ISLAMIC UNIVERSITY OF TECHNOLOGY (IUT)

ORGANIZATION OF ISLAMIC COOPERATION (OIC)
DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

EEE 4518 Project Report

3 DOF Robotic Arm

NAME : Fariha Mahjabin Islam STUDENT ID: 200021304

NAME : Al-Hasib Fahim STUDENT ID: 200021312

NAME: Tashnia Islam STUDENT ID: 200021330

NAME: Sahat Abrar Rafij STUDENT ID: 200021333

NAME: Fardun Islam Aumio STUDENT ID: 200021348

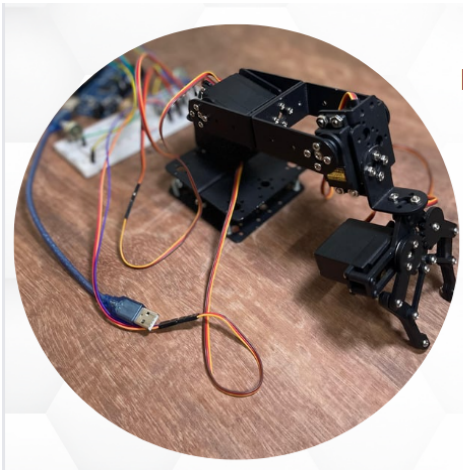


TABLE OF CONTENTS:

OBJECTIVE:

The primary objectives of the robotic arm project are as follows:

Precise Object Manipulation:

Design a robotic arm capable of accurately picking and placing objects with a high degree of precision, showcasing its versatility in various tasks.

Computer Vision Integration:

Implement computer vision using OpenCV to enable the robotic arm to detect and recognize objects. This includes the ability to identify object types and determine their colors.

Weight Measurement Capability:

Integrate load cells and HX711 amplifiers into the robotic arm system to enable accurate weight measurements of manipulated objects.

Comprehensive System Functionality:

Ensure seamless coordination between the robotic arm's mechanical movements, computer vision capabilities, and weight measurement functionalities, creating a holistic system for object manipulation and analysis.

Cost-Effective Design:

Develop the robotic arm system with a focus on cost-effectiveness, considering both material costs and component costs, to enhance the feasibility of implementation in diverse applications.

Reliability and Robustness:

Design the robotic arm system to operate reliably in various environments, ensuring robust performance and minimal downtime in practical applications.

These objectives collectively aim to create a robotic arm system that goes beyond basic manipulation tasks, incorporating advanced capabilities for object recognition and weight analysis, thereby enhancing its utility in industries requiring automation and intelligent decision-making processes.

INTRODUCTION:

A robotic arm is a mechanical system with multiple joints, mimicking the structure and function of a human arm. The degrees of freedom in a robotic arm determine its range of motion and versatility. The kinematics of the arm, involving the study of motion without considering the forces involved, play a crucial role in determining the arm's movements.

Our proposed robotic arm project aims to develop a sophisticated system capable of not only precise object manipulation but also intelligent analysis through advanced features such as Inverse Kinematics, Object Detection, Color Recognition, and Weight Determination.

Inverse Kinematics

Inverse Kinematics (IK) plays a pivotal role in determining the joint configurations required for the end-effector of the robotic arm to reach a specific position or orientation. By implementing IK algorithms, we aim to achieve precise and efficient motion control, enabling the robotic arm to navigate through complex spatial arrangements with accuracy. This will enhance the versatility of the robotic arm, allowing it to adapt to various tasks in diverse environments.

Object Detection

Object detection is crucial for the robotic arm to perceive its surroundings and interact with objects intelligently. Leveraging the capabilities of OpenCV, our system will be equipped to identify and locate objects within its field of view. This feature is integral for tasks such as picking and placing objects with varying shapes and sizes. The integration of object detection will contribute to the overall adaptability and autonomy of the robotic arm, making it a valuable asset in environments where object recognition is paramount.

Color Detection

Color detection complements object detection, providing the robotic arm with the ability to distinguish between objects based on their color properties. By incorporating color detection algorithms into the vision system, the robotic arm can intelligently categorize and interact with objects based on their visual characteristics. This feature enhances the arm's discrimination capabilities, facilitating applications where the color of objects is a critical factor in the manipulation or sorting process.

Weight Determination

The integration of load cells and HX711 amplifiers enables the robotic arm to accurately determine the weight of manipulated objects. This capability is essential for applications where the weight of objects plays a significant role, such as sorting, quality control, or material handling processes. By incorporating weight determination into our robotic arm system, we aim to provide a comprehensive solution for tasks that require not only spatial precision but also an understanding of the physical properties of the manipulated objects.

