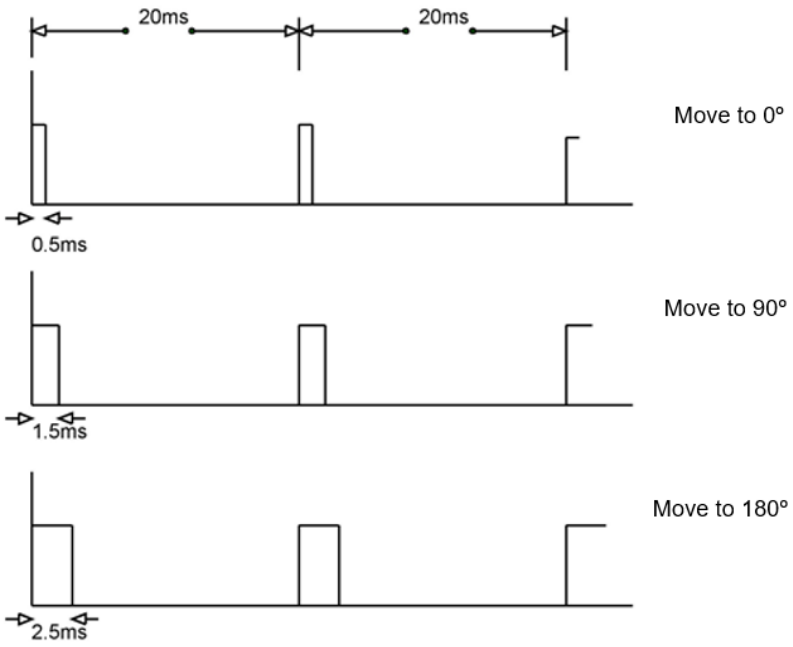
- **Working Principle:**

Frequency of PWM: MG995 takes in PWM signal of frequency 50Hz and any higher and lower frequency PWM will lead to error. As shown in figure every single cycle of PWM needs to be 20ms width for 50Hz frequency.

Duty cycle of PWM: The duty cycle of PWM (or ratio of ON time to total cycle time) determines the position of servo axis. If we provide a PWM signal of 0.5ms ON time over 20mS complete cycle, the servo axis will move to 0º.

And if we provide a PWM signal of 1.5ms ON time over 20ms complete cycle, the servo axis will move to 90º. If we provide a PWM signal of 2.5ms ON time over 20mS complete cycle, the servo axis will move to 180º.

Based on these standard values we can also calculate any other degree of rotation. After calculation we just have to adjust the duty cycle of the PWM for the servo to read the signal and change to that stated position.

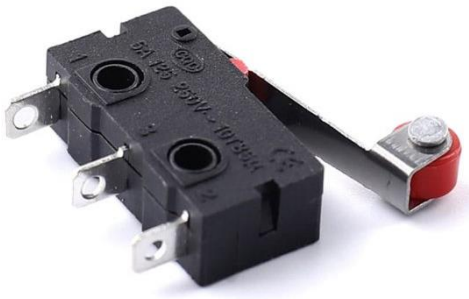


MG995 SERVO MOTOR PWM

3.2.5 Limit Switch (Sensor):

A limit switch is an electromechanical device that detects the presence or absence of an object or the position of a part. It operates by making or breaking electrical connections when a physical force is applied to its actuator. Limit switches are widely used in industrial applications for position detection, end-of-travel limits, and safety functions.

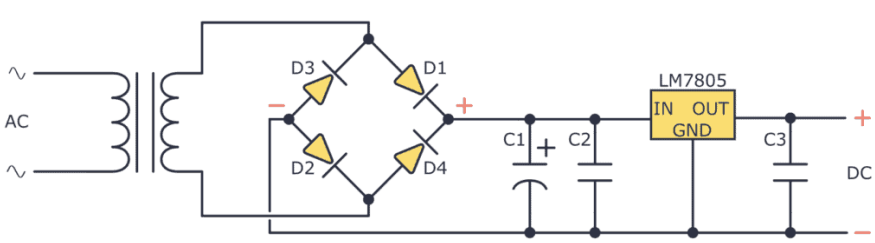
LIMIT SWITCH SCHEMATIC SYMBOL



LIMIT SWITCH

3.2.6 Dc Power Supply (Power Supply):

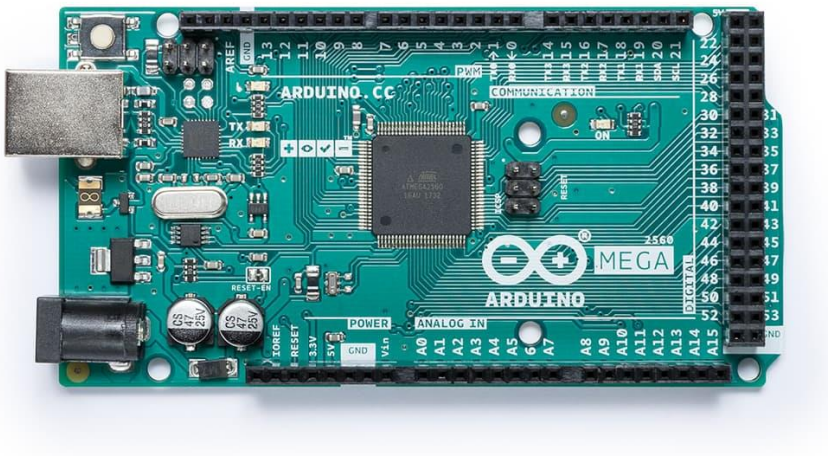
used to power the electrical components.



DC POWER SUPPLY CIRCUIT DIAGRAM

3.2.7 Arduino Mega 2560 (Motors Controller):

Arduino Mega 2560 is a microcontroller board based on the ATmega2560. It is designed for projects that require more input/output pins, memory, and additional functionalities compared to other Arduino boards like the Arduino Uno.



ARDUINO MEGA 2560

- **Specifications:**

  - Microcontroller: ATmega2560.
  - Operating Voltage: 5V.
  - Input Voltage: 6-20V.
  - Digital I/O Pins: 54 (of which 15 can be PWM).
  - Analog Input Pins: 16.
  - DC Current per I/O Pin: 20 mA.
  - DC Current for 3.3V Pin: 50 mA.
  - Flash Memory: 256 KB (8 KB used by bootloader).
  - SRAM: 8 KB.
  - EEPROM: 4 KB.
  - Clock Speed: 16 MHz.
  - Dimensions: 101.52 mm x 53.3 mm.

- **Pins Description:**

1. Power Pins:

- Vin: Input voltage to the board when using an external power source (6-12V).
- 5V: Regulated 5V output to power external components.
- 3.3V: 3.3V output generated by the onboard voltage regulator.
- GND: Ground pins.
- IOREF: Provides voltage reference for the microcontroller.

2. Digital Pins:

- The board has 54 digital I/O pins.
- Pins 0 to 53 can be configured as input or output.
- Pins 2 and 3 support external interrupts.
- PWM capable pins: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 44, 45,46.

3. Analog Input Pins:

- There are 16 analog input pins (A0 to A15).
- These pins can read signals from analog sensors and convert them to a digital value between 0 and 1023.

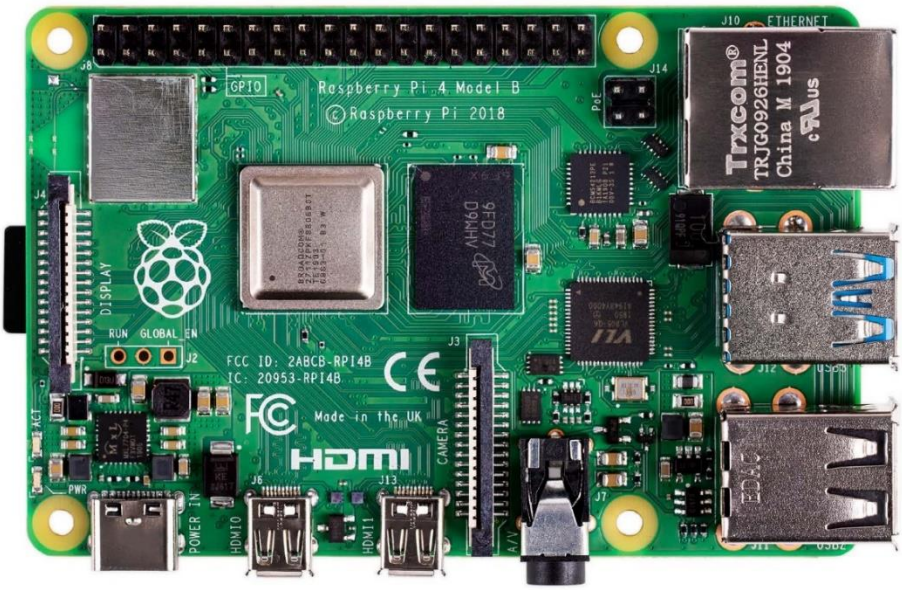
4. Communication Pins:

- UART (Serial Communication):
  - Serial 0: Pins 0 (RX), 1 (TX).
  - Serial 1: Pins 19 (RX), 18 (TX).
  - Serial 2: Pins 17 (RX), 16 (TX).
  - Serial 3: Pins 15 (RX), 14 (TX).
- SPI: Pins: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).
- I2C: SDA: Pin 20, SCL: Pin 21.

5. Special Function Pins:

- LED: Pin 13 has a built-in LED connected.
- Reset: Resets the microcontroller when activated.

3.2.8 Raspberry PI 4 (processor for machine vision and complex computations):

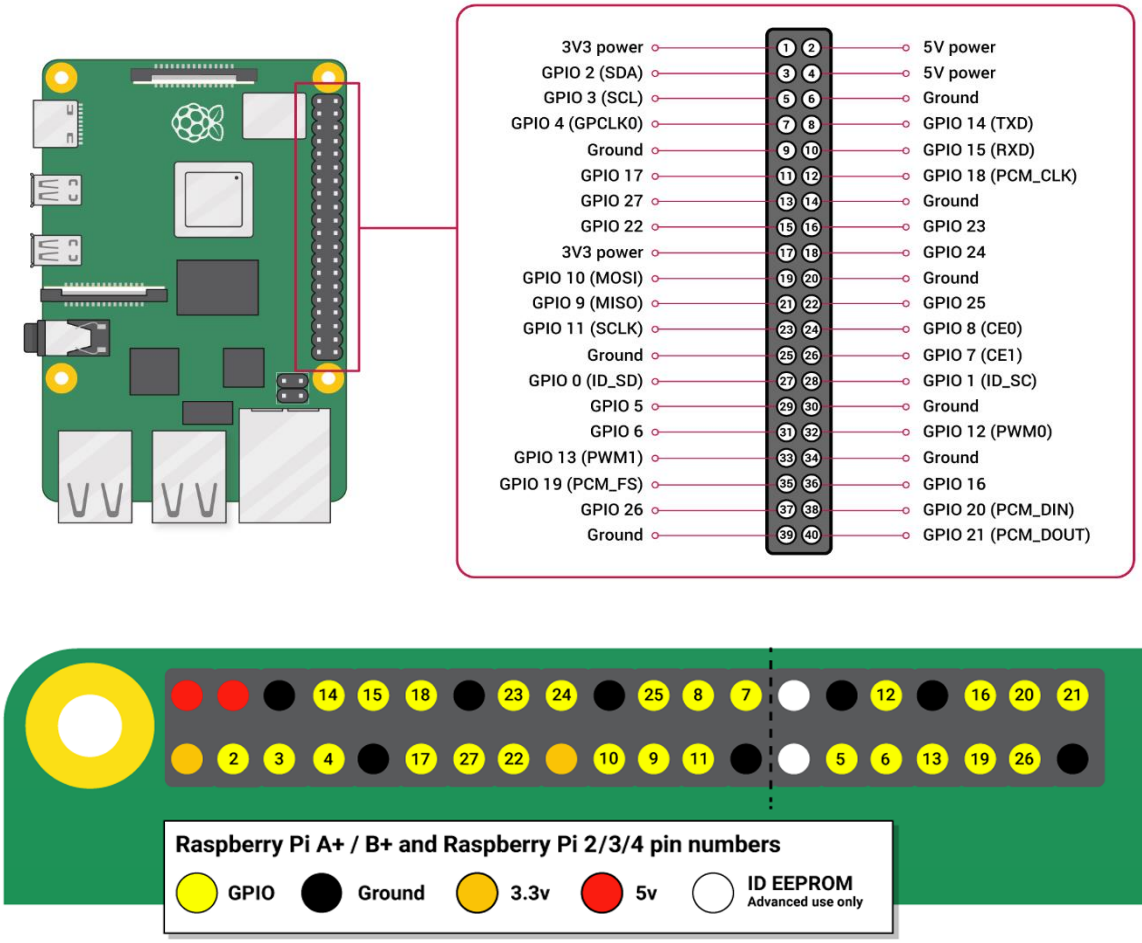


RASPBERRY PI 4

- **Specifications:**

- Processor: Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- Memory: Available in 2GB, 4GB, and 8GB LPDDR4 SDRAM variants
- Connectivity:2.4GHz and 5.0GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0 with BLE Gigabit Ethernet
- Video and Display: Dual micro-HDMI ports supporting up to 4Kp60 resolution
- Storage: microSD card slot for operating system and data storage
- USB Ports:
    - 2 USB 3.0 ports
    - 2 USB 2.0 ports
- Power Supply: 5V/3.0A DC via USB-C connector
- GPIO: 40-pin GPIO header
- Dimensions: Board Size: 85.6mm × 56.5mm.
- Operating System: Raspberry Pi OS and other Linux distributions.

- **GPIO Pinout:**



RASPBERRY PI 4 GENERAL PURPOSE INPUT/ OUTPUT PINS (GPIO)

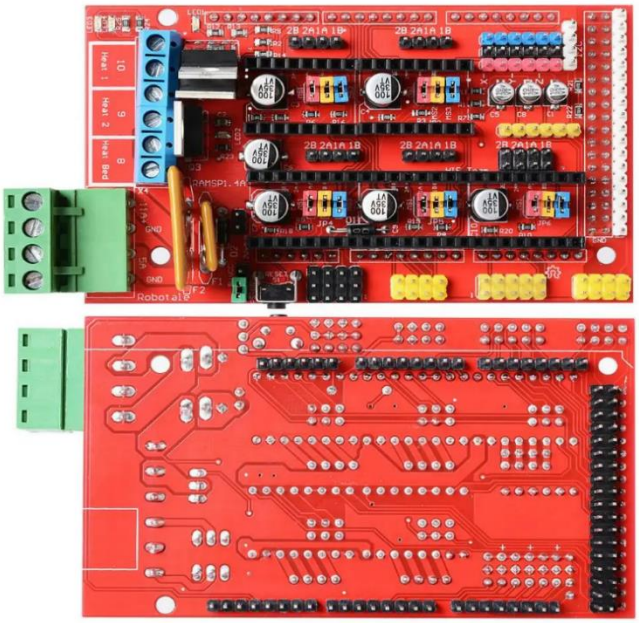
- **Connecting Raspberry pi 4 with Arduino Mega 2560:**

  - Serial Communication (UART): Use the GPIO pins on the Raspberry Pi (TXD and RXD) to establish a UART connection with the TX and RX pins on the Arduino Mega.

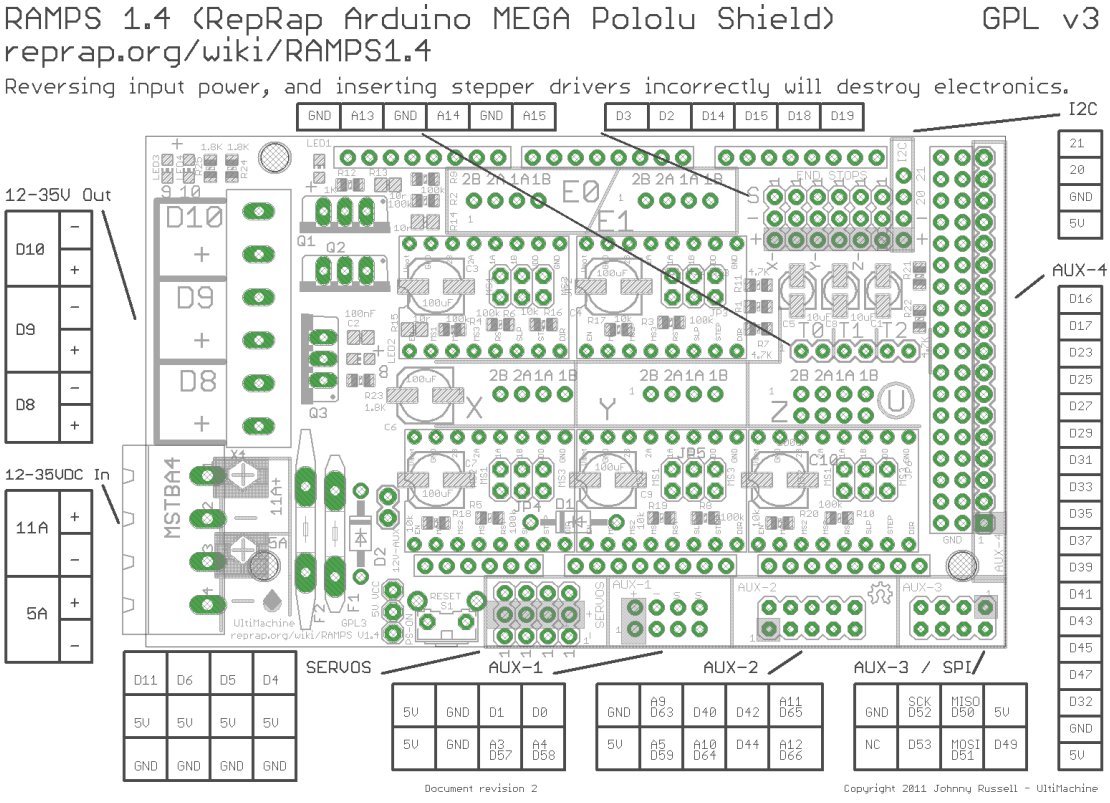
  - USB Connection: Connect the Arduino Mega to the Raspberry Pi via a USB cable, The Raspberry Pi will recognize the Arduino Mega as a serial device, Use Python libraries such as pyserial to send and receive data.

3.2.9 RepRap Arduino Mega Pololu Shield (RAMP1.4):

RAMPS stands for RepRap Arduino Mega Pololu Shield. It is a Motion Control System controller board designed to stack on an Arduino Mega 2560.Popular for its modular design and ease of integration into custom Motion Control System. Acts as the nerve center for controlling components like motors and Sensors (limit switches).



RAMP 1.4

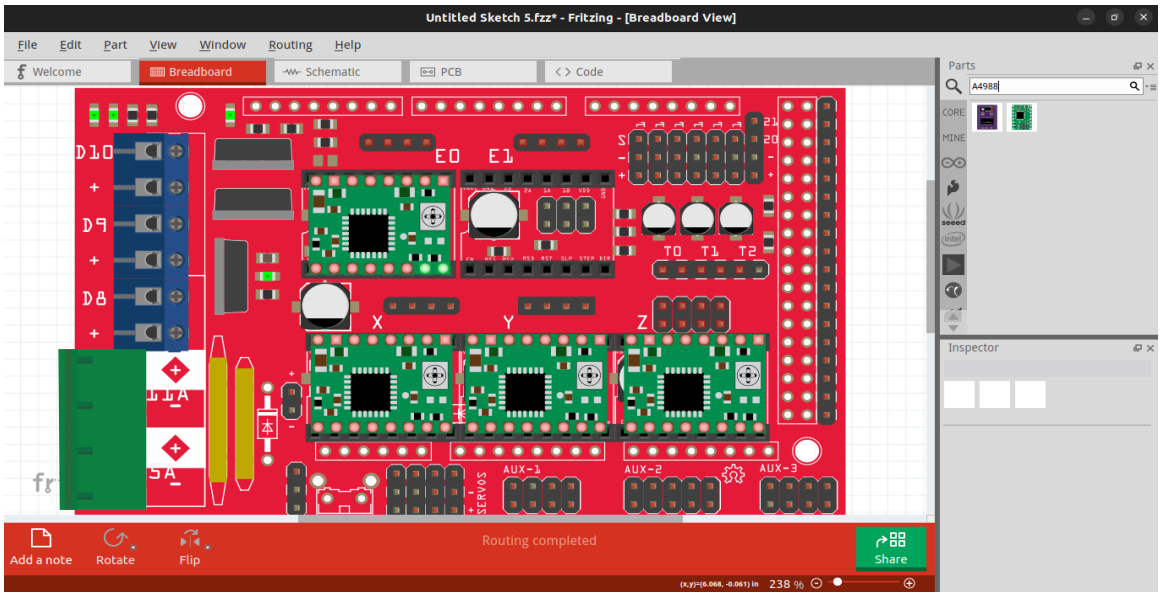


RAMP 1.4 SCHEMATIC

- **Specifications:**

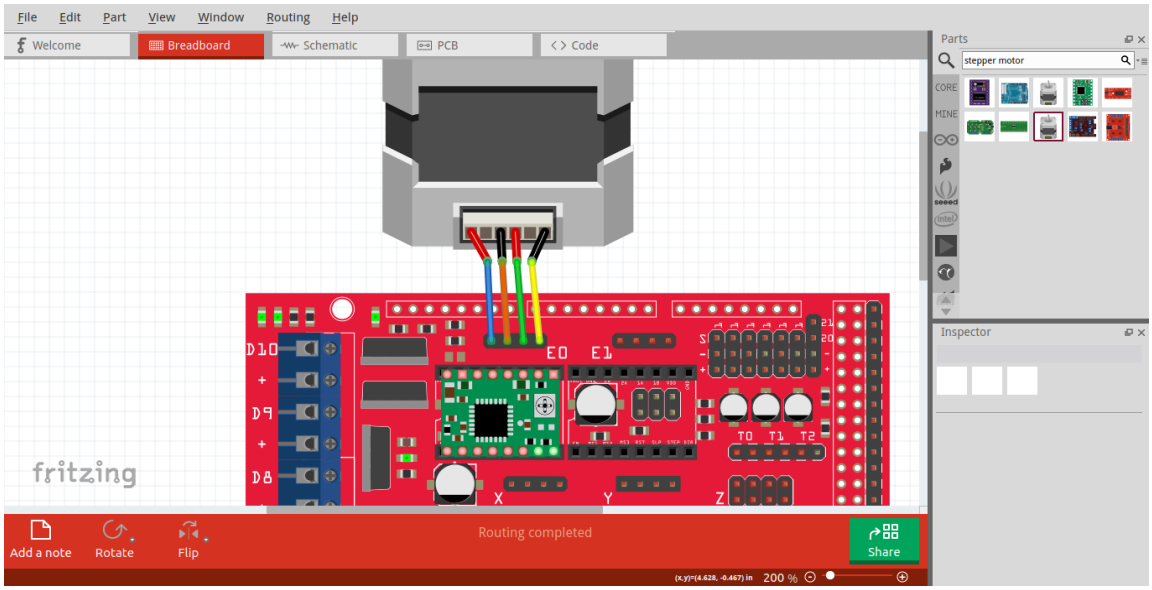
  - Works with Arduino Mega 2560.
  - Supports up to 5 stepper motor drivers (A4988).
  - Endstops: 3 sets of endstop connectors.
  - Handles 12V or 24V power supply.
  - Compatible with a variety of firmware like Marlin.

