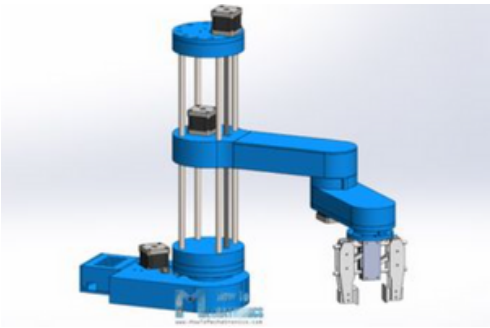
RefleX

Research:

In this project,We want to build a robotic arm that can basically detect the hand's movement and follow it using computer vision.Its use as a start will be for simple tasks such as grabbing little objects and moving them.

In our research the documenttion of these two arms helped us a lot to understand the mechanical concept.

SCARA ROBOTIC ARM:



The robot has 4 degrees of freedom and it's driven by 4 NEMA 17 stepper motors.
here we learnt about reduction system used to increase the speed of movement of the whole robotic arm employing a stepper motor in the base.

YOUPI Robotic Arm :

All the movement is assured by the system belt pulley .
we were able to manipulate this arm at the Aerobotix workspace

https://howtomechatronics.com/projects/scara-robot-how-to-build-your-own-arduino-based-robot/

https://www.hackster.io/Hammouda/robotic-arm-youpi-2020-b2f67a#toc-introduction-0

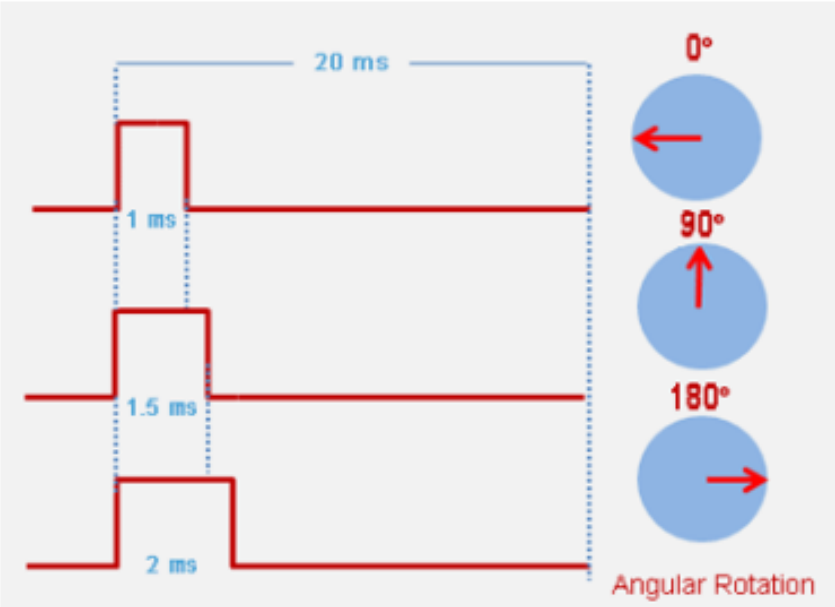
Actuator and Preactuator

1.Servo Motor Working Mechanism

It consists of three parts:
Controlled device
Output sensor
Feedback system

It is a closed-loop system where it uses a positive feedback system to control motion and the final position of the shaft. Here the device is controlled by a feedback signal generated by comparing output signal and reference input signal.
Here the reference input signal is compared to the reference output signal and the third signal is produced by the feedback system. And this third signal acts as an input signal to control the device. This signal is present as long as the feedback signal is generated or there is a difference between the reference input signal and reference output signal. So the main task of servomechanism is to maintain the output of a system at the desired value in the presence of noises.
Interfacing Servo Motors with Microcontrollers:
Servos have three wires coming out of them. Out of which two will be used for Supply (positive and negative) and one will be used for the signal that is to be sent from the MCU. An MG995 Metal Gear Servo Motor which is most commonly used for RC cars, humanoid bots etc. The picture of MG995 is shown below:

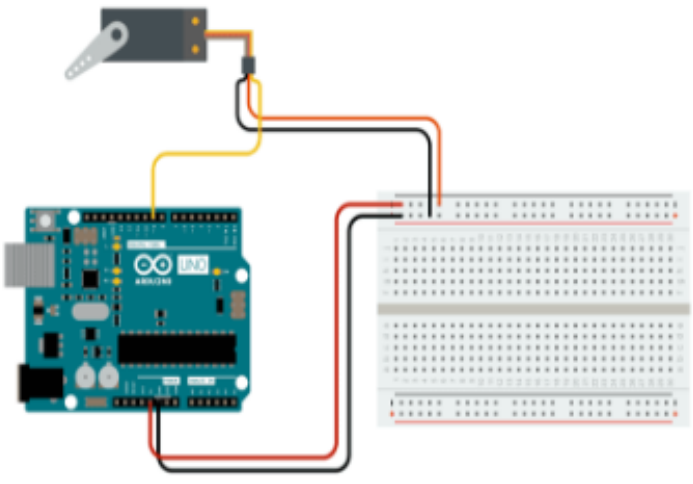
The color coding of your servo motor might differ hence check for your respective datasheet. All servo motors work directly with your +5V supply rails but we have to be careful on the amount of current the motor would consume if you are planning to use more than two servo motors a proper servo shield should be designed.



Servos are clever devices. Using just one input pin, they receive the position from the Arduino and they go there. Internally, they have a motor driver and a feedback circuit that makes sure that the servo arm reaches the desired position. But what kind of signal do they receive on the input pin?
It is a square wave similar to PWM. Each cycle in the signal lasts for 20 milliseconds and for most of the time, the value is LOW. At the beginning of each cycle, the signal is HIGH for a time between 1 and 2 milliseconds. At 1 millisecond it represents 0 degrees and at 2 milliseconds it represents 180 degrees. In between, it represents the value from 0–180. This is a very good and reliable method. The graphic makes it a little easier to understand.

Hardware Required :
Arduino Board
Servo Motor
hook-up wires

Following are the steps to connect a servo motor to the Arduino:
The servo motor has a female connector with three pins. The darkest or even black one is usually the ground. Connect this to the Arduino GND.
Connect the power cable that in all standards should be red to 5V on the Arduino.
Connect the remaining line on the servo connector to a digital pin on the Arduino.



Code for testing the servo motor with Arduino:
The following code will turn a servo motor to 0 degrees, wait 1 second, then turn it to 90, wait one more second, turn it to 180, and then go back.

```
 // Include the Servo library
#include <Servo.h>
// Declare the Servo pin
int servoPin = 3;
// Create a servo object
Servo Servo1;
void setup() {
  // We need to attach the servo to the used pin number
  Servo1.attach(servoPin);
}
void loop(){
  // Make servo go to 0 degrees
  Servo1.write(0);
  delay(1000);
  // Make servo go to 90 degrees
  Servo1.write(90);
  delay(1000);
  // Make servo go to 180 degrees
  Servo1.write(180);
  delay(1000);
}
```

The code simply declares the servo object and then initializes the servo by using the servo.attach() function. We shouldn't forget to include the servo library. In the loop(), we set the servo to 0 degrees, wait, then set it to 90, and later to 180 degrees.