Computer Vision with OpenCV

OpenCV (Open Source Computer Vision Library) is a powerful tool for computer vision applications. It provides a wide range of functions and algorithms for image processing, object detection, and pattern recognition. In our project, OpenCV will be utilized to enable the robotic arm to "see" and understand its environment, facilitating the detection of objects and their colors.

Keywords:

Inverse kinematics, servo motor, openCV, python, arduino

Inverse Kinematics:

A 3-DOF robotic arm possesses three independent ways it can move in space. These typically correspond to rotations about three orthogonal axes, allowing the end-effector to reach the target position. Each DOF is associated with a joint, and the combination of joint angles determines the arm's configuration.

The robotic arm is modeled as a kinematic chain, where each joint introduces a new DOF. The end-effector's position and orientation are determined by the collective configuration of these joints.

The objective of Inverse Kinematics in a 3-DOF robotic arm is to find the joint angles that achieve a desired end-effector position and orientation. This involves solving a set of nonlinear equations, usually described by trigonometric functions and geometric relations.

For certain 3-DOF robotic arms with simple geometries, analytical solutions might be derived directly. Closed-form expressions relating joint angles to end-effector pose provide an elegant solution, although this is not always feasible for more complex robotic arms.

In cases where analytical solutions are impractical, numerical methods come to the forefront. Iterative techniques like the Newton-Raphson method or optimization algorithms iteratively refine joint angles until they converge to a solution.

Calculation:

L1 be the distance between shoulder servo and elbow servo

L2 be the distance between elbow servo and gripper servo

x is the linear distance from the base of the arm where the object placed

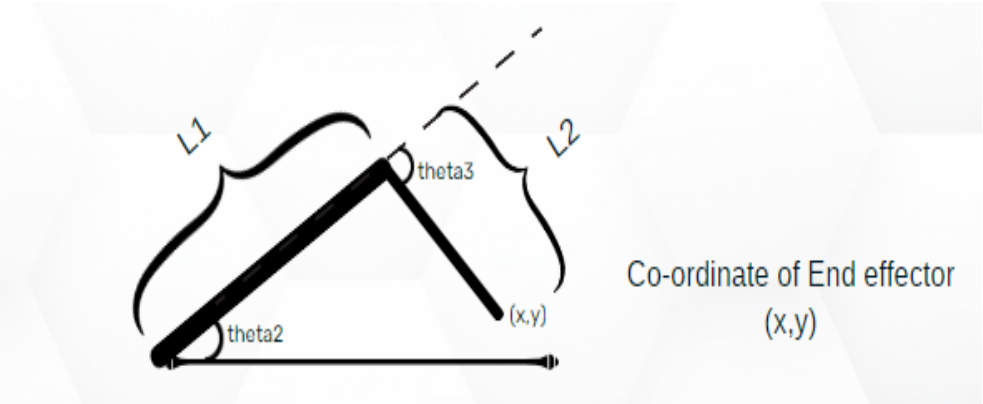
y is the height above which the object placed

theta2 is the angle between the base (horizontal surface) and

the shoulder link (L1)

theta3 is the angle between the direction of the shoulder link

and the elbow link (L2)



theta3 =

$$3.1416 - \frac{acos(-(pow(0.01*L1, 2) + pow(0.01*L2, 2) - pow(0.01*x, 2) - pow(0.01*y, 2)}{(2*0.01*L1*0.01*L2));}$$

theta2 = atan(y/x)+ $$\frac{atan(L2*sin(theta3))}{L1+L2*cos(theta3)}$$

Movement of Our Robotic Arm:

Connection Of Servo Motors:

- We connect the servo motor's signal wire (usually white or yellow) to a PWM-enabled pin on the Arduino (e.g., pin 9).
- Then we connect the servo motor's power (red) and ground (black or brown) wires to the appropriate 5V and GND pins on the Arduino or an external power supply.

Motor 1: Base Motor

The base motor is moved using **potentiometer.** It is used to move the arm in the z axis. Using the potentiometer through the arduino, we can control the angle of the motor 1.

After picking up the object, the base motor is used to rotate the whole arm to another direction for placing the object.

Motor 2: Shoulder Motor

This motor is moved using **Inverse kinematics.** The link attached to this motor is L1. After powering the motor, the motor will move the calculated angle Theta2 in the arduino Code using Inverse Kinematics. As a result the link will bend accordingly.

Motor 3: Elbow Motor

This is the second motor moved using **Inverse Kinematics.** The link attached to this motor is L2. it will move according to the angle Theta3 calculated using Inverse Kinematics in Arduino Code.

