



ABSTRACT



Abstract

This project aims to design, control and implement a four degree of freedom (4DOF) modified SCARA Robot to sort objects by their appearance with taking an advantage of machine vision and embedded systems technologies.

The first stage of the project after reviewing the previous works on SCARA robot and its applications is to design a modified one to be suitable for our project. Then the mechanical structure of the robot will be modeled.

The machine vision will determine the pose of each object with respect to the manipulator base frame. Based on these locations the path planner will generate the trajectory that the manipulator has to track.

The Robot is expected to sort objects based on their appearance with precise, accurate repeatable picking and placing missions.

Keywords: SCARA, machine vision, embedded systems, image processing, robotics control.

1



INTRODUCTION



1.1 Concept of the project



Figure 1: SCARA robot arm

Selective Compliance Assembly Robot Arm or SCARA robot was invented by Hiroshi Makino in 1978 is shown in Figure 1.1 Its arm is rigid in the Z-axis and pliable in the XY-axes, which allowed it to adapt to holes in the XY-axes. Because of its speed and simple design, this was and continues to be a groundbreaking robot in many industries.

There is a demand for such a robot in many industrial applications that need quick picking and placing operations since human employees cannot accomplish these tedious tasks.

1.2 The need recognition:

Some operations require sorting things by color, shape, or both faster than human efforts, necessitating the development of a high speed and precise robotic arm to carry out such missions.

Some of these sorting missions will be done on the horizontal plane, so that it is reasonable to construct a simple robotic structure to do these missions hence there is no need here to construct a complex robotic arm like German Aerospace Center (DLR) manipulator or anthropomorphic robotic arm, SCARA robotic structure is good choice in this case.

1.3 Project scope

This project aims to design, control and implement a 4DOF modified SCARA Robot to collect objects (fruits to be exported) which are on a horizontal plane and sort them into baskets by their colors and shapes with taking an advantage of image processing with camera vision.

The first stage of the project after determining the project limitations and assumptions is to design the manipulator. Then the mechanical structure of the manipulator will be modeled for the sake of controller design and simulation, more than one control algorithm will be used to control the manipulator to choose the most fitting controller for such a mission.

The first stage of the project after reviewing the previous works on SCARA robot and its applications is to design a modified one to be suitable for our project. Then the mechanical structure of the robot will be modeled.

The machine vision system will determine the pose of each object with respect to a reference frame, based on these poses the path planner will generate the trajectory that the manipulator must track. The Robot is expected to sort objects based on their appearance with precise, accurate repeatable picking and placing missions.

1.4 Project objectives

- ✓ To decrease the structural inertias by optimizing the mechanical structure of the robot to reduce actuator energy consumption by positioning the motors as near to the robot's base as achievable.
- ✓ To simplify the poses sensing operation by taking advantage of computer vision system.
- ✓ To build an intelligent automated sorting system.
- ✓ To use embedded systems technology to build an accurate robot control system.

1.5 Methodology

The methodology to make this project contains several stages, theoretical calculations, which contains the dynamic model and design the mechanical structure for the robot.

After the theoretical calculations, the mechanical structure will be modeled in SOLIDWORKS software to check the maximum deflection that occurs due to the maximum static and dynamic forces.

After that, MATLAB software will calculate the forward & inverse kinematics using a simple GUI.

After that, several conceptual designs will be considered to show the connections between several components of the project (computer, interfacing circuits, controller, drivers, and motors) and then choose the best design.

2



MECHANICAL DESIGN



2.1. Main design

The robot Arm consist of a lot of mechanical components like bearing, gears timing belt and lead screw. In the figures below it's obvious all the mechanical components in the project.

1. Component of Base
2. Revolute Joint 1
3. Prismatic Joint
4. Platform of Arm 1
5. Prismatic Top
6. Arm 1
7. Arm 1 Component
8. Arm 2 & Revolute joint 2
9. Revolute Joint 3
10. Gripper

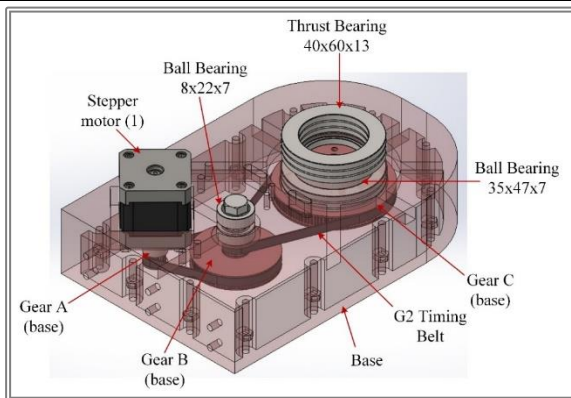


Figure 2.1: Step (1) Component of Base

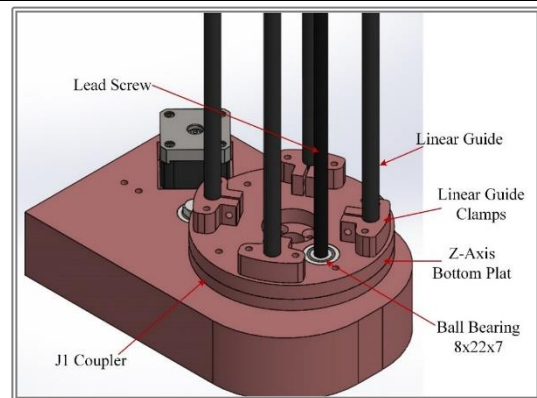


Figure 2.2: Step (2) Revolute Joint 1

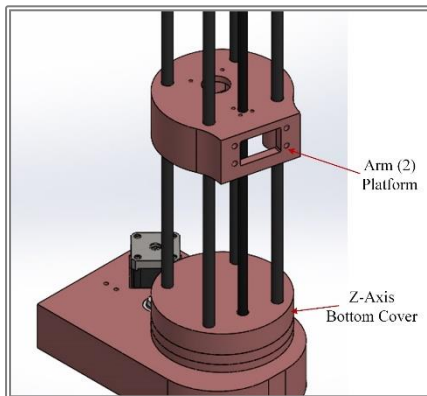


Figure 2.3: Step (3) Prismatic Joint

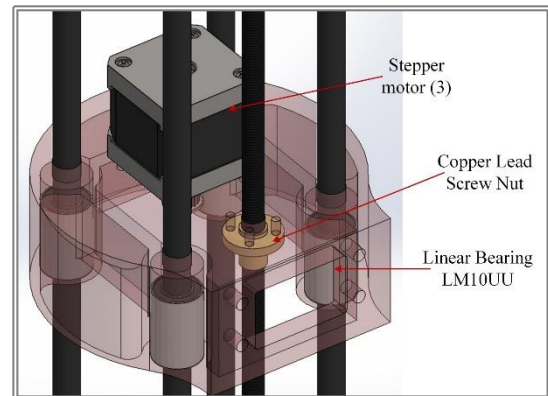


Figure 2.4: Step (4) Platform of Arm 1

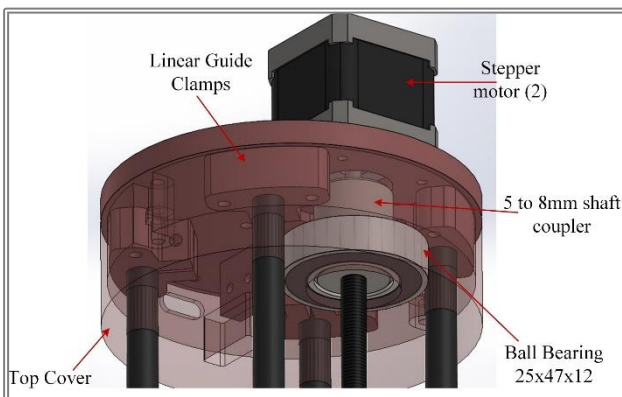


Figure 2.5: Step (5) Prismatic Top

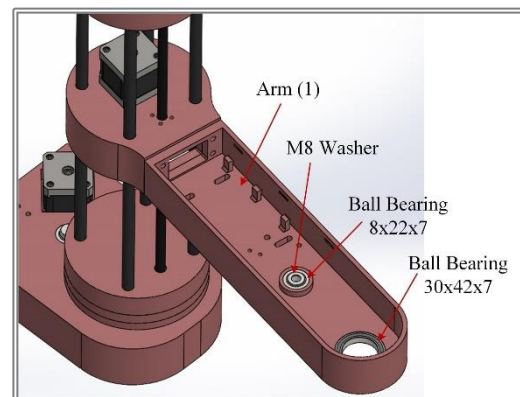


Figure 2.6: Step (6) Arm 1

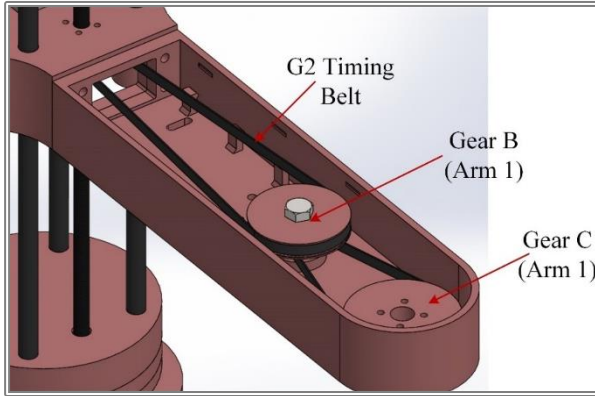


Figure 2.7: Step (7) Arm 1 Component

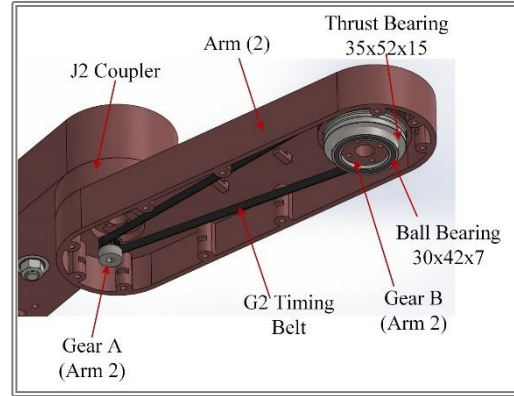


Figure 2.8: Step (8) Arm 2 & Revolute joint 2

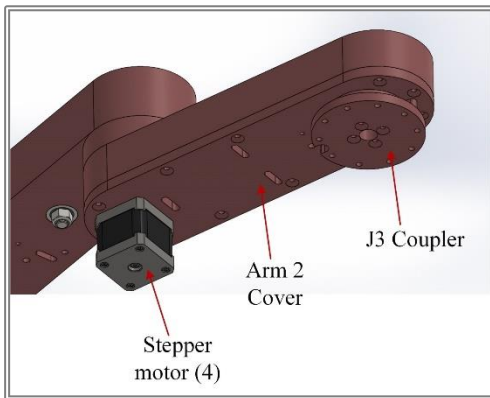


Figure 2.9: Step (9) Revolute Joint 3

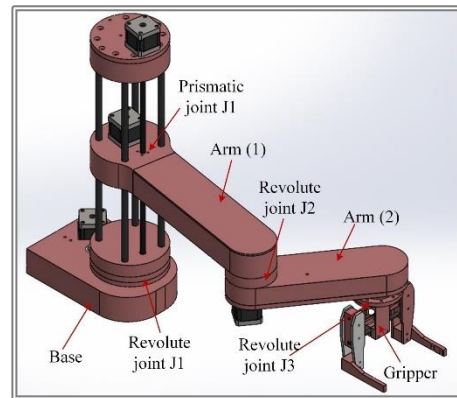


Figure 2.10: Step (10) Gripper

2.2. Constrains

The project has been designed on some constrains which are:

- ✓ Pay load = 1.5 kg
- ✓ Width of product = 100mm
- ✓ Workspace radius = 40 cm
- ✓ Accelerations
 - $\alpha_{j1} = 0.08 \text{ cm/s}^2$
 - $\alpha_{prismatic} = 0.0065 \text{ cm/s}^2$
 - $\alpha_{j2} = 0.47 \text{ cm/s}^2$
 - $\alpha_{j3} = 4.9 \text{ cm/s}^2$