- **RAMP1.4 and A4988 Stepper motor driver Connections:**



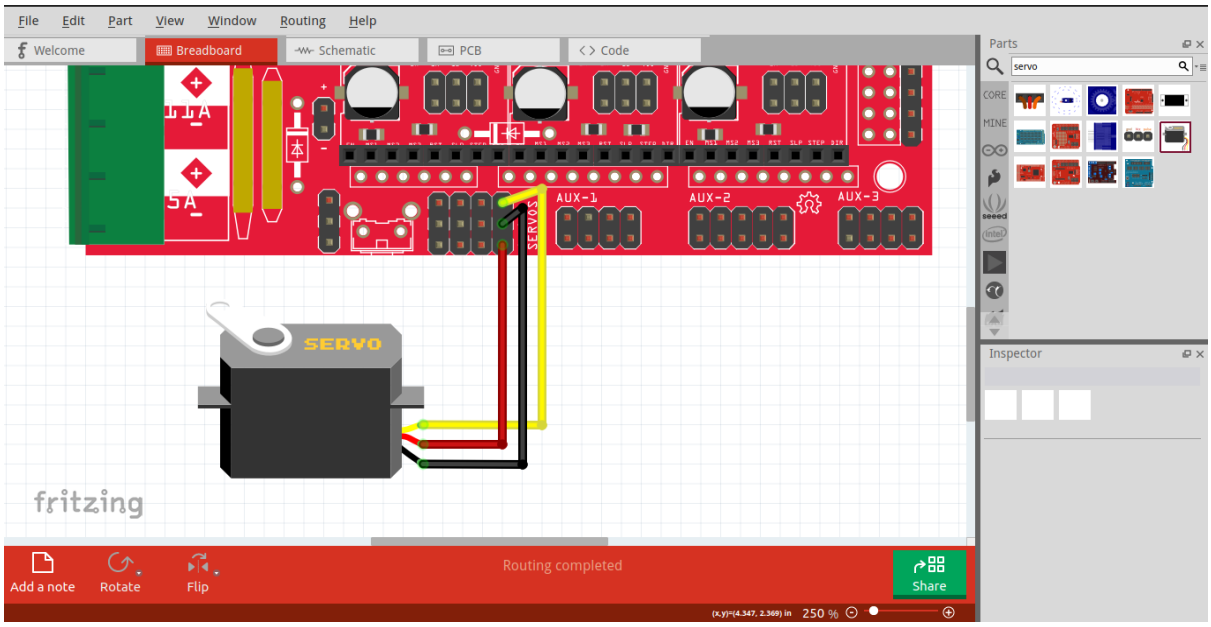
RAMP 1.4 AND A4988 STEPPER MOTOR DRIVER

- **RAMP1.4 and Stepper Motor Connections:**



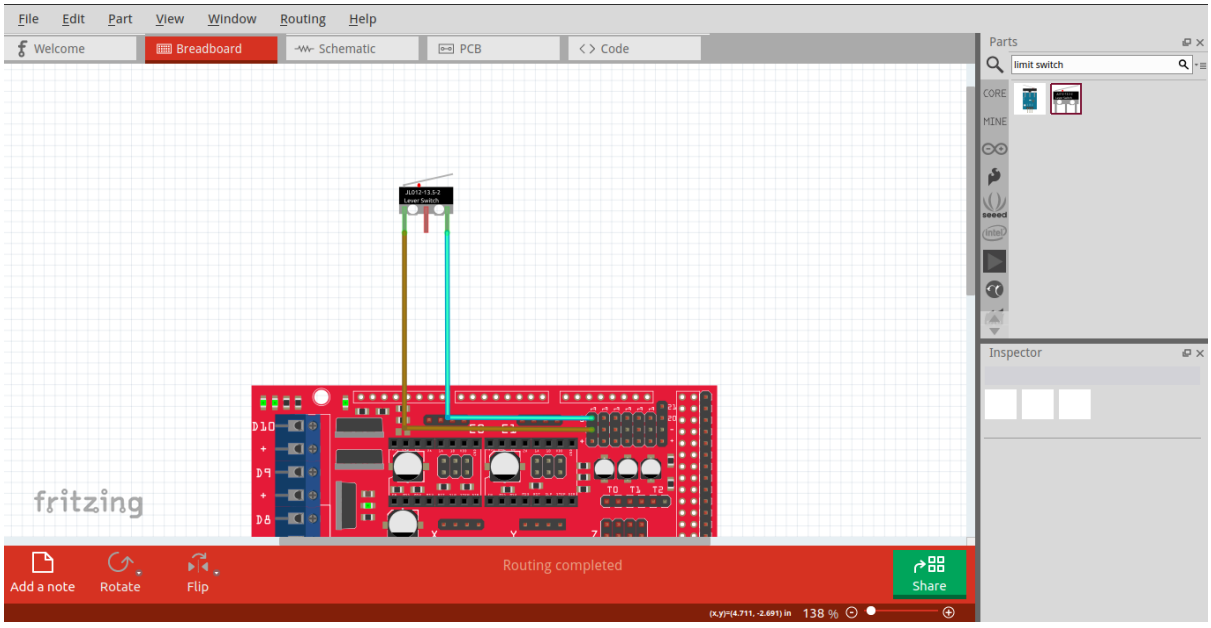
RAMP 1.4 AND STEPPER MOTOR CONNECTIONS

- **RAMP1.4 and Servo Motor Connections:**



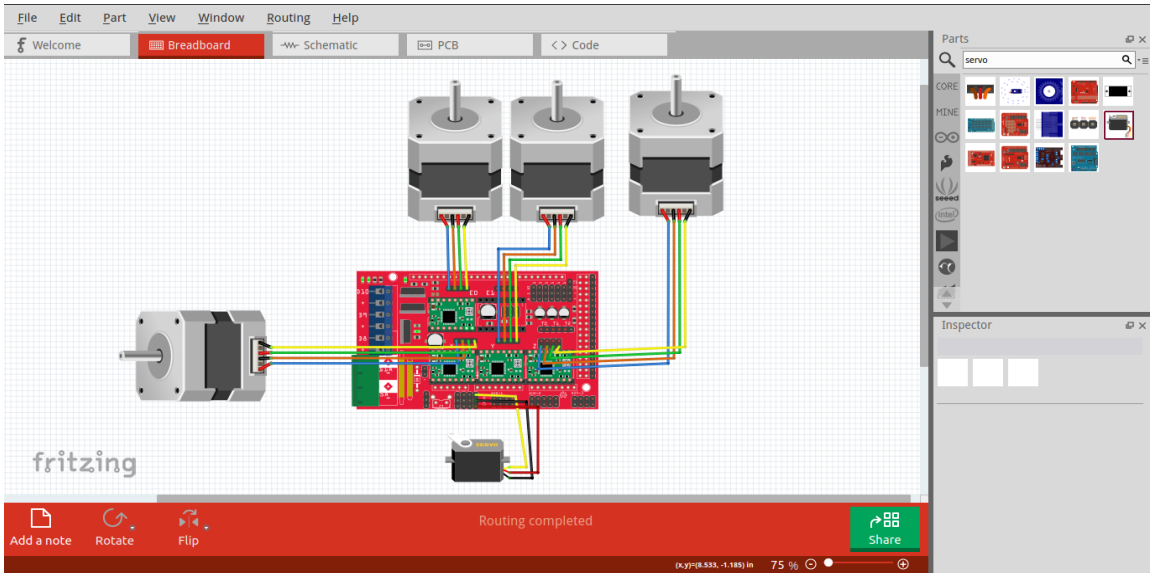
RAMP 1.4 AND SERVO MOTOR CONNECTIONS

- **RAMP1.4 and Limit Switch Connections:**



RAMP 1.4 AND LIMIT SWITCH CONNECTIONS

- **RAMP1.4 and All Motors Connections:**



RAMP 1.4 AND ALL MOTORS CONNECTIONS

3.2.10 Logitech C922 Pro Stream Webcam 1080P (Sensor):

LOGITECH C922 PRO STREAM WEBCAM 1080P

- **Specifications:**

  - Max Resolution: 1080p/30 fps - 720p/ 60 fps
  - Camera mega pixel: 3
  - Focus type: Autofocus
  - Lens type: Glass
  - Diagonal field of view (DFOV): 78°
  - Digital zoom: 1.2x

Chapter 4
ROBOT CONTROL

Chapter 4
ROBOT CONTROL

4.1 Introduction

This chapter reviews the practical methodologies for kinematics and dynamics modeling, essential to robotics. Kinematics describes the motion of robotic manipulators, focusing on positions, velocities, and accelerations, without considering mass or inertia. Dynamics examines how internal and external forces and torques interact with manipulators, ensuring balance under various conditions. Together, these models are critical for robotic design, simulation, and real-time control.

Kinematics and dynamics have been foundational to robotics research since its inception. Early studies positioned these topics as key areas of academic inquiry, resulting in significant theoretical advancements and a robust body of literature. Over time, these models have become indispensable for solving complex problems in robotic design, control, and simulation.

Kinematics modeling addresses forward and inverse kinematics problems, enabling precise calculation of a manipulator's position and orientation based on joint variables or vice versa. This capability is vital for applications requiring accurate end-effector positioning, such as manufacturing, assembly, and surgery. Dynamics modeling complements this by analyzing how forces and torques affect motion, offering insights into energy efficiency, system stability, and motor selection.

As robotics systems evolve, integrating kinematics and dynamics modeling is essential for high-performance robots in dynamic environments. In industrial automation, these models optimize path planning for smooth, efficient motion while reducing wear on components. In mobile robots and humanoids, dynamics ensures balance and stability, even on uneven terrain or during unexpected disturbances.

The integration of machine learning and artificial intelligence has further enhanced the utility of these models. AI-powered tools refine control strategies, enabling robots to learn and adapt to new tasks with minimal human intervention. This synergy between classical modeling techniques and modern computational advancements has driven innovations in autonomous systems, collaborative robotics, and human-robot interaction.

In summary, kinematics and dynamics modeling are vital for understanding robotic motion and designing efficient, robust, and intelligent systems. These methodologies bridge the gap between theoretical research and practical implementation, remaining central to innovation as robotics expands into new domains. This chapter explores these methodologies in depth, emphasizing their relevance and application in contemporary robotics.

4.2 FORWARD KINEMATICS

4.2.1 Introduction:

The forward kinematics equations focus on determining the position of the end-effector in the Cartesian coordinate system\ (x, y, z) \) based on the joint parameters:

1. *Rotational angle* \(\theta_1\) and \(\theta_2\), which control the motion in the horizontal plane.

2. *Prismatic joint extension* \(d_3\), which controls the vertical motion.

This way provides an explicit way to determine the end-effector's position in 3D space.

Forward kinematics determines the position of the end-effector based on joint angles. However, when using a PID controller, high accuracy is required to avoid errors in determining the end-effector's position.

4.2.2 Role of PID:

- The PID controller controls the joint angles to achieve the desired position.
- PID parameters must be tuned to minimize the error between the calculated end effector position and the actual position.
- It's important to handle oscillations when approaching the final position.

4.2.3 Forward Kinematics Equations:

Forward Kinematics equations in robotics are used to calculate the final position and orientation of the robot's end-effector based on the angles or displacements of each joint in the robotic arm.
These equations have several applications, including:

1. Determining the Position of the End-Effector:
Given the joint angles (or displacements for linear joints), you can use the Forward Kinematics equations to determine the position () and orientation of the end-effector.

2. Simulation and Analysis:
Forward Kinematics can be used to simulate the movement of the robot and analyze its performance, especially when designing or optimizing a robotic system.

3. Robot Control:
These equations are essential for programming precise robot movements. For example, if you want the end-effector to move to a specific location, Forward Kinematics helps you understand its final position based on the input joint values.

4. Result Verification:
When solving Inverse Kinematics (to calculate the joint angles required for the end-effector to reach a specific position), you can verify the results using Forward Kinematics.

5. Integration with Other Systems:
Forward Kinematics can be integrated with sensors or cameras to help the robot adjust its movements dynamically based on the environment or to align with a target.

6. Education and Training:
These equations are fundamental for understanding robotics. They help visualize how each joint contributes to the movement of the end-effector.

Steps for Applying Forward Kinematics:
1. Identify the type of robot (e.g., SCARA, anthropomorphic arm, or delta robot).
2. Use the Denavit-Hartenberg (DH) method to define the parameters for each joint (link lengths, joint angles, offsets, etc.).
3. Write the equations using transformation matrices to calculate the final pose (position and orientation) of the end-effector.

4.2.4 Calculate Forward Kinematics Equations:

- **Calculate The Degree of Freedom:**

Number of Degrees of Freedom of a Manipulator

Chebychev–Grübler–Kutzbach criterion

$$F = \lambda\,(n - j - 1) + \sum_{i=1}^{j}(f_i)$$

F: degrees of freedom of a mechanism

λ: degrees of freedom of the space in which a mechanism is intended to function (λ =6 for spatial mechanism, and λ=3 for planar mechanism)

n: number of links in the mechanism (including the fixed link)

j: number of joints in the mechanism

fi: the number of joints' freedom for every joint

fp: number of passive degrees of freedom in a mechanism

3D ⟶ $\lambda = 6$

We have four moving links and one fixed link. So

$$n = 5$$

We have four joints, three revolute and one prismatic. So

$$j = 4$$

Each joint has a DOF (Number of Movement).

Revolute ➔ 1 DOF

Prismatic ➔ 1 DOF

Spherical ➔ 3DOF

$$\sum_{i=1}^{j} (f_i) = 3 * 1 + 1 * 1 = 4$$

DOF = 4

- **Denavit – Hartenberg (D-H Parameters) Convention:**

Parameters: These help in defining the relationship between consecutive links.

1. **Simplification of Kinematics:** DH parameters simplify the kinematic analysis of robotic arms by providing a systematic way to describe the relationships between joint variables and link lengths.

2. **Easier Calculations:** By using DH parameters, the calculations for forward and inverse kinematics become more straightforward, allowing for quicker development and implementation of robotic algorithms.

3. **Visualization:** They help in visualizing the robot's configuration and movements in a clear and consistent manner.

Variable for prismatic joint (d_i)

Variable for revolute joint (θ_i)

DH parameters – Distal Table form

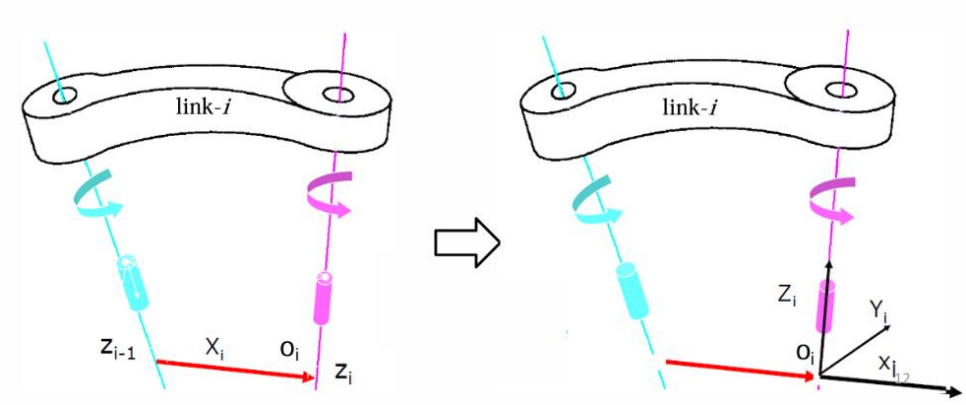
i	a_i	α_i	d_i	θ_i
1				
2				

Link Number

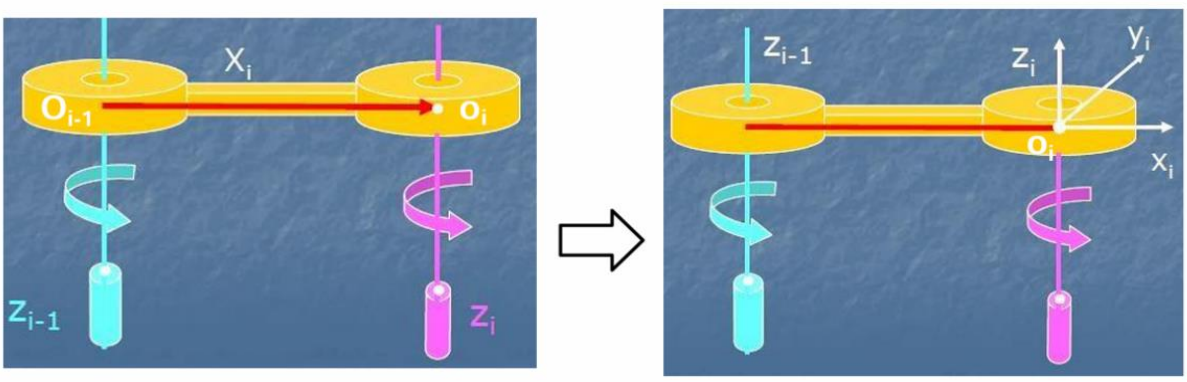
- **Steps to extract D-H Parameters :**

1. Assign Z-axis for all joints and end-effectors.
   - To assign the Z-axis for all joints and the end-effector according to the DH convention, follow these steps:
   A. For revolute joints: The Z-axis should align with the axis of rotation.
   B. For prismatic joints: The Z-axis should align with the direction of linear motion.
   C. For the end-effector: The Z-axis is chosen based on the desired final orientation or motion direction.
2. Base frame is arbitrary.
3. Assign X-axis for all frames.
   - Links 1, 2, …, n-1, we have 3-cases:
     - z_{i-1} and z_i are not coplanar
     - z_{i-1} is parallel to z_i
     - z_{i-1} intersects z_i

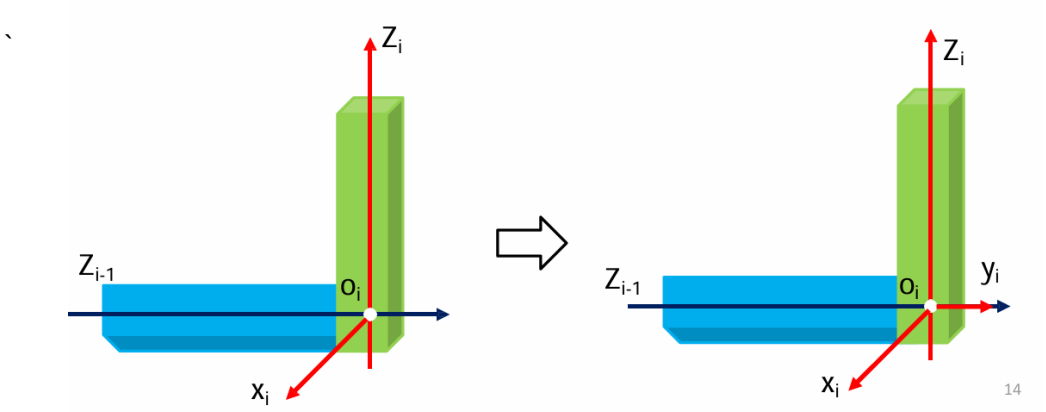
- **Case 1:** z_{i-1} and z_i are not coplanar



- **Case 2:** z_{i-1} is parallel to z_i



- **Case 3:** z_{i-1} intersects z_i



4. Find DH parameters (a_i , α_i , d_i , θ_i)

Through these four steps, a table of D-H parameters can be determined and filled with the values required to be used in finding the equations of motion.