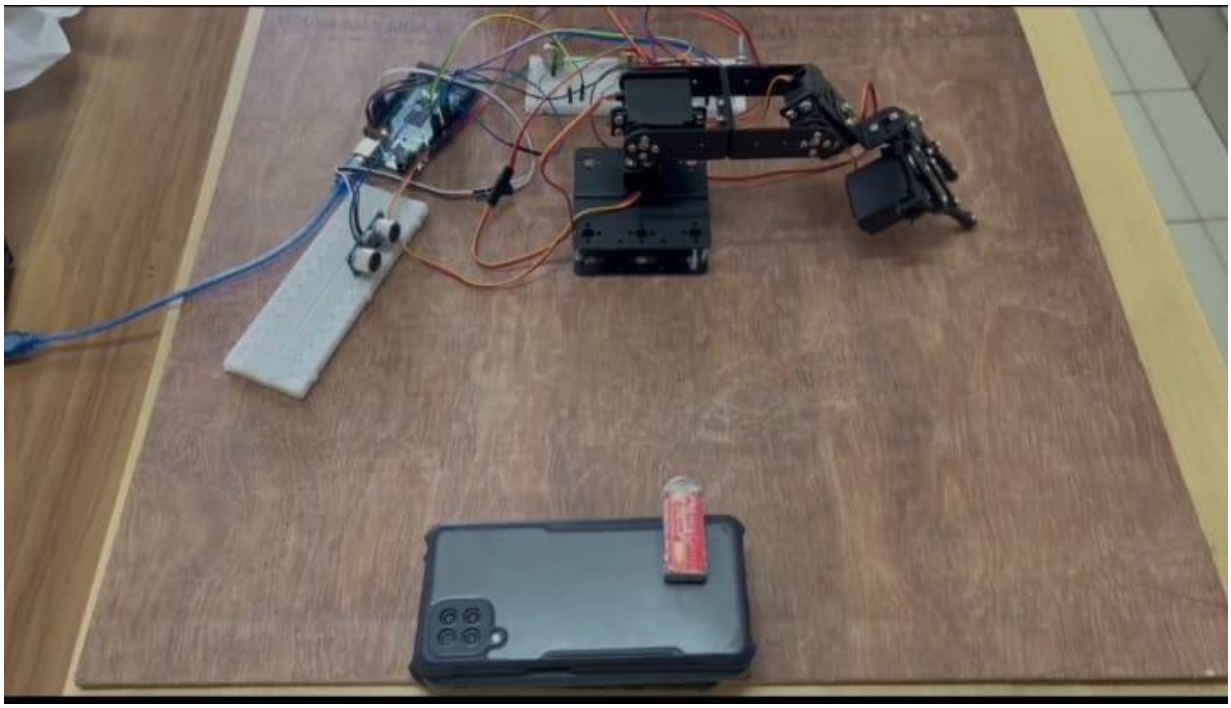
Motor 4: Gripper Motor

Gripper (Claw) control:

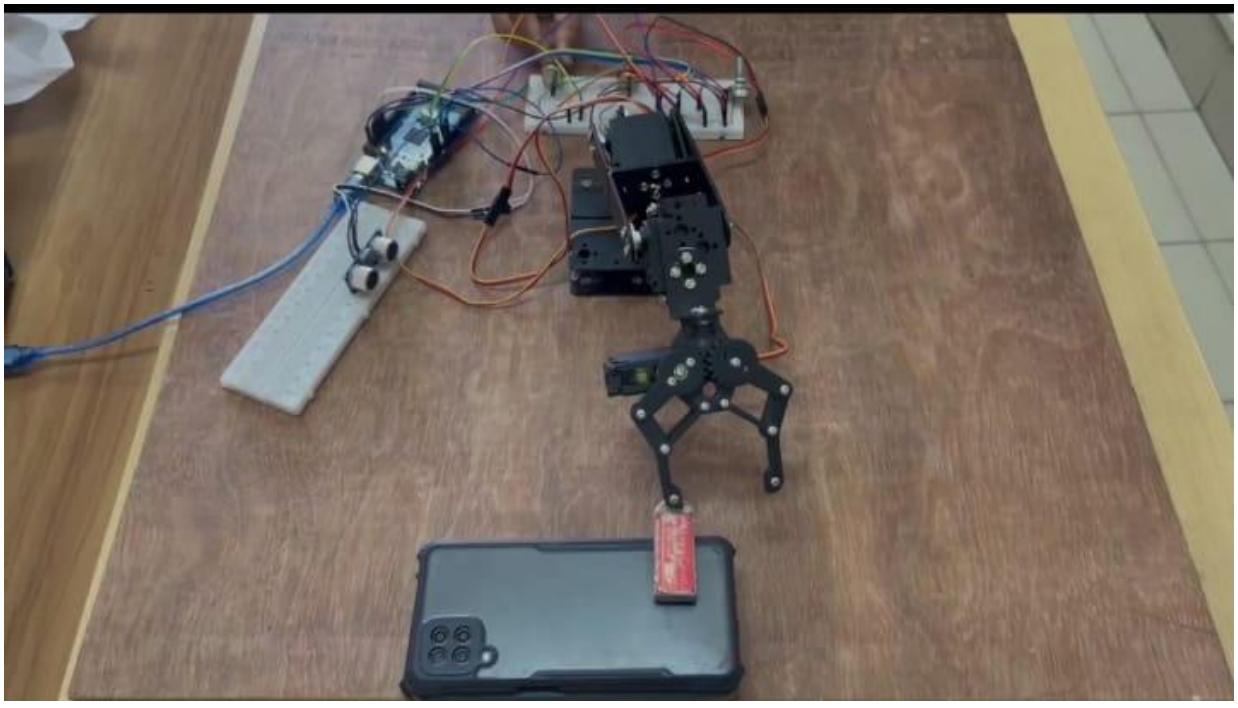
For controlling the gripper,a servo motor was connected to it. We used a potentiometer to move the servo to open and close the claw accordingly. By varying the resistance, it provides position values that can be read by the Arduino to determine the desired claw position.