2.3. Gear ratios

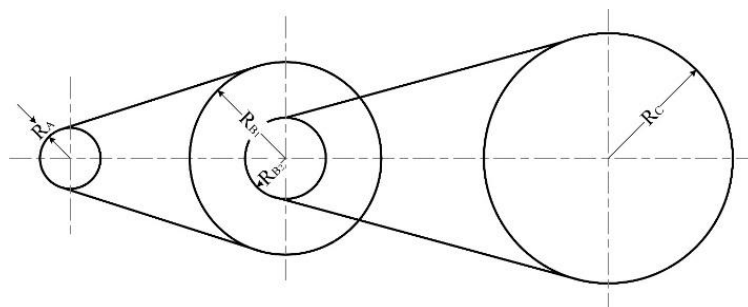


Figure 2.14: Gear Mechanisms Design

$$n_1 = \frac{R_{B_1}}{R_A} = \frac{\omega_A}{\omega_{B_1}}$$

$$n_2 = \frac{R_C}{R_{B_2}} = \frac{\omega_{B_2}}{\omega_C}$$

$$\omega_{B_1} = \omega_{B_2}$$

$$\omega_C = \frac{\omega_{B_2}}{n_2} = \frac{\omega_A}{n_1 n_2}$$

$$\theta_C = \frac{\theta_A}{n_1 n_2}$$

$$\frac{\omega_C}{\omega_A} = \frac{1}{n_1 n_2}$$

$$P = \tau \cdot \omega$$

$$\frac{\tau_A}{\tau_C} = \frac{\omega_C}{\omega_A}$$

$$\frac{\tau_A}{\tau_C} = \frac{1}{n_1 n_2}$$

$$\theta_{A(motor)} = \theta_{C(joint)} n_1 n_2$$

$$\tau_{A(motor)} = \frac{\tau_{C(joint)}}{n_1 n_2}$$

2.3.1. Base Gear Mechanism

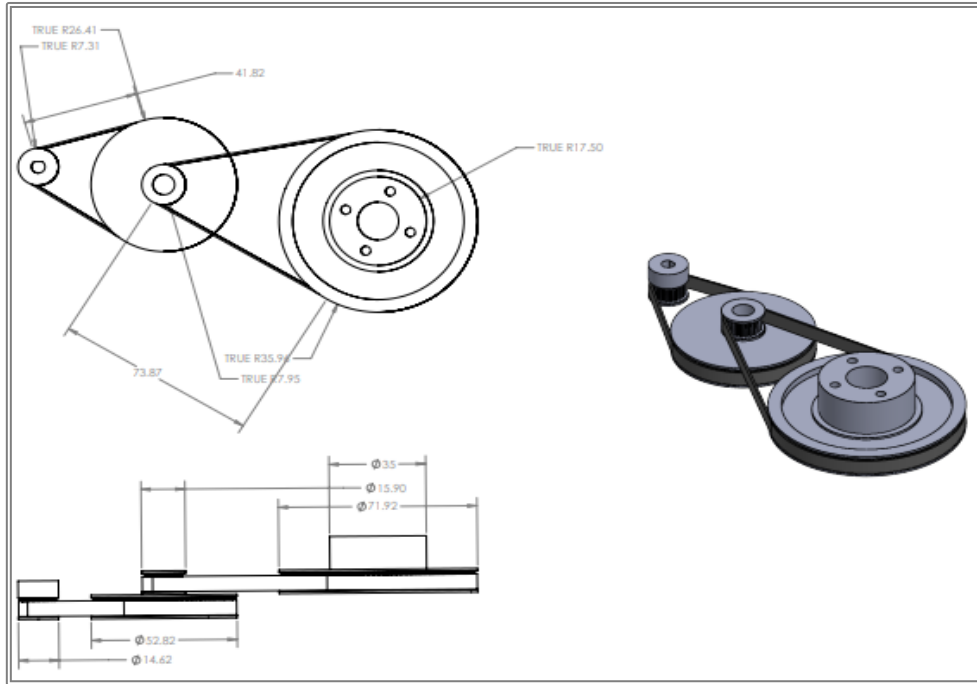


Figure 2.15: Base Gear Mechanisms Drawing

$$R_A = 7.31mm$$

$$R_{B_2} = 7.95mm$$

$$R_{B_1} = 26.41mm$$

$$R_C = 35.96mm$$

$$n_1 = \frac{26.41}{7.31} = 3.613$$

$$n_2 = \frac{35.96}{7.95} = 4.523$$

$$\theta_{A(motor)} = \theta_{C(joint)} \times 3.613 \times 4.523$$

$$\theta_{A(motor)} = 16.342 \times \theta_{C(joint)}$$

$$\tau_{A(motor)} = \frac{\tau_{C(joint)}}{16.342}$$

2.3.2. Arm (1) Gear Mechanism

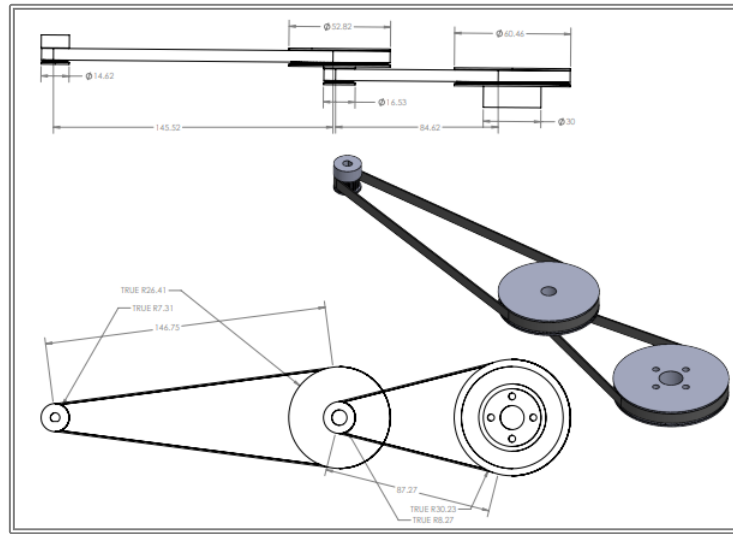


Figure 2.20: Arm (1) Gear Mechanisms Drawing

$$R_A = 7.31mm$$

$$R_{B_2} = 8.265mm$$

$$R_{B_1} = 26.41mm$$

$$R_C = 30.23mm$$

$$n_1 = \frac{26.41}{7.31} = 3.613$$

$$n_2 = \frac{30.23}{8.265} = 3.658$$

$$\theta_{A(motor)} = \theta_{C(Joint)} \times 3.613 \times 3.658$$

$$\theta_{A(motor)} = 13.215 \times \theta_{C(Joint)}$$

$$\tau_{A(motor)} = \frac{\tau_{C(joint)}}{13.215}$$

2.3.3. Arm (2) Gear Mechanism

$$R_A = 7.31mm$$

$$R_C = 29.595mm$$

$$n = \frac{29.595}{7.31} = 4.049$$

$$\theta_{A(motor)} = 4.049 \times \theta_{C(Joint)}$$

$$\tau_{A(motor)} = \frac{\tau_{C(joint)}}{4.049}$$

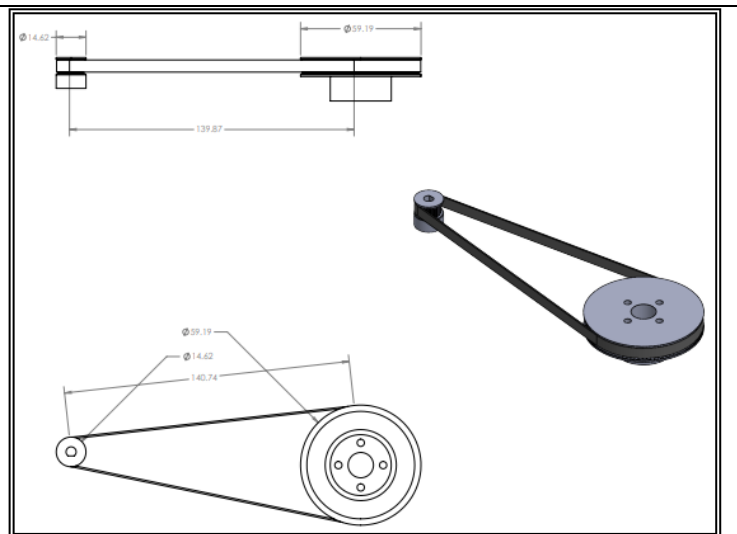


Figure 2.24: Arm (2) Gear Mechanisms Drawing

2.4. Gripper design

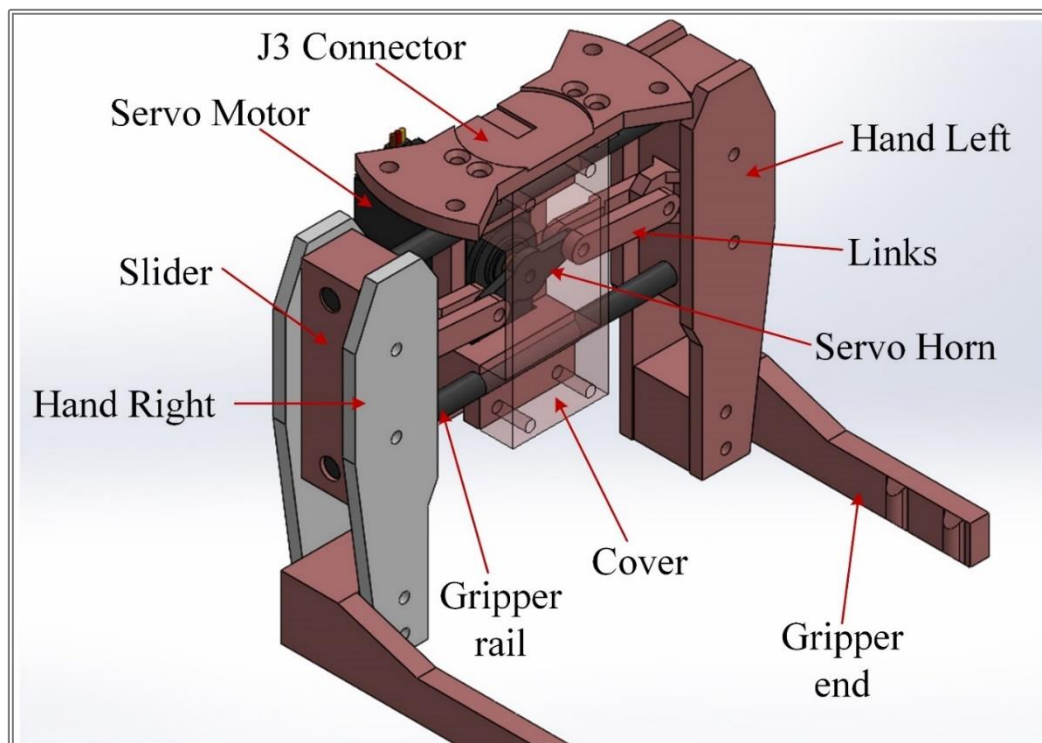


Figure 2.25: Gripper Construction and components

The gripper mechanism is the end effector of the robot arm, it's used to hold the product to move it from its position to another. The actuator of the gripper is servo motor with torque 11 kg.cm. The servo connected with links by revolute joint and links connected with sliders. 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.

2.5. Mass Moment of inertia

The moment of inertia of a rigid body is a quantity that determines the torque needed for a desired angular acceleration about a rotational axis, it's one of the main steps in mechanical design as it enables the designer to calculate the torque loaded on every motor and select the best motor which can provide the required torque and acceleration with lowest cost and best specifications.

In this design there are 4 stepper motors for 4 degree of freedom robot arm

1. Motor (1) moves the total robot
2. Motor (2) moves the prismatic joint
3. Motor (3) moves Arm (3)
4. Motor (4) Rotates the gripper

2.5.1. Motor (1)

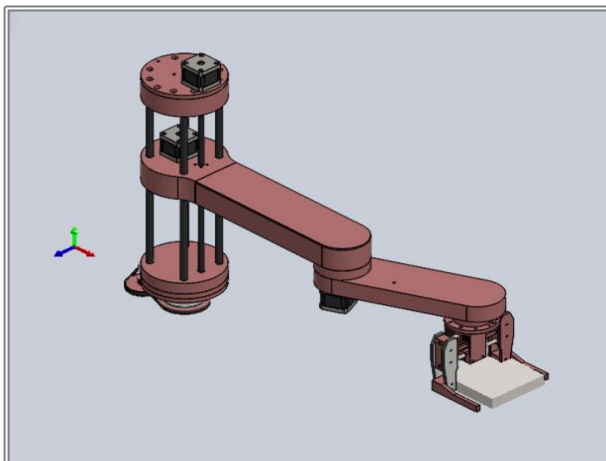


Figure 2.26: Mechanical Parts Loaded on motor (1)

Mass properties of inertia motor 1.png	
Configuration: Default	
Coordinate system: -- default --	
* Includes the mass properties of one or more hidden components/bodies.	
Mass = 6.41 kilograms	
Volume = 2539.15 cubic centimeters	
Surface area = 9957.84 square centimeters	
Center of mass: (centimeters)	
X = 47.53	
Y = 21.37	
Z = 0.50	
Principal axes of inertia and principal moments of inertia: (kilograms * square centimeters)	
Taken at the center of mass.	
Ix = (0.94, -0.31, -0.14)	Px = 873.83
Iy = (0.31, 0.95, 0.01)	Py = 3838.29
Iz = (0.13, -0.05, 0.99)	Pz = 4581.15
Moments of inertia: (kilograms * square centimeters)	
Taken at the center of mass and aligned with the output coordinate system.	
Lxx = 1225.19	Lyy = -857.23
Lyx = -857.23	Lyy = 3557.13
Lzx = -477.95	Lzy = 163.61
	Lzz = 4510.94

Figure 2.27: Mass moment of inertia on motor (1)

$$L_{yy} = 3557.13 \text{ kg.cm}^2$$

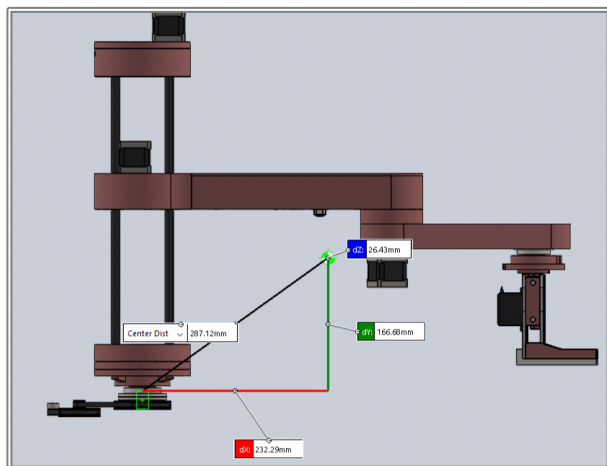


Figure 2.28: Center of mass motor (1) load

$$I = L_{yy} + M R^2$$

$$I = 3557.1 + 6.41 \times 23.23^2$$

$$I = 7016.2 \text{ Kg.cm}^2$$

$$\tau_{A(\text{motor})} = \frac{\tau_{C(\text{joint})}}{n_1 n_2} = \frac{I \alpha}{n_1 n_2}$$

$$\tau_{A(\text{motor})} = \frac{7016.2 \times 0.08}{16.342}$$

$$\tau = 34.35 \text{ N.cm}$$

2.5.2. Motor (2)

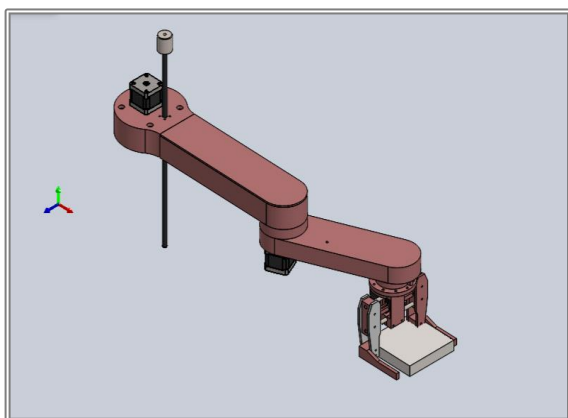


Figure 2.29: Mechanical Parts Loaded on motor (2)

Mass properties of inertia motor 2	
Configuration: Default	
Coordinate system: -- default --	
* Includes the mass properties of one or more hidden components/bodies.	
Mass = 4.17 kilograms	
Volume = 1669.58 cubic centimeters	
Surface area = 6532.72 square centimeters	
Center of mass: (centimeters)	
X = 59.90	
Y = 19.72	
Z = -0.91	
Principal axes of inertia and principal moments of inertia: (kilograms * square centimeters)	
Taken at the center of mass.	
Ix = (0.90, -0.39, -0.17)	Px = 158.90
Iy = (0.42, 0.88, 0.20)	Py = 1929.30
Iz = (0.07, -0.25, 0.97)	Pz = 2009.96
Moments of inertia: (kilograms * square centimeters)	
Taken at the center of mass and aligned with the output coordinate system.	
Lxx = 483.08	Lyy = -630.08
Lyx = -630.08	Lyy = 1685.79
Lzx = -269.26	Lzy = 134.88
	Lzz = 1956.28

Figure 2.30: Mass moment of inertia on motor (2)

$$L_{yy} = 1685.79 \text{ kg.cm}^2$$

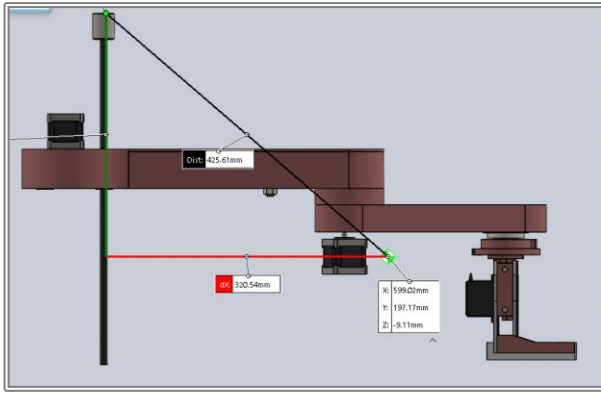


Figure 2.31: Center of mass motor (2) load

$$I = L_{yy} + M R^2$$

$$I = 1685.8 + 4.17 \times 32^2$$

$$I = 5928.9 \text{ Kg} \cdot \text{cm}^2$$

$$\tau = I \alpha$$

$$\tau = 5928.9 \times 0.0065$$

$$\tau = 38.5 \text{ N} \cdot \text{cm}$$

2.5.3. Motor (3)

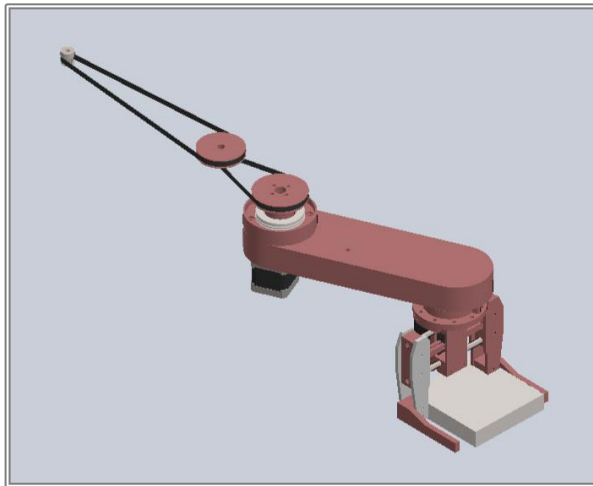


Figure 2.32: Mechanical Parts Loaded on motor (3)

Mass properties of inertia motor 3
Configuration: Default
Coordinate system: -- default --

* Includes the mass properties of one or more hidden components/bodies.