- **Final Process:**

Determine the position and orientation of a robotic arm's end effector based on joint angles and link lengths…
We substitute the values of dh parameters into the transformation matrix to find all matrices that express the movement of the link with the link before it…
After substitution in the matrix, we will get a number of matrices describing the movement of each link relative to the link before it (A_1^0 , A_2^1 , …) and their product gives a matrix (T_n^0) expressing the movement of the end effector relative to the base.

$$T_n^0 = A_1^0 A_2^1 \ldots\ldots A_n^{n-1}$$

$$A_i = Rot_{z,\theta_i} Trans_{z,d_i} Trans_{x,a_i} Rot_{x,\alpha}$$

$$A_i = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & c_\alpha & -s_\alpha & 0 \\ 0 & s_\alpha & c_\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

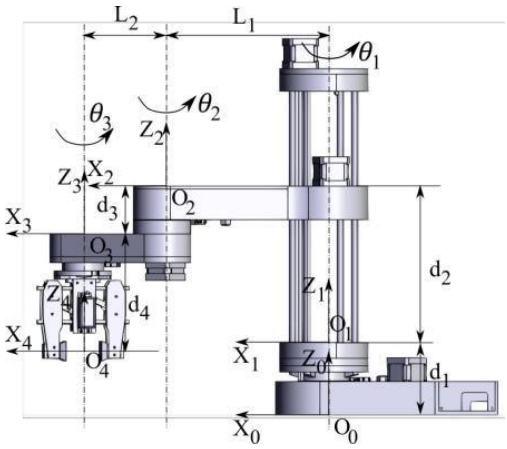
- **Final Equations:**

D-H Table :

Links	Angle Twist α_i	Link Length a_i	Joint Angle θ_i	Link offset d_i
1	0	0	θ_1	d_1
2	0	L_1	0	d_2
3	0	L_2	θ_2	d_3
4	0	0	θ_3	d_4

We will compensate here :

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



- **By Using The Values of D-H Parameters We can make the Transformation matrix:**

$$A_1^0 = \begin{bmatrix} C\theta_1 & -S\theta_1 & 0 & 0 \\ S\theta_1 & C\theta_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_2^1 = \begin{bmatrix} 1 & 0 & 0 & l_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_3^2 = \begin{bmatrix} C\theta_2 & -S\theta_2 & 0 & l_2C\theta_2 \\ S\theta_2 & C\theta_2 & 0 & l_2S\theta_2 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_4^3 = \begin{bmatrix} C\theta_3 & -S\theta_3 & 0 & 0 \\ S\theta_3 & C\theta_3 & 0 & 0 \\ 0 & 0 & 1 & -d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_n^0 = A_1^0 A_2^1 \ldots\ldots A_n^{n-1}$$

$$T_2^0 = A_1^0 * A_2^1 = \begin{bmatrix} C\theta_1 & -S\theta_1 & 0 & l_1C\theta_1 \\ S\theta_1 & C\theta_1 & 0 & l_1S\theta_1 \\ 0 & 0 & 1 & d_1 + d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_4^2 = A_3^2 * A_4^3 = \begin{bmatrix} C\theta_2C\theta_3 - S\theta_2S\theta_3 & -C\theta_2S\theta_3 - S\theta_2C\theta_3 & 0 & l_2C\theta_2 \\ S\theta_2C\theta_3 + C\theta_2S\theta_3 & -S\theta_2S\theta_3 + C\theta_2C\theta_3 & 0 & l_2S\theta_2 \\ 0 & 0 & 1 & d_3 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4^2 = A_3^2 * A_4^3 = \begin{bmatrix} C(\theta_2+\theta_3) & -S(\theta_2+\theta_3) & 0 & l_2C\theta_2 \\ S(\theta_2+\theta_3) & C(\theta_2+\theta_3) & 0 & l_2S\theta_2 \\ 0 & 0 & 1 & d_3 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_2^0 = A_1^0 * A_2^1 = \begin{bmatrix} C\theta_1 & -S\theta_1 & 0 & l_1C\theta_1 \\ S\theta_1 & C\theta_1 & 0 & l_1S\theta_1 \\ 0 & 0 & 1 & d_1 + d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_4^2 = A_3^2 * A_4^3 = \begin{bmatrix} C\theta_2C\theta_3 - S\theta_2S\theta_3 & -C\theta_2S\theta_3 - S\theta_2C\theta_3 & 0 & l_2C\theta_2 \\ S\theta_2C\theta_3 + C\theta_2S\theta_3 & -S\theta_2S\theta_3 + C\theta_2C\theta_3 & 0 & l_2S\theta_2 \\ 0 & 0 & 1 & d_3 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4^0 = T_2^0 * T_4^2 = \begin{bmatrix} C\theta_1C\theta_2C\theta_3 - C\theta_1S\theta_2S\theta_3 - S\theta_1S\theta_2C\theta_3 - S\theta_1C\theta_2S\theta_3 & -C\theta_1C\theta_2S\theta_3 - C\theta_1S\theta_2C\theta_3 - S\theta_1C\theta_2C\theta_3 + S\theta_1S\theta_2S\theta_3 & 0 & l_2C\theta_1C\theta_2 - l_2S\theta_1S\theta_2 + l_1C\theta_1 \\ S\theta_1C\theta_2C\theta_3 + C\theta_1S\theta_2S\theta_3 + S\theta_1C\theta_2S\theta_3 + C\theta_1S\theta_2C\theta_3 & -S\theta_1C\theta_2S\theta_3 + C\theta_1C\theta_2C\theta_3 + S\theta_1S\theta_2C\theta_3 - C\theta_1S\theta_2S\theta_3 & 0 & l_2S\theta_1C\theta_2 + l_2C\theta_1S\theta_2 + l_1S\theta_1 \\ 0 & 0 & 1 & d_1 + d_2 - d_3 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4^0 = \begin{bmatrix} C(\theta_1+\theta_2+\theta_3) & -S(\theta_1+\theta_2+\theta_3) & 0 & l_1C(\theta_1) + l_2C(\theta_1+\theta_2) \\ S(\theta_1+\theta_2+\theta_3) & C(\theta_1+\theta_2+\theta_3) & 0 & l_1S(\theta_1) + l_2S(\theta_1+\theta_2) \\ 0 & 0 & 1 & d_1 + d_2 - d_3 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$x = l_1C(\theta_1) + l_2C(\theta_1+\theta_2) \qquad z = d_1 + d_2 - d_3 - d_4$$

$$y = l_1S(\theta_1) + l_2S(\theta_1+\theta_2)$$

4.2.4 Robot Modeling and Simulation:

- **Introduction:**

  In this project, a robotic manipulator with a Revolute-Prismatic-Revolute-Revolute (RPRR) configuration is modeled and simulated using MATLAB's Robotics Toolbox. The RPRR robot consists of four links, where the first joint is revolute, the second is prismatic, and the third and fourth joints are revolute. Each joint contributes to the robot's degrees of freedom, allowing it to perform complex tasks in a three-dimensional workspace.

  The Denavit-Hartenberg (DH) parameter convention is employed to systematically define the geometry of each link and joint. These parameters specify the relative position and orientation of the robot's links, providing a standardized approach to robot kinematics. The prismatic joint is constrained to specific limits to ensure realistic operation.

  Joint variables, including revolute joint angles and prismatic joint extensions, are defined to simulate the robot's configuration. A 3D visualization of the robot is plotted within a defined workspace to analyze its motion and validate its design.

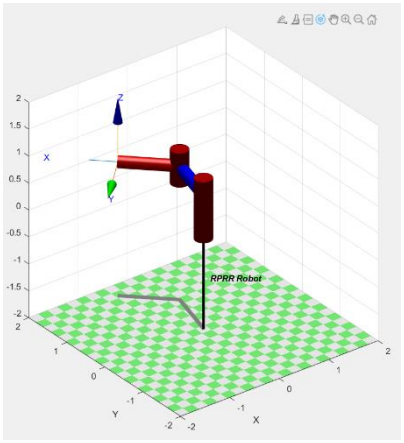
  This simulation provides an essential step toward understanding the kinematic behavior of robotic systems and serves as a foundation for further control and optimization studies in robotics applications.

- **Simulation Code:**

Editor - C:\bin\Untitled6.m

Untitled6.m

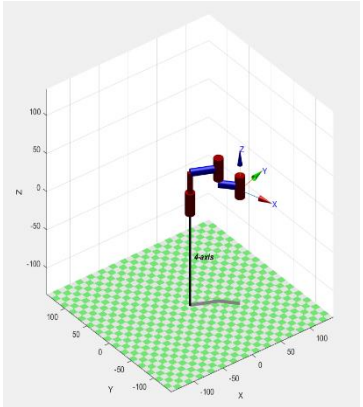
```
% Define links using DH parameters for Revolute-Prismatic-R-R robot
L1 = Link('d', 0, 'a', 0, 'alpha', 0, 'revolute');        % Revolute joint
L2 = Link('theta', 0, 'a', 0, 'alpha', 0, 'prismatic');   % Prismatic joint
L3 = Link('d', 0, 'a', 1, 'alpha', 0, 'revolute');        % Revolute joint
L4 = Link('d', 0, 'a', 1, 'alpha', 0, 'revolute');        % Revolute joint

% Set prismatic joint limits
L2.qlim = [0 1];  % Prismatic joint limits (e.g., 0 to 1 unit of extension)

% Create the robot
robot = SerialLink([L1 L2 L3 L4], 'name', 'RPRR Robot');

% Joint variables: θ1, d2 (extension), θ3, θ4
q = [pi/4, 0.5, pi/6, pi/3];

% Define workspace dimensions
workspace = [-2 2 -2 2 -2 2];

% Plot the robot
robot.plot(q, 'workspace', workspace);
```



Editor - C:\bin\Untitled1.m

Untitled20.m | Untitled1.m

```
% تعريف الروابط (Links)
L(1) = Link([0, 0, 0, 0]);
L(2)=Prismatic('theta',0,'a',40,'alpha',0);
L(3) = Link([0, -20, 30, 0]);
L(4) = Link([0, -5, 0, 0]);
Rob = SerialLink(L, 'name', '4-axis');
Rob.links(2).qlim = [10, 40];
for th = -pi/2 : 0.05 : (3/4)*pi
    Rob.plot([th 40 th th])
    pause(0.25)
end
```



4.3 INVERSE KINEMATICS

4.3.1 Introduction:

While forward kinematics deals with finding the position given the joint values, the inverse kinematics equations solve the opposite problem: determining the joint values needed to reach a specific position. For the SCARA robot, this involves solving the nonlinear equations for \(\theta_1\), \(\theta_2\), and \(d_3\), which requires careful consideration of the robot's workspace constraints and possible multiple solutions.

Inverse kinematics is used to calculate the joint angles required to achieve a specific end-effector position. Small errors in these calculations can lead to significant deviations in the robot's movement.

4.3.2 Role of PID:

- The PID controller controls the joint angles to reach the target position based on the inverse kinematics output.
- Balancing accuracy and response time is crucial to prevent delays in reaching the desired position.
- **Additional Considerations:** Feedforward control should be integrated with PID to account for the required speed or external forces acting on the system.

4.3.3 Inverse Kinematics Equations:

Inverse Kinematics equations in robotics are used to calculate the joint angles or displacements required for the robot's end-effector to reach a specific position and orientation in space. These equations are crucial for controlling robotic arms and solving motion-planning problems. Here's how they can be applied:

a. Calculating Joint Angles:

When you have a target position and orientation for the end-effector (and rotation angles), Inverse Kinematics is used to determine the necessary joint angles or displacements.

b. Path Planning and Motion Control:

Inverse Kinematics is essential for programming the robot to follow a specific trajectory or perform tasks such as picking and placing objects.

c. Integration with Sensors:

Inverse Kinematics helps robots interact with external systems, such as vision sensors or force sensors, by determining how to move the joints to respond to external stimuli.

d. Interaction with Forward Kinematics:

Once the joint angles are calculated using Inverse Kinematics, Forward Kinematics can be used to verify the accuracy of the solution by checking if the calculated joint angles result in the desired end-effector position.

➢ **Challenges in Inverse Kinematics:**

- **Multiple Solutions:** Some robotic configurations can have multiple valid joint angle solutions for the same end-effector position.
- **Singularities:** Certain configurations may lead to undefined or infinite solutions, known as singularities.

4.3.4 Calculate Inverse Kinematics Equations:

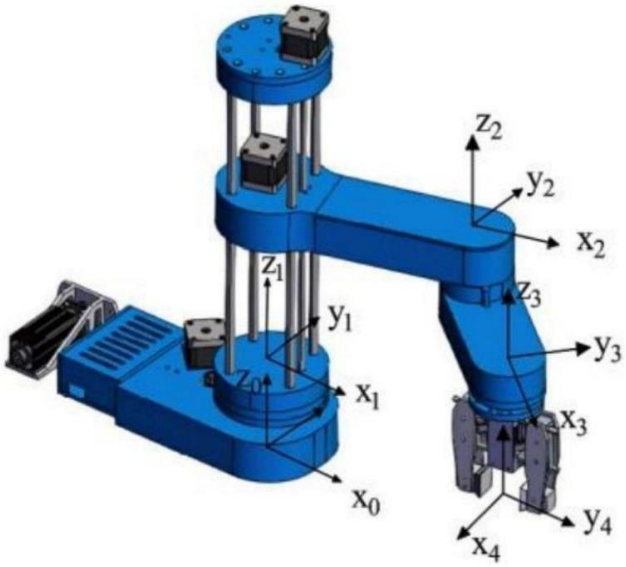
➢ **Steps for Solving Inverse Kinematics:**

1. **Define the Robot Structure:** Identify the type of robot and its degrees of freedom (DOF).
2. **Forward Kinematics Reference:** Use the Forward Kinematics equations to relate joint variables to the end-effector's pose.
3. **Set the Target Pose:** Specify the desired end-effector position and orientation in space.
4. **Solve the Equations:**
   - For simpler robots (e.g., 2-DOF), analytical solutions can be derived.
   - For more complex robots, numerical methods or optimization algorithms may be required.
5. **Verify the Solution:** Use Forward Kinematics to ensure the calculated joint values produce the desired pose.

- **Applications for Inverse Kinematics:**

  - **Industrial Automation:** Programming robotic arms for tasks such as welding, painting, or assembling parts.
  - **Robotic Surgery:** Precise control of surgical tools based on target positions.
  - **Animation and Gaming:** Controlling character limbs or objects in 3D space.
  - **Humanoid Robots:** Enabling realistic movement of robotic limbs.

- **Final Process:**



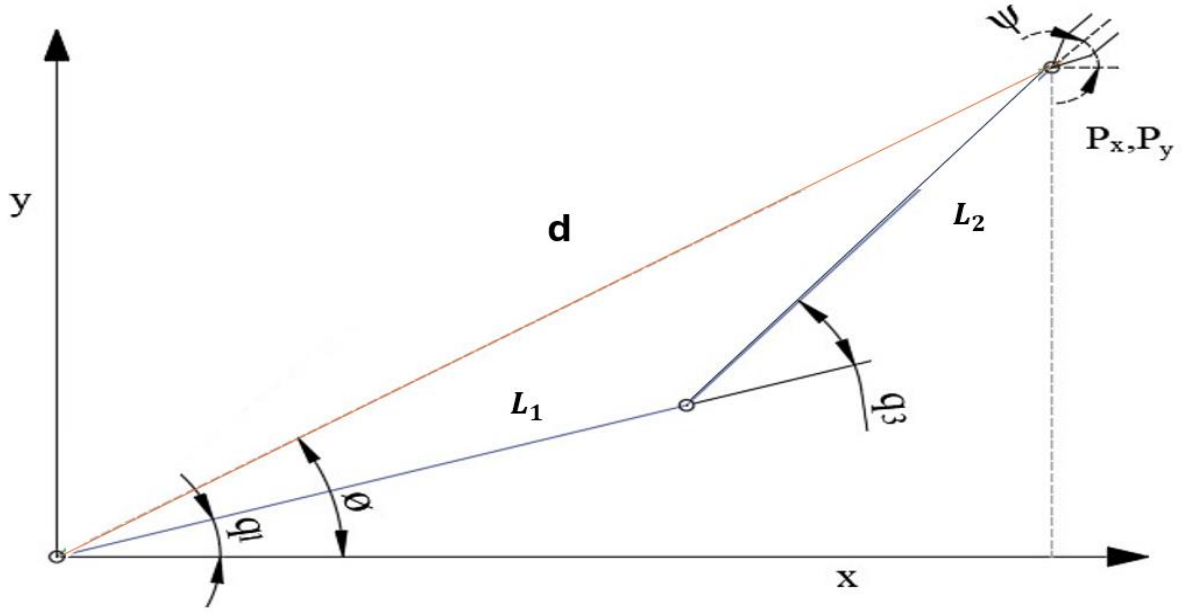


FIGURE 27

$$d = \sqrt{X^2 + Y^2} \qquad d^2 = X^2 + Y^2 \longrightarrow 1$$

$$d^2 = L_1^2 + L_2^2 - 2L_1L_2\cos\alpha \qquad \cos\alpha = \frac{d^2 - L_1^2 - L_2^2}{2L_1L_2} \longrightarrow 2$$

- **From Fig 27:**

θ_2=180 -α or $\pi - \alpha$

$\cos\theta_2 = \cos(\pi - \alpha)$ = $\cos\pi\cos\alpha + \sin\pi\sin\pi$

$$\cos\theta_2 = \frac{d^2 - L_1^2 - L_2^2}{2L_1L_2}$$

$$\theta_2 = \pm \cos^{-1}\left(\frac{X^2+Y^2-L_1^2-L_2^2}{2L_1L_2}\right) \longrightarrow 3$$

- **From Fig 27:**

$$\tan(\theta_1 + \beta) = \frac{Y}{X}$$

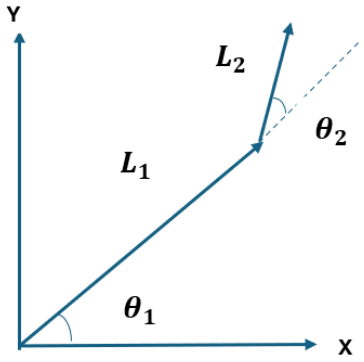
$$\theta_2 = \beta - \tan^{-1}\left(\frac{y}{X}\right) \longrightarrow 4$$

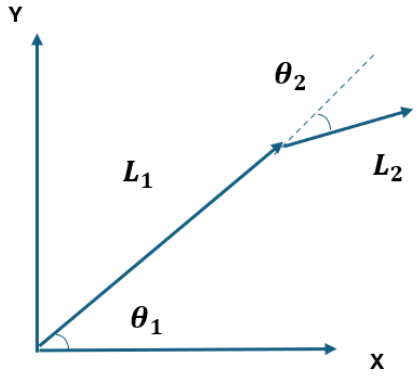
$$\tan\beta = \frac{L_2\sin\theta_2}{L_1+L_2\cos\theta_2}$$

$$\beta = \tan^{-1}\left(\frac{L_2\sin\theta_2}{L_1+L_2\cos\theta_2}\right) \longrightarrow 5$$

$$\theta_2 = \tan^{-1}\left(\frac{y}{X}\right) \pm \tan^{-1}\left(\frac{L_2\sin\theta_2}{L_1+L_2\cos\theta_2}\right)$$

➢ **Two Solutions:**





$$\theta_2 = +\cos^{-1}\left(\frac{X^2+Y^2-L_1^2-L_2^2}{2L_1L_2}\right)$$

$$\theta_1 = \tan^{-1}\left(\frac{y}{X}\right) - \tan^{-1}\left(\frac{L_2\sin\theta_2}{L_1+L_2\cos\theta_2}\right)$$

$$\theta_2 = -\cos^{-1}\left(\frac{X^2+Y^2-L_1^2-L_2^2}{2L_1L_2}\right)$$

$$\theta_1 = \tan^{-1}\left(\frac{y}{X}\right) + \tan^{-1}\left(\frac{L_2\sin\theta_2}{L_1+L_2\cos\theta_2}\right)$$

➢ $\theta_1 = \tan^{-1}\left(\frac{y}{X}\right) \pm \tan^{-1}\left(\frac{L_2\sin\theta_2}{L_1+L_2\cos\theta_2}\right)$

➢ $\theta_2 = \pm\cos^{-1}\left(\frac{X^2+Y^2-L_1^2-L_2^2}{2L_1L_2}\right)$

4.4 JACOBIAN KINEMATICS

4.4.1 Introduction:

The Jacobian matrix is used to relate linear and angular velocities of the end-effector to joint velocities. Any errors in these conversions can affect the smoothness and accuracy of the movement.

4.4.2 Role of PID:

- The PID controller adjusts the joint velocities to ensure the required end-effector velocities are achieved.
- The Jacobian must be continuously monitored to ensure system stability, especially in singular situations that can cause control issues.
- **Additional Considerations:** Use techniques like Pseudo-Inverse Jacobian to improve system stability at singularity points.

4.4.3 Jacobian in Robotics:

1. Definition of the Jacobian:

The Jacobian matrix links joint velocities to the end-effector's linear and angular velocities. It is a key tool for analyzing motion and controlling robots.

2. Key Applications:

a. **Joint Velocity Calculation:**
Used to calculate the required joint velocities to achieve a desired end-effector velocity.

b. **Path Planning and Control:**
Ensures smooth movement of the end-effector along a specific trajectory.

c. **Dynamic Adaptation:**
It enables the robot to adjust to environmental changes or interact with moving objects.

d. **Sensor Integration:**
Translates sensor signals (e.g., vision or force sensors) into joint movements.

e. **Feasibility Testing:**
Identifies singularities and ensures movements remain within the robot's workspace limits.

3. Challenges:

a. **Singularities:**
Occur when the Jacobian determinant is zero, leading to a loss of degrees of freedom.

b. **Computational Complexity:**
Calculating Jacobian becomes challenging for robots with many degrees of freedom.

c. Nonlinearity:

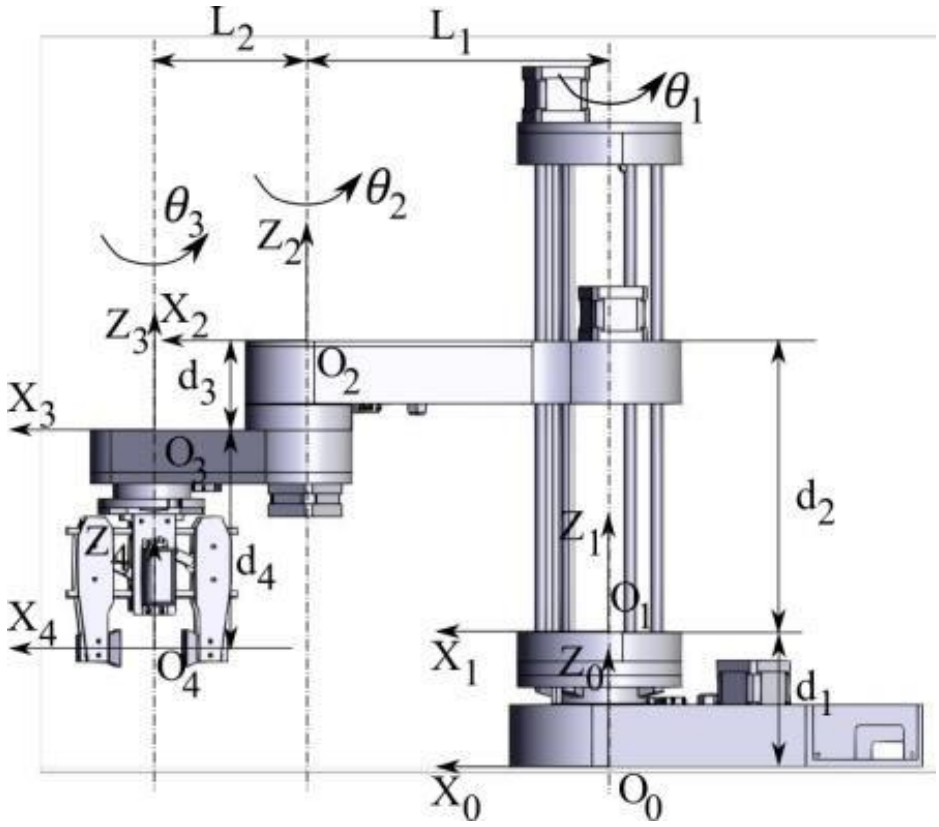
The Jacobian depends on the robot's joint configuration and changes dynamically during motion.

4. Steps for Using Jacobian:

a. Define the robot (links, joints, and degrees of freedom).
b. Derive the forward kinematics equations.
c. Compute the Jacobian matrix by differentiating the forward kinematics equations.
d. Solve the equations for velocity control or inverse kinematics.

4.4.4 Calculate Jacobians Equations:

Jacobian is a fundamental tool in robotics for motion analysis, solving kinematics, and enabling dynamic adaptation. Careful attention is required to address singularities and computational complexities.