Pictorial analysis of the Movement of the Robotic Arm:

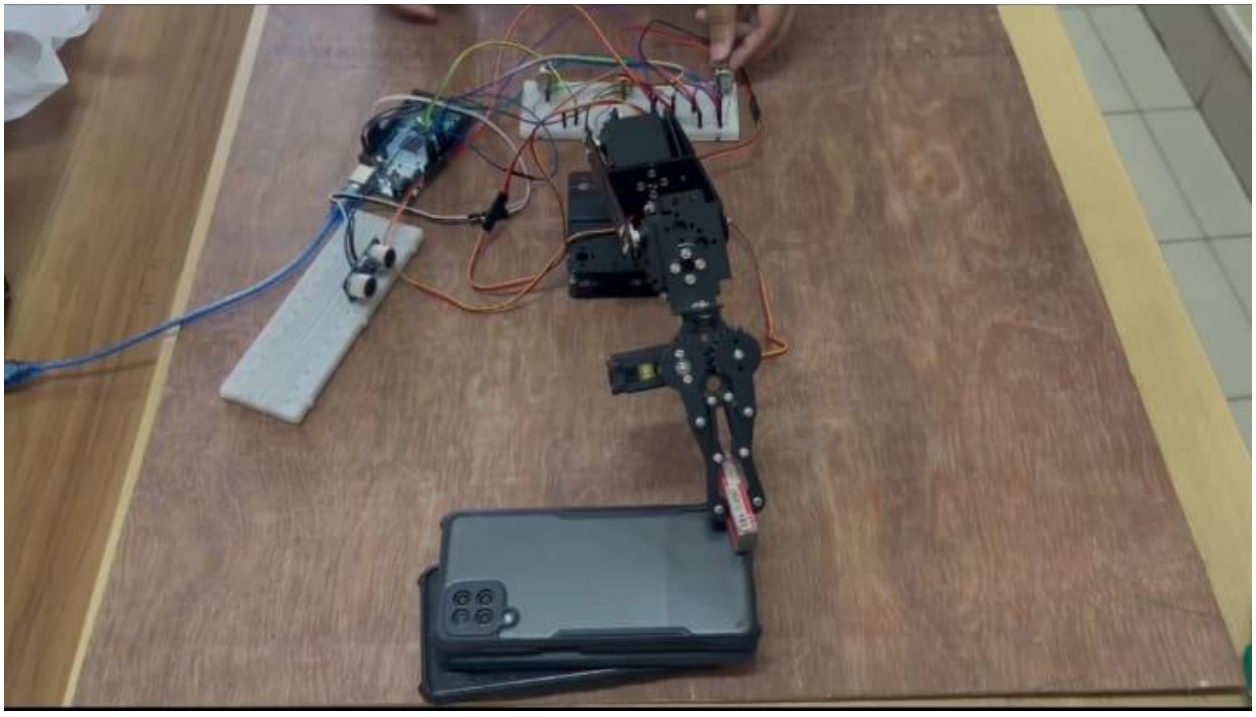
After giving the power supply, the arm will be at its resting position.