Mass = 2.84 kilograms

Volume = 991.29 cubic centimeters

Surface area = 3782.72 square centimeters

Center of mass: (centimeters)
X = 71.34
Y = 14.89
Z = -2.92

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

$I_x = (0.75, -0.60, -0.29)$	$P_x = 68.03$
$I_y = (0.56, 0.80, -0.20)$	$P_y = 355.29$
$I_z = (0.35, -0.01, 0.94)$	$P_z = 378.78$

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

$L_{xx} = 195.98$	$L_{xy} = -128.33$	$L_{xz} = -69.23$
$L_{yx} = -128.33$	$L_{yy} = 253.57$	$L_{yz} = 49.06$
$L_{zx} = -69.23$	$L_{zy} = 49.06$	$L_{zz} = 352.56$

Figure 2.33: Mass moment of inertia on motor (3)

$$L_{yy} = 253.56 \text{ kg} \cdot \text{cm}^2$$

$$I = L_{yy} + M R^2$$

$$I = 253.6 + 2.84 \times 15.85^2$$

$$I = 967 \text{ Kg} \cdot \text{cm}^2$$

$$\tau_{A(\text{motor})} = \frac{\tau_{C(\text{joint})}}{n_1 n_2} = \frac{I \alpha}{n_1 n_2}$$

$$\tau_{A(\text{motor})} = \frac{967 \times 0.47}{13.215}$$

$$\tau = 34.39 \text{ N} \cdot \text{cm}$$

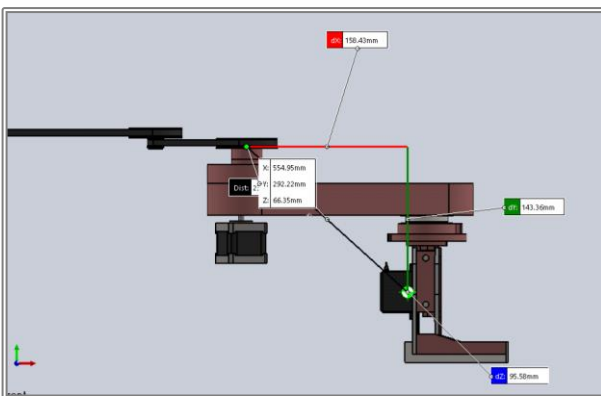


Figure 2.34: Center of mass motor (1) load

2.5.4. Motor (4)

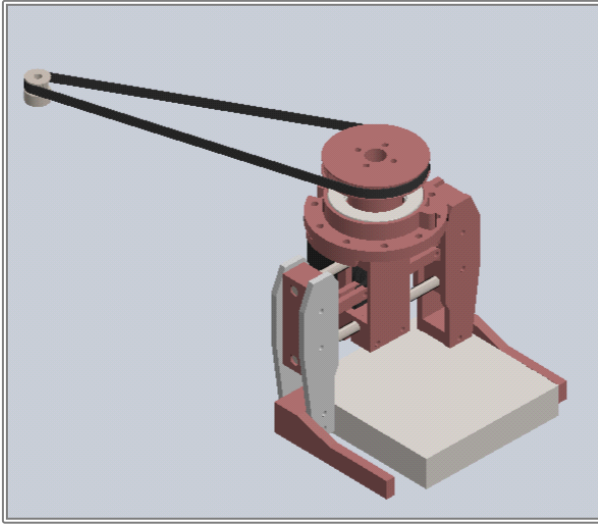


Figure 2.35: Mechanical Parts Loaded on motor (4)

Mass properties of inertia motor 4		
Configuration: Default		
Coordinate system: -- default --		
* Includes the mass properties of one or more hidden components/bodies.		
Mass = 2.15 kilograms		
Volume = 603.93 cubic centimeters		
Surface area = 2055.29 square centimeters		
Center of mass: (centimeters)		
X = 74.53		
Y = 11.75		
Z = -4.16		
Principal axes of inertia and principal moments of inertia: (kilograms * sqa		
Taken at the center of mass.		
lx = (-0.66, 0.70, 0.28)	Px = 39.04	
ly = (-0.49, -0.68, 0.54)	Py = 107.85	
lz = (0.57, 0.21, 0.80)	Pz = 111.55	
Moments of inertia: (kilograms * square centimeters)		
Taken at the center of mass and aligned with the output coordinate system.		
Lxx = 79.13	Lxy = -32.07	Lxz = -14.45
Lyx = -32.07	Lyx = 74.57	Lyz = 12.89
Lzx = -14.45	Lzy = 12.89	Lzz = 104.73

Figure 2.36: Mass moment of inertia on motor (4)

$$L_{yy} = 74.57 \text{ kg.cm}^2$$

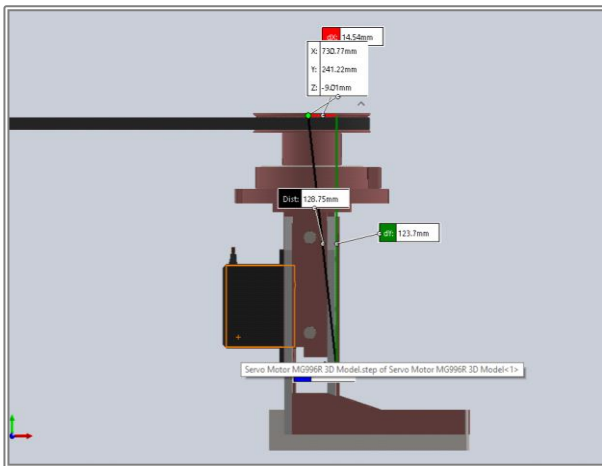


Figure 2.37: Center of mass motor (1) load

$$I = L_{yy} + M R^2$$

$$I = 74.6 + 2.15 \times 1.5^2$$

$$I = 79.5 \text{ Kg} . \text{ cm}^2$$

$$\tau_{A(\text{motor})} = \frac{\tau_{C(\text{joint})}}{n_1 n_2} = \frac{I \alpha}{n_1 n_2}$$

$$\tau_{A(\text{motor})} = \frac{79.5 \times 4.9}{13.215}$$

$$\tau = 29.5 \text{ N.cm}$$

2.6. Stress Analysis

Stress analysis is one of the main design steps as it views to the designer if his mechanism is safe to afford the loads and stresses which effect on it. We can save the money and lives using stress analysis as the mechanism may be damaged because of wrong calculations in the loads. In our project we use solid works as stress analysis program.

In this project we have focused on the effect of sheer stresses as the normal stresses is low and can be neglected

There are two main points fixing effected by shear stress

1. The screw nut connected with screw

2. Connecting points in arm (1)

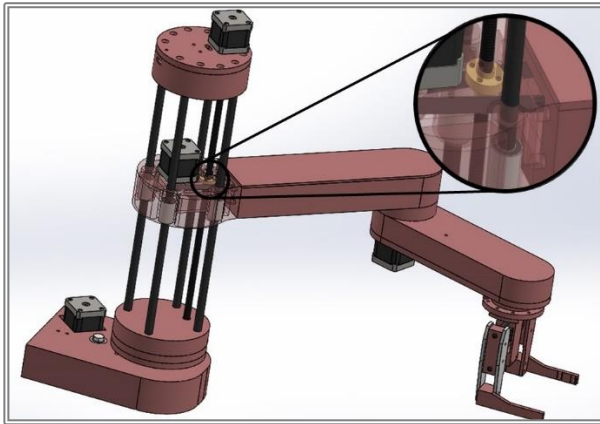


Figure 2.37: Lead screw Nut

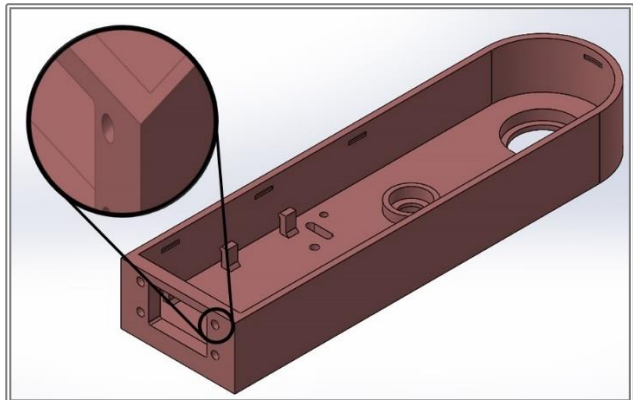


Figure 2.38: Fixing point in Arm (1)

2.6.1. Screw nut

The screw nut used in the project is made of copper and carry almost the total load of the arms, arms gear mechanisms, stepper motor, servo motor, gripper and product pay load. From figure (3.26) the copper yielding stress = 250:400 MPa depending on the increasing rate of load.

Figure (3.27) and figure (3.28) are result of stress analysis study on copper screw nut with pay load 5 kg. The result shows that the maximum stress on the nut is 217 MPa which is less than the minimum yield stress of the used material

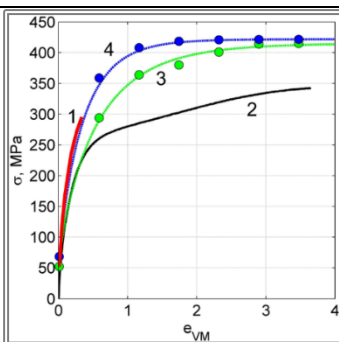


Figure 2.39: Copper stress strain curve

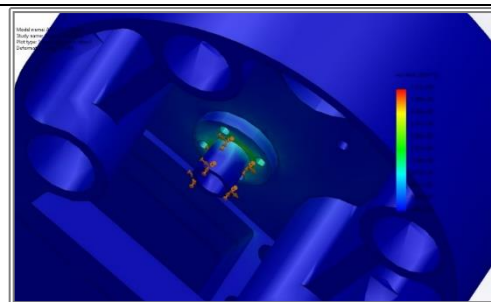


Figure 2.41: result of stress analysis study on the screw nut

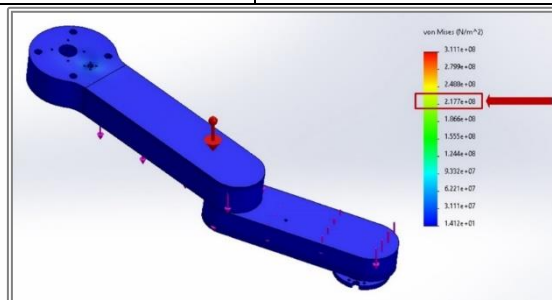


Figure 2.40: result of stress analysis study on the screw nut

2.6.2. Arm (1)

Arm (1) is a main part of the project. It is 3d printed from PLA+. The arm carries the load of the arm (2), Arm (1) and arm (2) gear mechanisms, gripper, stepper motor, servo motor and product pay load. From figure (3.29) the PLA+ yielding stress = 25 MPa.

Figure (3.30), figure (3.31) and figure (3.32) are result of stress analysis study on Arm (1) with pay load 5 kg. the result shows that the maximum stress on the arm body is 2.2 MPa and the maximum stress on the fixing points = 16.5 MPa which are less than the minimum yield stress of the used material

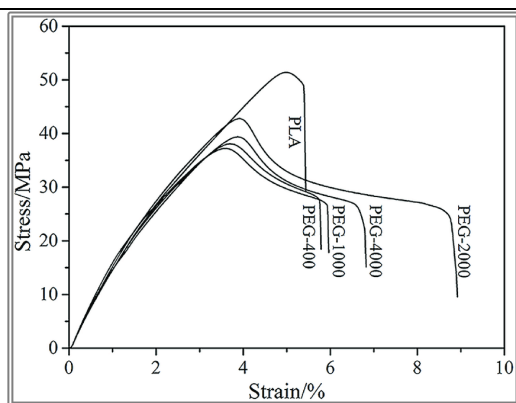


Figure 2.42: PLA+ stress strain curve

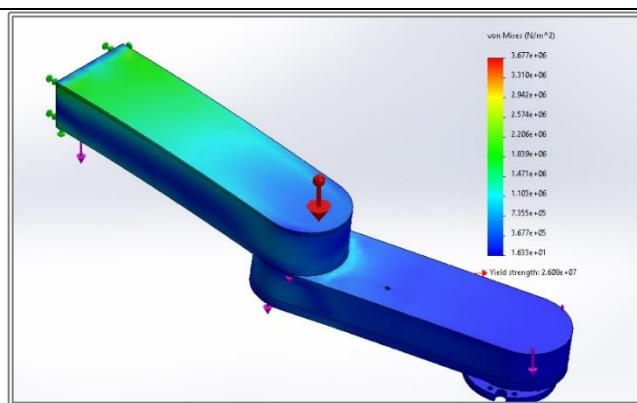


Figure 2.43: result of stress analysis study on the Arm's (1) body

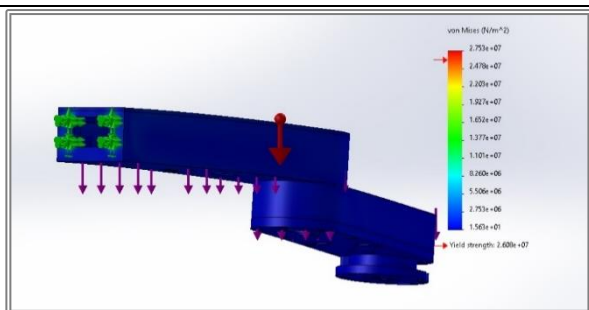


Figure 2.44: result of stress analysis study on the Arm's (1) fixing points

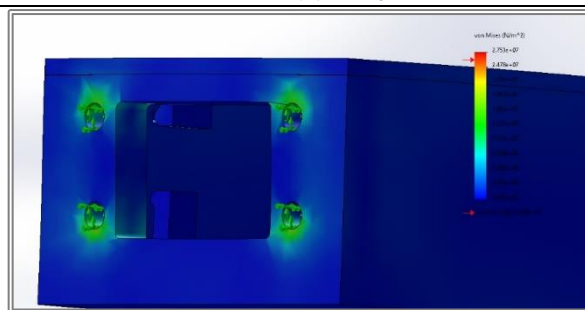


Figure 2.45: result of stress analysis study on the Arm's (1) fixing points

As a result of stress analysis studies, the fixing points loaded with shear stress and the design of the project are safe.

2.7. Fabrication

There a lot of parts has been fabricated using 3D printing technology with material PLA+

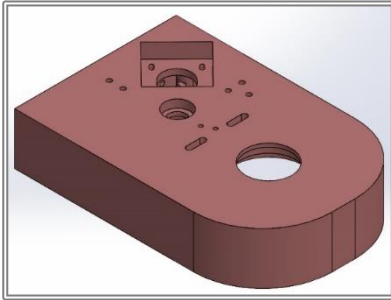


Figure 5.1: Base structure

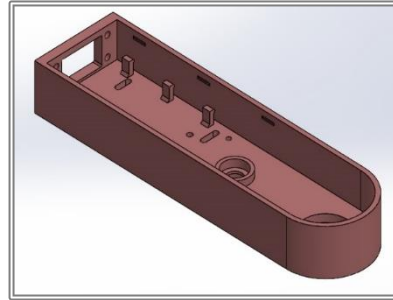


Figure 5.2: Arm (1)

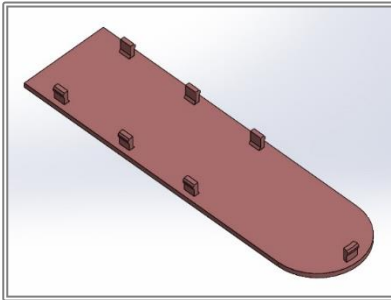


Figure 5.3: Arm (1) Cover

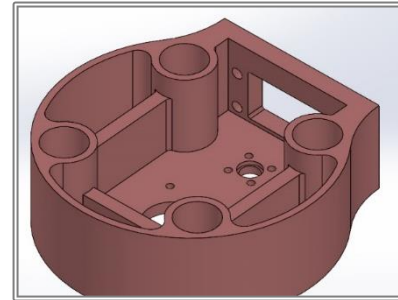


Figure 5.4: Arm (1) Platform

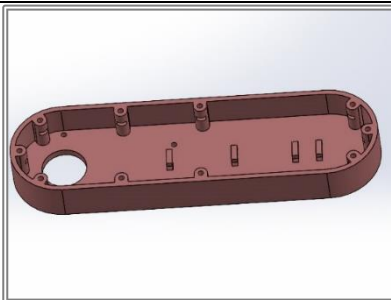


Figure 5.5: Arm (2)