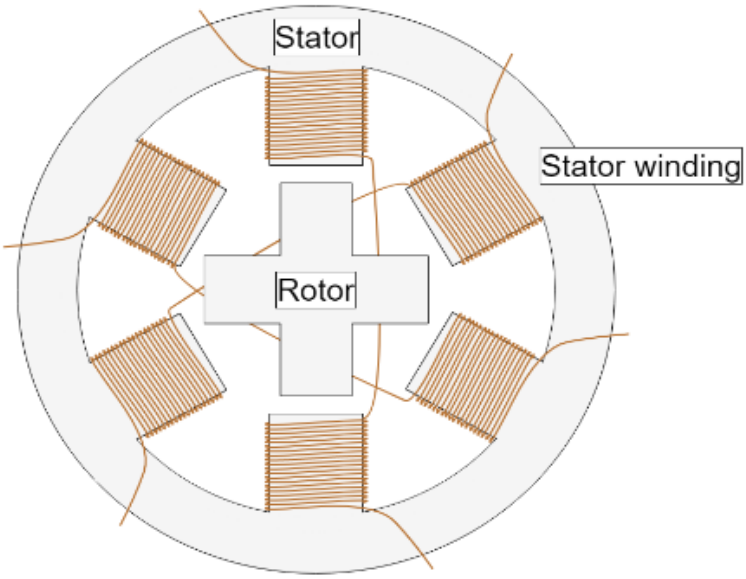
Controlling the exact pulse time

Arduino has a built-in function servo.write(degrees) that simplifies the control of servos. However, not all servos respect the same timings for all positions. Usually, 1 millisecond means 0 degrees, 1.5 milliseconds mean 90 degrees, and, of course, 2 milliseconds mean 180 degrees. Some servos have smaller or larger ranges.
For better control, we can use the servo.writeMicroseconds(us) function, which takes the exact number of microseconds as a parameter. Remember, 1 millisecond equals 1,000 microseconds.

2.Stepper motor :

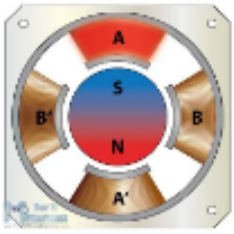
Stepper Motor Working Principles

As all with electric motors, stepper motors have a stationary part (the stator) and a moving part (the rotor). On the stator, there are teeth on which coils are wired, while the rotor is either a permanent magnet or a variable reluctance iron core. We will dive deeper into the different rotor structures later

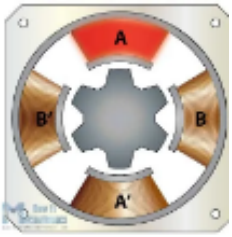


The basic working principle of the stepper motor is the following: By energizing one or more of the stator phases, a magnetic field is generated by the current flowing in the coil and the rotor aligns with this field. By supplying different phases in sequence, the rotor can be rotated by a specific amount to reach the desired final position.

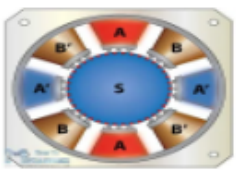
Stepper Motor Types



The Permanent Magnet stepper has a permanent magnet rotor which is driven by the stators windings. They create opposite polarity poles compared to the poles of the rotor which propels the rotor



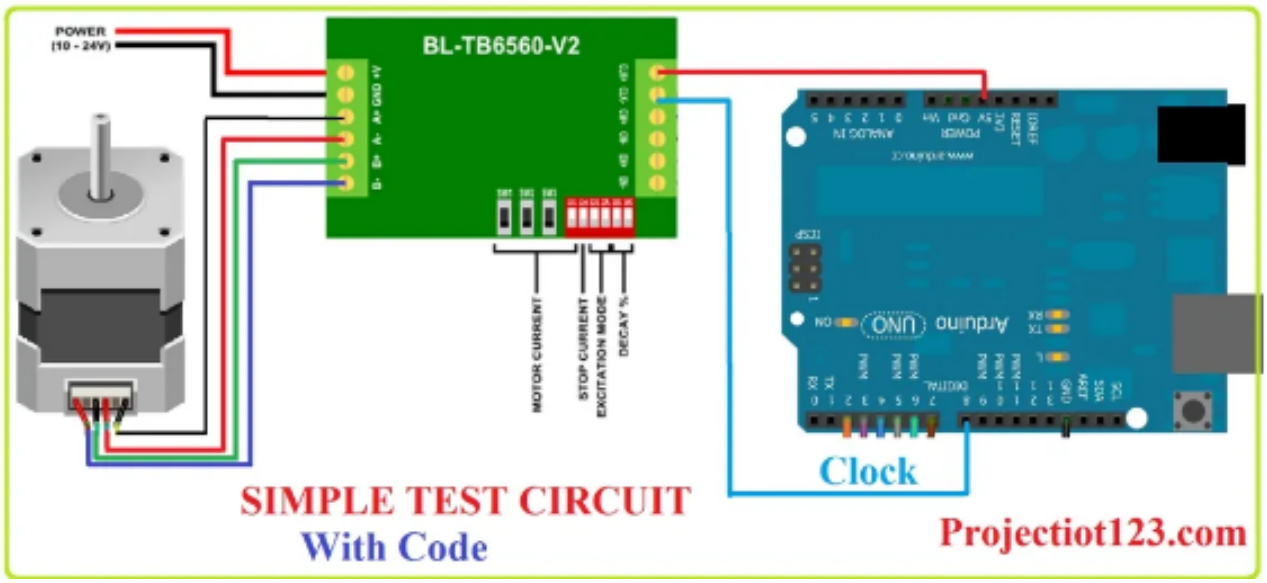
the Variable Reluctant stepper motor uses a non-magnetizes soft iron rotor. The rotor has teeth that are offset from the stator and as we active the windings in a particular order the rotor moves respectively so that it has minimum gab between the stator and the teeth of the rotor



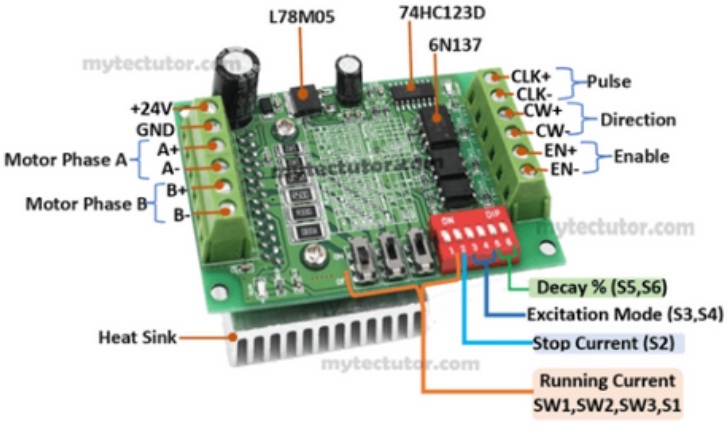
The Hybrid Synchronous motor is combinations of the previous two steppers. It has permanent magnet toothed rotor and also a toothed stator. The rotor has two sections, which are opposite in polarity and their teeth are offset as shown here