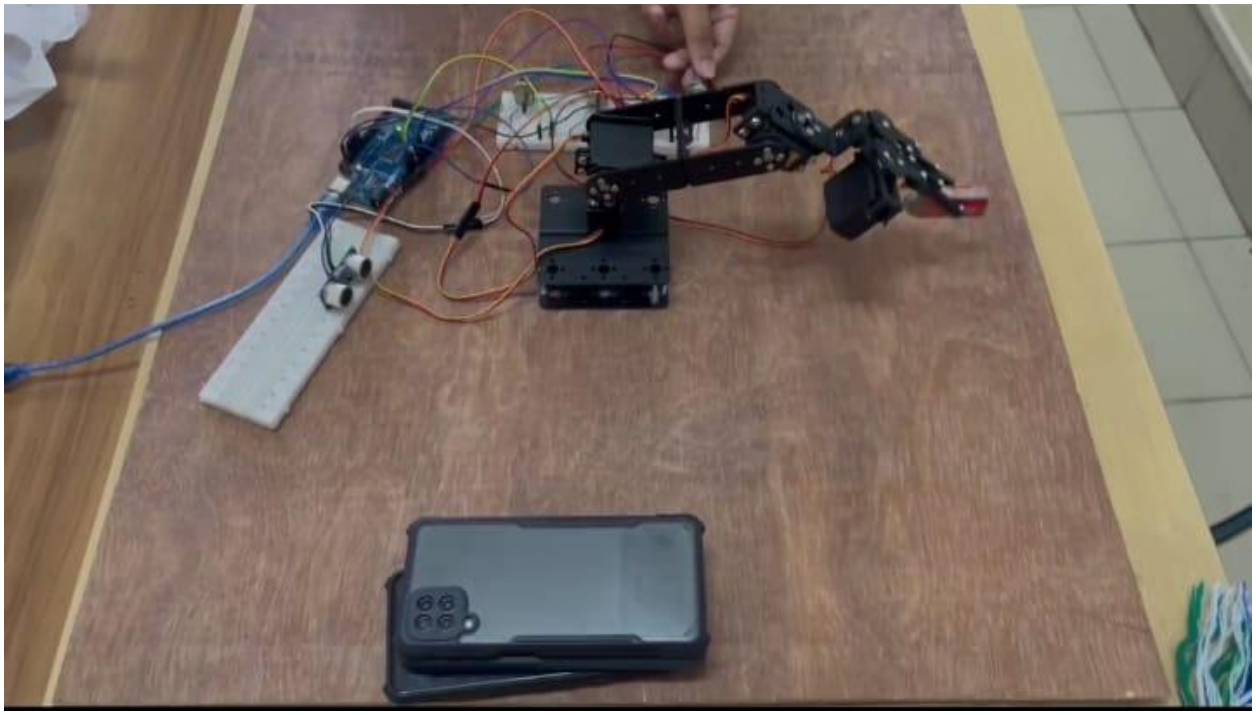
Then the shoulder and elbow servo will operate to go to the position of the object to pick up the object.



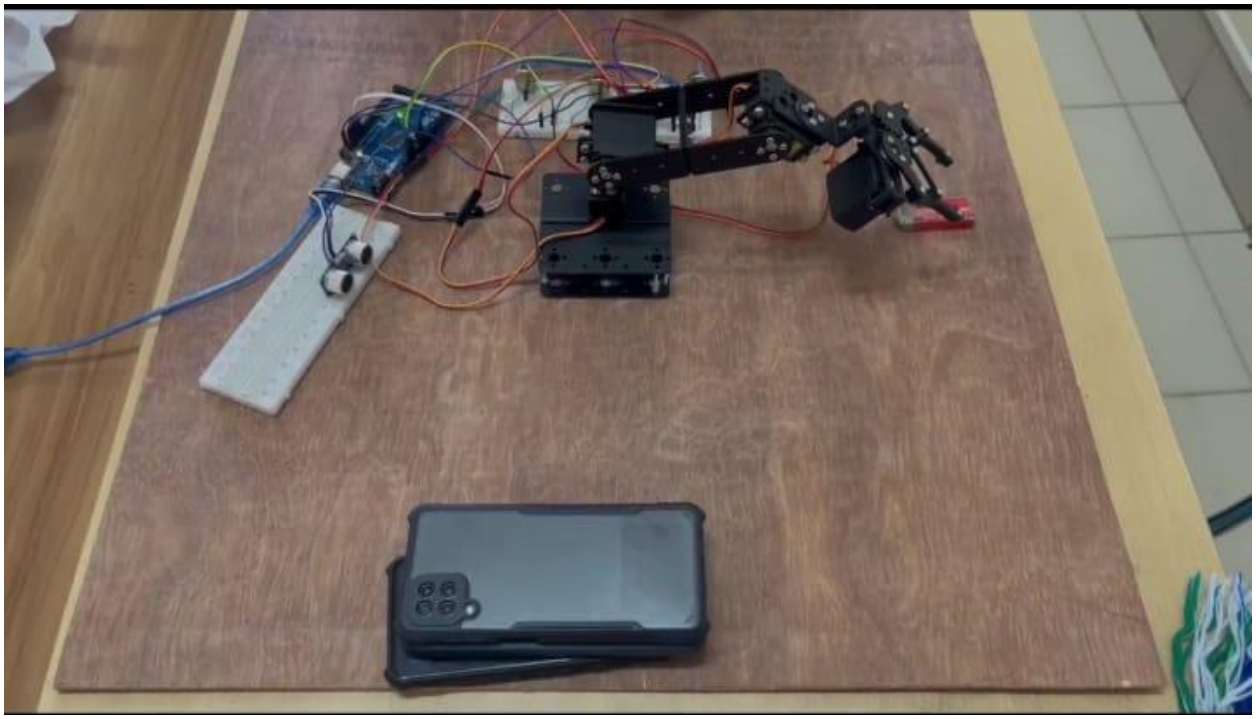
After that, the gripper servo will operate to grip the object