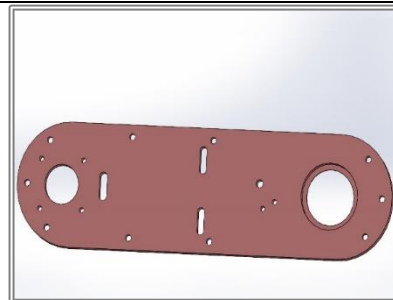


Figure 5.6: Arm (2) Cover

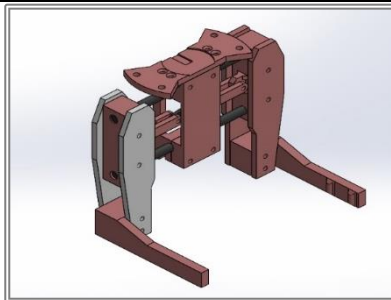


Figure 5.7: Gripper Mechanism

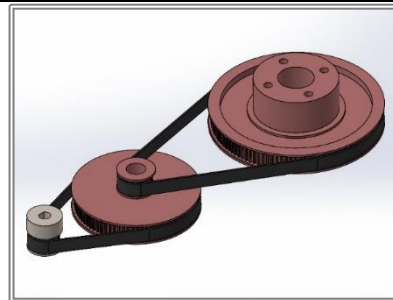


Figure 5.8: Base Gear Mechanism

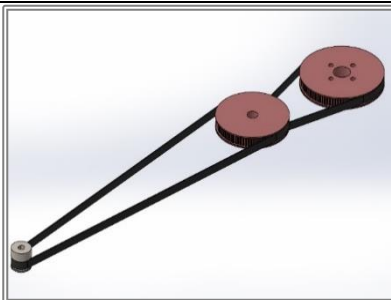


Figure 5.9: Arm (1) Gear Mechanism

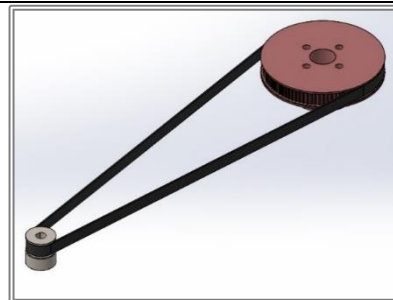


Figure 5.10: Arm (2) Gear Mechanism

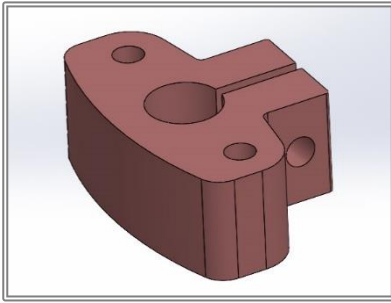


Figure 5.11: Linear guide Clamp

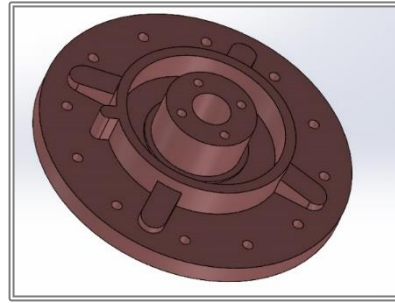


Figure 5.12: Revolute joint J1

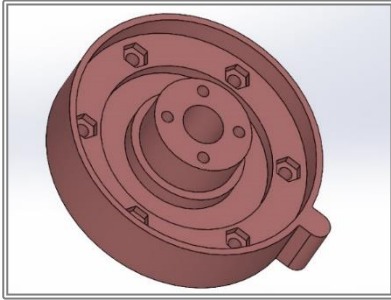


Figure 5.13: Revolute joint J2

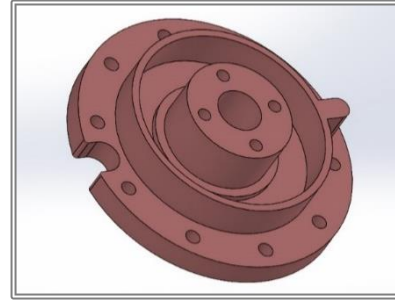


Figure 5.14: Revolute joint J3

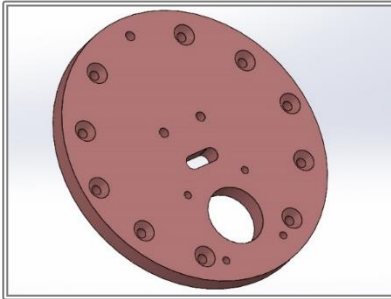


Figure 5.15: Prismatic joint Bottom Plate

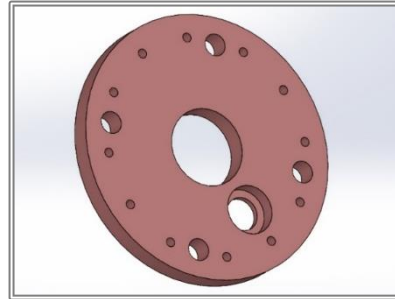


Figure 5.16: Prismatic joint top Plate

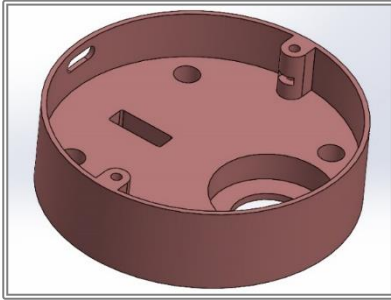


Figure 5.17: Prismatic joint top cover

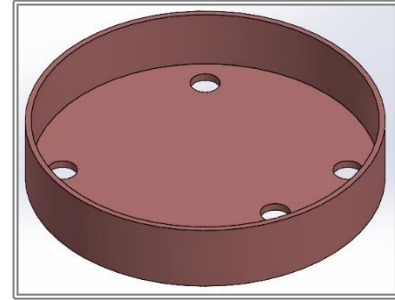


Figure 5.18: Prismatic joint bottom cover

2.8. Mechanical Components

2.8.1. Bearing

A bearing is a machine element that constrains relative motion to only the desired motion and reduces friction between moving parts. Most bearings facilitate the desired motion by minimizing friction. Bearings are classified broadly according to the type of operation, the motions allowed, or to the directions of the loads (forces) applied to the parts. This SCARA robot has three types of bearing which are rotary, linear and thrust bearing.

a. Rotary Bearing

The rotary bearing is a type of bearing which connect between two rotary elements and reduce the rotary friction. There are four different sizes of bearing in this project which are

- ✓ Ball Bearing 30x42x7mm
- ✓ Ball Bearing 25x47x 2mm
- ✓ Ball Bearing 35x47x7mm
- ✓ Ball Bearing 8x22x7mm



Figure 3.1: Rotary Ball Baring

b. Linear Bearing

This linear ball bearing is sort of the opposite of the radial ball bearings. It's intended to slide along a linear shaft, rather than rotate around it. These are very slim, and good for attaching a motion carriage onto a railing without adding a lot of weight. These are very basic bearings; they're meant for a stepper-motion controlled setup so they're not ultra-smooth. They're best used for DIY/hobby robotics projects. There is 4 linear bearings in the same size LM10UU whose dimensions 10x19x29



Figure 3.2: Linear Bearing

c. Thrust Bearing

A thrust bearing is a particular type of rotary bearing. Like other bearings they permanently rotate between parts, but they are designed to support a predominantly axial load. There are two different sizes of bearing in this project which are

- ✓ Thrust Bearing 40x60x13mm
- ✓ Thrust Bearing 35x52x12mm



Figure 3.3: Thrust Bearing

2.8.2. Timing belt

The timing belt is a method is used to transfer the motion between two gears instead of gear box in the aim to control the torque of the motor. The timing belt is better than the gear box as it has less friction than gear box. It is used GT2 timing belt to transfer the motion between gears in this project



Figure 3.4: GT2 timing belt

2.8.3. Lead screw

A leadscrew (or lead screw), also known as a power screw or translation screw, is a screw used as a linkage in a machine, to translate turning motion into linear motion. Because of the large area of sliding contact between their male and female members, screw threads have larger frictional energy losses compared to other linkages.



Figure 3.5: 8x400 mm leadscrew

They are not typically used to carry high power, but more for intermittent use in low power actuator and positioner mechanisms. Leadscrews are commonly used in linear actuators, machine slides (such as in machine tools), vises, presses, and jacks. Leadscrews are a common component in electric linear actuators. It is used a lead screw with specifications:

- ✓ Material: Stainless steel lead screw, copper nut
- ✓ Screw diameter: 8mm
- ✓ Length: 400mm
- ✓ Screw spacing: 2mm

2.8.4. Linear guide

The linear guide is a main method to facilitate the mechanism linear motion, make it more balanced and reduce the friction using linear bearing. There is 4 round linear guide 8mm diameter used in the prismatic joint to facilitate the linear motion in Z-Axis.



Figure 3.6: 10x400mm Linear guide

3



ELECTRICAL HARDWARE



Electrical components

3.1. Actuators

3.1.1. Stepper Motor (17HS4401S)

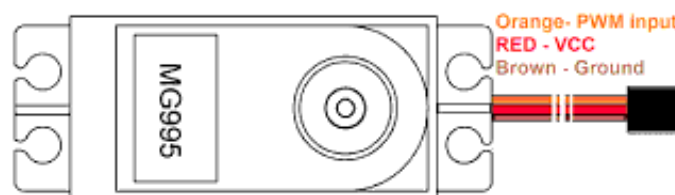


It has 1.8° per step for smooth motion and have a max current of 1.7A/phase , it could be driven easily with a motor driver and an adapter

Motor specifications:

- | | |
|--------------------------------------------|----------------------------------------------|
| ✓ Number of Phase: 2. | ✓ Holding Torque: 43Ncm. |
| ✓ Step Angle: 1.8°. | ✓ Shaft Diameter: ø5mm. |
| ✓ Phase Voltage: 2.6Vdc. | ✓ Motor Length: 40mm. |
| ✓ Phase Current: 1.7A. | ✓ Rotor Inertia: 54gcm ² . |
| ✓ Number of Wire: 4 (100cm Length). | ✓ Mass: 280g. |

3.1.2. Servo motor (MG995)



It comes with a standard 3-pin power and control cable

Mechanical Specification

- ✓ **Size:** 40.4*19.9*37.5mm
- ✓ **Weight:** 58g
- ✓ **Horn type:** Metal

Electrical Specification

- ✓ **Operating voltage:** 4.8V
- ✓ **Idle current:** 5ma
- ✓ **Peak stall torque:** 11 kg.cm
- ✓ **Running degree:** 180°±3°

3.2. DC Power Supply



The 230V to 12V 10A transformer is used to step down the mains voltage. The 10-ampere diode bridge is used to rectify the voltage coming from the transformer. The capacitors are used for filtering the voltage.

The maximum output current of the IC is 1.5A therefore we have used two TIP2955 transistors two boost the output current to 10A

3.3. Electronic components

3.3.1. Arduino Uno (Controller)

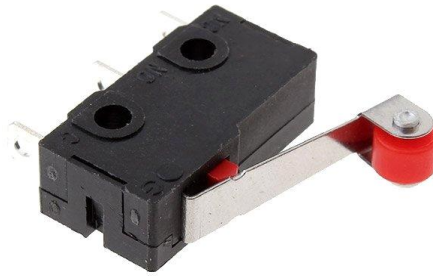
It can be powered by the USB cable or by an external 9-volt battery, though it accepts voltages between 7 and 20 volts.



Technical specifications

- ✓ Microcontroller: Microchip ATmega328P[7]
- ✓ Analog Input Pins: 6
- ✓ DC Current per I/O Pin: 20 mA
- ✓ DC Current for 3.3V Pin: 50 ma
- ✓ Operating Voltage: 5 Volts
- ✓ Input Voltage: 7 to 20 Volts
- ✓ Digital I/O Pins: 14

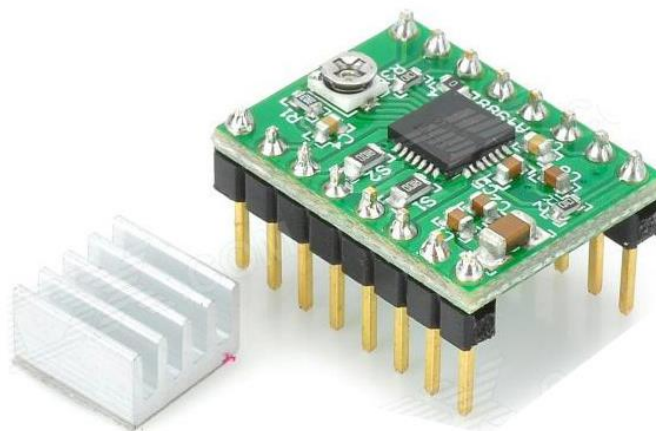
3.3.2. Limit switches (sensors)



The electronic position switches (limit switches) make it possible to flexibly monitor position ranges. It is also used for protecting motors from over (run)

3.3.3. Stepper Motor Driver (A4988)

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, or one for controlling the rotation direction and the other for controlling the steps.



General Specifications:

- ✓ Simple step and direction control interface
- ✓ Five different step resolutions

Technical Specifications:

- | | |
|-------------------------------------|--------------------------------------------------|
| ✓ Minimum operating voltage: 8 V | ✓ Maximum logic voltage: 5.5 V |
| ✓ Maximum operating voltage: 35 V | ✓ Micro step resolutions: 1, 1/2, 1/4, 1/8, 1/16 |
| ✓ Continuous current per phase: 1 A | ✓ Size: 0.6" × 0.8" |
| ✓ Maximum current per phase: 2 A | |
| ✓ Minimum logic voltage: 3 V | |

Connection Schematic:

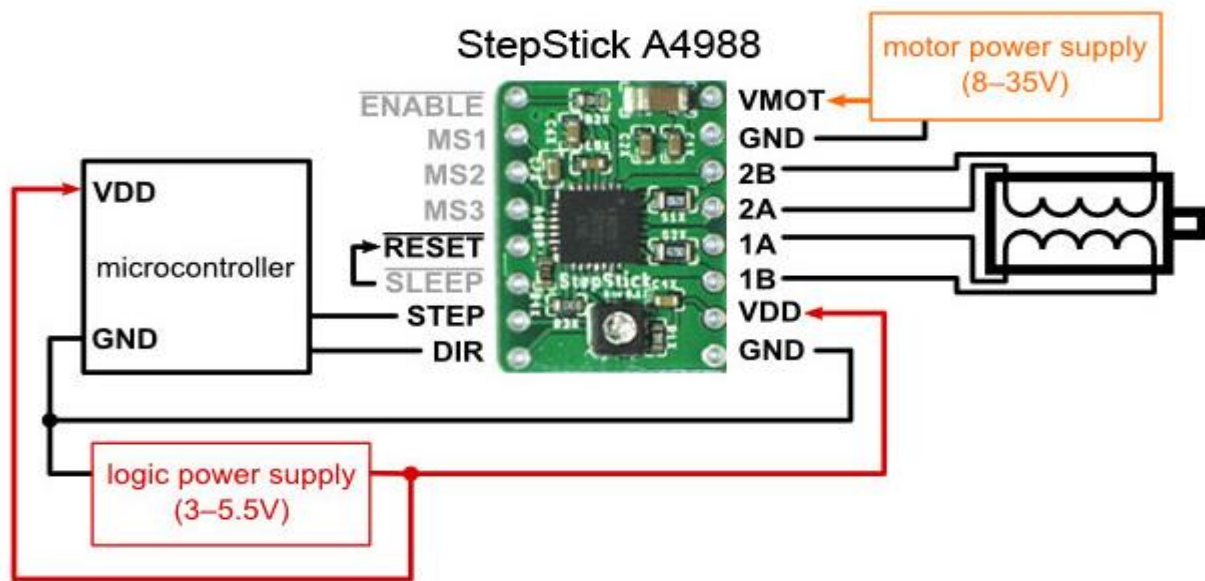
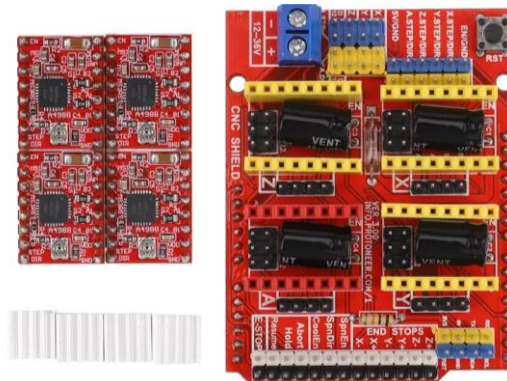


Figure 3.23: A4988 Connection Schematic

3.3.4. CNC Shield:

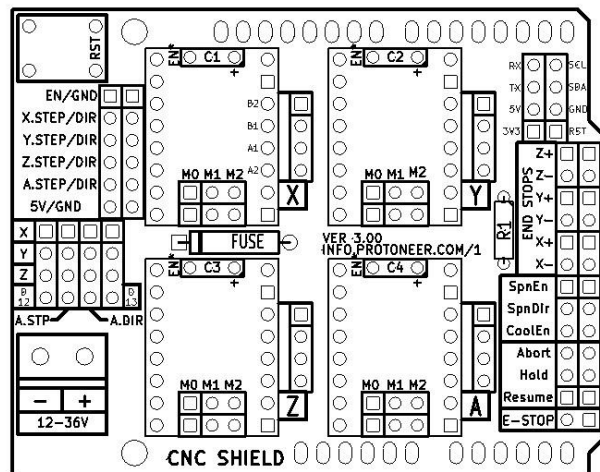


The shield itself consists of the red board in the picture with 4 smaller boards that are used to control motors. The three boards are called the motor driver boards and correspond to the X-, Y-, and Z-axes, and a fourth axis called 'A'. This shields provides an Arduino microcontroller with the power necessary to drive stepper motors and run all the other functions that contribute to the robot operation.

SPECIFICATIONS:

- ✓ **Motor Voltage:** 8 V to 35 V
- ✓ **Logic Circuits Voltage:** 3 V to 5.5 V
- ✓ **Current:** 2 A (MAX)
- ✓ **Five step resolutions:** full, 1/2, 1/4, 1/8 and 1/16
- ✓ **Protection:** under-voltage, over-current and over-temperature
- ✓ External resources

Schematic:

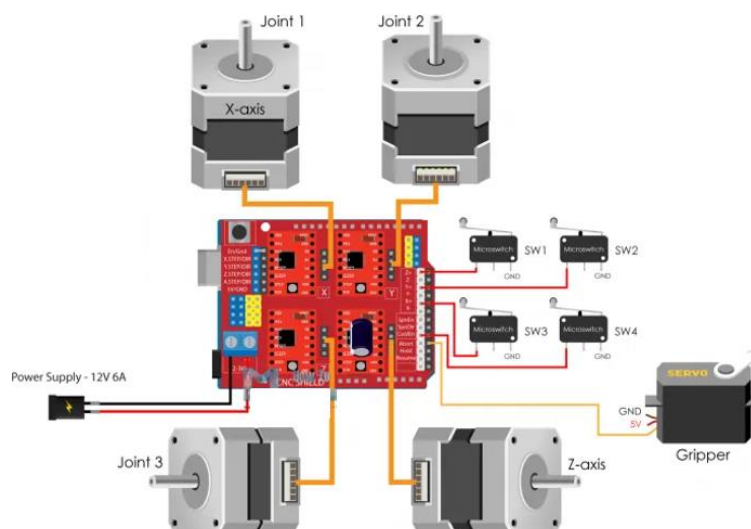


The 3 pins (MS1, MS2 and MS3) are for selecting one of the five step resolutions according to the below truth table. These pins have internal pull-down resistors so if we leave them disconnected, the board will operate in full step mode.

MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step

3.4. SCARA Circuit diagram

The following figure showing the circuit diagram of our SCARA robot and how everything is connected.



4



KINEMATICS, CONTROL & GUI



4.1. Kinematics

The goal of deriving the forward kinematics is to determine the pose (position and orientation) of the end-effector as a function of the joint variables, with respect to the base frame. An x-y-z frame will be attached to each joint to describe its pose with respect to the base frame.

On the contrary, the robot inverse kinematics finds the joint angles from the given end-effector and its orientation.

There is no unique way to attach these frames. A general, systematic method called Denavit-Hartingberg convention will be used to attach these frames.

The DH parameters are specified in the table below.

3.4.1. Calculate DH-parameter from robot.

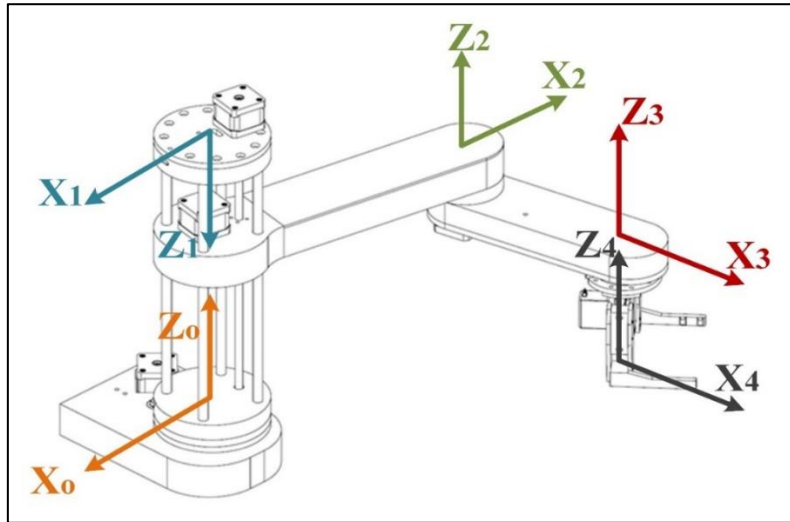


Table : DH parameters for the modified SCARA robot.

DH Parameter	a	alpha	d	theta
1	31	0	0	θ_1
2	0	0	D_2	0
3	20	0	0	θ_2
4	7.2	0	0	θ_3

3.4.2. Transformation matrix:

The total transformation for each frame to its previous frame is given by:

$A_i^{i-1}(q_i) = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -s_{\theta_i}c_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$	<p>Where</p> <p>q_i is Joint i variable</p> <p>$C\theta_i = \cos(\theta_i)$</p> <p>$S\theta_i = \sin(\theta_i)$</p> <p>$C\alpha_i = \cos(\alpha_i)$</p> <p>$S\alpha_i = \sin(\alpha_i)$</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The homogeneous transformation for that end-effector to the base frame given by:

$$T_4^0 = A_1^0 A_2^1 A_3^2 A_4^3$$

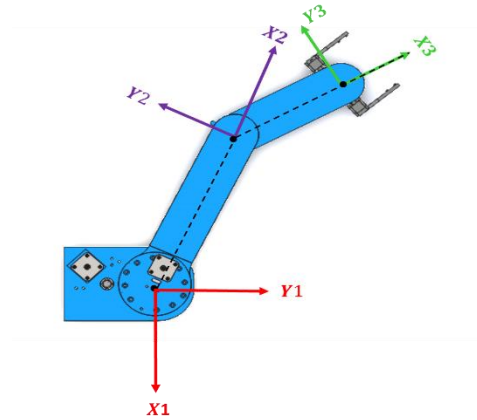
$$T_4^0 = \begin{bmatrix} \cos(\theta_{123}) & -\sin(\theta_{123}) & 0 & 7.2\cos(\theta_{123}) + 20\cos(\theta_{12}) + 31\cos(\theta_1) \\ \sin(\theta_{123}) & \cos(\theta_{123}) & 0 & 7.2\sin(\theta_{123}) + 20\sin(\theta_{12}) + 31\sin(\theta_1) \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where

$$C123 = \cos(\theta_1 + \theta_2 + \theta_3)$$

$$S124 = \sin(\theta_1 + \theta_2 + \theta_3).$$

3.4.3. Forward Kinematics:



The robot forward kinematics calculates the end-effector position and orientation given the joint angles and link lengths. The end-effector position is define by:

$$P = f(\theta, L)$$

Where, θ comprises the joint angles and L is made up of all the link lengths.

We derived the joint angles using the trigonometric relations in a right triangle. The joint positions are:

$$P_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}; \quad P_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} L_{12}c_1 \\ L_{12}s_1 \end{bmatrix};$$

$$P_3 = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} = \begin{bmatrix} L_{12}c_1 + L_{23}c_{1+2} \\ L_{12}s_1 + L_{23}s_{1+2} \end{bmatrix}$$

$$P_e = \begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix} = \begin{bmatrix} L_1C_1 + L_2C_{12} + L_3C_{123} \\ L_1S_1 + L_2S_{12} + L_3S_{123} \\ d \end{bmatrix}$$

We take the vector sum of all the joint position and yield the forward kinematic equation for as:

Where γ is the orientation of the end effector.

3.4.4. Inverse kinematics:

In inverse kinematics problem, the task is to find the joint angles given position p , orientation γ , and the link lengths. Initially, we solve the position of P_3 and get:

$$P_3 = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} = \begin{bmatrix} x_e - L_{34}\cos(\gamma) \\ y_e - L_{34}\cos(\gamma) \end{bmatrix}$$

$$\gamma = \theta_1 + \theta_2 + \theta_3$$

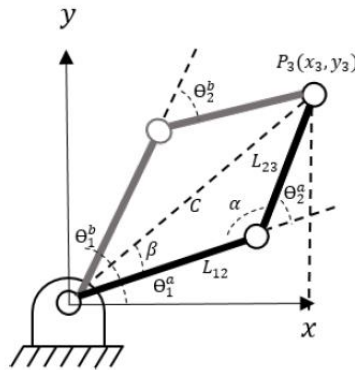
$$P_e = \begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix} = \begin{bmatrix} L_1 C_1 + L_2 C_{12} + L_3 C_{123} \\ L_1 S_1 + L_2 S_{12} + L_3 S_{123} \\ d \end{bmatrix}$$

Hence P_3 is obtained by, we solve the angles α and β as:

$$\alpha = \cos^{-1} \left(\frac{x_3^2 + y_3^2 - L_{12}^2 - L_{23}^2}{2L_{12}L_{23}} \right)$$

$$\beta = \sin^{-1} \left(\frac{L_{23} \sin \alpha}{\sqrt{x_3^2 + y_3^2}} \right).$$

These angles are used to determine the joint angles θ_1 and θ_2 . We keep in mind that there are two set of solution for the joint angles in 2R inverse kinematics problem. We refer joint 3 as the wrist, and joint 2 as the elbow.



We compute the joint angles θ_1 and θ_2 by considering the elbow-up and elbow-down configuration.

The joint angles θ_1 , θ_2 and θ_3 are:

$$\theta_1^a = \left(\tan^{-1} \frac{y_3}{x_3} - \beta \right); \quad \theta_1^b = \left(\tan^{-1} \frac{y_3}{x_3} + \beta \right)$$

$$\theta_2^a = (180 - \alpha); \quad \theta_2^b = -(180 - \alpha)$$

$$\theta_3^a = \gamma - \theta_1^a - \theta_2^a; \quad \theta_3^b = \gamma - \theta_1^b - \theta_2^b$$

Where the a and b notation denotes the elbow-up and elbow-down configuration respectively.

4.2. Workspace

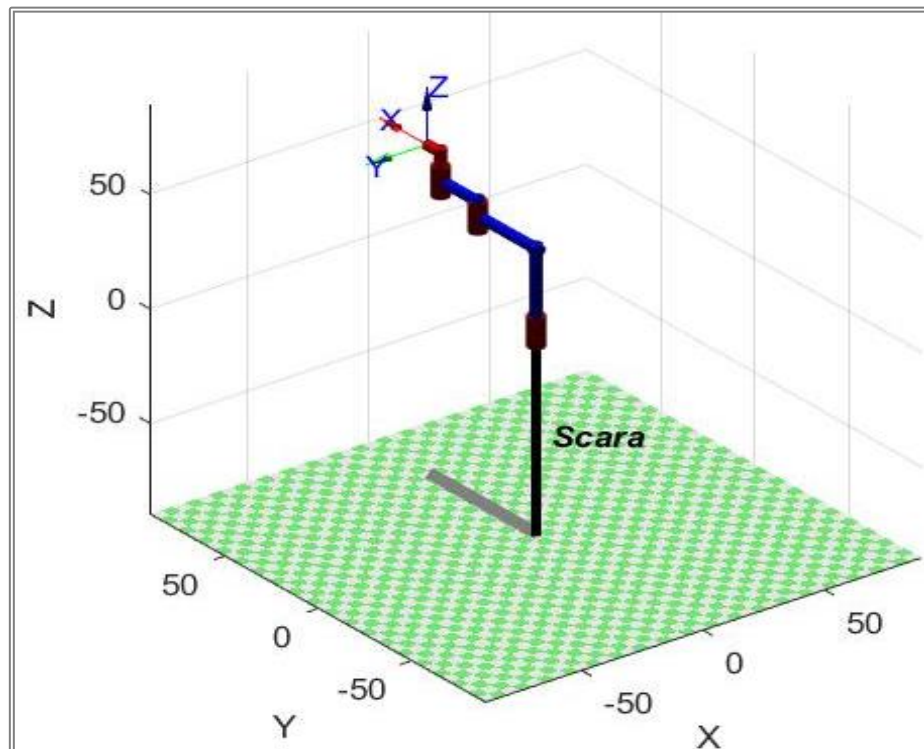
4.2.1. Using MATLAB Peter cork toolbox.

```

4 - syms d2 th1 th3 th4
5 - L(1)= Link( [0 0 31 0 0], 'standard');
6 - L(2)= Link( [0 0 0 0 1 ], 'standard');
7 - L(3)= Link( [0 0 20 0 0 ] , 'standard');
8 - L(4)= Link( [0 0 7.2 0 0], 'standard');
9 - Rb = SerialLink (L);
10
11 |
12 - Rb.plot([0 0 0 0 ], 'workspace', [-90 90 -90 90 -90 90])
13
14

```

4.2.2. Robot simulation

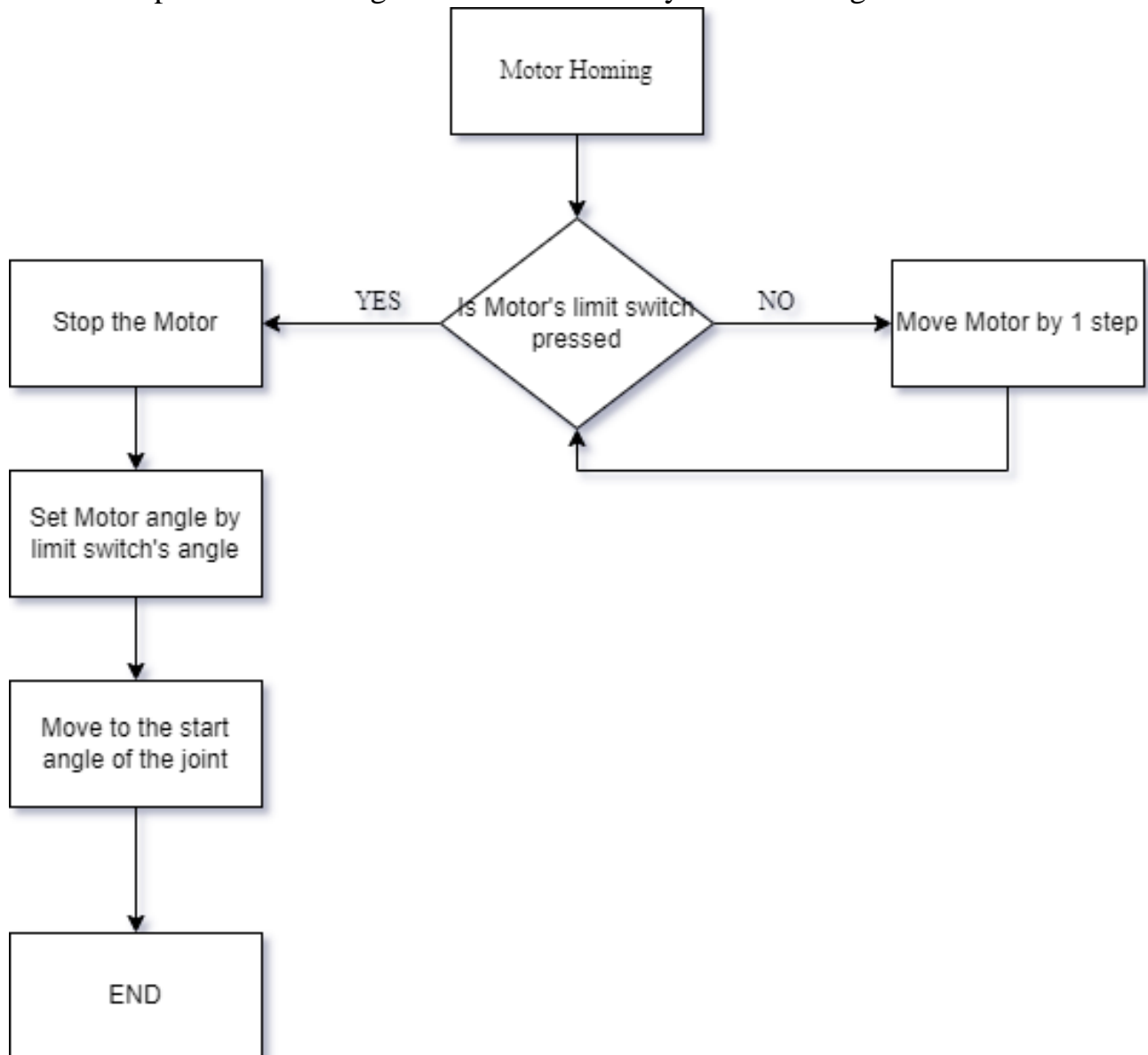


4.3. SCARA Joints Control

4.3.1. Position Homing:

After system reboot the robot motors must go through homing process to determine the position of the Joints, that's done using Limit Switches fixed at known positions.

The process for a single motor illustrated by the following flow chart:

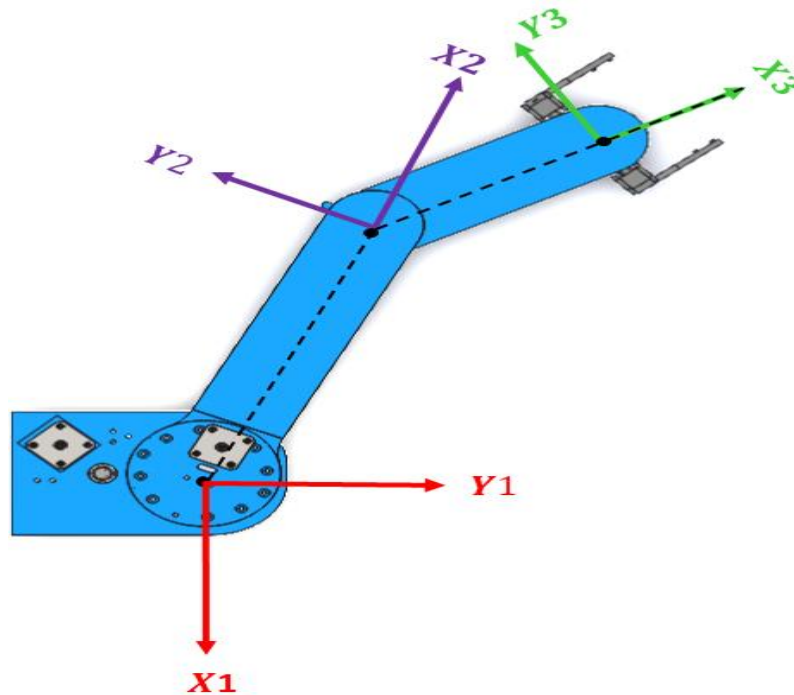


- ✓ At first the motor moves towards the limit switch.
- ✓ When the joint press the switch motor stops and the angle is saved to establish the coordinate system of the joint.
- ✓ Then move the motor to the start angle specified in the code.