$\because\ Z_{i-1}^{0} = R_{i-1}^{0} * K$

For prismatic ⟶ $J_i = \begin{bmatrix} Z_{i-1}^{0} \\ 0 \end{bmatrix}$

For Revolute ⟶ $J_i = \begin{bmatrix} Z_{i-1}^{0}(O_n^0 - O_{i-1}^0) \\ Z_{i-1}^{0} \end{bmatrix}$

$$\because J_1 = \begin{bmatrix} z_0^0(o_4^0 - o_0^0) \\ z_0^0 \end{bmatrix} \quad \because J_2 = \begin{bmatrix} z_1^0 \\ 0 \end{bmatrix}$$

$$\because J_3 = \begin{bmatrix} z_2^0(o_4^0 - o_2^0) \\ z_2^0 \end{bmatrix} \quad \because J_4 = \begin{bmatrix} z_3^0(o_4^0 - o_3^0) \\ z_3^0 \end{bmatrix}$$

$$\because Z_0^0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \because Z_1^0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \because Z_2^0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \because Z_3^0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\because O_0^0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \because O_1^0 = \begin{bmatrix} 0 \\ 0 \\ d_1 \end{bmatrix} \quad \because O_2^0 = \begin{bmatrix} L_1C_1 \\ L_1S_1 \\ d_1 + d_2 \end{bmatrix} \quad \because O_3^0 = \begin{bmatrix} L_1C_1 + L_2C_{12} \\ L_1S_1 + L_2S_{12} \\ d_1 + d_2 - d_3 \end{bmatrix}$$

$$\because O_4^0 = \begin{bmatrix} L_1C_1 + L_2C_{12} \\ L_1S_1 + L_2S_{12} \\ d_1 + d_2 - d_3 - d_4 \end{bmatrix}$$

$$\therefore Z_0^0(O_4^0 - O_0^0) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} * \begin{bmatrix} L_1C_1 + L_2C_{12} \\ L_1S_1 + L_2S_{12} \\ d_1 + d_2 - d_3 - d_4 \end{bmatrix} =$$

$$\begin{bmatrix} +j & -i & +k \\ 0 & 0 & 1 \\ L_1C_1 + L_2C_{12} & L_1S_1 + L_2S_{12} & d_1 + d_2 - d_3 - d_4 \end{bmatrix} = \begin{bmatrix} -L_1S_{12} \\ L_2C_{12} \\ 1 \end{bmatrix}$$

$$\therefore Z_3^0(O_4^0 - O_3^0) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -d_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} +j & -i & +k \\ 0 & 0 & 1 \\ 0 & 0 & -d4 \end{bmatrix}$$

$$\therefore \mathbf{J} = \begin{bmatrix} j_v \\ j_w \end{bmatrix} = \begin{bmatrix} -l_1S_1 - l_2S_{12} & 0 & -l_2S_{12} & 0 \\ l_1C_1 + l_2C_{12} & 0 & -l_2C_{12} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

4.5 Dynamics

4.5.1 Introduction:

Kinematic and dynamic modeling are very important issues for design, simulation, and control of robotic manipulators. Kinematics describes the motion of the robotic manipulator without considering forces and torques causing this motion. Kinematic modeling is used in many fields of robotics such as robot simulation; modeling of mechanisms, actuators, and sensors; on-line and off-line robot control applications. Therefore, derivation of a kinematic model of robotic manipulators is one of the most important phases of robot issues.

Dynamic modeling of robotic manipulators is a cumbersome issue. It provides a relationship between forces acting on robotic manipulators and acceleration that the robotic manipulators generate. Dynamic equations are important for simulation of robot motions, and design of control algorithms. Lagrange-Euler and Newton-Euler formulation are well-known approaches for dynamic analysis of robot manipulators. In the standard Lagrange-Euler formulation, the robotic system is analyzed based on its kinetic and potential energy. The Newton-Euler formulation is different from Lagrange-Euler formulation since each link of the manipulator is considered separately while deriving dynamic equations. There is not a strict answer to which method is better than the other. The main goal is to obtain the dynamic model of the robotic manipulator as fast as possible. In the Newton Euler method, two types of forces (given and the constraint forces) applied to the robotic system are considered. The main drawback of the Newtonian approach is that these forces are not easily computable in general.

Lagrange- Euler and Newton-Euler formulation are two well-known methods for dynamic analysis of robot manipulators. In this project, Lagrange-Euler formulation is selected for performing dynamic modeling of 4-DOF (Degrees of Freedom) RPRR type serial robot manipulator. All of the computations related to dynamic equations are performed on Mathematica software. Using Lagrange-Euler method yields simple and well- structured dynamic equations of 4-DOF RPRR type serial robot manipulator. It is verified that implementation of Lagrange-Euler method to the dynamics equations of serial robotic manipulator requires easy, systematic and steady forward processes.

4.5.2 Role of PID:

- PID is used to adjust the torque applied at each joint based on dynamic calculations.
- Gravity compensation should be included as part of the control to reduce the burden on PID, especially in vertical movements.
- Tuning PID ensures high stability and avoids oscillations caused by nonlinear dynamics.

- **Additional Considerations:** Feedforward control can be integrated with PID to predict and compensate for expected accelerations and forces, improving performance
- **Tips for Tuning PID Controllers for These Considerations:**

1. **Precise Calibration:** The PID parameters (Kp), (Ki), and (Kd) need to be finely tuned to reduce position or velocity errors while minimizing overshot and response time.
2. **Kinematic Feedback:** Position, velocity, and torque feedback from sensors must be integrated into the control loop for accurate performance.
3. **Filtering Techniques:** Filters can be applied to sensor feedback signals to reduce noise and oscillations, ensuring smoother motion.
4. **Stability:** Ensuring the system's dynamic stability is key. PID control can be augmented with techniques like *Adaptive Control* or *Feedforward Dynamics Compensation* for more complex systems.

4.5.3 The Lagrange-Euler Dynamic Model:

The dynamic model of a manipulator with [n] joints can be expressed through Equation.

$$D(q)\ddot{q} + H(q,\dot{q})\dot{q} + G(q) = \tau \longrightarrow (1)$$

T: Vector of generalized forces (*n*×1 dimension).

D: Inertia matrix (*n*×*n* dimension).

H: Vector of centrifugal and Coriolis forces (*n*×1 dimension).

$\boldsymbol{q}$: Components of the joint position vector.

$\dot{\boldsymbol{q}}$: Components of the joint speed vector.

G: Vector of gravitational force (*n*×1 dimension).

$\ddot{\boldsymbol{q}}$: Vector of joint acceleration (*n*×1dimension).

- **The equations of motion for an n-link manipulator are:**

$$\sum_{j=1}^{n} D_{ij}(q)\ddot{q}_j + H_{ikm}\dot{q}_k\,\dot{q_m} + G(q) = Q_i \longrightarrow (2)$$

- **$D_{ij}(q)$ is the inertia matrix is of size n×n:**

$$D_{ij} = \sum_{i=1}^{n} \left(J_{D_i}^{T} \cdot m_i \cdot J_{D_i} + \frac{1}{2} J_{R_i}^{T} \cdot I_i \cdot J_{R_i} \right) \longrightarrow (3)$$

- **H_{ikm} is the Christoffel Symbols:**

$$H_{ikm} = \sum_{j=1}^{n} \sum_{k=1}^{n} \left(\frac{\partial D_{ij}}{\partial q_k} - \frac{1}{2} \frac{\partial D_{jk}}{\partial q_i} \right) \longrightarrow (4)$$

- **$G(q)$ is the gravitational vector:**

$$G(q) = \sum_{j=1}^{n} m_j g^T J_{Dj}^{(i)} \longrightarrow (5)$$

4.5.4 Dynamic Modeling of SCARA Robot:

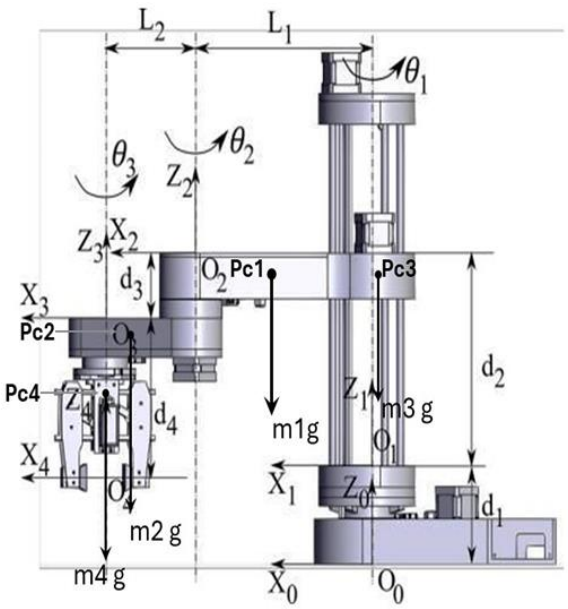


FIGURE 28
(SCARA ROBOT CONFIGURED WITH RPRR 4 DEGREES OF FREEDOM)

- use fix notation as follows: For **i = 1, 2,3,4 θ_i** denotes the joint angle, which also serves as a generalized coordinate; **m_i** denotes the mass of link ***i* and *L_i*** denotes the length of link ***i* and *L_{ci}*** denotes the distance from the previous joint to the center of mass of link ***i*** and ***L_i*** denotes the moment of inertia of link ***i*** about an axis coming out of the page, passing through the center of mass of link ***i.***
- We will make effective use of Jacobian expressions in computing kinetic energy. Since we are using joint variables as the generalized coordinates, it follows that we can use the contents of Section 1.

- **Link Positions and Velocities:**

Centers of mass of the various links as a function of the generalized coordinates

$$p_{c1} = \begin{bmatrix} {}^1/_2\, l_1 C\theta_1 \\ {}^1/_2\, l_1 S\theta_1 \\ d_1 + d_2^* \end{bmatrix} \longrightarrow (6) \qquad p_{c2} = \begin{bmatrix} l_1 C\theta_1 + {}^1/_2\, l_2 C\theta_1\theta_2 \\ l_1 S\theta_1 + {}^1/_2\, l_2 S\theta_1\theta_2 \\ d_1 + d_2^* - d_3 \end{bmatrix} \longrightarrow (7)$$

$$p_{c3} = \begin{bmatrix} 0 \\ 0 \\ d_1 + d_2^* \end{bmatrix} \longrightarrow (8) \qquad p_{c4} = \begin{bmatrix} l_1 C\theta_1 + l_2 C\theta_1\theta_2 \\ l_1 S\theta_1 + l_2 S\theta_1\theta_2 \\ d_1 + d_2^* - d_3 - d_4 \end{bmatrix} \longrightarrow (9)$$

- **Utilizing the Jacobian Matrix**

From equations **(6), (7), (8), and (9)** by differentiating the end position components with respect to the joint variables, equations **(10), (11), (12), and (13)** are derived, representing the velocities at (m1), (m2), (m3), and (m4).

$$\because J_{Dn} = \begin{bmatrix} \frac{dp_{cn}}{d\theta_1} & \frac{dp_{cn}}{d\theta_2} & \frac{dp_{cn}}{dd_2^*} & \frac{dp_{cn}}{d\theta_3} \end{bmatrix}$$

$$\therefore J_{D1} = \begin{bmatrix} \frac{dp_{c1}}{d\theta_1} & \frac{dp_{c1}}{d\theta_2} & \frac{dp_{c1}}{dd_2^*} & \frac{dp_{c1}}{d\theta_3} \end{bmatrix} = \begin{bmatrix} -{}^1/_2\, l_1 S\theta_1 & 0 & 0 & 0 \\ -{}^1/_2\, l_1 C\theta_1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \longrightarrow (10)$$

$$\because J_{D2} = \begin{bmatrix} \frac{dp_{c2}}{d\theta_1} & \frac{dp_{c2}}{d\theta_2} & \frac{dp_{c2}}{dd_2^*} & \frac{dp_{c2}}{d\theta_3} \end{bmatrix}$$

$$\therefore J_{D2} = \begin{bmatrix} -l_1 S\theta_1 - {}^1/_2\, l_2 S\theta_1\theta_2 & -{}^1/_2\, l_2 S\theta_1\theta_2 & 0 & 0 \\ l_1 C\theta_1 + {}^1/_2\, l_2 C\theta_1\theta_2 & {}^1/_2\, l_2 C\theta_1\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \longrightarrow (11)$$

$$\therefore J_{D3} = \begin{bmatrix} \frac{dp_{c3}}{d\theta_1} & \frac{dp_{c3}}{d\theta_2} & \frac{dp_{c3}}{dd_2^*} & \frac{dp_{c3}}{d\theta_3} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \longrightarrow (12)$$

$$\therefore J_{D4} = \begin{bmatrix} \frac{dp_{c4}}{d\theta_1} & \frac{dp_{c4}}{d\theta_2} & \frac{dp_{c4}}{dd_2^*} & \frac{dp_{c4}}{d\theta_3} \end{bmatrix} = \begin{bmatrix} -l_1 S\theta_1 - l_2 S\theta_1\theta_2 & -l_2 S\theta_1\theta_2 & 0 & 0 \\ l_1 C\theta_1 + l_2 C\theta_1\theta_2 & l_2 C\theta_1\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \longrightarrow (13)$$

- **The angular velocity**:

at each joint and its corresponding joint variables are analyzed and represented as follows:

$$J_{R1} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \dot{\theta}_1 \qquad J_{R2} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 + \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

$$J_{R3} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 + \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ d_2^* \end{bmatrix}$$

$$J_{R4} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 + \dot{\theta}_2 + \dot{\theta}_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ d_2^* \\ \dot{\theta}_3 \end{bmatrix}$$

- **Compute the Inertia matrix D(q):**

Now we are ready to apply the formulation of the inertia matrix D(q) that have size 4×4:

$$\mathbf{D(q)} = \sum_{i=1}^{n} \left(\mathbf{J_{D_i}^T . m_i . J_{D_i}} + \frac{1}{2} \mathbf{J_{R_i}^T . I_i . J_{R_i}} \right)$$

$$\mathbf{D(q)} = \mathbf{m_1\, J_{D_1}^T\, J_{D_1}} + \frac{1}{2} \mathbf{I_1\, J_{R_1}^T\, J_{R_1}} + \mathbf{m_2\, J_{D_2}^T\, J_{D_2}} + \frac{1}{2} \mathbf{I_2\, J_{R_2}^T\, J_{R_2}} + \mathbf{m_3\, J_{D_3}^T\, J_{D_3}}$$

$$+ \frac{1}{2} \mathbf{I_3\, J_{R_3}^T\, J_{R_3}} + \frac{1}{2} \mathbf{I_4\, J_{R_4}^T\, J_{R_4}}$$

➢ Inertia matrix for Joint 1 (n=1):

$$\mathbf{D_1 = m_1\, J_{D_1}^T\, J_{D_1}} = \begin{bmatrix} \frac{l_1^2 . m_1}{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & m_1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{D_2} = \frac{1}{2} \mathbf{I_1\, J_{R_1}^T\, J_{R_1}} = \begin{bmatrix} 1/2\, I_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

➢ Inertia matrix for Joint 2 (n=2):

$$\mathbf{D_3 = m_2\, J_{D_2}^T\, J_{D_2}} = \begin{bmatrix} \frac{(m_2(4l_1^2 + 4C\theta_2\, l_1\, l_2 + l_2^2))}{4} & \frac{(l_2\, m_2\, (l_2 + 2l_1 C\theta_2))}{4} & 0 & 0 \\ \frac{(l_2\, m_2\, (l_2 + 2l_1 C\theta_2))}{4} & \frac{(l_2^2\, m_2)}{4} & 0 & 0 \\ 0 & 0 & m_2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$D_4 = \frac{1}{2} I_2 J_{R_2}^T J_{R_2} = \begin{bmatrix} 1/2\, I_2 & 1/2\, I_2 & 0 & 0 \\ 1/2\, I_2 & 1/2\, I_2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- **Inertia matrix for Joint 3 (n=3):**

$$D_5 = m_3 J_{D_3}^T J_{D_3} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & m_3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$D_6 = \frac{1}{2} I_3 J_{R_3}^T J_{R_3} = \begin{bmatrix} 1/2\, I_3 & 1/2\, I_3 & 0 & 0 \\ 1/2\, I_3 & 1/2\, I_3 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- **Inertia matrix for Joint 4 (n=4):**

$$D_7 = m_4 J_{D_4}^T J_{D_4} = \begin{bmatrix} m_4(l_1^2 + 2C\theta_2\, l_1\, l_2 + l_2^2) & 0 & 0 & 0 \\ l_2\, m_4(l_2 + l_1\, C\theta_2) & l_2^2\, m_4 & 0 & 0 \\ 0 & 0 & m_4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$D_8 = \frac{1}{2} I_4 J_{R_4}^T J_{R_4} = \begin{bmatrix} 1/2\, I_4 & 1/2\, I_4 & 0 & 1/2\, I_4 \\ 1/2\, I_4 & 1/2\, I_4 & 0 & 1/2\, I_4 \\ 0 & 0 & 0 & 0 \\ 1/2\, I_4 & 1/2\, I_4 & 0 & 1/2\, I_4 \end{bmatrix}$$

- **Computing Inertia Matrix Parameters**:

$$D = \begin{bmatrix} D_{11} & D_{12} & D_{13} & D_{14} \\ D_{21} & D_{22} & D_{23} & D_{24} \\ D_{31} & D_{32} & D_{33} & D_{34} \\ D_{41} & D_{42} & D_{43} & D_{44} \end{bmatrix}$$

Inertia matrix $D(q)$ =

$$D_1 + D_2 + D_3 + D_4 + D_5 + D_6 + D_7 + D_8$$

Carrying out the above multiplications and using the standard trigonometric identities $\sin^2(\theta) + \cos^2(\theta) = 1, \cos(\alpha)\cos(\beta) + \sin(\alpha)\sin(\beta) = \cos(\alpha - \beta)$ leads to :

$$D_{11} = \frac{l_1^2 m_1}{4} + \frac{\left(m_2\left(4l_1^2 + 4C\theta_2 l_1 l_2 + l_2^2\right)\right)}{4} + m_4\left(l_1^2 + 2C\theta_2 l_1 l_2 + l_2^2\right) + \frac{1}{2}(I_1 + I_2 + I_3 + I_4)$$

$$D_{12} = \frac{(l_2 m_2(l_2 + 2l_1 C\theta_2))}{4} + l_2 m_4(l_2 + l_1 C\theta_2) + \frac{1}{2}(I_2 + I_3 + I_4)$$

$$D_{13} = \text{zero} \quad \& \quad D_{14} = \frac{1}{2}I_4$$

$$D_{21} = \frac{l_2 m_2(l_2 + 2l_1 C\theta_2)}{4} + l_2 m_4(l_2 + l_1 C\theta_2) + \frac{1}{2}(I_2 + I_3 + I_4)$$

$$D_{22} = \frac{l_2^2 m_2}{4} + l_2^2 m_4 + \frac{1}{2}(I_2 + I_3 + I_4)$$

$$D_{23} = \text{zero} \quad \& \quad D_{24} = \frac{1}{2}I_4 \quad \& \quad D_{31} = \text{zero} \quad \& \quad D_{32} = \text{zero}$$

$$D_{33} = m_1 + m_2 + m_3 + m_4$$

$$D_{34} = \text{zero}$$

$$D_{41} = \frac{1}{2}I_4$$

$$D_{42} = \frac{1}{2}I_4$$

$$D_{43} = \text{zero}$$

$$D_{44} = \frac{1}{2}I_4$$

(14)

Notice...

$$D_{13} = D_{23} = D_{31} = D_{32} = D_{34} = D_{43} = Zero$$

$$D_{14} = D_{24} = D_{41} = D_{42} = D_{44} = \frac{1}{2}I_4$$

- **Compute the Christoffel symbols H_{ikm}:**

Now we can compute the $\mathbf{H_{ikm}}$ Christoffel symbols using the definition **[14]** This gives:

➢ $\mathbf{H_{ikm}}$ Christoffel symbols:

$$\mathbf{H_{ijk}} = \sum_{j=1}^{n}\sum_{k=1}^{n}\left(\frac{\mathbf{dD_{ij}}}{\mathbf{dq_k}} - \mathbf{1/_2}\ \frac{\mathbf{dD_{jk}}}{\mathbf{dq_i}}\right)$$

We know that **n=4** that is the degree of freedom of the system **(q1, q2, d3, q4)**

$$\mathbf{H_1} = \mathbf{H_{111}\dot{q}_1\dot{q}_1 + H_{122}\dot{q}_2\dot{q}_2 + H_{133}\dot{q}_3\dot{q}_3 + H_{144}\dot{q}_4\dot{q}_4 + H_{121}\dot{q}_2\dot{q}_1 + H_{123}\dot{q}_2\dot{q}_3}$$
$$\mathbf{+H_{124}\dot{q}_2\dot{q}_4 + H_{131}\dot{q}_3\dot{q}_1 + H_{132}\dot{q}_3\dot{q}_2 + H_{134}\dot{q}_3\dot{q}_4 + H_{112}\dot{q}_1\dot{q}_1 + H_{113}\dot{q}_1\dot{q}_3}$$
$$\mathbf{+H_{114}\dot{q}_1\dot{q}_4 + H_{141}\dot{q}_4\dot{q}_1 + H_{142}\dot{q}_4\dot{q}_2 + H_{143}\dot{q}_4\dot{q}_3}$$

$$\frac{dD_{12}}{d\theta_2} = -\,1/_2\, l_1 l_2 m_2 \sin(\theta_2) - l_1 l_2 m_4 \sin(\theta_2)$$

$$\mathbf{H_{111}} = \frac{\mathbf{dD_{11}}}{\mathbf{dq_1}} - \mathbf{1/_2}\ \frac{\mathbf{dD_{11}}}{\mathbf{dq_1}} = \mathbf{0 - 1/_2 * 0 = Zero}$$

$$\mathbf{H_{122}} = \frac{\mathbf{dD_{12}}}{\mathbf{dq_2}} - \mathbf{1/_2}\ \frac{\mathbf{dD_{22}}}{\mathbf{dq_1}}$$

$$\mathbf{H_{122}} = \mathbf{l_1 l_2 \sin(\theta_2)\,(-1/_2\, m_2 - m_4)}$$

$$\mathbf{H_{133}} = \frac{\mathbf{dD_{13}}}{\mathbf{dd_2^*}} - \mathbf{1/_2}\ \frac{\mathbf{dD_{33}}}{\mathbf{d\theta_1}} = \mathbf{0 - 1/_2 * 0 = Zero}$$

$$\mathbf{H_{144}} = \frac{\mathbf{dD_{14}}}{\mathbf{d\theta_3}} - \mathbf{1/_2}\ \frac{\mathbf{dD_{44}}}{\mathbf{d\theta_1}} = \mathbf{0 - 1/_2 * 0 = Zero}$$

$$\mathbf{H_{121}} = \frac{\mathbf{dD_{12}}}{\mathbf{d\theta_1}} - \mathbf{1/_2}\ \frac{\mathbf{dD_{21}}}{\mathbf{d\theta_1}} = \mathbf{0 - 1/_2 * 0 = Zero}$$

$$\mathbf{H_{123}} = \frac{\mathbf{dD_{12}}}{\mathbf{dd_2^*}} - \mathbf{1/_2}\ \frac{\mathbf{dD_{23}}}{\mathbf{d\theta_1}} = \mathbf{0 - 1/_2 * 0 = Zero}$$

$$\mathbf{H_{124}} = \frac{\mathbf{dD_{12}}}{\mathbf{d\theta_3}} - \mathbf{1/_2}\ \frac{\mathbf{dD_{24}}}{\mathbf{d\theta_1}} = \mathbf{0 - 1/_2 * 0 = Zero}$$

$$\mathbf{H_{131}} = \frac{\mathbf{dD_{13}}}{\mathbf{d\theta_1}} - \mathbf{1/_2}\ \frac{\mathbf{dD_{31}}}{\mathbf{d\theta_1}} = \mathbf{0 - 1/_2 * 0 = Zero}$$

$$\mathbf{H_{132}} = \frac{\mathbf{dD_{13}}}{\mathbf{d\theta_2}} - \mathbf{1/_2}\ \frac{\mathbf{dD_{32}}}{\mathbf{d\theta_1}} = \mathbf{0 - 1/_2 * 0 = Zero}$$