Then the base servo will operate to change the position of the object



Then the gripper servo will open to place the object



Limitations of Movement Through Kinematics:

1. The motors can move only from 0 degree to 180 Degree and vice versa.
2. The end effectors can reach a fixed distance through movement of links.
3. The arm has a fixed workspace.

Code for Movement of the Motors:

Controlling Motor 2 and Motor 3 using Inverse Kinematics:

```
#include <Servo.h>
// Define servo objects for each joint (3 DOF)
Servo shoulderServo;
Servo elbowServo;

const float L1 = 10.0;  // Length of the first link
const float L2 = 8.0;   // Length of the second link
float pi = 3.1416;
bool isConditionMet1 = false;

float x;
float y;
float theta1; // Angle of the first motor (in radian)
float theta1_deg; // Angle of the first motor (in degree)
float theta2; // Angle of the second motor (in radian)
float theta2_deg; // Angle of the second motor (in degree)
float theta3; // Angle of the second motor (in radian)
float theta3_deg; // Angle of the second motor (in degree)
int pos2;
int pos3;

void setup() {

  // Initialize servos and ultrasonic sensor
  shoulderServo.attach(10);
  elbowServo.attach(11);

  shoulderServo.write(90);
  delay(200);
  elbowServo.write(0);
  delay(200);// Initialize serial communication for debugging
  Serial.begin(9600);
}
// If the object is within a certain range, pick it up
void loop() {
  if (!isConditionMet1) {
```

```
    // Adjust this threshold as needed
    move(20, 0);
    delay(2000);
  }
   /*if (!isConditionMet2) {
    // Adjust this threshold as needed
    placeObject(24,0) ;
    delay(2000);
  }  */
}
void move(float x, float y) {
  theta3 = 3.1416- acos(-(pow(0.01*x, 2) + pow(0.01*y, 2) - pow(0.01*L1,
2) - pow(0.01*L2, 2)) / (2 *0.01* L1 * L2));

  theta2 = atan(y/x)+ atan((L2*sin(theta3))/(L1+L2*cos(theta3)));
  theta2_deg =90 + theta2 * 180 / pi;
  theta3_deg = theta3 * 180 / pi;

  Serial.print("theta2_deg: ");
  Serial.println(theta2_deg);
  Serial.print("theta3_deg: ");
  Serial.println(theta3_deg);

  for (int pos2 = 80; pos2 <= theta2_deg; pos2 += 1) {
    // STEPS FOR 1 degree
    shoulderServo.write(pos2);
    Serial.print("pos2: ");
    Serial.println(pos2);
    delay(15);  // Adjust the delay as needed
  }
  for (int pos3 = 0; pos3 <= theta3_deg; pos3 += 1) {
    // STEPS FOR 1 degree
    elbowServo.write(pos3);
    Serial.print("pos3: ");
    Serial.println(pos3);
    delay(2);  // Adjust the delay as needed
  }
  isConditionMet1 = true;
}
```

Explanation:

- The servo library is first included in the arduino library.
- The variables L1, L2 , pi, x, y, theta2 , theta3 and boolean flag are declared after that.
- In the setup function, the servos are attached to the PWM pins of the arduino consecutively.
- Then in the main void loop, the function Move function is called with the values of x and y.
- In the Move function, the **inverse kinematics** formula for calculating theta3 and theta2 have been provided.
- Then the calculated angles are converted to degrees.
- In the two consequent for loops, the motors are made to move from their initial position to the required theta2 and theta3 consecutively.
- The pos2 and pos3 values are written to the **motor 2 and motor 3** consecutively. In this way we can control the speed of the motors by giving delays after every one degree.