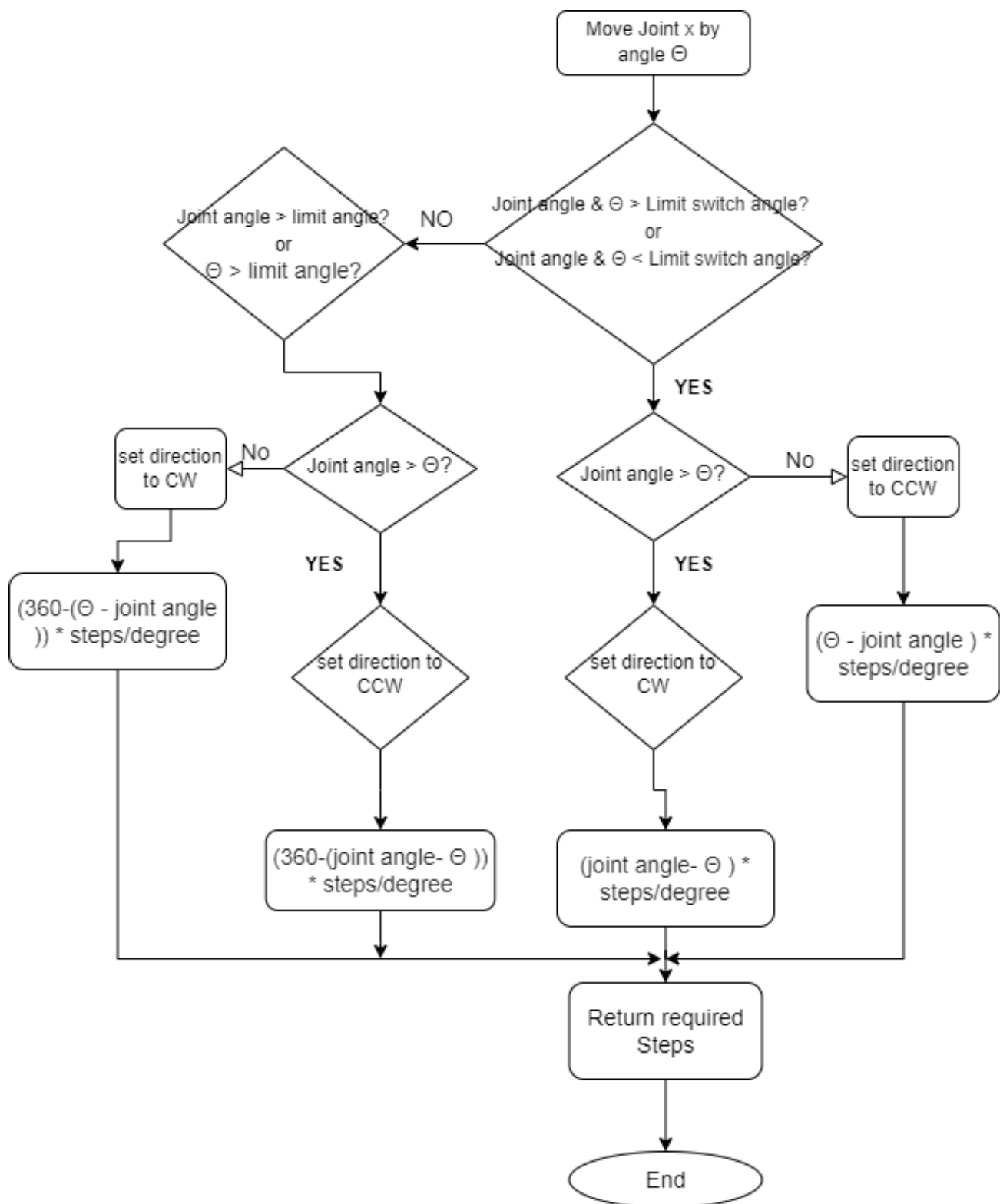
4.3.2. Joints Motion according to the coordinate system:

a) Revolute Joints:

The coordinate system is established after the Homing process as now each joint's angle is known that's done from the Forward Kinematics process by setting each joint's Axis as follow:



- ✓ As the angles are known now we can control the SCARA motion by changing each joint angle value.
- ✓ That's done by a simple function that determines the direction of rotation for the motor and the exact number of steps needed to reach the desired angle.
- ✓ The Mechanism is illustrated by the following flow chart for a single joint:



- ✓ At first determine whether both the current joint angle and desired angle Θ is bigger or smaller than the limit switch fixed angle.
 - If that's true then determine the bigger angle of them for joint angle $> \Theta$ set the direction to clock wise, and calculate the required steps by the formula:

$$(\text{Joint angle} - \Theta) * \text{Steps/degree}.$$
 - For $\Theta > \text{current joint angle}$: set the direction to counter clock wise, calculate the steps by this formula:

$$(\Theta - \text{Joint angle}) * \text{Steps/degree}.$$
- ✓ If one of the angles exceeds the limit switch angles the process needs to change in order to avoid the fixed limit switch:
 - If joint angle $> \Theta$ set the direction to counter clock wise, and calculate the steps by the formula:

$$(360 - (\text{joint angle} - \Theta)) * \text{Steps/degree}$$
 - For $\Theta > \text{joint angle}$ set the direction to clock wise, and calculate the required steps by the formula:

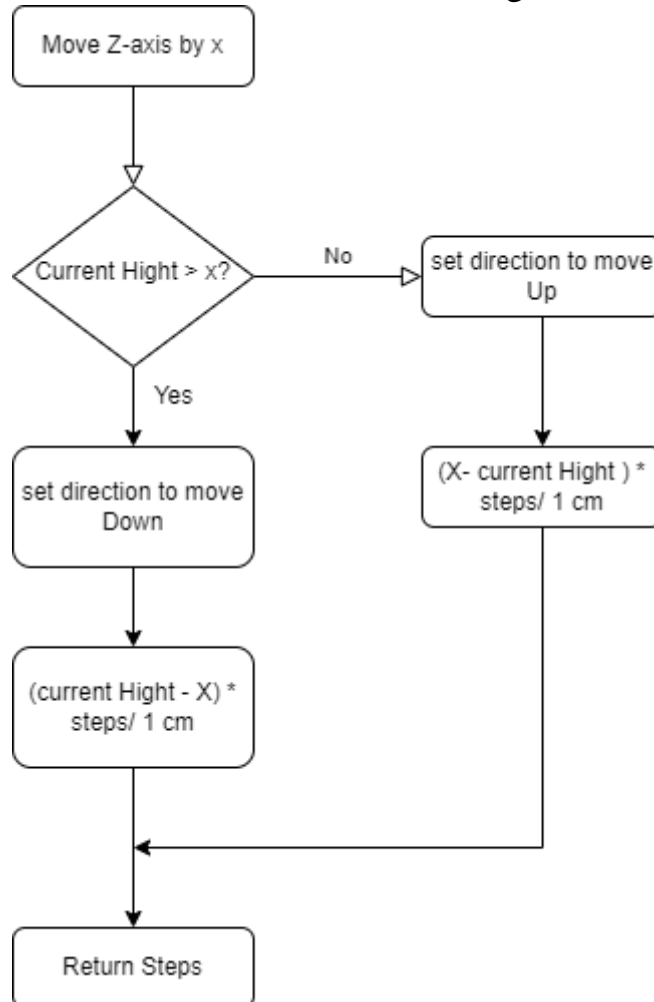
$$(360 - (\Theta - \text{joint angle})) * \text{Steps/degree}$$
- ✓ At the End return the calculated steps with the direction to the Joint's motor to accomplish the motion correctly.

b) Prismatic Joint:

The Z-axis motion is caused by stepper motor coupled to lead screw.

After the Homing process the prismatic joint has reached its maximum height and this height is known and recorded in the controller.

In order to change the height to desired height, the direction and steps is determined by mechanism illustrated in the following flow chart:



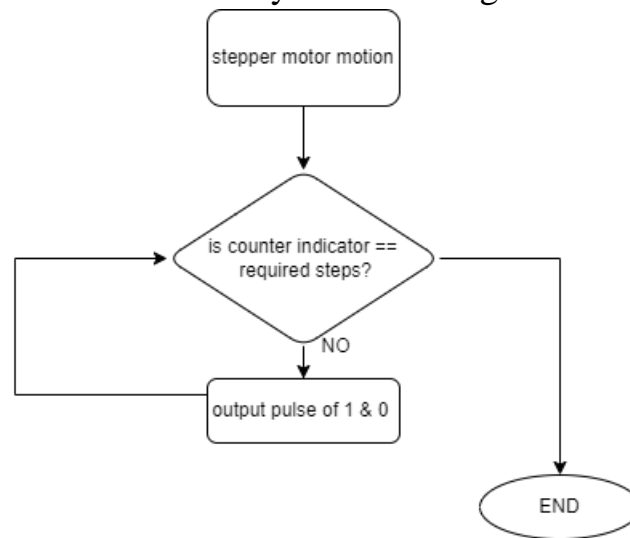
- ✓ If current height > desired height set the direction to move down(clock wise) and calculate the required steps to reach this position by the formula:
 $(\text{Current Height} - X) \times \text{Steps/1 cm}$
- ✓ For desired height > current height set the direction to move up(counter clock wise) and calculate the required steps to reach the position by the formula:
 $(X - \text{Current Height}) \times \text{Steps/1 cm}$

4.3.3. Stepper Motor Motion:

In order to move stepper motor of type(Nema 17) it takes pulses to the Step pin of 1 & 0 each pulse moves the motor by one step and to complete full cycle it needs 800 pulse (quarter resolution)

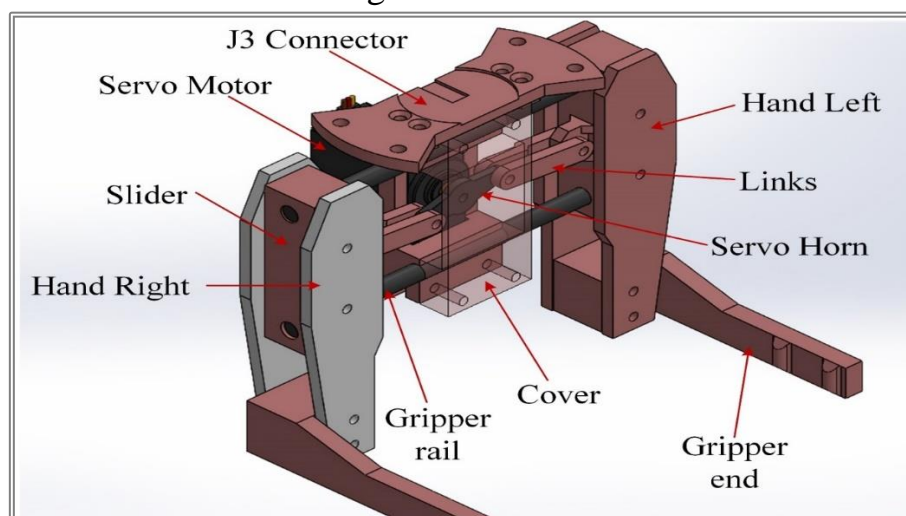
For setting the motor's direction there's a Direction pin that takes 0 (low) for clock wise direction and 1(High) for counter clock wise direction.

Motor motion mechanism illustrated by the following flow chart:



4.3.4. Gripper Control:

- ✓ Gripper is controlled by using a Servo motor of type (MG996R) for opening and closing the gripper.
- ✓ Servo motor is controlled by pulse width modulation (PWM) at 50Hz frequency and delay period 20ms.
- ✓ Rotation angle is determined by the formula: $(\text{rotation angle} / 180) + 1$.
- ✓ By taking advantage of the Arduino's servo library, the desired rotation angle is sent to the required function.
- ✓ The gripper is fully open at angle = 180 and the width = 12cm, fully closed at angle = 0 and the width = 7cm.
- ✓ For specific objects with specific width the gripper can close but not fully by sending the desired angle to Servo motor.
- ✓ For 1cm the servo needs 36 degree of rotation.



4.4. GUI (Graphical User Interface)

4.4.1. MegunoLink – Overview

MegunoLink is a customizable interface tool for Arduino sketches. It allows the user to make a GUI to easily send and receive data to and from the Arduino based on the USB serial communication.

With MegunoLink, we can build a user interface from controls such as buttons and text boxes that runs on PC and send messages to your Arduino program and then the messages is processed on the Arduino to change configurations. So, we used it to easily control our robot as will be discussed.

4.4.2. Serial Communication:

The title itself explains its meaning; the word “serial” means in series and “communication” means to communicate. In Arduino, “Serial Communication” means to transfer data in series to another device.

In Arduino, we can do serial communication either with a computer or some other devices via USB plug and TX/RX pins of Arduino.

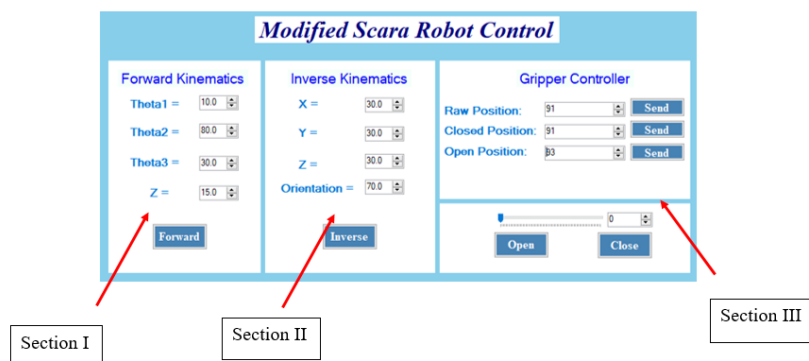
The serial communication in Arduino is done through the pins which are dedicated for this purpose.

The communication between Arduino and computer is established at a specific baud rate. The baud rate specifies how fast the data is sent over the serial line or in simple terms, the speed of serial communication. The common baud rate for Arduino Uno is 9600.

To start serial communication the baud rate set for Arduino and computer must be the same, if baud rate for both is set at 9600 baud, then to transmit 1 bit of data it will take $1/9600 \text{ sec} = 0.014 \text{ millisecond}$. The output of the serial communication can be seen on the serial monitor.

4.4.3. GUI Implementation:

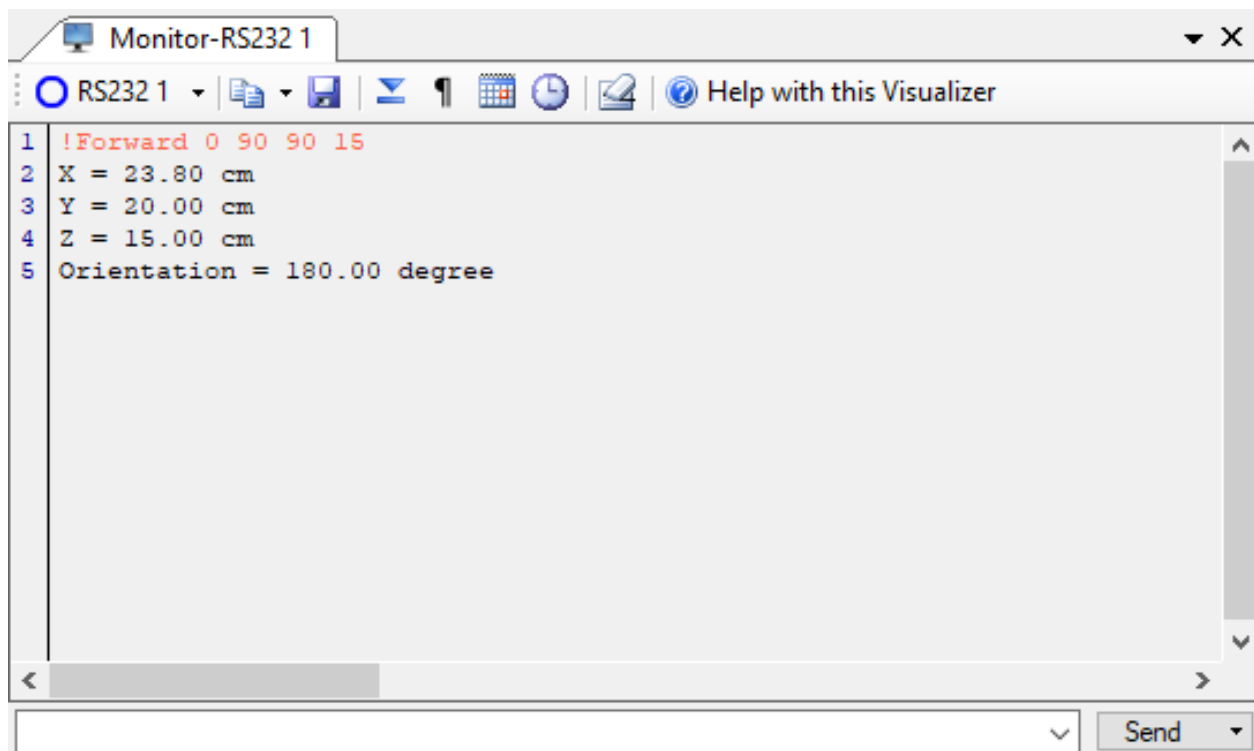
The GUI is spitted into three sections as shown in figure, the first section is for forward kinematics, the second for inverse kinematics, and the third is for controlling the gripper.