➢ **Continuation of: Computing Christoffel Symbols:**

$$H_{134} = \frac{dD_{13}}{d\theta_3} - {}^{1}/_{2}\ \frac{dD_{34}}{d\theta_1} = 0 - {}^{1}/_{2} * 0 = \text{Zero}$$

$$H_{112} = \frac{dD_{11}}{d\theta_2} - {}^{1}/_{2}\ \frac{dD_{12}}{d\theta_1}$$

$$H_{112} = -m_2 l_1 l_2 \sin(\theta_2) - 2m_4 l_1 l_2 \sin(\theta_2) * l_1 l_2 \sin(\theta_2)(-m_2 - 2m_4)$$

$$H_{113} = \frac{dD_{11}}{dd_2^*} - {}^{1}/_{2}\ \frac{dD_{13}}{d\theta_1} = 0 - {}^{1}/_{2} * 0 = \text{Zero}$$

$$H_{114} = \frac{dD_{14}}{d\theta_3} - {}^{1}/_{2}\ \frac{dD_{14}}{d\theta_1} = 0 - {}^{1}/_{2} * 0 = \text{Zero}$$

$$H_{141} = \frac{dD_{14}}{d\theta_1} - {}^{1}/_{2}\ \frac{dD_{41}}{d\theta_1} = 0 - {}^{1}/_{2} * 0 = \text{Zero}$$

$$H_{142} = \frac{dD_{14}}{d\theta_2} - {}^{1}/_{2}\ \frac{dD_{42}}{d\theta_1} = 0 - {}^{1}/_{2} * 0 = \text{Zero}$$

$$H_{143} = \frac{dD_{14}}{dd_2^*} - {}^{1}/_{2}\ \frac{dD_{43}}{d\theta_1} = 0 - {}^{1}/_{2} * 0 = \text{Zero}$$

$$H_1 = l_1 l_2 \sin(\theta_2)\left(-{}^{1}/_{2}\, m_2 - m_4\right)\dot{\theta}_2^2 + l_1 l_2 \sin(\theta_2)(-m_2 - 2m_4)\,\dot{\theta}_1\dot{\theta}_2 \quad \Rightarrow \quad (15)$$

$$H_{211} = \frac{dD_{21}}{d\theta_1} - {}^{1}/_{2}\ \frac{dD_{11}}{d\theta_2} = 0 - {}^{1}/_{2} * (-m_2 l_1 l_2 \sin(\theta_2) - 2m_4 l_1 l_2 \sin(\theta_2))$$

$$H_{211} = {}^{1}/_{2}\, l_1 l_2 \sin(\theta_2)\,(m_2 + 2m_4)$$

$$H_{222} = \frac{dD_{22}}{d\theta_2} - {}^{1}/_{2}\ \frac{dD_{22}}{d\theta_2} = \text{Zero}$$

$$H_{233} = \frac{dD_{23}}{dd_2^*} - {}^{1}/_{2}\ \frac{dD_{33}}{d\theta_2} = \text{Zero}$$

$$H_{244} = \frac{dD_{24}}{d\theta_3} - {}^{1}/_{2}\ \frac{dD_{44}}{d\theta_2} = \text{Zero}$$

$$H_{221} = \frac{dD_{22}}{d\theta_1} - {}^{1}/_{2}\ \frac{dD_{21}}{d\theta_2} = {}^{1}/_{2}\, l_1 l_2 \sin(\theta_2)({}^{1}/_{2}\, m_2 + m_4)$$

$$H_{223} = \text{Zero}$$

$$H_{224} = \text{Zero}$$

$$H_{232} = \text{Zero}$$

➢ **Continuation of: Computing Christoffel Symbols:**

$$H_{231} = \text{Zero}$$

$$H_{234} = \text{Zero}$$

$$H_{212} = \frac{dD_{24}}{d\theta_2} - \frac{1}{2}\frac{dD_{12}}{d\theta_2} = -m_2 l_1 l_1 s\theta_2 - 2m_4 l_1 l_1 s\theta_2 + \frac{1}{2} m_2 l_1 l_1 s\theta_2$$

$$H_{223} = \text{Zero} \quad : \quad H_{224} = \text{Zero} \quad : \quad H_{232} = \text{Zero}$$

$$H_{231} = \text{Zero} \quad : \quad H_{234} = \text{Zero}$$

$$H_{212} = \frac{dD_{24}}{d\theta_2} - \frac{1}{2}\frac{dD_{12}}{d\theta_2} = -m_2 l_1 l_1 s\theta_2 - 2m_4 l_1 l_1 s\theta_2 + \frac{1}{2} m_2 l_1 l_1 s\theta_2 + m_4 l_1 l_1 s\theta_2$$

$$H_{212} = \frac{-1}{2} m_2 l_1 l_1 s\theta_2 - m_4 l_1 l_1 s\theta_2 = l_1 l_1 s\theta_2 \left(\frac{-1}{2} m_2 - m_4\right)$$

$$H_{213} = \frac{dD_{21}}{d\theta_3} - \frac{1}{2}\frac{dD_{13}}{d\theta_2} = \text{Zero} = 0$$

$$H_{214} = \frac{dD_{21}}{d\theta_3} - \frac{1}{2}\frac{dD_{14}}{d\theta_2} = \text{Zero} = 0$$

$$H_{241} = \text{Zero} \quad : \quad H_{242} = \text{Zero} \quad : \quad H_{243} = \text{Zero}$$

$$H_2 = \frac{1}{2} l_1 l_1 s\theta_2 (m_2 + 2m_4)\, \theta_1^2 + \frac{1}{2} l_1 l_1 s\theta_2 \left(\frac{1}{2} m_2 + m_4\right) \theta_2' \theta_1'$$

$$-l_1 l_1 s\theta_2 \left(\frac{1}{2} m_2 + m_4\right) \theta_1' \theta_2'$$

$$\boxed{H_2 = \frac{1}{2} l_1 l_1 s\theta_2 (m_2 + 2m_4)\, \theta_1^2 + \frac{1}{2} l_1 l_1 s\theta_2 \left(\frac{1}{2} m_2 + m_4\right) \theta_1' \theta_2'} \quad \Rightarrow \quad (16)$$

$$H_{311} = \frac{dD_{31}}{d\theta_1} - \frac{1}{2}\frac{dD_{11}}{d.d_2^*} = \text{Zero}$$

$$H_{322} = \text{Zero} \quad : \quad H_{344} = \text{Zero} \quad : \quad H_{321} = \frac{dD_{32}}{d\theta_1} - \frac{1}{2}\frac{dD_{21}}{d.d_2^*} = \text{Zero}$$

$$H_{323} = \text{Zero} \quad : \quad H_{324} = \text{Zero} \quad : \quad H_{331} = \text{Zero}$$

$$H_{332} = \text{Zero} \quad : \quad H_{334} = \text{Zero} \quad : \quad H_{312} = \frac{dD_{31}}{d\theta_2} - \frac{1}{2}\frac{dD_{12}}{d.d_2^*} = \text{Zero}$$

$$H_{313} = \text{Zero} \quad : \quad H_{314} = \text{Zero} \quad : \quad H_{341} = \text{Zero} \quad : \quad H_{342} = \text{Zero}$$

$$H_{343} = \text{Zero}$$

➢ **Continuation of: Computing Christoffel Symbols:**

H_3= Zero = 0 ⟶ (17)

H_4= Zero = 0 ⟶ (18)

✓ **From 15,16,17 and 18:**

$$H = \begin{bmatrix} H_1 \\ H_2 \\ H_3 \\ H_4 \end{bmatrix} = \begin{bmatrix} l_1 l_2 \sin(\theta_2)\left(-{}^{1}/_{2}\, m_2 - m_4\right)\dot{\theta}_2^2 + l_1 l_2 \sin(\theta_2)(-m_2 - 2m_4)\,\dot{\theta}_1\dot{\theta}_2 \\ \frac{1}{2} l_1 l_1 s\theta_2 (m_2 + 2m_4)\,\theta_1^2 + \frac{1}{2} l_1 l_1 s\theta_2 \left(\frac{1}{2} m_2 + m_4\right)\theta_1'\theta_2' \\ 0 \\ 0 \end{bmatrix}$$

$$C(q, \dot{q}) = \begin{bmatrix} l_1 l_1 s\theta_2(-m_2 - 2m_4)\theta_2' & l_1 l_1 s\theta_2\left(-\frac{1}{2} m_2 - m_4\right)\theta_2' & 0 & 0 \\ \frac{1}{2} l_1 l_1 s\theta_2 (m_2 + 2m_4)\theta_1' & \frac{-1}{2} l_1 l_1 s\theta_2 \left(\frac{1}{2} m_2 + 2m_4\right)\theta_1' & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} \theta_1' \\ \theta_2' \\ d_2^* \\ \theta_3' \end{bmatrix}$$

- **Computing the Gravity matrix $G(q)$:**
  - **The gravity components:**

$$G(q) = \sum_{j=1}^{n} m_j\, g^T J_{Dj}^i$$

- Where m_j is the mass of the g^T is the vector gravity and

$$g = \begin{bmatrix} 0 \\ -g \\ 0 \end{bmatrix} \longrightarrow$$ **where g is the gravity (9.81 m/sec)**

$\because\ G_1 = m_1 g^T J_{D1}^1 + m_2 g^T J_{D2}^1 + m_3 g^T J_{D3}^1 + m_4 g^T J_{D4}^1$

$$\therefore G_1 = m_1 [0 \quad -g \quad 0] \begin{bmatrix} \frac{-1}{2} l_1 s\theta_1 \\ \frac{1}{2} l_1 c\theta_1 \\ 0 \end{bmatrix} + m_2 [0 \quad -g \quad 0] \begin{bmatrix} -l_1 s\theta_1 - \frac{1}{2} l_2 s\theta_1\theta_2 \\ l_1 c\theta_1 + \frac{1}{2} l_2 c\theta_1\theta_2 \\ 0 \end{bmatrix}$$

$$+ m_3 [0 \quad -g \quad 0] \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + m_4 [0 \quad -g \quad 0] \begin{bmatrix} -l_1 s\theta_1 - l_2 s\theta_1\theta_2 \\ l_1 c\theta_1 + l_2 c\theta_1\theta_2 \\ 0 \end{bmatrix}$$

$$G_1 = \frac{-1}{2} m_1\, g l_1 c\theta_1 - (m_2\, g(l_1 c\theta_1 + \frac{1}{2} l_2 c\theta_1\theta_2)) - (m_4\, g(l_1 c\theta_1 + \frac{1}{2} l_2 c\theta_1\theta_2)) \quad \Rightarrow \quad (19)$$

$$\because\ G_2 = m_1 g^T J D_1^2 + m_2 g^T J D_2^2 + m_3 g^T J D_3^2 + m_4 g^T J D_4^2$$

$$\therefore\ G_2 = m_1 [0 \quad -g \quad 0] \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + m_2 [0 \quad -g \quad 0] \begin{bmatrix} \frac{-1}{2} l_1 s\theta_1\theta_2 \\ \frac{1}{2} l_1 c\theta_1\theta_2 \\ 0 \end{bmatrix} + m_3 [0 \quad -g \quad 0] \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$+ m_4 [0 \quad -g \quad 0] \begin{bmatrix} -l_2 s\theta_1\theta_2 \\ l_2 c\theta_1\theta_2 \\ 0 \end{bmatrix}$$

$$G_2 = \frac{-1}{2} m_2\, g l_2 c\theta_1\theta_2 - m_4\, g l_2 c\theta_1\theta_2 \quad \Rightarrow \quad (20)$$

➢ **Continuation of: Computing Gravity matrix:**

$$\because\ G_3 = m_1 g^T J D_1^3 + m_2 g^T J D_2^3 + m_3 g^T J D_3^3 + m_4 g^T J D_4^3$$

$$\therefore\ G_3 = m_1 [0 \quad -g \quad 0] \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + m_2 [0 \quad -g \quad 0] \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + m_3 [0 \quad -g \quad 0] \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$+ m_4 [0 \quad -g \quad 0] \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0 \qquad G_3 = 0 \quad \Rightarrow \quad (21)$$

$$\because\ G_4 = m_1 g^T J D_1^4 + m_2 g^T J D_2^4 + m_3 g^T J D_3^4 + m_4 g^T J D_4^4$$

$$\therefore\ G_4 = m_1 [0 \quad -g \quad 0] \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + m_2 [0 \quad -g \quad 0] \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + m_2 [0 \quad -g \quad 0] \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$+ m_4 [0 \quad -g \quad 0] \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad G_4 = -0 \quad \Rightarrow \quad (22)$$

$$G(q) = \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix} = \begin{bmatrix} \frac{-1}{2} m_1\, g l_1 c\theta_1 - (m_2\, g(l_1 c\theta_1 + \frac{1}{2} l_2 c\theta_1\theta_2)) - (m_4\, g(l_1 c\theta_1 + \frac{1}{2} l_2 c\theta_1\theta_2)) \\ \frac{-1}{2} m_2\, g l_2 c\theta_1\theta_2 - m_4\, g l_2 c\theta_1\theta_2 \\ 0 \\ 0 \end{bmatrix}$$

- **Computing the Torques of the Motors:**

  Finally, we can write down the dynamical equations of the system as in equation **(1).** Substituting for the various quantities in this equation **(D(q), H(q) and G(q)) and omitting zero** terms leads to:

$$\tau = \begin{bmatrix} \tau_1 \\ \tau_2 \\ F \\ \tau_3 \end{bmatrix} = \begin{bmatrix} D_{11}\theta_1'' + D_{12}\theta_2'' + D_{14}\theta_3'' + H_{11}\theta_1' + H_{12}\theta_2' + G_1 \\ D_{21}\theta_1'' + D_{22}\theta_2'' + D_{24}\theta_3'' + H_{21}\theta_1' + H_{22}\theta_2' + G_2 \\ D_{33}d_3'' \\ D_{41}\theta_1'' + D_{42}\theta_2'' + D_{44}\theta_3'' \end{bmatrix}$$

$$\tau_1 = D_{11}\theta_1'' + D_{12}\theta_2'' + D_{14}\theta_3'' + H_{11}\theta_1' + H_{12}\theta_2' + G_1 \longrightarrow \text{Base Motor Torque}$$

$$\tau_2 = D_{21}\theta_1'' + D_{22}\theta_2'' + D_{24}\theta_3'' + H_{21}\theta_1' + H_{22}\theta_2' + G_2 \longrightarrow \text{Link Motor Torque}$$

$$F = D_{33}d_3'' \longrightarrow \text{Ball Screw Motor Force}$$

$$\tau_3 = D_{41}\theta_1'' + D_{42}\theta_2'' + D_{44}\theta_3'' \longrightarrow \text{End Effector Motor Torque}$$

4.5.5 Computed-Torque Control:

- **Introduction:**

The control technique implemented is the computed torque control (CTC) which is classified as model-based control. Thus, we design a trajectory-following controller that tracks the desired position, the desired velocity, and the desired acceleration which provide a closed loop control with the desired frequencyfor each joint separately. $\zeta_{i_des}$and a desired damping ration ω_{ni_des} This is achieved by understanding the general form for the dynamics equations as shown in (1):

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau \quad \Longrightarrow \quad 1$$

- Where q is a vector of the joint variables and τ is the torque applied at the joints.
- The system of the dynamic model of the robot is non-linear and it is preferred to make it linear or reduce the nonlinearity or remove it.
- The system should have linearization.
- The computed torque control or feedback linearization method is used to reduce
  (eliminated theoretically) the non-linearity of the system.
- We can control the robot to follow the desired path, by introducing a computed
  torque control law as below:

$$D(q)\ddot{q} + H(q,\dot{q})\dot{q} + G(q) = \tau$$

- We need first to define the nonlinear parts in the dynamic equation and then apply
  the controller (control law) to reduce (eliminate) the nonlinearity.
- $H(q,\dot{q})\dot{q} + G(q)$ represents the nonlinear parts in the equation.
- The input to the dynamic model should make two things:
  1- try to overcome the nonlinearity of the system (τ)
  2- apply a controller to follow the desired path $\Longrightarrow$ $e = q_d - q \ , \dot{e} = \dot{q} - \dot{q_d}$
- **Types of control:**
  - **Computed Torque Control:** Calculates the required torque for each axis based on the robot's dynamic model to ensure precise control.
  - **Model Predictive Control:** Uses a mathematical model to predict and optimize control based on future predictions.
  - **Neural Network-Based Control:** Utilizes neural networks to enhance control strategies based on learning from data.
  - **Adaptive Control:** Adjusts control strategies based on changes in the system or environment.
- We will use **Computed Torque Control (CTC).**

- **Computed-Torque Control**

❑ Derivation of inner feedforward loop

- The robot dynamics:

$$\mathbf{D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau}$$

- For convenience $\boldsymbol{C(q,\dot{q})\dot{q} + G(q) = N(q,\dot{q})}$**:**

$$\mathbf{D(q)\ddot{q} + N(q,\dot{q}) = \tau}$$

- Define tracking error:

$$\mathbf{e(t) = q_d - q_a}$$

where, q_d is a desired trajectory.
where, q_a is an actual trajectory.

- Differentiate the error twice:

$$\mathbf{e = q_d - q}$$
$$\mathbf{\dot{e} = \dot{q}_d - \dot{q}}$$
$$\mathbf{\ddot{e} = \ddot{q}_d - \ddot{q}_a}$$

- PD Controller (to calculate motor currents)

$$\mathbf{i_n = k_p(q_d - q_a) + k_v(\dot{q}_d - \dot{q}_a) + k_i \int (q_d - q_a)}$$

- Motor Dynamics (to calculate motor torques – neglect the motors dynamics)

$$\mathbf{\tau_n \cong i_n}$$

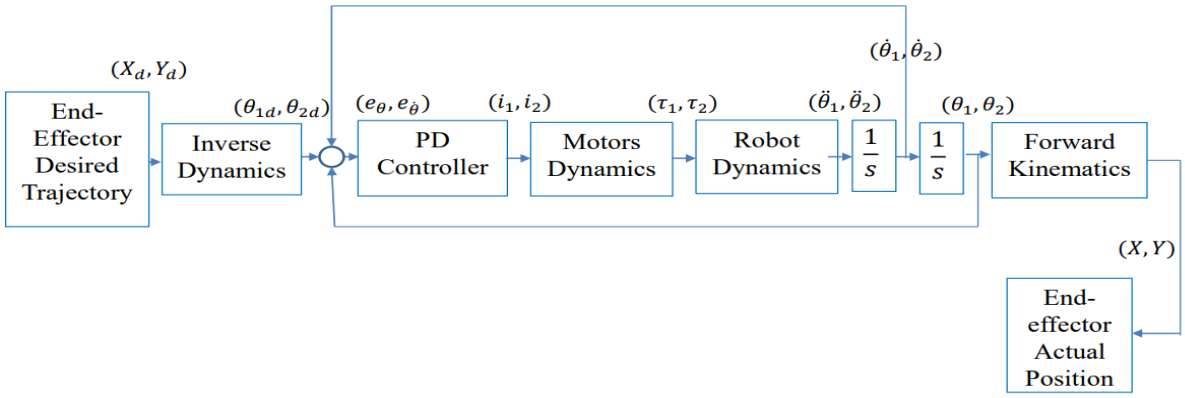
- Robot Dynamics (to calculate actual accelerations)

$$\mathbf{D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau}$$

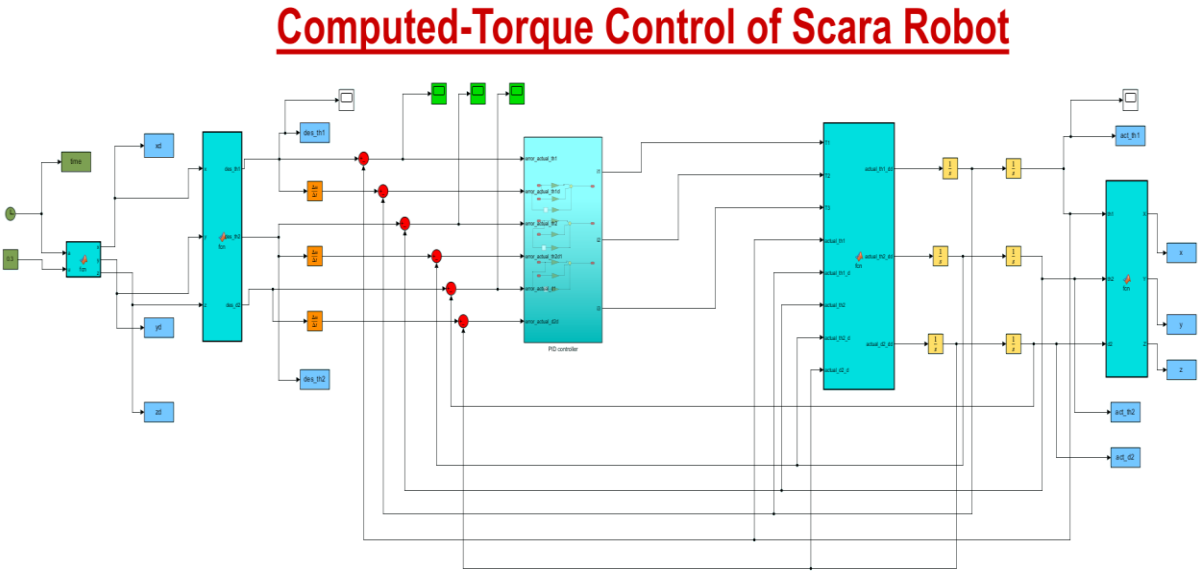
➢ we can write it:

$$\mathbf{\ddot{q} = D^{-1}(\tau - C(q,\dot{q})\dot{q} - G(q))}$$
$$\mathbf{\dot{q} = \int \ddot{q}}$$
$$\mathbf{q = \int \dot{q}}$$