Controlling the Motor 1 and Motor 4 using potentiometer:

Connection of the Potentiometer:

- We connect one leg of the first potentiometer to the 5V pin on the Arduino.
- Then connect the other leg of the first potentiometer to the GND pin on the Arduino.
- And the middle leg of the first potentiometer to an analog input pin (e.g., A0) on the Arduino.

Code:

```
#include <Servo.h>
Servo servo1;  // create servo object to control the servo
int potpin = A0;  // analog pin connected to the potentiometer
int val;  // variable to store the potentiometer value
int angle;  // variable to store the angle for the servo

void setup() {
  servo1.attach(9);  // attach the servo to digital pin 9
  Serial.begin(9600);  // initialize serial communication at 9600 baud
rate
}
void loop() {
```

```
  // Read the potentiometer value
  val = analogRead(potpin);
  // Map the potentiometer value to servo angle
  angle = map(val, 0, 1023, 0, 180);  // map the potentiometer value to
servo angle (0 to 180 degrees)
  // Set the servo angle
  servo1.write(angle);  // set the angle for the servo
  // Print servo angle to the serial monitor
  Serial.print("Angle: ");
  Serial.println(angle);
  delay(10);  // wait 10 milliseconds before updating the servo
}
```

Explanation:

- **#include <Servo.h>**: This line includes the Servo library, enabling control of servo motors.
- **Servo clawServo;**: Declares an instance of the Servo class called **clawServo** to control the servo motor.
- **int potPin = A0**;: Defines the analog input pin A0 to which the potentiometer is connected.
- **void setup()**: The **setup()** function attaches the servo motor to pin 9 of the Arduino.
- **void loop()**: The **loop()** function continuously performs the following:
    - Reads the analog value from the potentiometer using **analogRead(potPin).**
    - Maps the potentiometer value (ranging from 0 to 1023) to the servo angle range (0 to 180 degrees) using **map().**
    - Controls the servo motor's position based on the mapped angle using **clawServo.write(angle).**
    - Adds a small delay for stability using **delay(100).**

This code will read the potentiometer's position and map its values to control the servo motor connected to pin 9 of the Arduino, which could simulate the movement of a robotic claw based on the potentiometer's position.

This code is integrated with the main code for two motors.

Object distance measurement:

For this, we used an ultrasonic sensor(HC-SR04). This sensor is used for distance measurement in various projects. It consists of an ultrasonic transmitter, receiver, and control circuitry on a breadboard. This sensor can be used for robotics, automation, obstacle detection, distance monitoring etc.

In our 3-DOF robotic arm, we used the ultrasonic sensor with an Arduino Uno connected to a laptop**. By using this, we will get the desired position of the end-effector, that is, the x coordinate of the position of the object.**

For this, we need–

- Arduino Uno
- Ultrasonic sensor-HC-SR04
- USB cable to connect the Arduino to laptop
- Arduino IDE

Connection:

1. We connect the VCC pin of the ultrasonic sensor to 5V on the Arduino Mega.
2. And the GND pin of the ultrasonic sensor to GND on the Arduino Mega.
3. Then we connect the Trig pin of the ultrasonic sensor to a digital pin on the Arduino (e.g., Pin 7).
4. And the Echo pin of the ultrasonic sensor to another digital pin on the Arduino (e.g., Pin 8).