For forward kinematics you are only required with angles and it provides you the end effector position and in Inverse Kinematics provide the end effector position (coordinate axes) and GUI will provide you the required angles to achieve that position. It also controls the rotation angle of the servo motor based on the dimensions of the product to be controlled.

4.4.4.1. Section I Forward Kinematics:

1. This section works as follow:
2. It takes the joints angles: Theta1, Theta2, Theta3, and the Z height as inputs from the user.
3. When forward button is pressed, the application sends the values to the MCU via USB serial communications.
4. The MCU receives the data and starts to do its function by converting the given angle to steps and then rotating the motor until reaching the calculated steps number.
5. By substituting in the previously discussed equations eq() of forward kinematics, the end-effector position and its orientation are calculated.
6. Finally, the end effector position and orientation are displayed on the serial monitor as shown

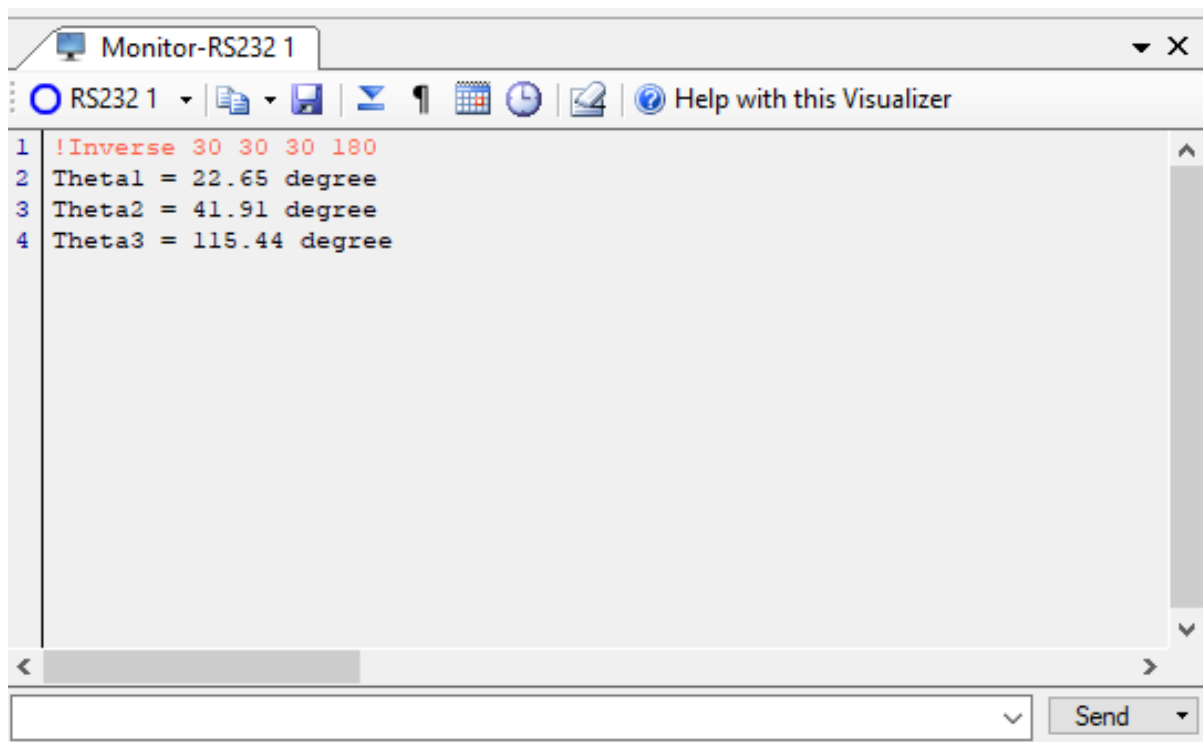


```
1 !Forward 0 90 90 15
2 X = 23.80 cm
3 Y = 20.00 cm
4 Z = 15.00 cm
5 Orientation = 180.00 degree
```


4.4.4.2. Section II Inverse Kinematics:

This section works as follow:

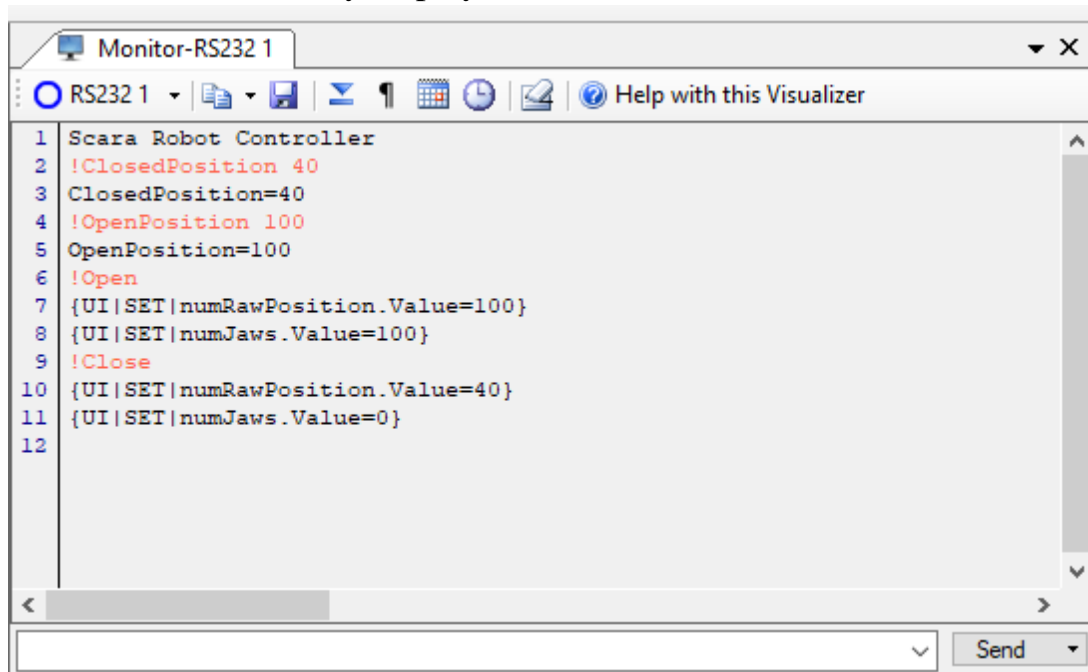
1. It takes X, Y, Z, and orientation of the end effector as inputs from the user in order to find the rotation angles according to the previously mentioned equations.
2. When inverse button is pressed, the application sends the values to the MCU via USB serial communications.
3. The MCU receives the data and starts to do its function by substituting in the previously discussed equations eq() of inverse kinematics in order to calculate the values of the joints angles.
4. Then, the angles are converted to steps and the motors are forced to rotate until reaching the calculated steps number.
5. Finally, the the joints angles: Theta1, Theta2, Theta3, and the Z height are displayed on the serial monitor as shown



4.4.4.3. Section III Gripper Controller:

This section works as follow:

1. It allows the user to send a raw position angle in order to move the servo motor by any desired angle.
2. By using trial and error approach we start to give the program, values of closed and open positions depending on the width of the product to be controlled.
e.g. : in case of a product of juice can with a determined width to allow the gripper to be closed and open by angles 100, 40 degrees, respectively, the user gives these two values to the GUI through its specified labels, then they are stored in the EEPROM in order to be used again.
3. Once open and closed positions are stored, we can easily open and close the gripper using the open and close buttons or changing the angle with any value between using the slider by a percentage between 0 and 100%.
4. These values are finally displayed on the serial monitor as shown



```
1 Scara Robot Controller
2 !ClosedPosition 40
3 ClosedPosition=40
4 !OpenPosition 100
5 OpenPosition=100
6 !Open
7 {UI|SET|numRawPosition.Value=100}
8 {UI|SET|numJaws.Value=100}
9 !Close
10 {UI|SET|numRawPosition.Value=40}
11 {UI|SET|numJaws.Value=0}
12
```

5



IMAGE PROCESSING



Image processing is a method to perform some operations on an image, to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Nowadays, image processing is among rapidly growing technologies. It forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps:

- ✓ Importing the image via image acquisition tools
- ✓ Analyzing and manipulating the image
- ✓ Output in which result can be altered image or report that is based on image analysis.

There are two types of methods used for image processing namely, analogue, and digital image processing. Analogue image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. Digital image processing techniques help in manipulation of the digital images by using computers. The three general phases that all types of data must undergo while using digital technique are pre-processing, enhancement, and display, information extraction.

5.1. Image Processing Applications

Almost in every field, digital image processing puts a live effect on things and is growing with time to time and with new technologies.

5.1.1. Image Sharpening and Reconstruction

It refers to the process in which we can modify the look and feel of an image. It basically manipulates the images and achieves the desired output. It includes conversion, sharpening, blurring, detecting edges, retrieval, and recognition of images.

5.1.2. Medical Field

Image Processing is used for various tasks like PET scan, X-Ray Imaging, Medical CT, UV imaging, Cancer Cell Image processing, and much more. The introduction of Image Processing to the medical technology field has greatly improved the diagnostics process

5.1.3. Robot and Machine Vision

There are several robotic machines which work on the digital image processing. Through image processing technique robot finds their ways, for example, hurdle detection robot and line follower robot. Also, one of the most interesting and useful applications of Image Processing is in Computer Vision. Computer Vision is used to make the computer see, identify things, and process the whole environment. An

important use of Computer Vision is Self-Driving cars, Drones etc. CV helps in obstacle detection, path recognition, and understanding the environment. Also, in Traffic Sensing Technologies and phones and devices with camera system attached to it

5.1.4. Pattern recognition

It involves the study of image processing, it is also combined with artificial intelligence such that computer-aided diagnosis, handwriting recognition, and images recognition can be easily implemented. Now a days, image processing is used for pattern recognition.

5.1.5. Video processing and Real-Time Processing

It is also one of the applications of digital image processing. A collection of frames or pictures are arranged in such a way that it makes the fast movement of pictures. It involves frame rate conversion, motion detection, reduction of noise and space conversion etc. This leads us to the main application of our project which is Quality Control on a Product.

5.2. Quality Control in an Industrial System

Now a days industries require accurate and timely information to improve their quality and increase the production of goods. Thus, digital image processing can be used in this area mainly for the detection of faulted parts or missing part. Automated visual inspection is the key factor behind all the industrial application of digital image processing. The proposed project mainly deals with analysis to find faulty industrial parts and correcting it. Automated visual inspection uses digital camera that captures image which is coming out of product line. The captured image is then fed to computer for further processing. The typical image processing task is to find the faulty piece by comparing the user requirements and manufactured product image. This proposed idea will employ verification of measurements of all the pieces that are coming out of product line. Based on the error rate modification in the design can be pursued. This proposed idea increases the speed and accuracy and avoids human errors which are common in quality testing. The proposed project overcomes the weakness in the existing system. Thus, manual inspection can be replaced with the proposed idea and hence productivity can be increased.

5.3. Our target

Therefore, Our Target Is to do an improved quality check on our products and it depends on some factors which some of them are co-dependent on each other:

1. Workspace Ahead that the camera sees, and the arm operates inside.
2. Camera Orientation regarding other components.
3. Arm Orientation and motion effects on the workspace.
4. Product Orientation and what point of view of it the camera catches.
5. Accuracy and Time of Processing of the data extracted from the product and system.
6. Whether The Purpose is Sorting Different Products or Checking on The Same Product.

5.4. Selecting the Product and Criteria of Quality Check

Mainly it depends on the application of the system and the manufacturing techniques regarding the product specifications and requirements. Therefore, we selected a suitable product that consists with our application. Some Local Boxes of Juice.

5.5. Criteria of Quality Check:

1. Coloration and how consistent and smooth it is regarding to the original photo
2. Color Intensity: where the system checks for the required degree of the color
3. Size of the product: the system evaluates the dimensions of the product from a certain pre-recognized angle and decides upon it.
4. Shapes and cravings: for the original design to be maintained, the system checks for all the details
5. Attached Components: if a product like ours has a part stick to it for example like the Drinking Straw
6. Production Date and instructions: the writing provided by the manufacturer and other policy to be accurate and thorough, the system checks for them although the need a very high-quality equipment to provide a good detection.

5.6. System Components and Part

5.6.1. The Hardware Part:

- a. **The Sensing Element (The Camera):** It's critical to select a camera that fulfill the requirements of the system and needs to operate in a fully functional way.

Note that there are systems that develop very fine results using more than one camera, but they are very sophisticated and complex in a very advanced level and requires more time and resources. **So,** Hence The main basis for choosing the camera would be resolution, frames per second, and the wideness of angle and after dedicated research we settled on “Razer Kiyo “which has the next specifications:

Camera Specifications

- ✓ Connection type: USB 2.0

- ✓ Image resolution: 4 Megapixels
- ✓ Video Resolution: 1080p @ 30FPS / 720p @ 60FPS / 480p @ 30FPS / 360p @ 30FPS
- ✓ Video encoding: YUY2/NUPEG or H.264
- ✓ Still Image Resolution: 2688x1520
- ✓ Image Quality settings customization: Yes
- ✓ Diagonal Field of View (FOV): 81.6
- ✓ Focus Type: Auto
- ✓ Mounting Options: L-shape joint and Tripod
- ✓ Cable Length: 1.5 meters braided cable

Ring Light specifications

- ✓ Illumination 12 white LEDs
- ✓ Color Temperature: 5600K "daylight"
- ✓ LED Diffuser: Milky White
- ✓ Buttons: 12 step ring dials
- ✓ Brightness: 10 Lux @ 1 m

Microphone Specifications

- ✓ Audio Codec: 16bit 48 KHZ
- ✓ Polar patterns: Omnidirectional
- ✓ Sensitivity. -38dB

Approximate Size & Weight

- ✓ Diameter: 68.7 mm/ 2.7 in
- ✓ Height: 50 mm/ 1.96 in
- ✓ Weight: 170 g / 0.374 Ibs

b. ***The Processing Unit*** which in this case we used our Personal Laptop as Runner for our programs. However we faced some complications as the laptop overheated during the training process in order to solve this we cleaned the laptop from the dust, applied new thermal paste, and added thermal pads to the VRAM chips of the GPU.

c. ***Attachments and Secondary tools:*** Cables, Tripod and covers.

5.6.2. Software Part and Applications we installed for machine learning and AI :

a. Anaconda3

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

b. Python

Python is used along with Tensorflow for machine learning/deep learning applications.

c. Protoc (protocol buffers)

Protocol Buffers is a free and open-source cross-platform data format used to serialize structured data. It is useful in developing programs to communicate with each other over a network or for storing data.

d. LabelImg (python library)

labelImg is a graphical image annotation tool and label object bounding boxes in images

e. Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits

f. Jupyter notebook

Project Jupyter is a community run project with a goal to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages". We use this as our compiler for python code.

g. Matlab

We used the computer vision toolbox which contained the ground truth labeler app which is similar to that of the function of 'labelimg' however we were unable to continue with the training and evaluation due to licensing, coding, and function issues, In addition to low tutorial count which wouldn't help us.

5.7. Steps of the Image Processing Operation

1. We install all the needed application and modules.

We install python, TensorFlow and all the dependencies that are needed.

2. Taking images

We capture the data/images we need from the camera using opencv functions along with python, we can adjust the number of pictures taken depending on the needed and the accuracy.

3. Labeling the pictures

Using the labeling library, each and every image is taken will be labeled, this method is named as feature extraction as we extract the shape/object in the picture or image that needs to be detected, Image labeling is the process of identifying and marking various details in an image. Image labeling is useful when automating the process of generating meta data or making recommendations to users based on details in their images.

4. Training the model

After we gathered and labeled our images, we distribute the images along two folders, test and train, most of the data goes into the train and the rest is in testing. As a general rule, 70% of the data goes to the train folder and 30% is in the test.