- **Blook Diagram for Control**



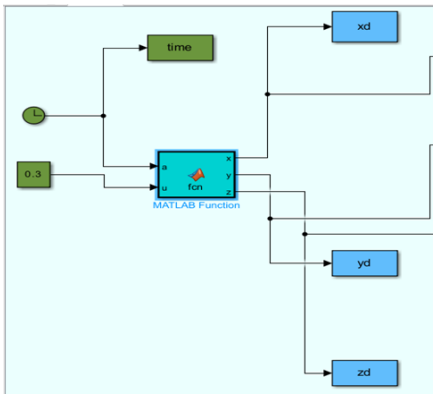
- **Simulink Simulation**



- **Simulink Simulation**

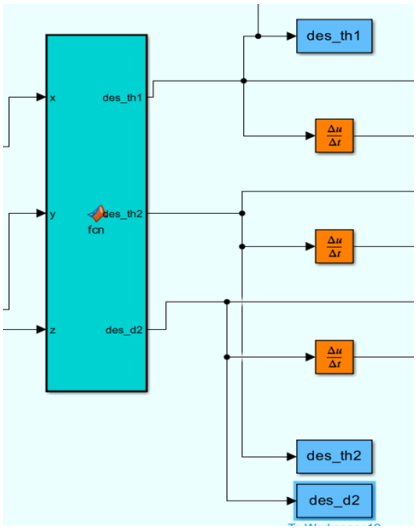
1. **Trajectory (MATLAB function).**

```
function [x,y,z] = fcn(a,u)

x = 1 +u*cosd(a);
y = 1 +u*sind(a);
z=8;
end
```



2. Inverse Kinematics (MATLAB function).

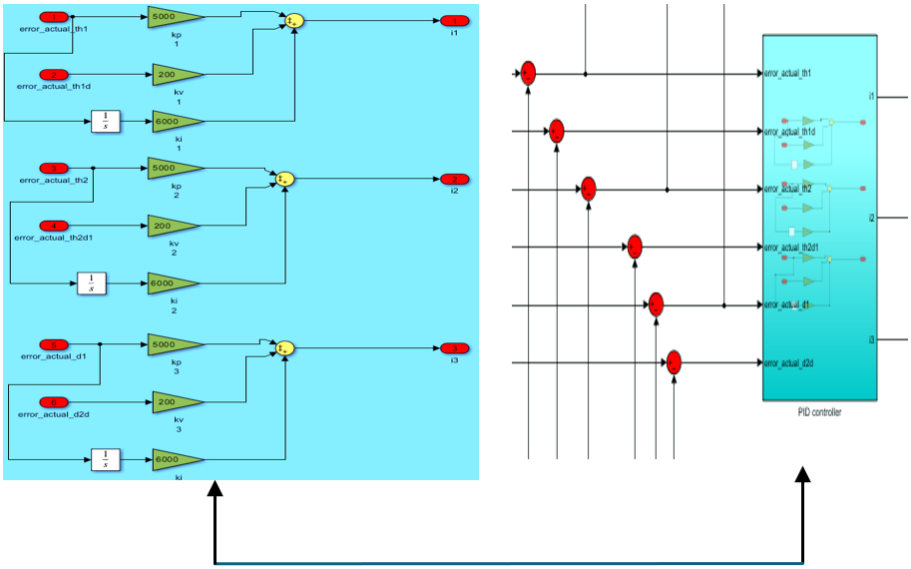
```
function [des_th1, des_th2,des_d2]=fcn(x,y,z)
l1=2;
l2=2;
d1=2;
d3=2;
d4=2;
c2=(x^2+y^2-l1^2-l2^2)/(2*l1*l2);
s2=(1 - c2^2)^0.5;
k1=l1+l2*c2;
k2=l2*s2;
des_th1=atan2(y,x)-atan2(k2, k1);
des_th2=acos(c2);
des_d2=z-d1+d3+d4;
end
```



3. PID Controller:

- PID computed-torque control is very effective if all the arm parameters are known and there is no disturbance. → it gives a nonzero steady-state error.
- Consequently, we add an integral controller in the feedback loop.
- PD Controller (to calculate motor currents)

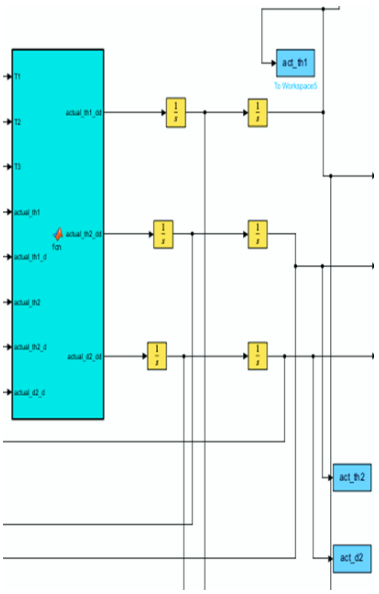
$$i_n = k_p(q_d - q_a) + k_v(\dot{q}_d - \dot{q}_a) + k_i \int (q_d - q_a)$$



Joint no	K_p	K_V	K_i
1	5000	200	6000
2	5000	200	6000
3	5000	200	6000

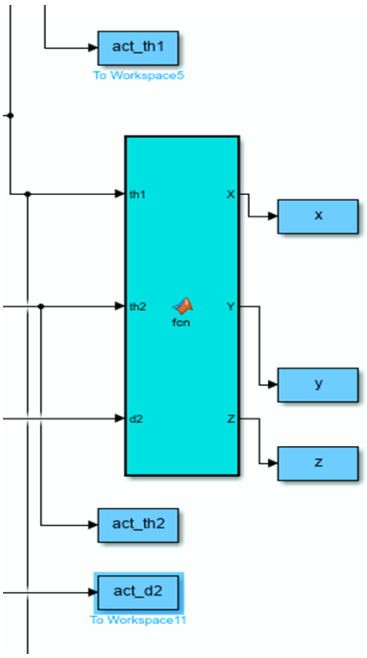
4. Robot dynamic model (MATLAB function):

```
function [actual_th1_dd, actual_th2_dd,actual_d2_dd] = fcn(T1,T2,T3, actual_th1,actual_th1_d,actual_th2,actual_th2_d,actual_d2_d)
m1=1;
m2=1 ;
m3=1;
l1=2 ;
l2=2  ;
I1= 0.001;
I2= 0.001 ;
I3=0.001;
g=9.81;
%Lc1=.5*L1;
%Lc2=.5*L2;
th1 = actual_th1;
th2 = actual_th2;
th1d = actual_th1_d;
th2d = actual_th2_d;
%d2=actual_d2;
d2d=actual_d2_d;
M = [l1^2*m1/4+(m2*(4*l1^2+4*cos(th2)*l1*l2+l2^2))/4+0.5*(I1+I2+I3)  (12*m2*(12+2*l1*cos(th2)))/4+0.5*(I2) 0;
    (12*m2*(12+2*l1*cos(th2)))/4+ 0.5*(I2)            |       (12^2*m2)/4+0.5*(I2) 0;
    0                                                  0                          m1+m2+m3];

C= [-0.5*m2*l1*l2*sin(th2)*th2d   -0.5*m2*l1*l2*sin(th2)*(th2d) 0;
   0.5*m2*l1*l2*sin(th2)*th1d  -0.5*m2*l1*l2*sin(th2)*th1d 0;
   0                            0                          0];
V=C*[th1d;th2d;d2d];
G=[0.5*m1*g*l1*cos(th1)+(m2*g*(l1*cos(th1)+0.5*l2*cos(th1+th2)));
   0.5*m2*g*l2*cos(th1+th2);
   -g*(m1+m2+m3)];
T=[T1;T2;T3];
actual_th_dd= inv(M)*(T-V-G);
v=actual_th_dd(1);
w=actual_th_dd(2);
c=actual_th_dd(3);
actual_th1_dd =v;
actual_th2_dd= w;
actual_d2_dd=c;
end
```



5. Forward Kinematics (MATLAB function)

```
function [X,Y,Z]=fcn(th1,th2,d2)
l1=2;
l2=2;
d1=2;
d3=2;
d4=2;
X=l1*cos(th1)+l2*cos(th1+th2);
Y=l1*sin(th1)+l2*sin(th1+th2);
Z=d1+d2-d3-d4;
end
```



6. SYSTEM SIMULATION

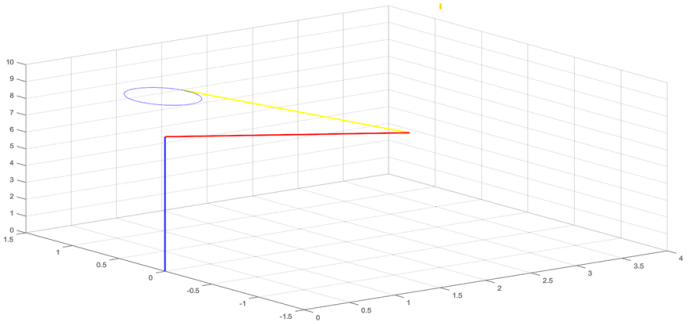


FIGURE 29 SYSTEM SIMULATION WITH FIRST TRAJECTORY

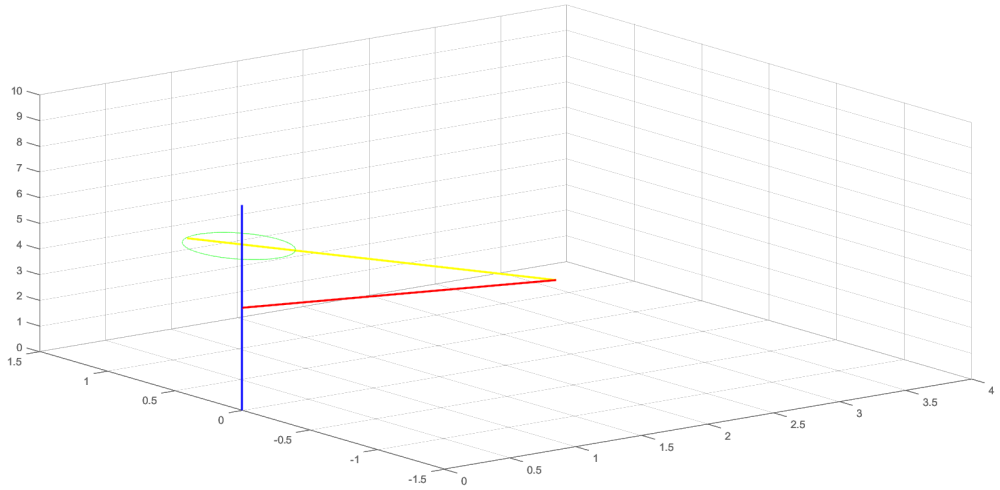


FIGURE 30 SYSTEM SIMULATION WITH SECOND TRAJECTORY

7. SYSTEM CTC AND OUTPUT

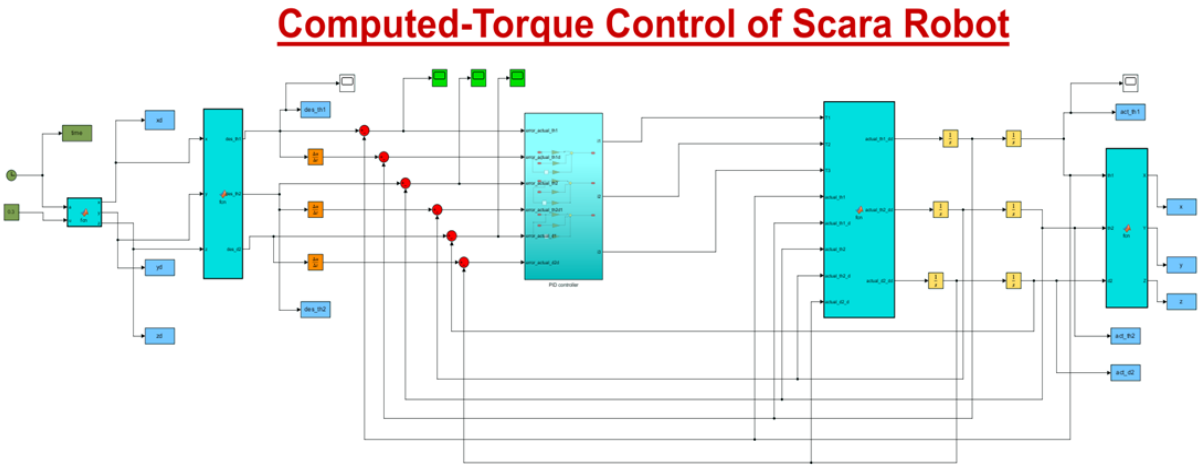


FIGURE 31 SIMULINK OF CTC

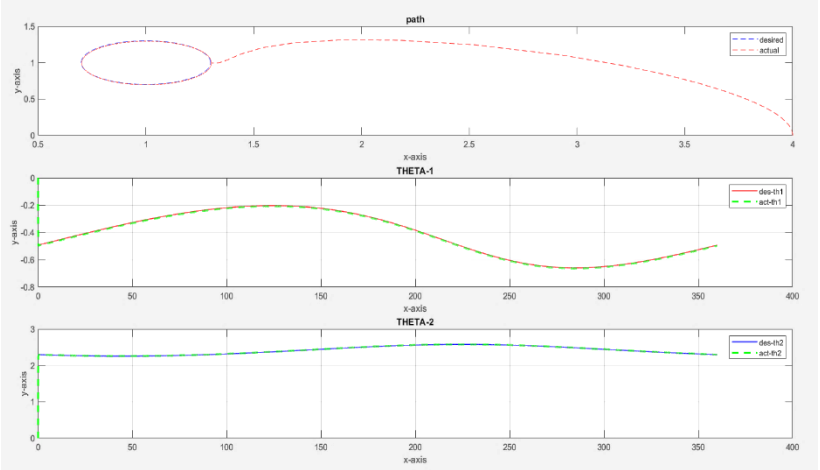


FIGURE 32 OUTPUT OF SYSTEM SIMULATION

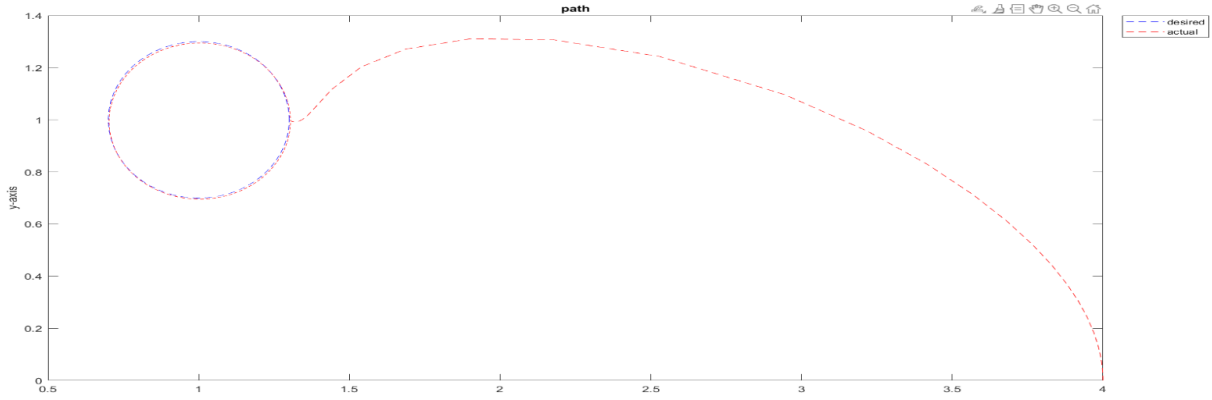


FIGURE 33

A COMPARISON BETWEEN THE DESIRED AND ACTUAL TRAJECTORY

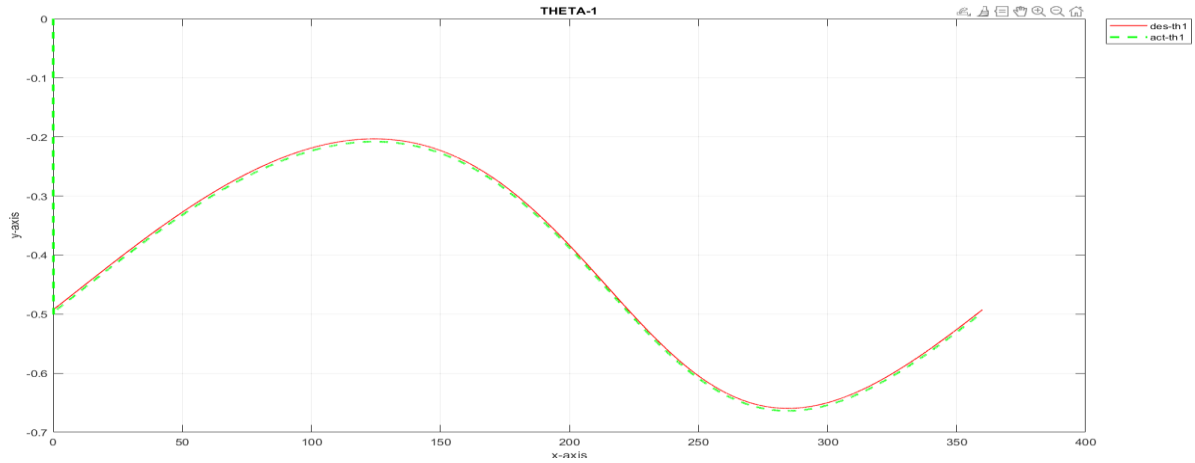


FIGURE 34

A COMPARISON BETWEEN THE DESIRED RESPONSE AND THE SIMULATED RESPONSE FOR THE FIRST JOINT

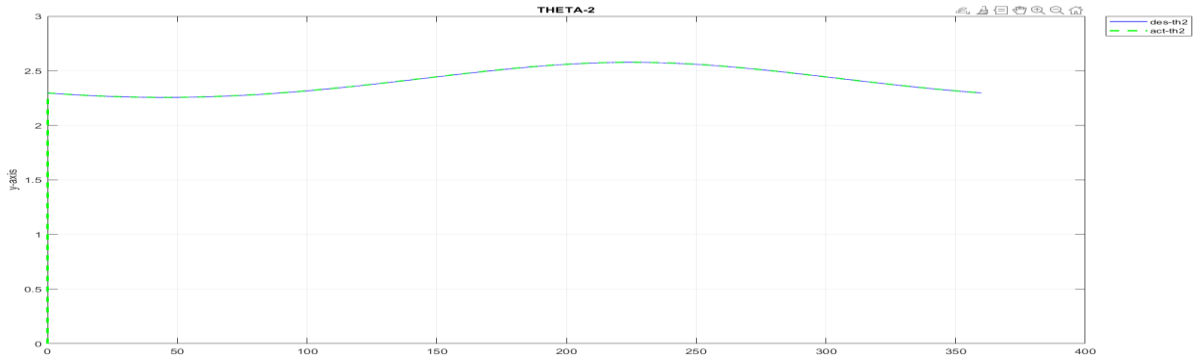


FIGURE 35

A COMPARISON BETWEEN THE DESIRED RESPONSE AND THE SIMULATED RESPONSE FOR THE SECOND JOINT

Chapter 5
MACHINE VISION

Chapter 5
MACHINE VISION

5.1 Introduction to Machine Vision

Overview of SCARA Robots in Pack-and-Place Operations SCARA (Selective Compliance Assembly Robot Arm) robots are widely utilized in pack-and-place applications due to their speed, precision, and efficiency. These robots excel in repetitive tasks that require consistent accuracy, such as picking items from a conveyor belt and placing them into packaging. Their rigid vertical axis and selective compliance in the horizontal plane make them ideal for planar motion tasks in packaging lines.

Role of Machine Vision in Pack-and-Place Applications Machine vision enhances SCARA robots by equipping them with the ability to "see" and interpret their environment. In pack-and-place operations, machine vision is used to identify, locate, and classify objects for accurate picking and placement. This technology enables robots to adapt to dynamic environments, such as handling randomly oriented items or sorting products by size or shape.

5.1.1 Key Benefits of Machine Vision for SCARA Pack-and-Place:

1. **Increased Accuracy:** Machine vision ensures precise object detection and alignment, reducing errors during placement.
2. **Improved Flexibility:** Robots can handle a variety of items without mechanical adjustments by using vision-based guidance.
3. **Higher Throughput:** Vision systems enable faster decision-making, allowing robots to operate efficiently in high-speed production lines.
4. **Reduced Waste:** By accurately detecting defective or misplaced items, vision systems help maintain product quality and reduce waste.

5.1.2 Workflow of Machine Vision in Pack-and-Place Operations:

1. **Image Capture:** The camera captures an image of the workspace, including objects on a conveyor or in a bin.
2. **Object Detection and Analysis:** Vision software processes the image to identify objects, their positions, and orientations.
3. **Communication:** The vision system sends data to the robot controller, specifying where and how to pick the object.
4. **Robot Action:** The SCARA robot picks the object and places it in the designated location, such as a package or tray.

5.1.3 Applications for Machine Vision in Pack-and-Place Tasks:

1. **Random Bin Picking:** Identifying and picking items from a bin with varying orientations.
2. **Conveyor Tracking:** Detecting and picking items moving on a conveyor belt.
3. **Sorting and Packaging:** Categorizing items by size, shape, or color and placing them in appropriate packaging.
4. **Defect Detection:** Inspecting items for defects before placement to ensure only quality products are packaged.

5.1.4Challenges in Machine Vision Integration:

1. **Dynamic Environments:** Variations in lighting or object appearance can impact detection accuracy.
2. **Speed Requirements:** High-speed production lines demand rapid image processing and robot response times.
3. **Calibration:** Precise calibration between the vision system and SCARA robot is essential for accurate operation.
4. **Cost:** Initial setup, including cameras, software, and integration, can be costly.

5.1.5 Conclusion

Machine vision significantly enhances the capabilities of SCARA robots in pack-and-place operations, enabling higher precision, adaptability, and efficiency. By integrating vision systems, SCARA robots can handle complex tasks, improve productivity, and maintain high-quality standards in packaging processes. As machine vision technology advances, it will continue to drive innovation in automation, unlocking new possibilities for industrial applications.

5.2 Object Detection Code Explanation

5.2.1 Importing Libraries

```
import cv2
from ultralytics import YOLO
```

- **OpenCV (Open-Source Computer Vision Library[cv2]):** is an open-source computer vision and machine learning software library. It is designed to help developers create real-time computer vision applications and includes hundreds of functions for processing images and videos. OpenCV can be used for a wide variety of tasks, including image manipulation, object detection, and video processing.
    - **Key Features of OpenCV:**
        - **Image Processing:**
            - Functions to manipulate and analyze images, such as resizing, cropping, filtering, and transforming.
            - Common operations include edge detection (Canny), thresholding, and color space conversions (e.g., RGB to Grayscale).
        - **Object Detection and Recognition:**
            - Integration with machine learning models to detect objects, faces, or specific patterns in images or videos.
            - Built-in support for pre-trained models like Haar cascades and DNN (Deep Neural Networks).
        - **Video Processing:**
            - Real-time capture and processing from video sources like webcams, cameras, or video files.
            - Frame-by-frame manipulation and display.
        - **Geometric Transformations**:
            - Operations like rotation, scaling, translation, and perspective transformations.
        - **Integration with Deep Learning Models.**

- **YOLO (You Only Look Once) from ultralytics:**

  YOLO is a real-time object detection system using deep learning to predict object classes and bounding boxes in a single pass, unlike traditional multi-step methods. The Ultralytics version is an optimized, user-friendly implementation designed for efficient detection in images, videos, and streams. It is widely adopted for its accuracy, speed, and simplicity.