Stepper Motor Control

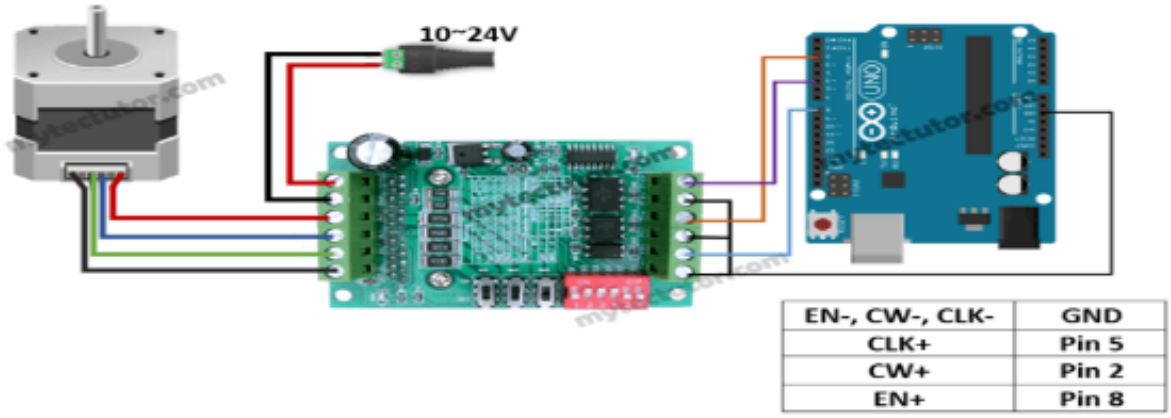
We have seen previously that the motor coils need to be energized, in a specific sequence, to generate the magnetic field with which the rotor is going to align. Several devices are used to supply the necessary voltage to the coils, and thus allow the motor to function properly. Starting from the devices that are closer to the motor we have:
A transistor bridge is the device physically controlling the electrical connection of the motor coils. Transistors can be seen as electrically controlled interrupters, which, when closed allow the connection of a coil to the electrical supply and thus the flow of current in the coil. One transistor bridge is needed for each motor phase.
A pre-driver is a device that controls the activation of the transistors, providing the required voltage and current, it is in turn controlled by an MCU.
An MCU is a microcontroller unit, which is usually programmed by the motor user and generates specific signals for the pre-driver to obtain the desired motor behavior



TB6560 Stepper Motor Driver
Specifications
Working voltage of 10~35V DC. Recommended supply voltage of 24V DC.
Rated maximum output current of ± 3A with 3.5A peak.
Suitable for 2 or 4-phase, 4 or 6 wire stepper motors with a maximum load current of 3A.
Low voltage shutdown, overheating and overcurrent protection circuit to ensure optimal performance.
Adjustable load current protection and decay modes.
Excitation modes: 1/2, 1/8, 1/16 step



Connecting TB6560 Stepper motor driver to Arduino:
The connection is done as shown in the schematic below. I have used a common cathode configuration where all the negative control signals and enable are all connected to ground.



EN-, CW-, CLK-	GND
CLK+	Pin 5
CW+	Pin 2
EN+	Pin 8

The pulse (CLK+), direction (CW+) and enable (EN+) are connected to Arduino pins 5, 2 and 8 respectively.
You can leave the enable pins (EN- and EN+) unconnected meaning that the enable pin is always LOW and the driver is always enabled. However, if you want to control the motor driver using software then you need to connect EN- to ground and EN+ to a digital pin of Arduino as I have done.
Do not connect stepper motors with a current rating of more than 3A to the driver.

Code for controlling TB6560 Stepper motor driver with Arduino:
A microcontroller like Arduino can be used to control the speed, number of revolutions and direction of rotation of the stepper motor. I'll use the code below as an example of how to move the stepper motor from 0° to 360°, then back to 0°, and so on. In my case I have set the driver to 1/8 step mode and using a 12V power supply with 1A.

CODE :

```
const int stepPin = 5;
const int dirPin = 2;
const int enPin = 8;
void setup() {

  // Sets the two pins as Outputs
  pinMode(stepPin,OUTPUT);
  pinMode(dirPin,OUTPUT);

  pinMode(enPin,OUTPUT);
  digitalWrite(enPin,LOW);
```

```
}
void loop() {

  digitalWrite(dirPin,HIGH); // Enables the motor to move in a particular direction
  // Makes 200 pulses for making one full cycle rotation
  for(int x = 0; x < 800; x++) {
 digitalWrite(stepPin,HIGH);
 delayMicroseconds(500);
 digitalWrite(stepPin,LOW);
 delayMicroseconds(500);
  }
  delay(1000); // One second delay

  digitalWrite(dirPin,LOW); //Changes the rotations direction
  // Makes 400 pulses for making two full cycle rotation
  for(int x = 0; x < 800; x++) {
 digitalWrite(stepPin,HIGH);
 delayMicroseconds(500);
 digitalWrite(stepPin,LOW);
 delayMicroseconds(500);
  }
  delay(1000);
}
```

Code description :

First the step (CLK+), direction (CW+) and enable (ENA+) pins are declared with their corresponding Arduino pins as 2, 5 and 8 respectively.

In the setup() section, all the motor control pins are declared as digital OUTPUT. I also enable the driver by setting the enable pin LOW.

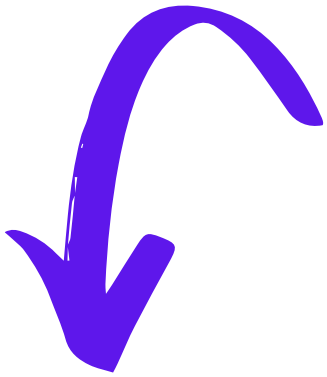
The loop() section determines the number of steps the motor will take. The four lines of code below will send a pulse to the step pin resulting in one microstep.

```
digitalWrite(stepPin, HIGH);
delayMicroseconds(stepDelay);
digitalWrite(stepPin, LOW);
delayMicroseconds(stepDelay);
```

Controlling the number of steps or revolutions:

The for loop repeats the above lines of code a given number of times which represents the steps per revolution. For example, if the driver is set to 1/8 step mode, then the code in the for loop has to be executed 1600 times to get 1 revolution.