Note that the training can be done by outside images that are taken by other cameras. But it's necessary when running the detection that the captured images specifications consists with the ones of the detecting camera

5. Evaluation of the model

Evaluation of the model is basically checking the performance of how well the model does in the detection process, the main things we focus on is the loss, Mean average precision or MAP, and average recall or AP.

6. Detection from camera or existing image

After we trained and evaluated our model, we put it into the test. We can either use the model either on the camera or testing on the image.

7. Synchronizing a Serial Data Communication With The Microprocessor

We exchange signals through serial communications with the micro and trying to sync the timing and responses together so both the system and the microprocessor have space to actuate properly leading to a full functional quality control system.

But we couldn't fulfill this step due to some complications from both sides of our system and the processor, also the same mentioned complications from the matlab application

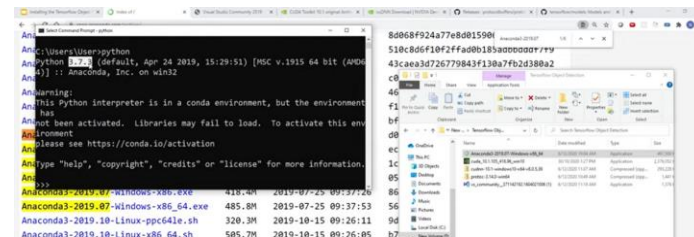
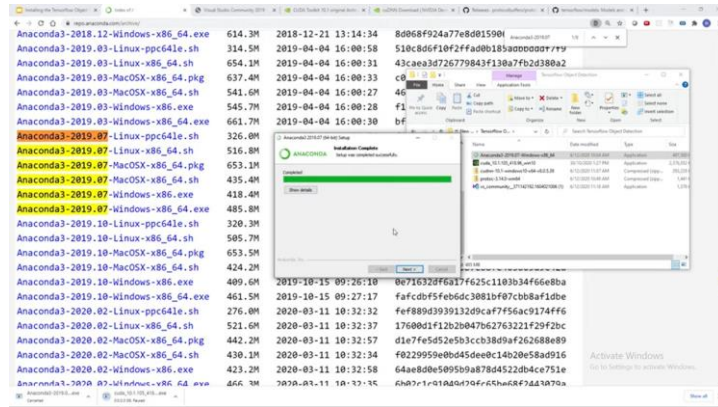
5.8. Working with Software

5.9. Object detection

1. installation and setup.
2. collecting images and labeling
3. training models
4. detecting objects
5. freezing and conversion
6. performance Tuning

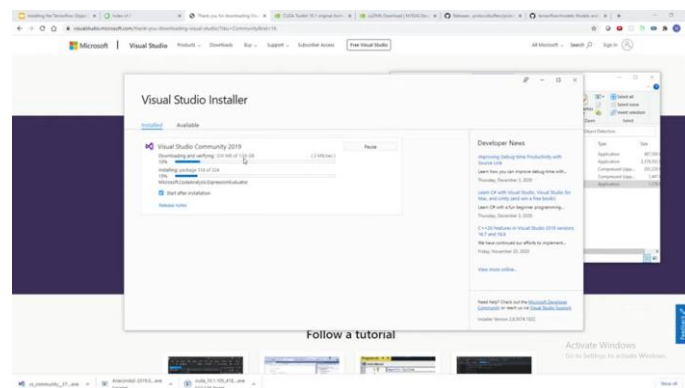
5.9.1. install and setup:

1. Camera
2. Python using anaconda



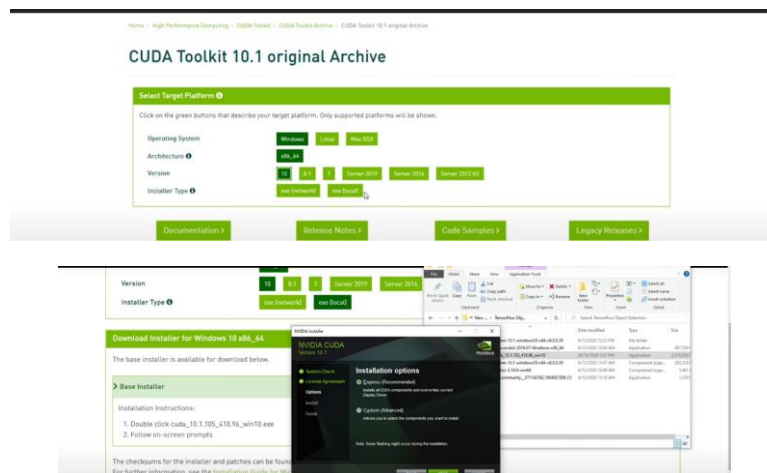
3. visual C++ build tool

Tensorflow relies on C++ build tools and CUDA so downloaded and install both visual studio and Nvidia CUDA.



4. CUDA and CUDNN for NVIDIA

CUDA and CUDDN are optional used by NVIDIA GPU to speed up deep learning.



5. Protoc
6. Tensorflow Object Detection (Dependencies)

5.9.2. Collecting images and labeling

We will be using the webcam for detection. If you are doing this on your machine you need to be able to have access to your webcam to be able to collect images using the webcam. When it comes to labelling, the library that is going to be used is called 'labelimg'. This allows you to label different objects within an image. The way how we use this library is that we press 'w' on the keyboard or manually select a rectangular shape around the object, these labels should be selected as narrow as possible to make sure that your model is able to detect specifically that object and avoid false detections. This helps a lot in boosting the detection performance. Taking pictures within different angels and different positions is also critical as it eases the detection process and is more effective for detecting. Another performance measure that would help is how much data/pictures are taken, the more the data, the better it is for the model for the training/testing process. We start off by 10 to 20 pictures for each class and train the model. If we find the accuracy is not enough, we increase the number of pictures taken.

5.9.3. Training the model

We train the model by using a model called mobile net which is a specific deep learning architecture, but there are other options of different types of models depending on the need, accuracy, latency, etc. Before training our model, we need to install a number of dependencies, the major one being TensorFlow. When it comes to training the model, we use a pretrained model as part of the training process and then we modify it and build on the pretrained model to get our custom model for our object detection purpose. Another thing to keep in mind, is the number of steps in training. The initial value is 4000 steps, but we can increase that number which increases training time resulting in better accuracy. This process takes all the data that has been gathered, as well as the labels, converting both the training and testing data to TensorFlow that can be used for training.

5.9.4. Evaluating the model

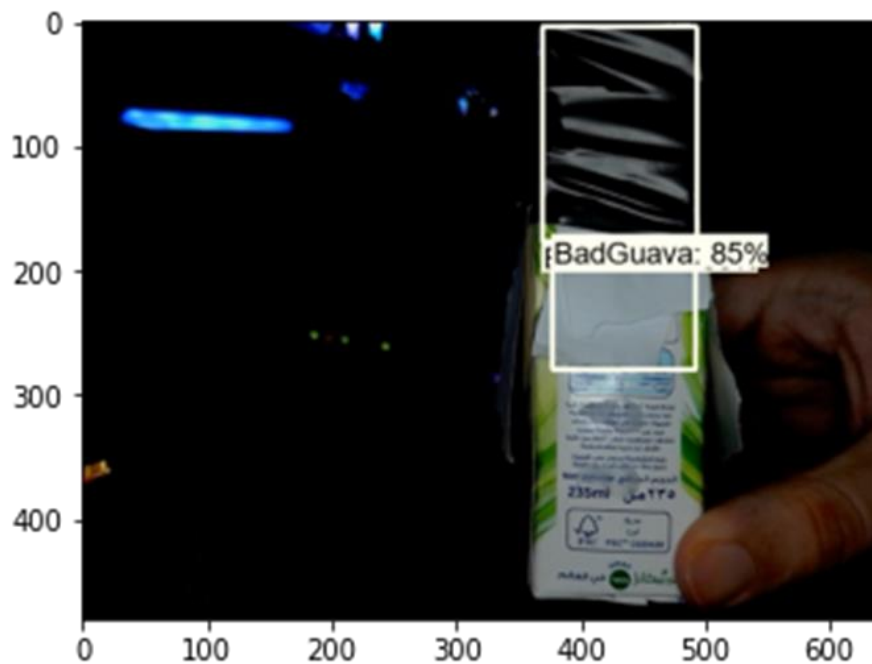
After the training is done, we can evaluate our model to calculate average precision, average recall, and loss. Mean average precision or MAP is the proportion of how many detections where correct, the equation for this is $TP/(TP+FP)$. TP being true position and FP being false position. While Average recall is the proportion of the number of actual objects that were captures. The equation being $TP/(TP+FN)$. FN being false negative.

		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	Sensitivity $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	Specificity $\frac{TN}{(TN + FP)}$
		Precision $\frac{TP}{(TP + FP)}$	Negative Predictive Value $\frac{TN}{(TN + FN)}$	Accuracy $\frac{TP + TN}{(TP + TN + FP + FN)}$

There are two methods for object detection.

1. Object detection from existing image

Where image is already taken and stored in the image directory and we can use that image to detect the object within it and it would display a rectangular box around the object. As the below figure shows the coordinates of X and Y being the resolution of the picture.



2. object detection using external device

Where the object is detected using a webcam,

The object is put in front of the camera and it will display a rectangular shape around the object, as the object moves the rectangular box moves as well showing the accuracy of how well the object is detected. We position the camera in front of the object, as we adjust the best lighting conditions for better accuracy and faster detection.

The code for both camera detection and existing image will be available in the appendix.



5.9.5. Freezing the graph (saving a slice of the model)

Freezing the model means producing a singular file containing information about the graph and checkpoint variables. This eliminates additional information saved in the checkpoint files such as the gradients at each point, which are included so that the model can be reloaded, and continue training from where you left off. As this is not needed when serving a model purely for inference they are discarded in freezing. It will be available at exported files that is in the workspace. These have same output of the pre-trained models so it exactly mimics what is in the model. This makes it available to be used elsewhere in other projects. So in Simple Words, It is the equivalent of saving a portion of your model to be later used when needed. As you continue to train your model, it builds up checkpoints as you go. You can pickup on your last checkpoint and continue to build on it.

checkpoint	01-Jul-22 7:23 PM
ckpt-1.data-00000-of-00001	01-Jul-22 7:07 PM
ckpt-1.index	01-Jul-22 7:07 PM
ckpt-2.data-00000-of-00001	01-Jul-22 7:11 PM
ckpt-2.index	01-Jul-22 7:11 PM
ckpt-3.data-00000-of-00001	01-Jul-22 7:15 PM
ckpt-3.index	01-Jul-22 7:15 PM
ckpt-4.data-00000-of-00001	01-Jul-22 7:19 PM
ckpt-4.index	01-Jul-22 7:19 PM
ckpt-5.data-00000-of-00001	01-Jul-22 7:23 PM
ckpt-5.index	01-Jul-22 7:23 PM
pipeline.config	01-Jul-22 7:05 PM

5.9.6. Exporting The Model to TFJS

For further the use of the frozen graphs files, they should be converted and exported into different types and formats. so, it becomes a deep learning model saved as JSON file. This can be used In JavaScript applications such as react application and Tensorflow.js in java script. It represents the full architecture, weights and everything attached.. Keep in mind that some Image-based models (Mobile Net, Squeeze Net, add more if you tested) are the most supported But Models with control flow ops are not yet supported.

TensorFlow.js converter is an open source library that is used to load a pre-trained Tensor Flow Saved Model, Frozen Model or Session Bundle into the browser and run inference through TensorFlow.js. It is a 2-step process to import your model through a series of coding steps and filing but the sums as:

1. A python pip package to convert a Tensor Flow Saved Model/Frozen Model/Session Bundle to a web friendly format. If you already have a converted model, or are using an already hosted model (e.g. Mobile Net), skip this step.
2. Javascript API. For loading and running inference.

Exporting to TFLite Formats (Final Detection File Model)

TFLite is another file format that is typically used for mobile application or machines that cannot or will not be running a full version of Tensorflow object detection. The command is separated into two parts. First is to export Tensorflow light graph by TF-light-conversion. And then export it as end-to-end model.

This is the representation of a full run through of Tensorflow object detection. Converting the normal tensorflow files into TFLite is used in devices that are not powerful for light use applications.

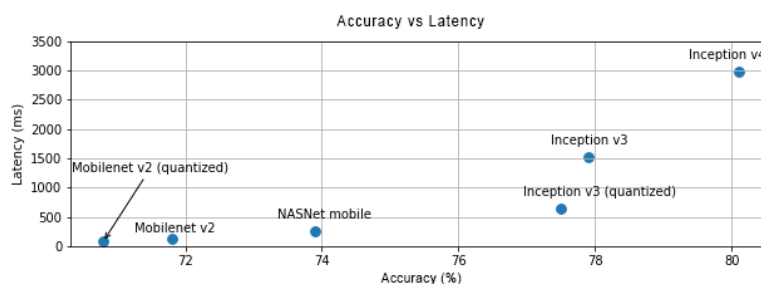
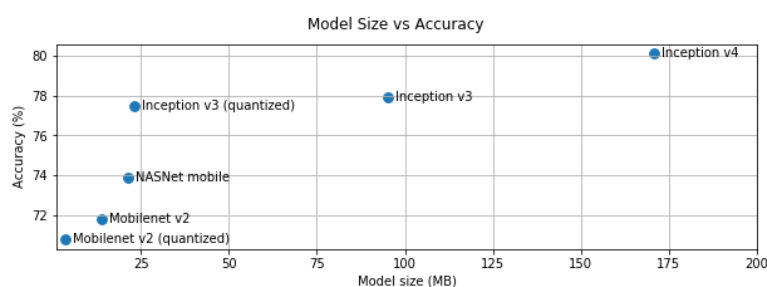
Zip Exported Files

Just a simple step of finalizing packing the whole detection run through process by putting all of the files in one. Some easy lines of codes elaborated in the appendix of this part.

5.9.7. Performance Tuning and Enhancement

This is for when your model doesn't perform well. Mobile and embedded devices have limited computational resources, so it is important to keep your application resource efficient. We have compiled a list of best practices and strategies that can be used to improve a Tensorflow model performance. Involving these techniques to an existing model is as effective as creating a new model but with some enhancements. Going through the same operation steps gives a well edited version of the same output. After applying most of the techniques, the new model adds a new checkpoint because of the larger training in images and real-time.

- ✓ **Adding more images of low performing classes to the training set.** Also taking them from different positions to generalize. This improves detection by great margin. The hazy images despite the lack of precision, they also help to upgrade the detection. Manual labeling is used here as auto label technique isn't always accurate. So it's really about image quality which its aim is to collect images with the intended object at a variety of different angles and lighting conditions.
- ✓ **Train longer for much more steps.** This gives a chance to reduce losses, generalize and perform better. By editing all codes and process to suit the new step target for a better tuning. An important step is to update the pipeline which happens as we copy our new pipeline config so for it to copy the pre-trained pipeline config into the new tune model that is extracted into the newly created folder. After the new training steps the evaluation matrix is projected to notice the improvement of the mean average indicating values.
- ✓ **Change Architecture.** From the Tensorflow model Zoo We choose another architecture that might be slower but with better detection. Simply the change is in the time per frame and the accuracy metric numbers. When using different pre-trained models the most important thing is to ensure the required fields are completed inside the pipeline.config file.
- ✓ **Choose the best model for the task.** Depending on the task, you will need to make a tradeoff between model complexity and size. If your task requires high accuracy, then you may need a large and complex model. For tasks that require less precision, it is better to use a smaller model because they not only use less disk space and memory, but they are also generally faster and more energy efficient. For example, graphs below show accuracy and latency tradeoffs for some common image classification models.



One example of models optimized for mobile devices are MobileNets, which are optimized for mobile vision applications. TensorFlow Hub lists several other models that have been optimized specifically for mobile and embedded devices. We can retrain the listed models by using transfer learning. Then Checking out the transfer learning tutorials using TensorFlow Lite Model Maker.

- ✓ **Profile the model.** Once you have selected a candidate model that is right for your task, it is a good practice to profile and benchmark your model. TensorFlow Lite benchmarking tool has a built-in profiler that shows per operator profiling statistics. This can help in understanding performance bottlenecks and which operators dominate the computation time. we can also use TensorFlow Lite tracing to profile the model in your Android application, using standard Android system tracing, and to visualize the operator invocations by time with GUI based profiling tools.
- ✓ **Optimize the model.** Model optimization aims to create smaller models that are generally faster and more energy efficient, so that they can be deployed on mobile devices. TensorFlow Lite supports multiple optimization techniques, such as quantization.
- ✓ **Evaluate whether your model benefits from using hardware accelerators available on the device.** TensorFlow Lite has added new ways to accelerate models with faster hardware like GPUs, DSPs, and neural accelerators. Typically, these accelerators are exposed through delegate submodules that take over parts of the interpreter execution. TensorFlow Lite can use delegates by: Using Android's Neural Networks API. You can utilize these hardware accelerator backends to improve the speed and efficiency of your model.