  - **Key Features of YOLO from Ultralytics:**
    - **Real-Time Object Detection:**
      - Detects objects in images or videos in real-time with high accuracy.
      - Processes an entire image in a single pass, making it faster than many other models.
    - **High Accuracy:**
      - Real-Time Performance: Processes images and videos in real-time with high accuracy.
      - Optimized for both CPU and GPU environments, enabling deployment on a wide range of devices.
      - Efficient architecture minimizes latency, making it suitable for applications requiring instant results.
    - **Scalability:**
      - Offers models of varying sizes to balance between speed and accuracy based on hardware capabilities.
      - Suitable for deployment on:
        1. **Edge Devices**: Lightweight models like YOLOv8n for mobile and embedded systems.
        2. **High-Performance Servers**: Larger models like YOLOv8x for maximum accuracy.
    - **Custom Training:**
      - Supports training on custom datasets with ease, allowing users to adapt the model to specific use cases.
      - Provides tools for:
        1. Annotating datasets.
        2. Augmenting data to improve model robustness.
        3. Monitoring training metrics in real-time.
    - **Easy to Use.**

5.2.2Setting up the Webcam (Camera Capture)

```
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1280)
```

- **cv2.VideoCapture(0):** This command opens the default webcam (device index 0 or 1 refers to the primary camera on the system). The **cap** object is used to capture video frames from the webcam.

- **cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280):** This sets the video capture width to 1280 pixels, ensuring the video resolution is wide enough for object detection.

- **cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1280):** Similar to the previous line, this sets the height of the video frame to 1280 pixels.

5.2.3Loading the YOLO Model

```
model = YOLO("yolov8x-worldv2.pt")
model.set_classes(["glasses", "bottle", "cellphone", "cup"])
```

- **model = YOLO("yolov8x-worldv2.pt"):** This loads a pre-trained YOLO model from the specified file yolov8x-worldv2.pt. The model is a version of YOLOv8, and it's trained to recognize various objects.

- **model.set_classes(...):** This limits the model to detect only certain objects specified in the list, which in this case are glasses, bottles, cellphones, and cups. By default, YOLO can detect thousands of classes, but here we restrict it to just four.as shown in **Fig 36,37**.

FIGURE 36(GLASSES)

FIGURE 37(CUP)

5.2.4Defining Scaling Factors for Object Measurements

```
object_cm = 23

object_pixel = 218

scale_factor = object_cm / object_pixel
```

- **To calculate the object cm and pixel numbers, you must first calibrate the camera using the camera calibration code.**

➢ **Calibration Process:** This code allows the user to select two points on the frame using the mouse. After selecting the points, the code calculates the distance between them and displays it on the screen in pixels.

- **Functionality Overview:**
  - **The webcam is open using `cv2.VideoCapture(0)`.**
  - **The `draw_circle` function is defined to respond to mouse clicks and store the clicked points.**
  - **When two points are clicked, the distance between them is calculated using `math.hypot`.**
  - **The distance in pixels is displayed on the frame.**
  - **The code continues to display and update the video until the user presses the "Esc" key.**
- **Main Functionality:**
  - **Calculate and display the distance between the points in pixels.**
  - **Determining the value of the `scale_factor` in the main code.**

➢ **How It Works**

FIGURE 38

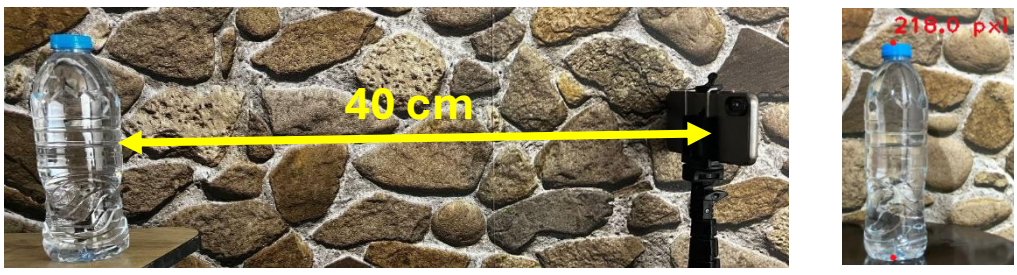


FIGURE 39

- **In Figure 38**, the distance between the camera and the object was **80 cm**, resulting in scale_factor equals **23 cm** per **251 pixels.**
- **In Figure 39,** the distance between the camera and the object was **40 cm**, resulting in scale_factor equals **23 cm** per **218 pixels**.

✓ **This indicates that as the distance between the object and the camera changes, the number of pixels changes for the same length.**

5.2.5 Starting the Video Capture Loop

```
while True:
    ret, frame = cap.read()
    if not ret:
        print("Failed to capture frame from webcam. Exiting!!!")
        break
```

- **while True:** This initiates an infinite loop that continuously captures frames from the webcam, allowing real-time object detection.
- **ret, frame = cap.read():**Captures a frame from the webcam. ret is a boolean indicating whether the frame was successfully captured, and frame holds the actual image data.
- **if not ret:** If capturing a frame fails (for example, the webcam is not accessible), the loop breaks, and a message is printed.

5.2.6 Running YOLO on the Frame

```
results = model(frame, imgsz=90)
```

- **results = model(frame, imgsz=90):** Passes the captured frame (image) to the YOLO model for object detection. **imgsz=90** resizes the input image to 90x90 pixels before processing to speed up inference. This is a typical optimization technique to balance speed and accuracy.

5.2.7 Processing the Detection Results

```
for box in results[0].boxes:
    x1, y1, x2, y2 = map(int, box.xyxy[0])
    label = results[0].names[int(box.cls[0])]
    confidence = box.conf[0]
```

- **for box in results[0].boxes:** Iterating through all detected objects in the frame.
- **x1, y1, x2, y2 = map(int, box.xyxy[0]):** Extracts the coordinates of the bounding box (top-left and bottom-right corners) and converts them into integers. These coordinates are essential for drawing rectangles around detected objects.
- **label = results[0].names[int(box.cls[0])]:** Retrieves the label (class name) of the detected object, such as "glasses," "bottle," "cellphone," or "cup".
- **confidence = box.conf[0]:** Retrieves the confidence score of the detection, which represents how confident the model is in its prediction for this object (ranging from 0 to 1).

5.2.8 Calculating Dimensions of the Detected Objects

```
object_width_pixels = x2 - x1
object_height_pixels = y2 - y1
object_width_cm = object_width_pixels * scale_factor
object_height_cm = (object_height_pixels * scale_factor) - 1.8
object_area_cm = object_width_cm * object_height_cm
```

- **object_width_pixels and object_height_pixels:** Calculate the width and height of the object in pixels.
- **object_width_cm = object_width_pixels * scale_factor:** Converts the width from pixels to centimeters using the scale_factor.
- **object_height_cm = (object_height_pixels * scale_factor) - 1.8:** Converts the height from pixels to centimeters, and subtracts a **small error (1.8 cm)** to adjust the measurement (possibly to account for distortion or offset).

- **object_area_cm = object_width_cm * object_height_cm:** Calculates the area of the detected object in square centimeters.

5.2.9 Drawing Bounding Boxes and Displaying Information

```
cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 0), 2)
cv2.putText(frame, f"{label} {confidence:.2f}", (x1, y1 - 10),
        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 0), 2)
cv2.putText(frame, f"W: {object_width_cm:.2f} cm", (x1, y1 + 20),
        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
cv2.putText(frame, f"H: {object_height_cm:.2f} cm", (x1, y1 + 40),
        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
cv2.putText(frame, f"Area: {object_area_cm:.2f} cm2", (x1, y1 + 60),
        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
```

- **cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 0), 2):** Draws a blue rectangle around the detected object using the bounding box coordinates (x1, y1, x2, y2).
- **cv2.putText(frame, f"{label} {confidence:.2f}", ...):** Displays the label of the object (e.g., "glasses") and its confidence score just above the bounding box. Same in other lines.

5.2.10 Displaying the Processed Frame

```
cv2.imshow("Webcam", frame)
```

- **cv2.imshow("Webcam", frame):** Displays the frame (with bounding boxes and text annotations) in a window titled "Webcam".

5.2.11Breaking the Loop on 'q' Key Press

```
if cv2.waitKey(1) == ord("q"):
    break
```

- **cv2.waitKey(1):** Waits for a key press for 1 millisecond and checks if the 'q' key is pressed.

- **`if cv2.waitKey(1) == ord("q")`:** If the **'q'** key is pressed, it breaks out of the loop and stops the program.

5.2.12 Releasing the Webcam and Closing Windows

```
cap.release()

cv2.destroyAllWindows()
```

- **`cap.release()`:** Releases the webcam resource, which is necessary to clean up after use.

- **`cv2.destroyAllWindows()`:** Closes any OpenCV-created windows (like the "Webcam" display window).

5.3 Object Detection Process Steps:

5.3.1 First Step:

You need to know the dimensions of the original object to determine the scale factor, as shown in Figure 40 and Figure 41.

FIGURE 40

FIGURE 41

- **In Figure 40**, the height of the object is 23 cm.
- **In Figure 41**, the width of the object is 8 cm.

5.3.2 Second Step:

Run the camera calibration code and click the left mouse button to mark the start and end points of the object to determine how many pixels correspond to the object's height, as shown in figure 42.

FIGURE 42

5.3.3 Third Step:

Change the values of object_cm and object_pixel in object detection code.

```
object_cm = 23

object_pixel = 218

scale_factor = object_cm / object_pixel
```

5.3.4 Fourth step:

Run the object detection code, the code will open a window (webcam window) that contains the detected object inside a blue rectangle representing the dimensions of this object, as shown in figure 6.

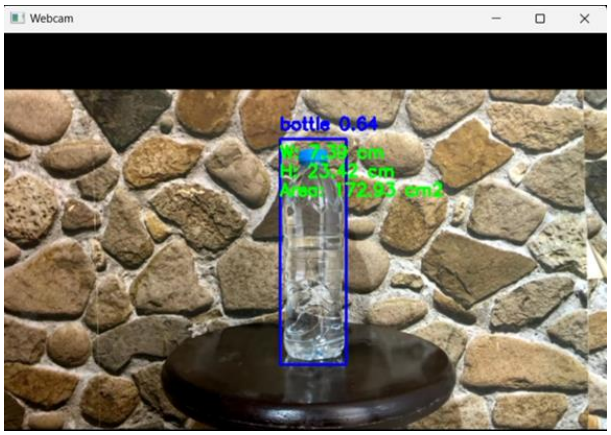


FIGURE 43

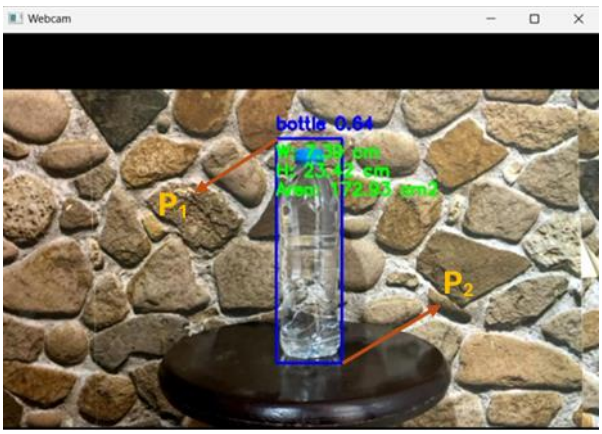


FIGURE 44

- **In Figure 43,** the code detects the object and places it inside a blue rectangle**.**
- The blue rectangle is drawn using two points **[$p_1(x_1, y_1)$, p_2 (x_2, y_2)]**, as shown in **Figure 44.**

```
cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 0), 2)
```

- **You get two points from the function of processing the detection results in the code.**

```
for box in results[0].boxes:
    x1, y1, x2, y2 = map(int, box.xyxy[0])
    label = results[0].names[int(box.cls[0])]
    confidence = box.conf[0]
```

- **You use these points to calculate the height and the width pixels in the code.**

```
object_width_pixels = x2 - x1
object_height_pixels = y2 - y1
```

- **The code will convert the pixels to centimeters by multiplying the result by the scale factor.**

```
object_width_cm = object_width_pixels * scale_factor
object_height_cm = (object_height_pixels * scale_factor) - 1.8
object_area_cm = object_width_cm * object_height_cm
```

- **And display the results of height, width and area.**

```
cv2.putText(frame, f"W: {object_width_cm:.2f} cm", (x1, y1 + 20),
        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
cv2.putText(frame, f"H: {object_height_cm:.2f} cm", (x1, y1 + 40),
        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
cv2.putText(frame, f"Area: {object_area_cm:.2f} cm2", (x1, y1 + 60),
        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
```

5.4 Summary of This Program

This program uses the **YOLO** model (from **Ultralytics**) to **detect objects** in real-time from a webcam feed. It **measures the size of each object in centimeters** based on a known reference (like a bottle), draws bounding boxes around detected objects, and displays real-world size information (width, height, and area) on the screen. The program continues running until the user presses the "q" key to exit.

APPENDIX

MATLAB FUNCTION

PART1: The Trapezoidal Motion Profile of Motors

- **Motion Profile of motor (4) MATLAB code:**

```
% Parameters
t_total = 9; % Total time (s)
alpha_max = 0.9; % Maximum angular acceleration (rad/s^2)
ta = 0.857; % Acceleration time (s)
tm = t_total - 2 * ta; % Constant velocity time (s)
td = ta; % Deceleration time (s)
omega_max = alpha_max * ta; % Maximum angular velocity (rad/s)
displacement = 360; % Total displacement (degrees)
displacement_rad = deg2rad(displacement); % Convert to radians

% Time vectors
dt = 0.01; % Time step
t1 = 0:dt:ta; % Time for acceleration phase
t2 = ta+dt:dt:ta+tm; % Time for constant velocity phase
t3 = ta+tm+dt:dt:ta+tm+td; % Time for deceleration phase

% Angular acceleration profile
alpha1 = alpha_max * ones(size(t1)); % Acceleration phase
alpha2 = zeros(size(t2)); % Constant velocity phase
alpha3 = -alpha_max * ones(size(t3)); % Deceleration phase

% Angular velocity profile
omega1 = alpha_max * t1; % Acceleration phase
omega2 = omega_max * ones(size(t2)); % Constant velocity phase
omega3 = omega_max - alpha_max * (t3 - (ta + tm)); %
Deceleration phase

% Angular position profile
theta1 = 0.5 * alpha_max * t1.^2; % Position during
acceleration
theta2 = theta1(end) + omega_max * (t2 - ta); % Position
during constant velocity
theta3 = theta2(end) + omega_max * (t3 - (ta + tm)) - 0.5 *
alpha_max * (t3 - (ta + tm)).^2; % Position during
deceleration

% Combine profiles
t = [t1, t2, t3];
alpha = [alpha1, alpha2, alpha3];
omega = [omega1, omega2, omega3];
theta = [theta1, theta2, theta3];
```

```
% Total rotation in radians and degrees
theta_total = theta(end);
theta_total_deg = rad2deg(theta_total);

% Plotting
figure;
subplot(3,1,1);
plot(t, alpha, 'LineWidth', 2);
grid on;
title('Angular Acceleration');
xlabel('Time (s)');
ylabel('Angular Acceleration (rad/s^2)');
subplot(3,1,2);
plot(t, omega, 'LineWidth', 2);
grid on;
title('Angular Velocity');
xlabel('Time (s)');
ylabel('Angular Velocity (rad/s)');
subplot(3,1,3);
plot(t, theta, 'LineWidth', 2);
grid on;
title('Angular Position');
xlabel('Time (s)');
ylabel('Angular Position (rad)');
```

- **Motion Profile of motor (3) MATLAB code:**

```
% Parameters
t_total = 10; % Total time (s)
alpha_max = 0.5; % Maximum angular acceleration (rad/s^2)
ta = 1.47; % Acceleration time (s)
tm = t_total - 2 * ta; % Constant velocity time (s)
td = ta; % Deceleration time (s)
omega_max = alpha_max * ta; % Maximum angular velocity (rad/s)
displacement = 360; % Total displacement (degrees)
displacement_rad = deg2rad(displacement); % Convert to radians

% Time vectors
dt = 0.01; % Time step
t1 = 0:dt:ta; % Time for acceleration phase
t2 = ta+dt:dt:ta+tm; % Time for constant velocity phase
t3 = ta+tm+dt:dt:ta+tm+td; % Time for deceleration phase

% Angular acceleration profile
alpha1 = alpha_max * ones(size(t1)); % Acceleration phase
alpha2 = zeros(size(t2)); % Constant velocity phase
alpha3 = -alpha_max * ones(size(t3)); % Deceleration phase
% Angular velocity profile
omega1 = alpha_max * t1; % Acceleration phase
omega2 = omega_max * ones(size(t2)); % Constant velocity phase
```

```
omega3 = omega_max - alpha_max * (t3 - (ta + tm)); %
Deceleration phase

% Angular position profile
theta1 = 0.5 * alpha_max * t1.^2; % Position during
acceleration
theta2 = theta1(end) + omega_max * (t2 - ta); % Position
during constant velocity
theta3 = theta2(end) + omega_max * (t3 - (ta + tm)) - 0.5 *
alpha_max * (t3 - (ta + tm)).^2; % Position during
deceleration

% Combine profiles
t = [t1, t2, t3];
alpha = [alpha1, alpha2, alpha3];
omega = [omega1, omega2, omega3];
theta = [theta1, theta2, theta3];

% Total rotation in radians and degrees
theta_total = theta(end);
theta_total_deg = rad2deg(theta_total);
% Plotting
figure;
subplot(3,1,1);
plot(t, alpha, 'LineWidth', 2);
grid on;
title('Angular Acceleration');
xlabel('Time (s)');
ylabel('Angular Acceleration (rad/s^2)');
subplot(3,1,2);
plot(t, omega, 'LineWidth', 2);
grid on;
title('Angular Velocity');
xlabel('Time (s)');
ylabel('Angular Velocity (rad/s)');
subplot(3,1,3);
plot(t, theta, 'LineWidth', 2);
grid on;
title('Angular Position');
xlabel('Time (s)');
ylabel('Angular Position (rad)');
```

- **Motion Profile of motor (2) MATLAB code:**

```
% Parameters
total_distance = 0.3; % Total distance (m)
total_time = 12;      % Total time (s)
pitch = 0.0025;       % Pitch (m/rev)

% Time distribution
T_a = total_time / 3; % Acceleration time (s)
T_c = total_time / 3; % Constant velocity time (s)
T_d = total_time / 3; % Deceleration time (s)

% Maximum velocity
V_max = total_distance / (T_a + T_c); % Maximum velocity (m/s)

% Acceleration
a = V_max / T_a; % Acceleration (m/s^2)

% Time vectors for each phase
t_acc = linspace(0, T_a, 100); % Acceleration phase
t_const = linspace(T_a, T_a + T_c, 100); % Constant velocity
phase
t_dec = linspace(T_a + T_c, total_time, 100); % Deceleration
phase

% Position profiles (in meters)
x_acc = 0.5 * a * t_acc.^2; % Position during acceleration (m)
x_const = x_acc(end) + V_max * (t_const - T_a); % Position
during constant velocity (m)
x_dec = x_const(end) + V_max * (t_dec - (T_a + T_c)) ...
- 0.5 * a * (t_dec - (T_a + T_c)).^2; % Position during
deceleration (m)

% Convert position to millimeters
x_acc = x_acc * 1000; % Position during acceleration (mm)
x_const = x_const * 1000; % Position during constant velocity
(mm)
x_dec = x_dec * 1000; % Position during deceleration (mm)
x = [x_acc, x_const, x_dec]; % Combined position profile in mm

% Velocity profiles
v_acc = a * t_acc; % Velocity during acceleration
v_const = V_max * ones(size(t_const)); % Velocity during
constant velocity
v_dec = V_max - a * (t_dec - (T_a + T_c)); % Velocity during
deceleration

% Angular velocity of the motor
omega_acc = v_acc / pitch; % Angular velocity during
acceleration
```

```
omega_const = v_const / pitch; % Angular velocity during
constant velocity
omega_dec = v_dec / pitch; % Angular velocity during
deceleration

% Angular acceleration of the motor
alpha_acc = (a / pitch) * ones(size(t_acc)); % Angular
acceleration during acceleration
alpha_const = zeros(size(t_const)); % Angular acceleration
during constant velocity
alpha_dec = -(a / pitch) * ones(size(t_dec)); % Angular
acceleration during deceleration
alpha = [alpha_acc, alpha_const, alpha_dec];

% Combine time and profiles
t = [t_acc, t_const, t_dec];
v = [v_acc, v_const, v_dec];
omega = [omega_acc, omega_const, omega_dec];

% Plotting
figure;

% Position profile
subplot(4, 1, 1);
plot(t, x, 'b', 'LineWidth', 1.5);
grid on;
title('Position');
xlabel('Time (s)');
ylabel('Position (mm)');
legend({'Position'});

% Velocity profile
subplot(4, 1, 2);
plot(t, v, 'r', 'LineWidth', 1.5);
grid on;
title('Velocity');
xlabel('Time (s)');
ylabel('Velocity (m/s)');
legend({'Velocity'});

% Angular velocity profile
subplot(4, 1, 3);
plot(t, omega, 'g', 'LineWidth', 1.5);
grid on;
title('Angular Velocity');
xlabel('Time (s)');
ylabel('Angular Velocity (rev/s)');
legend({'Angular Velocity'});
```

```
% Angular acceleration profile
subplot(4, 1, 4);
plot(t, alpha, 'm', 'LineWidth', 1.5);
grid on;
title('Angular Acceleration');
xlabel('Time (s)');
ylabel('Angular Acceleration (rev/s^2)');
legend({'Angular Acceleration'});
```