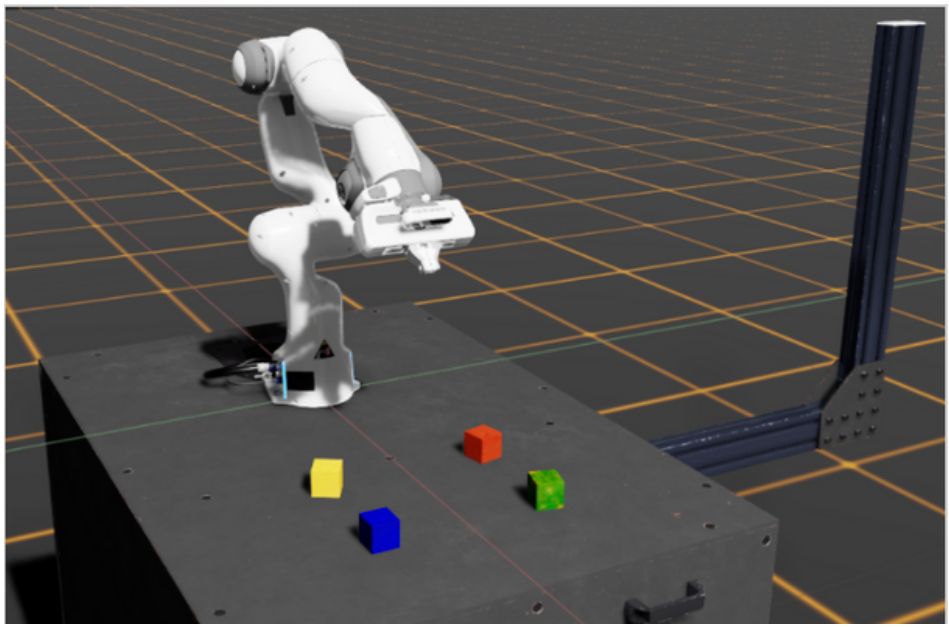
```
for(int i = 0; i < step; i++) {
   digitalWrite(stepPin,HIGH);
   delayMicroseconds(stepDelay);
   digitalWrite(stepPin,LOW);
   delayMicroseconds(stepDelay);
  }
```



EXPLICATION :

I have used the map() function to divide the 360⁰ into the corresponding number of steps to be taken by the motor to complete one revolution.
Controlling motor rotation direction.
The direction of rotation of the stepper motor is determined by setting the direction (CW+) pin either HIGH or LOW and depending on how you connected the stepper motor, when the CW pin is set HIGH, the motor will turn clockwise and when set LOW it turns counter clockwise.
Controlling speed of rotation of the motor.
The speed of the stepper motor is determined by the frequency of the pulses we send to the STEP pin which is set using the delayMicroseconds() function. The shorter the delay, the higher the frequency and therefore the faster the motor runs.

RefleX is a pick and place robotic arm that uses computer vision to detect objects and follow human arm movement.



Object Detection :

To create a computer vision sorting system for our robotic system, we will use OpenCV. To identify objects, the technique to be used is contouring. Contouring is a computer vision technique that simply joins all continuous points of the same color and intensity.

By applying this technique, we can identify objects based on color, shape, and size without using multiple techniques for each parameter. Instead, we can carry out the selection by simply classifying the contour. The process of identifying objects based on our three parameters are elaborated in further detail in this subsection. The block diagram shown in Fig.1 demonstrates the flow of the color sorting algorithm used for our design. The images captured by the camera are in the Red, Green, and Blue (RGB) color space and must be converted to the HSV color space to sort by color because the HSV space allows for more precise color selection.

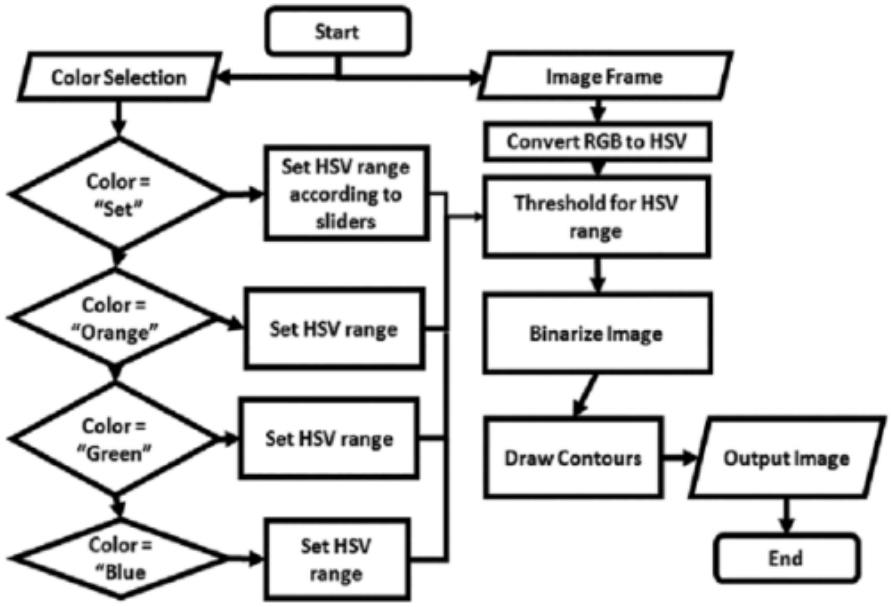


Figure 1: The flowchart demonstrates the algorithm used in order to sort objects according to color using HSV color space.

Once converted to HSV, the image is binarized such that the desired color is assigned a value of one, and the remainder of the image is assigned a value of 0, and then a contour can be drawn about the desired object. After color selection, the object is sorted according to shape. The flowchart given in Fig. 2 summarizes the process of shape sorting. To sort according to shape, we categorize the contour created during color sorting. Three easily distinguishable shapes are selected as our valid possibilities. These shapes are square, triangle, and rectangle as these shapes are easily recognized, but any shape may have been chosen. A polygon is approximated to the contour. The approximated polygon features are compared to our specified features to ascertain whether an object is of the desired shape.

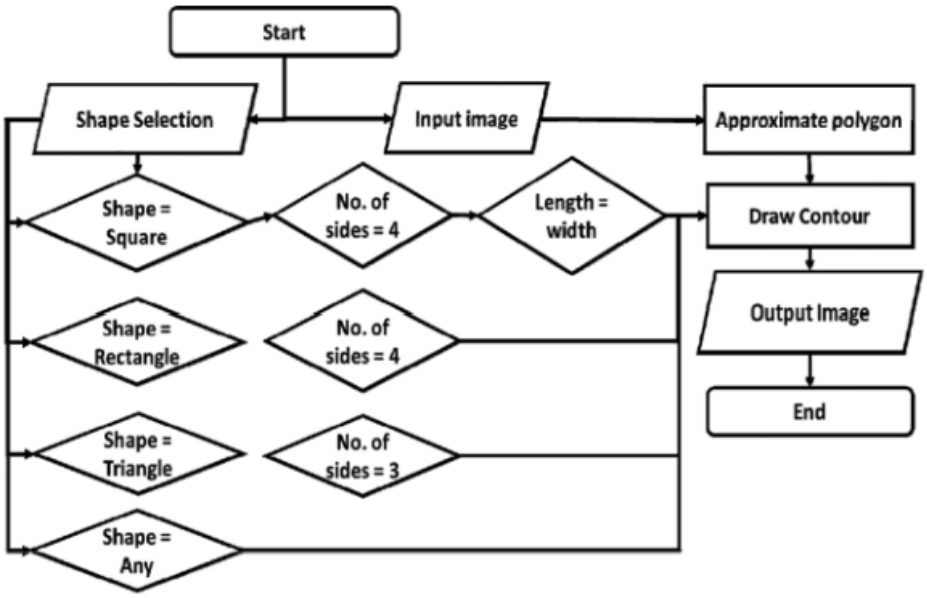


Figure 2: Flowchart demonstrating algorithm used in order to sort objects according to shape by approximating polygon to objects placed in the workspace

To accomplish size sorting, we will create an algorithm in accordance with Fig. 3. To ascertain the size of objects, the operator chooses one of the four sizes they are selecting for. The area constrained within the contour will be compared to the selected size to ascertain if an object complies with the user selection.

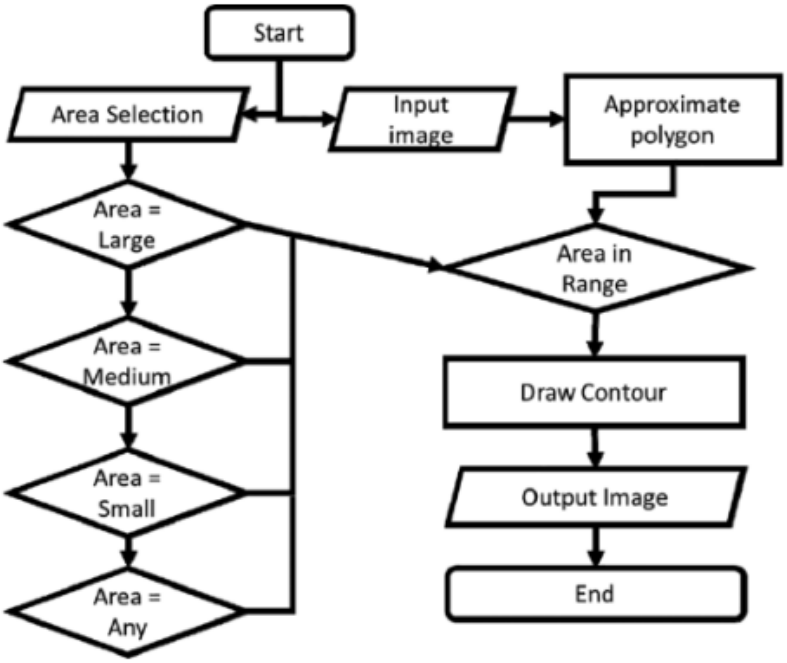
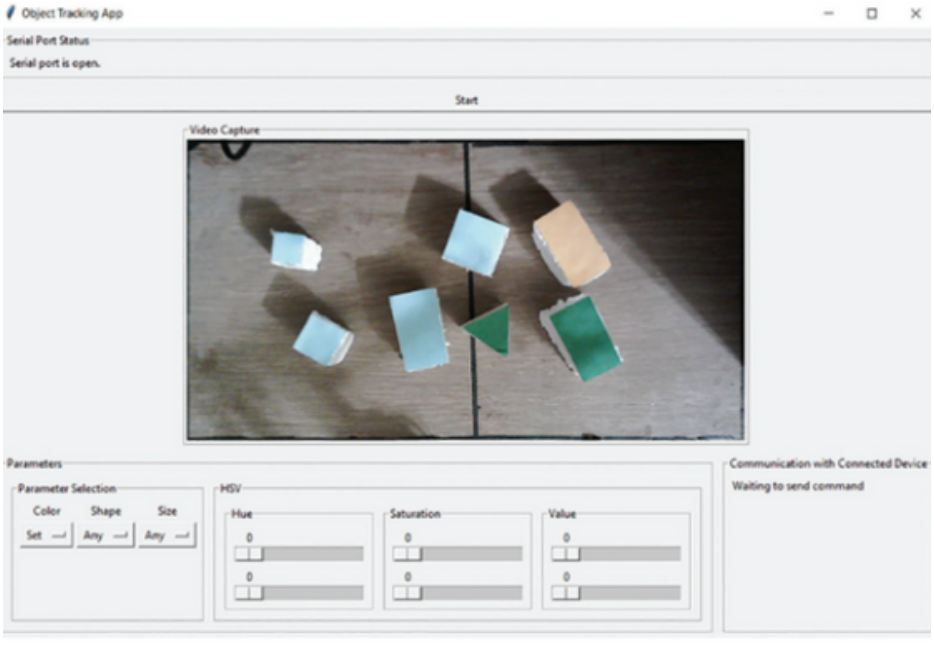


Figure 3: Flowchart demonstrating size sorting algorithm where area taken up by approximated polygon is used in order to identify targeted objects

The algorithms outlined in this subsection act as the backend of this system and are abstracted away by the forward-facing interface with which the system user interacts. The interface design allows users to take full advantage of the complex sorting accomplished by the combination of the three sorting algorithms without requiring any knowledge of the system's inner workings. The interface will be the part of the system that greatly improves interfacing between the human operator and the machine.



Detecting human arm movement :

- After detecting objects, RefleX will fix a webcam to the hand of the user and then the user will make a movement and then Reflex will follow him. This part is the hard one that we will develop an algorithm to detect the arm movement and repeat as seen.
- In this Robot Arm, we will develop an algorithm to control jointed arm robots for pick and place operations using image processing. A web camera will be used to control the shoulder arm within the work volume by taking the images of two reference points on the shoulder arm and object as a third reference point. Computers process the images and determine the center distances between the shoulder arm reference points and the object. Based on the difference in the center distances, the computer will actuate the shoulder arm towards the object until the difference becomes zero. The upper and forearms are actuated using reverse kinematics. The algorithm will be evaluated in a simple computer controlled jointed arm robot for pick and place operations.

Simple applications computer Vision:

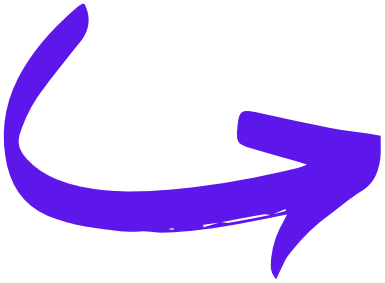
HandTracking

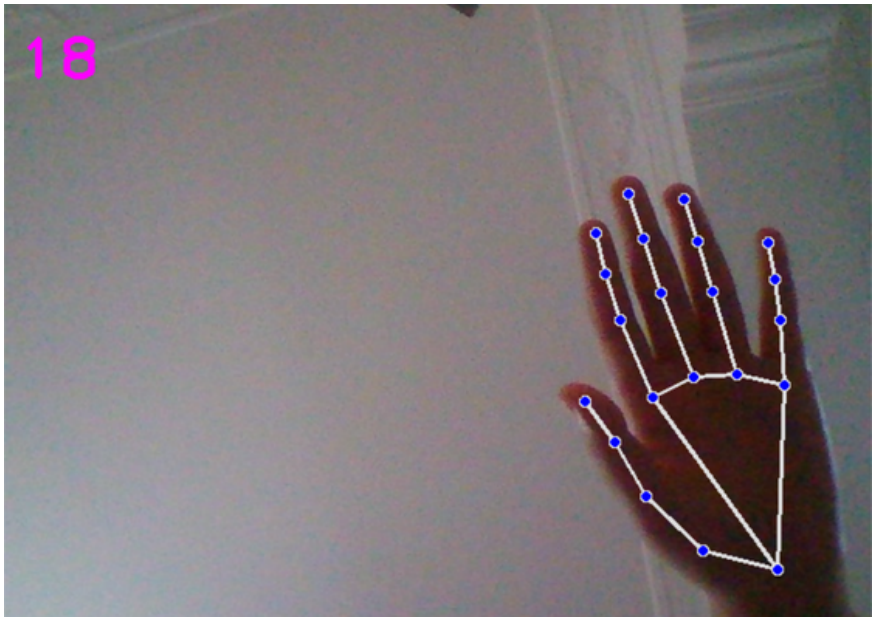
```python
import cv2
import mediapipe as mp
import time


class handDetector():
    def __init__(self, mode=False, maxHands=2, detectionCon=0.5, trackCon=0.5):
        self.mode = mode
        self.maxHands = maxHands
        self.detectionCon = detectionCon
        self.trackCon = trackCon
        self.modelComplex = 1
        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(self.mode, self.maxHands, self.modelComplex,
                                        self.detectionCon, self.trackCon)
        self.mpDraw = mp.solutions.drawing_utils

    def findHands(self, img, draw=True):

        self.results = self.hands.process(img)
        # print(results.multi_hand_landmarks)

        if self.results.multi_hand_landmarks:
            for handLms in self.results.multi_hand_landmarks:
                if draw:
                    self.mpDraw.draw_landmarks(img, handLms,
                                               self.mpHands.HAND_CONNECTIONS)
        return img
```

```python
def findPosition(self, img, handNo=0, draw=True):

    lmList = []
    if self.results.multi_hand_landmarks:
        myHand = self.results.multi_hand_landmarks[handNo]
        for id, lm in enumerate(myHand.landmark):
            # print(id, lm)
            h, w, c = img.shape
            cx, cy = int(lm.x * w), int(lm.y * h)
            # print(id, cx, cy)
            lmList.append([id, cx, cy])

    return lmList
```

```python
def main():
    pTime = 0
    cTime = 0
    cap = cv2.VideoCapture(0)

    detector = handDetector()
    while True:
        success, img = cap.read()
        if (success == True):
            imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img = detector.findHands(imgRGB)
            img = cv2.cvtColor(imgRGB, cv2.COLOR_RGB2BGR)
            lmList = detector.findPosition(img)
            if len(lmList) != 0:
                print(lmList[4])

            cTime = time.time()
            fps = 1 / (cTime - pTime)
            pTime = cTime

            cv2.putText(img, str(int(fps)), (10, 70), cv2.FONT_HERSHEY_PLAIN, 3,
                        (255, 0, 255), 3)

            cv2.imshow("Image", img)
            cv2.waitKey(1)

if __name__ == "__main__":
    main()
```

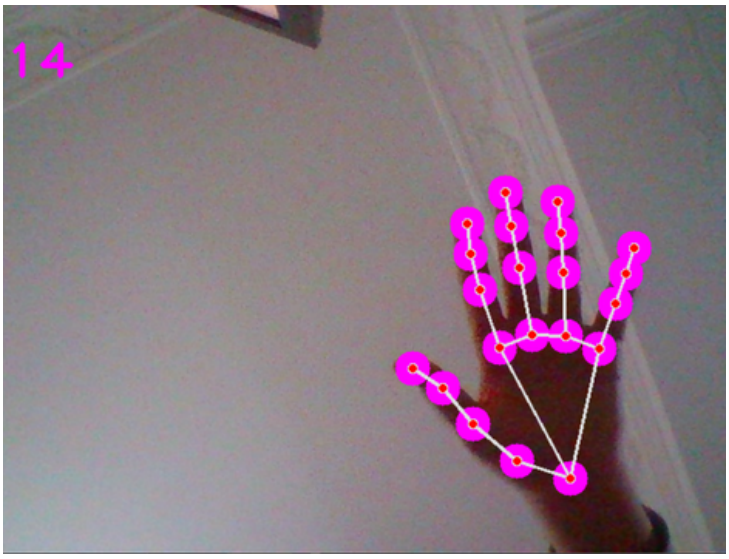
if __name__ == "__main__"





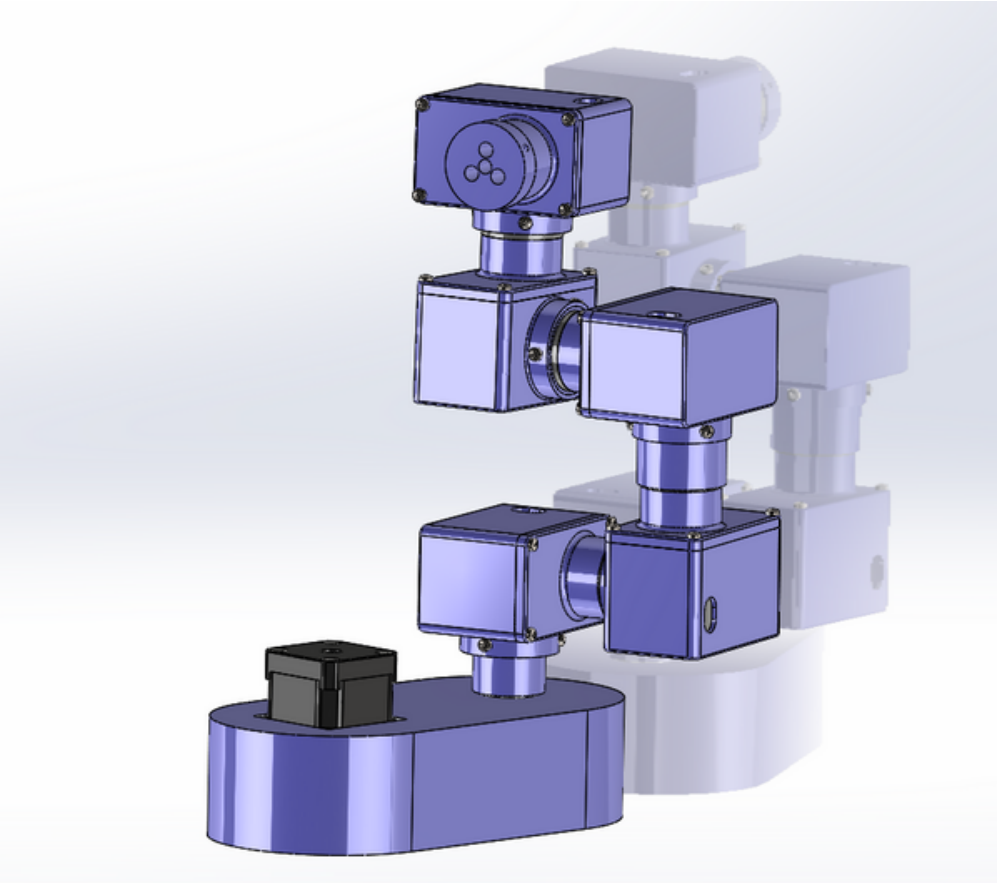
HandTrackingMin

```python
import cv2
import mediapipe as mp
import time

cap = cv2.VideoCapture(0)

mpHands = mp.solutions.hands
hands = mpHands.Hands()
mpDraw = mp.solutions.drawing_utils

pTime = 0
cTime = 0

while True:
    success, img = cap.read()
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    results = hands.process(imgRGB)
    # print(results.multi_hand_landmarks)

    if results.multi_hand_landmarks:
        for handLms in results.multi_hand_landmarks:
            for id, lm in enumerate(handLms.landmark):
                # print(id, lm)
                h, w, c = img.shape
                cx, cy = int(lm.x * w), int(lm.y * h)
                print(id, cx, cy)
                # if id == 4:
                cv2.circle(img, (cx, cy), 15, (255, 0, 255), cv2.FILLED)

        mpDraw.draw_landmarks(img, handLms, mpHands.HAND_CONNECTIONS)

    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime

    cv2.putText(img, str(int(fps)), (10, 70), cv2.FONT_HERSHEY_PLAIN, 3,
                (255, 0, 255), 3)

    cv2.imshow("Image", img)
    cv2.waitKey(1)
```