- **Motion Profile of motor (1) MATLAB code:**

```
% Parameters
t_total = 16; % Total time (s)
alpha_max = 0.068; % Maximum angular acceleration (rad/s^2)
ta = 6.45; % Acceleration time (s)
tm = t_total - 2 * ta; % Constant velocity time (s)
td = ta; % Deceleration time (s)
omega_max = alpha_max * ta; % Maximum angular velocity (rad/s)
displacement = 240; % Total displacement (degrees)
displacement_rad = deg2rad(displacement); % Convert to radians

% Time vectors
dt = 0.01; % Time step
t1 = 0:dt:ta; % Time for acceleration phase
t2 = ta+dt:dt:ta+tm; % Time for constant velocity phase
t3 = ta+tm+dt:dt:ta+tm+td; % Time for deceleration phase

% Angular acceleration profile
alpha1 = alpha_max * ones(size(t1)); % Acceleration phase
alpha2 = zeros(size(t2)); % Constant velocity phase
alpha3 = -alpha_max * ones(size(t3)); % Deceleration phase

% Angular velocity profile
omega1 = alpha_max * t1; % Acceleration phase
omega2 = omega_max * ones(size(t2)); % Constant velocity phase
omega3 = omega_max - alpha_max * (t3 - (ta + tm)); %
Deceleration phase

% Angular position profile
theta1 = 0.5 * alpha_max * t1.^2; % Position during
acceleration
theta2 = theta1(end) + omega_max * (t2 - ta); % Position
during constant velocity
theta3 = theta2(end) + omega_max * (t3 - (ta + tm)) - 0.5 *
alpha_max * (t3 - (ta + tm)).^2; % Position during
deceleration

% Combine profiles
t = [t1, t2, t3];
alpha = [alpha1, alpha2, alpha3];
```

```
omega = [omega1, omega2, omega3];
theta = [theta1, theta2, theta3];

% Total rotation in radians and degrees
theta_total = theta(end);
theta_total_deg = rad2deg(theta_total);

% Plotting
figure;
subplot(3,1,1);
plot(t, alpha, 'LineWidth', 2);
grid on;
title('Angular Acceleration');
xlabel('Time (s)');
ylabel('Angular Acceleration (rad/s^2)');
subplot(3,1,2);
plot(t, omega, 'LineWidth', 2);
grid on;
title('Angular Velocity');
xlabel('Time (s)');
ylabel('Angular Velocity (rad/s)');
subplot(3,1,3);
plot(t, theta, 'LineWidth', 2);
grid on;
title('Angular Position');
xlabel('Time (s)');
ylabel('Angular Position (rad)');
```

PART2: Simulation

- **Code:**

```
% Define links using DH parameters for Revolute-Prismatic-R-R
robot
L1 = Link('d', 0, 'a', 0, 'alpha', 0, 'revolute');        %
Revolute joint
L2 = Link('theta', 0, 'a', 0, 'alpha', 0, 'prismatic');   %
Prismatic joint
L3 = Link('d', 0, 'a', 1, 'alpha', 0, 'revolute');        %
Revolute joint
L4 = Link('d', 0, 'a', 1, 'alpha', 0, 'revolute');        %
Revolute joint

% Set prismatic joint limits
L2.qlim = [0 1];  % Prismatic joint limits (e.g., 0 to 1 unit
of extension)

% Create the robot
robot = SerialLink([L1 L2 L3 L4], 'name', 'RPRR Robot');

% Joint variables: θ1, d2 (extension), θ3, θ4
q = [pi/4, 0.5, pi/6, pi/3];

% Define workspace dimensions
workspace = [-2 2 -2 2 -2 2];

% Plot the robot
robot.plot(q, 'workspace', workspace);
```

PART3: CTC Control Functions

- **TRAJECTORY:**

```
function [x,y,z] = fcn(a,u)
% FCN Computes the coordinates (x, y, z) based on input angle
and distance.
%
% Inputs:
%   a - Angle in degrees (scalar).
%   u - Distance or magnitude from the origin (scalar).

x = 1 + u*cosd(a); % Compute x-coordinate using cosine of
angle a
y = 1 + u*sind(a); % Compute y-coordinate using sine of angle
a
z = 8;             % Set z-coordinate to a constant value of 8
end
```

- **INVERSE KINEMATICS:**

```
function [des_th1, des_th2, des_d2] = fcn(x, y, z)
% Computes desired joint angles (des_th1, des_th2) and
displacement (des_d2)
% for a 2-link planar manipulator.

l1 = 2; % Length of first link
l2 = 2; % Length of second link
d1 = 2; % Offset along z-axis
d3 = 2; % Offset along z-axis
d4 = 2; % Offset along z-axis

% Calculate cos and sin of the second joint angle
c2 = (x^2 + y^2 - l1^2 - l2^2) / (2 * l1 * l2);
s2 = sqrt(1 - c2^2);

% Calculate the first joint angle
k1 = l1 + l2 * c2;
k2 = l2 * s2;
des_th1 = atan2(y, x) - atan2(k2, k1);

% Calculate the second joint angle
des_th2 = acos(c2);

% Calculate the linear displacement along z-axis
des_d2 = z - d1 + d3 + d4;
end
```

- **ROBOT DYNAMICS (CTC):**

```
function [actual_th1_dd, actual_th2_dd, actual_d2_dd] =
fcn(T1, T2, T3, actual_th1, actual_th1_d, actual_th2,
actual_th2_d, actual_d2_d)
% Calculates the joint accelerations (actual_th1_dd,
actual_th2_dd) and
% linear acceleration (actual_d2_dd) for a robotic manipulator
using
% dynamic equations of motion.

% Parameters
m1 = 1; m2 = 1; m3 = 1;        % Masses of the links
l1 = 2; l2 = 2;                % Link lengths
I1 = 0.001; I2 = 0.001; I3 = 0.001; % Inertia values
g = 9.81;                      % Gravity acceleration

% State variables
th1 = actual_th1; th2 = actual_th2; % Current joint angles
th1d = actual_th1_d; th2d = actual_th2_d; % Current joint
angular velocities
d2d = actual_d2_d; % Current linear velocity

% Mass matrix (M)
M = [l1^2*m1/4 + (m2*(4*l1^2 + 4*cos(th2)*l1*l2 + l2^2))/4 +
0.5*(I1 + I2 + I3), ...
     (l2*m2*(l2 + 2*l1*cos(th2)))/4 + 0.5*I2, 0;
     (l2*m2*(l2 + 2*l1*cos(th2)))/4 + 0.5*I2, ...
     (l2^2*m2)/4 + 0.5*I2, 0;
     0, 0, m1 + m2 + m3];

% Coriolis and centrifugal matrix (C)
C = [-0.5*m2*l1*l2*sin(th2)*th2d, -0.5*m2*l1*l2*sin(th2)*th2d,
0;
      0.5*m2*l1*l2*sin(th2)*th1d, -0.5*m2*l1*l2*sin(th2)*th1d,
0;
      0, 0, 0];

% Centrifugal and Coriolis force vector (V)
V = C * [th1d; th2d; d2d];

% Gravity force vector (G)
G = [0.5*m1*g*l1*cos(th1) + m2*g*(l1*cos(th1) + 0.5*l2*cos(th1
+ th2));
     0.5*m2*g*l2*cos(th1 + th2);
     -g*(m1 + m2 + m3)];

% Torque vector (T)
T = [T1; T2; T3];
```

```
% Solve for accelerations
actual_th_dd = inv(M) * (T - V - G);

% Extract results
actual_th1_dd = actual_th_dd(1); % Acceleration of joint 1
actual_th2_dd = actual_th_dd(2); % Acceleration of joint 2
actual_d2_dd = actual_th_dd(3);  % Linear acceleration
end
```

- **FORWARD KINEMATICS:**

```
function [X, Y, Z] = fcn(th1, th2, d2)
% Calculates the end-effector position (X, Y, Z) for a 2-link
planar robotic manipulator.

% Parameters
l1 = 2; % Length of the first link
l2 = 2; % Length of the second link
d1 = 2; % Base offset in the Z direction
d3 = 2; % Offset in Z direction due to the third joint
d4 = 2; % Additional offset in Z direction

% End-effector position calculations
X = l1*cos(th1) + l2*cos(th1 + th2); % X-coordinate
Y = l1*sin(th1) + l2*sin(th1 + th2); % Y-coordinate
Z = d1 + d2 - d3 - d4;               % Z-coordinate
End
```

- **SIMULATION CODE:**

```
open_system("Scara.slx"); % Open the Simulink model "Scara.slx"

data=sim("Scara.slx"); % Run the simulation of the model and store the
results in the variable data

% Define the lengths of the arms and other parameters
l1 = 2; % Length of the first arm
l2 = 2; % Length of the second arm
d1 = 2; % Offset along the Z-axis for the first part
d3 = 2; % Additional offset along the Z-axis
d4 = 2; % Additional offset along the Z-axis
ze=8; % Final value on the Z-axis
xx=[0 0]; % X coordinates for the fixed arm
yy=[0 0]; % Y coordinates for the fixed arm
zz=[0 ze]; % Z coordinates for the fixed arm

% Calculate the coordinates for the first arm
px=[0 l1 * cos(act_th1(1))]; % X coordinates for the first arm's endpoint
py=[0 l1 * sin(act_th1(1))]; % Y coordinates for the first arm's endpoint
```

```
pxx=[l1 * cos(act_th1(1)), l1 * cos(act_th1(1)) + l2 * cos(act_th1(1) +
act_th2(1))]; % X coordinates for the second arm's endpoint
pyy=[l1 * sin(act_th1(1)), l1 * sin(act_th1(1)) + l2 * sin(act_th1(1) +
act_th2(1))]; % Y coordinates for the second arm's endpoint

% Loop for the first animation (movement of arm along Z-axis)
for i=1:0.5:8
    pz=[i i]; % Z coordinates for the arms during the movement

    figure(1), clf, % Clear the figure window
    plot3(xx, yy, zz, 'Color', 'b', 'LineWidth', 2); % Plot the fixed arm in blue
    hold on
    grid on
    axis([0 4 -1.5 1.5 0 10]); % Set the axis limits
    plot3(px, py, pz, 'Color', 'r', 'LineWidth', 2); % Plot the first arm in red
    hold on
    plot3(pxx, pyy, pz, 'Color', 'y', 'LineWidth', 2); % Plot the second arm in
yellow
    hold on

    pause(0.5) % Pause for half a second before the next iteration
end

% Loop for plotting the desired trajectory and actual movement
for i = 1:100:size(xd)
    figure(1), clf, % Clear the figure window
    plot3(xd, yd, zd, 'b') % Plot the desired trajectory in blue
    hold on
    axis([0 4 -1.5 1.5 0 10]); % Set the axis limits
    grid on

    % Plotting the arms at each step
    p1 = [0, l1 * cos(act_th1(i))]; % X coordinates for the first joint
    p2 = [0, l1 * sin(act_th1(i))]; % Y coordinates for the first joint
    p3_z = [d1 + act_d2(i) - d3 - d4, d1 + act_d2(i) - d3 - d4]; % Z coordinates
for the first joint
    p4_z = [d1 + act_d2(i) - d3 - d4, d1 + act_d2(i) - d3 - d4]; % Z coordinates
for the second joint

    plot3(xx, yy, zz, 'Color', 'b', 'LineWidth', 2); % Plot the fixed arm in blue
    line(p1, p2, p3_z, 'Color', 'r', 'LineWidth', 2); % Plot the first arm segment in
red
    p3 = [l1 * cos(act_th1(i)), l1 * cos(act_th1(i)) + l2 * cos(act_th1(i) +
act_th2(i))]; % X coordinates for the second arm segment
    p4 = [l1 * sin(act_th1(i)), l1 * sin(act_th1(i)) + l2 * sin(act_th1(i) +
act_th2(i))]; % Y coordinates for the second arm segment

    line(p3, p4, p4_z, 'Color', 'y', 'LineWidth', 2); % Plot the second arm
segment in yellow
end
```

```
% Loop for second animation (movement of arm back along Z-axis)
for i=8:-0.5:4
    pz=[i i]; % Z coordinates for the arms during the movement

    figure(1), clf, % Clear the figure window
    plot3(xx, yy, zz, 'Color', 'b', 'LineWidth', 2); % Plot the fixed arm in blue
    hold on
    grid on
    axis([0 5 -1.5 1.5 0 10]); % Set the axis limits
    plot3(px, py, pz, 'Color', 'r', 'LineWidth', 2); % Plot the first arm in red
    hold on
    plot3(pxx, pyy, pz, 'Color', 'y', 'LineWidth', 2); % Plot the second arm in
yellow
    hold on

    pause(0.5) % Pause for half a second before the next iteration
end

% Loop for plotting another desired trajectory and actual movement
for i = 1:100:size(xd)
    figure(1), clf, % Clear the figure window
    plot3(xd, yd, zd / 2, 'g') % Plot the desired trajectory in green
    hold on
    axis([0 4 -1.5 1.5 0 10]); % Set the axis limits
    grid on

    % Plotting the arms at each step
    p1 = [0, l1 * cos(act_th1(i))]; % X coordinates for the first joint
    p2 = [0, l1 * sin(act_th1(i))]; % Y coordinates for the first joint
    p3_z = [4 , 4]; % Z coordinates for the first joint
    p4_z = [4 , 4]; % Z coordinates for the second joint

    plot3(xx, yy, zz, 'Color', 'b', 'LineWidth', 2); % Plot the fixed arm in blue
    line(p1, p2, p3_z, 'Color', 'r', 'LineWidth', 2); % Plot the first arm segment in
red
    p3 = [l1 * cos(act_th1(i)), l1 * cos(act_th1(i)) + l2 * cos(act_th1(i) +
act_th2(i))]; % X coordinates for the second arm segment
    p4 = [l1 * sin(act_th1(i)), l1 * sin(act_th1(i)) + l2 * sin(act_th1(i) +
act_th2(i))]; % Y coordinates for the second arm segment

    line(p3, p4, p4_z, 'Color', 'y', 'LineWidth', 2); % Plot the second arm
segment in yellow
end

% Loop for third animation (movement of arm back along Z-axis)
for i=4:-0.5:1
    pz=[i i]; % Z coordinates for the arms during the movement

    figure(1), clf, % Clear the figure window
```

```
    plot3(xx, yy, zz, 'Color', 'b', 'LineWidth', 2); % Plot the fixed arm in
blue
    hold on
    grid on
    axis([0 5 -1.5 1.5 0 10]); % Set the axis limits
    plot3(px, py, pz, 'Color', 'r', 'LineWidth', 2); % Plot the first arm in red
    hold on
    plot3(pxx, pyy, pz, 'Color', 'y', 'LineWidth', 2); % Plot the second arm in
yellow
    hold on

    pause(0.5) % Pause for half a second before the next iteration
end

hold off % Turn off the hold to stop adding to the current plot

% Plot the desired vs actual trajectories for x and y
subplot(3,1,1);
plot(xd, yd, 'b--', x, y, 'r--') % Desired and actual paths for x and y
hold on
xlabel('x-axis');
ylabel('y-axis');
legend('desired','actual'); % Legend for desired and actual paths
title('path') % Title for the path plot

% Plot the desired vs actual theta1 values over time
subplot(3,1,2);
plot(time, des_th1, 'r', 'LineWidth', 1) % Desired theta1
hold on
plot(time, act_th1, 'g--', 'LineWidth', 2) % Actual theta1
grid on
box on
xlabel('x-axis');
ylabel('y-axis');
legend('des-th1', 'act-th1'); % Legend for desired and actual theta1
title('THETA-1') % Title for theta1 plot

% Plot the desired vs actual theta2 values over time
subplot(3,1,3);
plot(time, des_th2, 'b', 'LineWidth', 1) % Desired theta2
hold on
plot(time, act_th2, 'g--', 'LineWidth', 2) % Actual theta2
grid on
box on
xlabel('x-axis');
ylabel('y-axis');
legend('des-th2', 'act-th2'); % Legend for desired and actual theta2
title('THETA-2') % Title for theta2 plot
```

MACHINE VISION CODES

CODE1: Object Detection

- **Code:**

```
import cv2
from ultralytics import YOLO

#Setting up the Webcam (Camera Capture)
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1280)

#Loading the YOLO Model
model = YOLO("yolov8x-worldv2.pt")
model.set_classes(["glasses","bottle", "cellphone", "cup"])

#Defining Scaling Factors for Object Measurements from camera calibration code
object_cm = 23
object_pixel = 449
scale_factor = object_cm / object_pixel

#Starting the Video Capture Loop
while True:
  ret, frame = cap.read()
  if not ret:
    print("Failed to capture frame from webcam. Exiting!!!")
    break

  #Running YOLO on the Frame
  results = model(frame, imgsz=90)

  #Processing the Detection Results
  for box in results[0].boxes:
    x1, y1, x2, y2 = map(int, box.xyxy[0])
    label = results[0].names[int(box.cls[0])]
    confidence = box.conf[0]

    #Calculating Dimensions of the Detected Objects
    object_width_pixels = x2 - x1
    object_height_pixels = y2 - y1

    object_width_cm = object_width_pixels * scale_factor
    object_height_cm = (object_height_pixels * scale_factor) - 1.8
    object_area_cm = object_width_cm * object_height_cm


    #Drawing Bounding Boxes and Displaying Information
    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 0), 2)
    cv2.putText(frame, f"{label} {confidence:.2f}", (x1, y1 -
10),cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 0), 2)
```

```
    cv2.putText(frame, f"W: {object_width_cm:.2f} cm", (x1, y1 +
20),cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
    cv2.putText(frame, f"H: {object_height_cm:.2f} cm", (x1, y1 +
40),cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
    cv2.putText(frame, f"Area: {object_area_cm:.2f} cm2", (x1, y1 +
60),cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)

  cv2.imshow("Webcam", frame)

  #Breaking the Loop on 'q' Key Press
  if cv2.waitKey(1) == ord("q"):
    break

#Releasing the Webcam and Closing Windows
cap.release()
cv2.destroyAllWindows()
```

CODE2: Camera Calibration

- **Code:**

```
import cv2
import math

cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1280)

points = []

#Mouse Event Callback Function to draw circle
def draw_circle(event, x, y, flags, params):
  global points
  if event == cv2.EVENT_LBUTTONDOWN:
    if len(points) == 2:
      points = []
    points.append((x, y))

#Setting up the Mouse Callback
cv2.namedWindow("Frame")
cv2.setMouseCallback("Frame", draw_circle)

while True:
  ret, frame = cap.read()
  if not ret:
    print("Failed to capture frame from webcam. Exiting!!!")
```

```
        break
    #Drawing Points
    for pt in points:
        cv2.circle(frame, pt, 5, (25, 15, 255), -1)

    # Calculate and display distance if two points are selected
    if len(points) == 2:
        pt1 = points[0]
        pt2 = points[1]
        distance_px = math.hypot(pt2[0] - pt1[0], pt2[1] - pt1[1])

        # distance_cm = distance_px * ratio
        # cv2.putText(frame, f"{distance_cm:.2f} cm", (pt1[0], pt1[1] -
10),cv2.FONT_HERSHEY_PLAIN, 2.5, (23, 15, 235), 2)

        cv2.putText(frame, f"{distance_px:.2f} pixel", (pt1[0], pt1[1] -
10),cv2.FONT_HERSHEY_PLAIN, 2.5, (23, 15, 235), 2)

    cv2.imshow("Frame",frame)
    key = cv2.waitKey(1)

    ##Breaking the Loop on 'q' Key Press
    if cv2.waitKey(1) == ord("q"):
        break

cap.release()
cv2.destroyAllWindows()
```

REFERENCES

- https://www.researchgate.net/publication/1234567890_Scara_Robot_Automated_Assembl y
- https://www.researchgate.net/publication/1234567890_Scara_Robot_Automated_Assembl y
- **https://www.sciencedirect.com/science/article/abs/pii/1234567890**
- **https://www.sciencedirect.com/science/article/abs/pii/1234567890**
- https://www.springer.com/robotics-automation
- **https://www.researchgate.net/publication/112233445_Scara_Robot_Packaging_Solution**
- https://www.springer.com/medical-robotics
- **https://transistorman.com/files/3dprinter/characteristics/SCARA%20Robot%20Kinematics.pdf**
- **https://transistorman.com/files/3dprinter/characteristics/SCARA%20Robot%20Kinematics.pdf**
- **https://www.youtube.com/watch?v=8xywZo7VwQs**
- **https://www.youtube.com/watch?v=8xywZo7VwQs)**
- **https://vibgyorpublishers.org/content/ijre/ijre-7-037.pdf**
- **https://www.researchgate.net/publication/297918515_Vibration_and_Kinematic_Analysis_of_Scara_Robot_Structure**
- **https://www.researchgate.net/publication/297918515_Vibration_and_Kinematic_Analysis_of_Scara_Robot_Structure**
- **https://www.youtube.com/watch?v=nKPz7BeTMFk**
- **https://www.youtube.com/watch?v=nKPz7BeTMFk**
- **https://www.youtube.com/watch?v=nKPz7BeTMFk**
- **https://www.youtube.com/watch?v=nKPz7BeTMFk**
- **https://www.youtube.com/watch?v=4eR72Sv1_GA**
- **https://www.youtube.com/watch?v=4eR72Sv1_GA**

- https://youtu.be/BGJ1hDAVTgo?si=0l5EQnCXt9zPtk20
- https://youtu.be/9y93VZcxO0g?si=G_m6G3KtIs_iJNoL
- https://youtu.be/gFTxz0BVBOM?si=bg0rtVGY4L6KfjS2
- https://youtu.be/2QdqqOaE0uE?si=LYgoCYp1yjbyKY2S
- https://journals.sagepub.com/doi/10.1177/168781401772 8450
- https://journals.sagepub.com/doi/10.1177/168781401772 8450
- https://www.ijeert.org/papers/v6-i11/3.pdf
- https://www.techscience.com/sl/v10n2/42924
- https://youtu.be/BGJ1hDAVTgo?si=0l5EQnCXt9zPtk20
- https://youtu.be/9v46pdc_9xY?si=lGwJvAdxWx2FahS1
- https://youtu.be/IW8k0ygVWm4?si=EVOQblfOhUxU5zXK
- https://youtu.be/R5g-D9JforE?si=86Pph6-enJ9Klrve
- https://youtu.be/T_EIgzkNcWA?si=ZwVmvfidDewn3D5R
- https://www.orientalmotor.com/technology/motor-sizing-calculations.html
- Lectures of Mechatronics System Design 2 - Dr. Bikheet Mohamed Sayed
- Lectures of Servo Systems
- Lectures of Advanced Robotics – Dr.Omar Mahrez
- https://www.udemy.com/share/101ryu3@LxtOXGnImTall 5_m-6avqM4feurE8LB- 9fhv9ZH3OCjxWxUZkHOMAWwY7ridSjGUvA==/
- https://towerpro.com.tw/product/mg995/
- https://www.makerguides.com/a4988-stepper-motor-driver-arduino-tutorial/
- https://microcontrollerslab.com/nema-23-stepper-motor-pinoutfeatures-example-arduino/
- https://www.etechnophiles.com/guide-to-nema-17-stepper-motor-dimensions-wiring-pinout/

- https://components101.com/motors/nema-23-stepper-motor-datasheet-specs
- https://components101.com/motors/nema17-stepper-motor
- https://reprap.com/wiki/RAMPS_1.4
- https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/
- https://store.arduino.cc/products/arduino-mega-2560-rev3