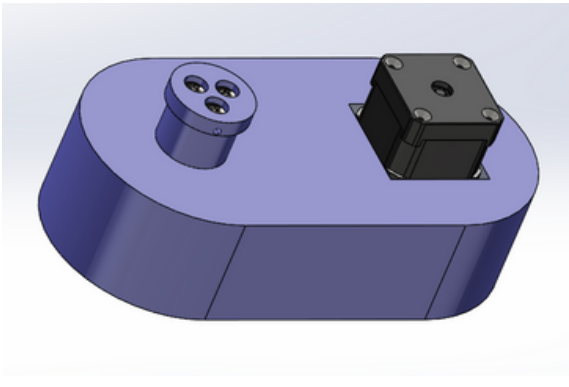


Mechanical Concept

The full Assembly

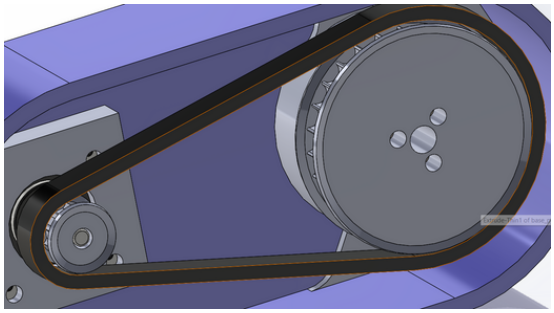


The Base

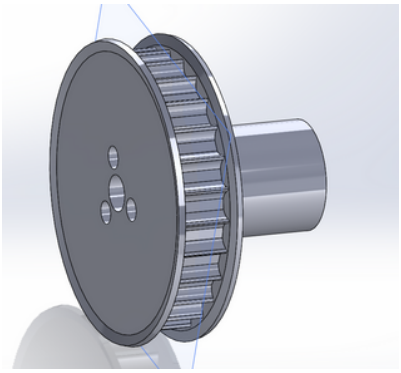


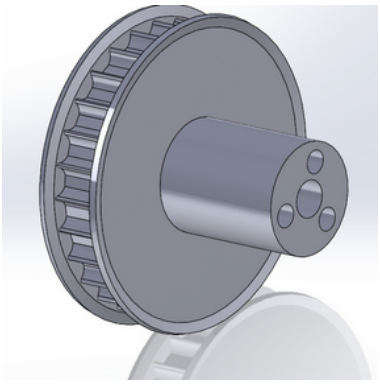
In the base , we chose to work with stepper motor and a reduction system (r=1/3) with belt pulley to increase the torque

The reduction system

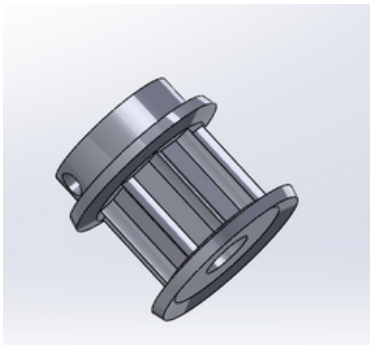


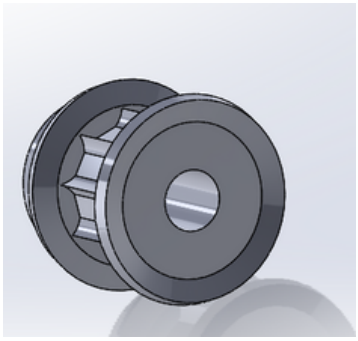
The big Pulley

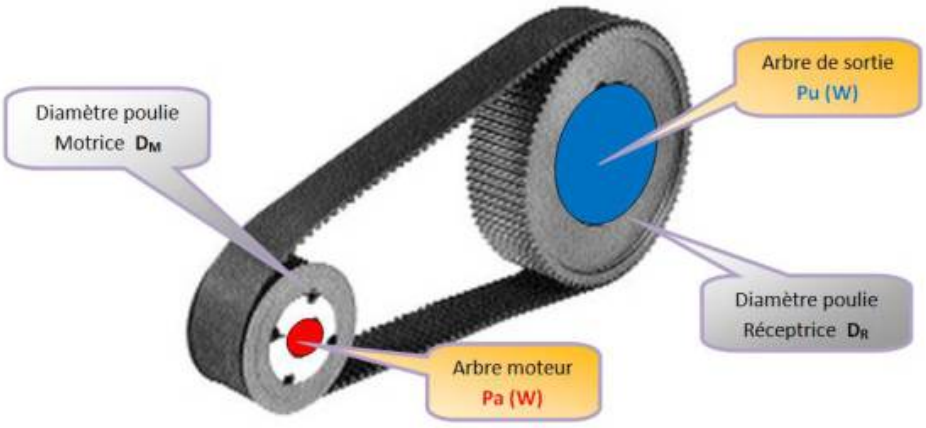


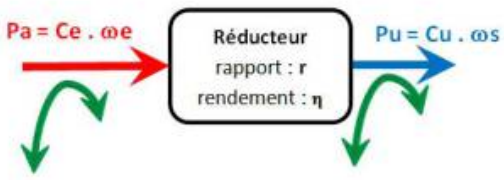


The little Pulley

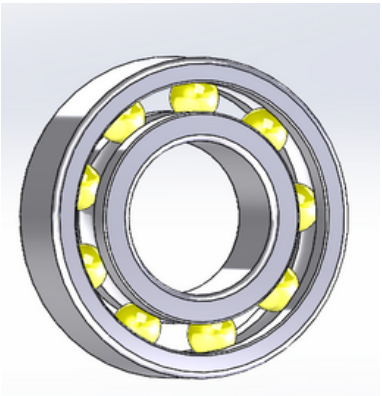


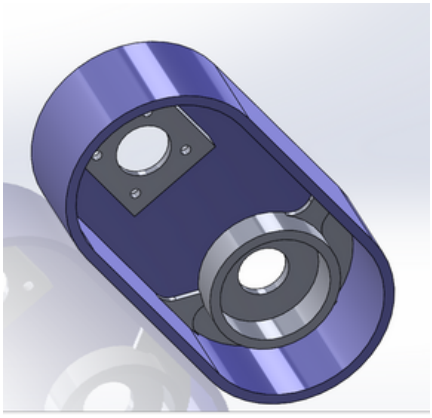


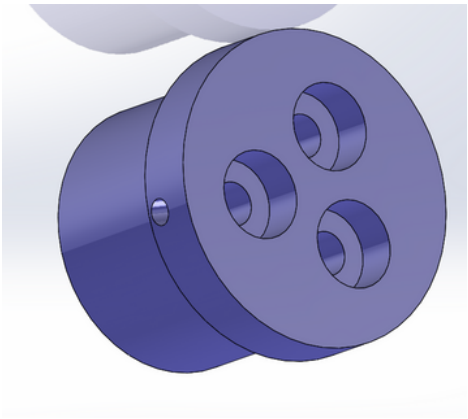




$$\eta = \frac{Pu}{Pa} \qquad r = \frac{D_M}{D_R} = \frac{\omega s}{\omega e}$$

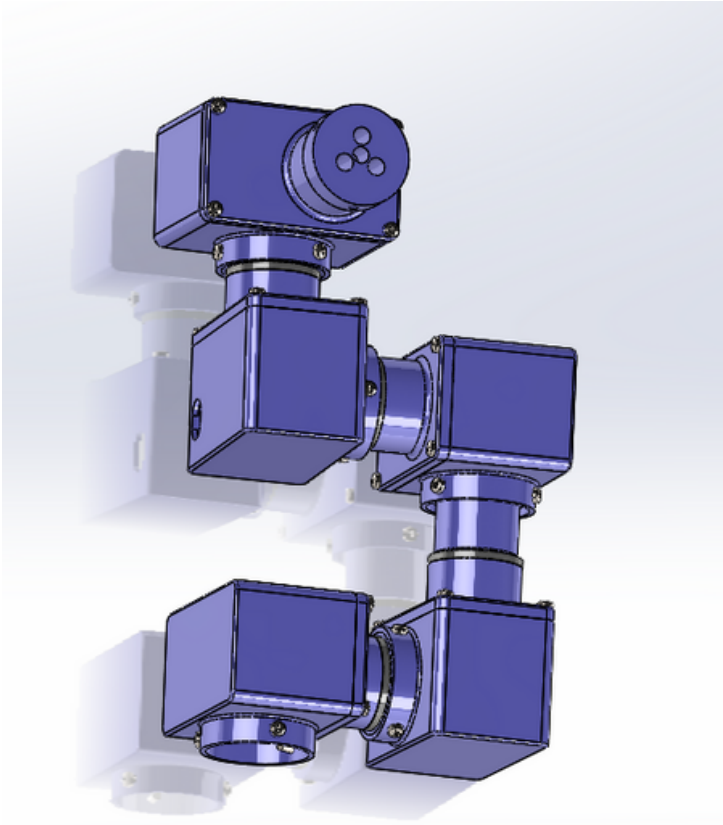




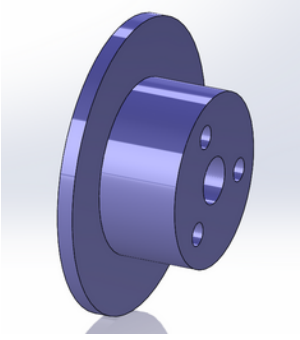


With these parts we are able to transfert the movement and assuring the rotation of the arm.

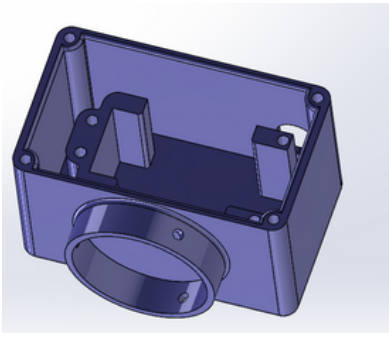
The top Assembly



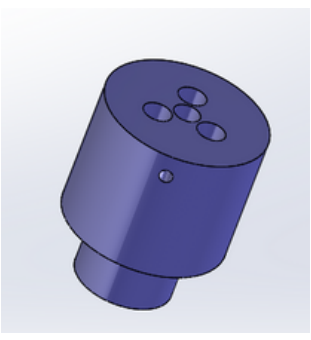
The parts:



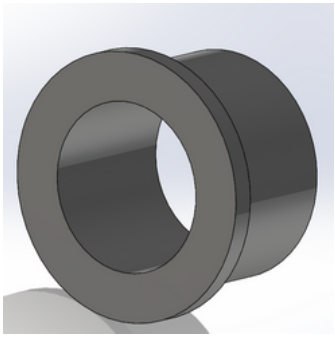
Bottom bearing



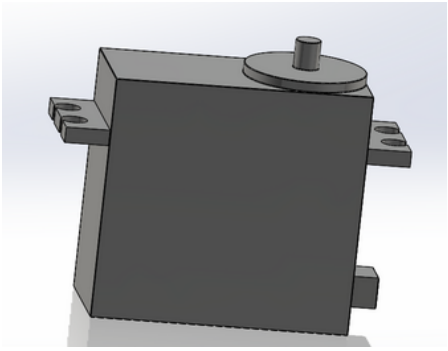
Bottom box



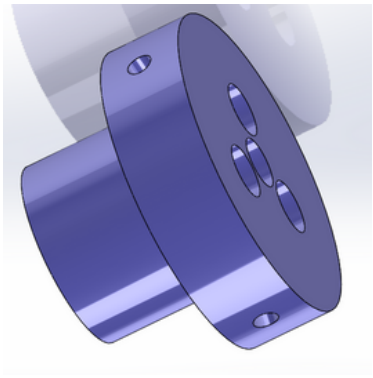
Connector bearing



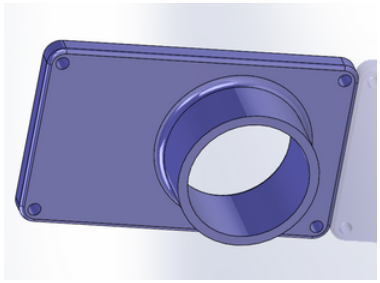
INGUS bearing



Servo motor



top gearing



Box cover

The Box