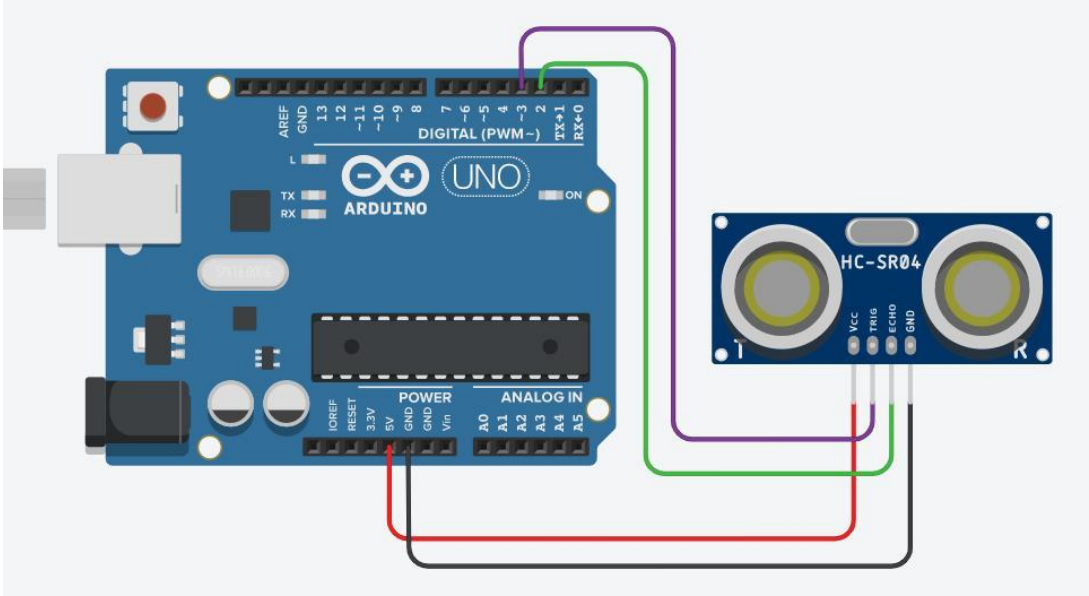


Fig: sample of an HC sensor connection

Working procedure:

1. The ultrasonic sensor detects objects in front of it by measuring the time it takes for an ultrasonic pulse to travel from the sensor to the object and back.
2. The distance to the object is calculated based on the time it takes for the echo signal to return.
3. At first the sensor is triggered by setting the Trig pin (Trigger) to a HIGH state for a brief period (10 microseconds) and then setting it back to LOW.
4. The sensor sends out an ultrasonic pulse, which travels through the air, hits an object (if there's any object in its path), and bounces back to the sensor.
5. The Echo pin of the sensor is connected to the Arduino and is used to measure the time taken for the pulse to return.
6. The distance is calculated by dividing the time taken by 2 and then multiplying by the speed of sound in the air (approximately 29.1 microseconds per centimeter).
7. The calculated distance is then printed to the Serial Monitor, and you can see it on your laptop screen.

Arduino code:

```
#define trigPin 7
#define echoPin 8

bool objectDetected = false;

void setup() {
  Serial.begin(9600); // Initialize serial communication
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop() {
  if (!objectDetected) {
    long duration, distance;

    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
```

```
    duration = pulseIn(echoPin, HIGH);
    distance = (duration / 2) / 29.1; // Convert to centimeters

    Serial.print("Distance: ");
    Serial.print(distance);
    Serial.println(" cm");

    objectDetected = true; // Set flag to indicate object detected

    // Break the loop after measuring and displaying the distance
    while (true) {
      // Infinite loop to stop further iterations
    }
  }
}
```

Explanation of the code:

1. **trigPin** and **echoPin** are defined as 7 and 8 respectively, representing the trigger and echo pins of the ultrasonic sensor.
2. **objectDetected** is a boolean flag used to control whether the object's distance has been detected.
3. **pinMode(trigPin, OUTPUT)** and **pinMode(echoPin, INPUT)** set the trigger pin as an output and the echo pin as an input.
4. The **loop()** function contains an **if** statement that checks if the **objectDetected** flag is **false**. This flag indicates whether the object's distance has been detected or not.
5. The duration of the pulse returning from the object is measured using **pulseIn()** function.
6. **objectDetected** is set to **true**, indicating that the object's distance has been detected.
7. After setting **objectDetected** to **true**, the code enters an infinite loop (**while (true)**) to stop further iterations of the loop, effectively breaking the loop and preventing additional distance measurements.
8. This code structure allows for a single measurement and display of the distance once the **objectDetected** flag is **false,** and it breaks the loop immediately after displaying that single distance.

Finally, we integrated this code with the main code of our robotic arm to ensure the proper working of the arm.

Weight Measurement with Load Cells:

Load cells are sensors designed to measure force or weight. They operate on the principle of deformation of a material under a load, leading to a change in electrical resistance. Here's a breakdown of the working principles:

- Strain Gauge Configuration:

Load cells typically contain strain gauges, which are thin conductive elements applied to a flexible material. When the load cell deforms under a load, the strain gauges also deform, causing a change in their electrical resistance.

- Wheatstone Bridge Circuit:

Strain gauges are configured in a Wheatstone bridge circuit. A Wheatstone bridge is a network of four resistive arms forming a diamond shape. Changes in the resistance of the strain gauges lead to imbalances in the bridge circuit.

- Balanced and Unbalanced States:

In the absence of a load, the Wheatstone bridge is balanced, and there is no output voltage. When a load is applied, the bridge becomes unbalanced, resulting in a voltage output proportional to the applied force.

- Output Signal:

The output signal is typically very small, often in the millivolt range. This signal needs to be amplified and converted into a usable form for further processing.

<u>Working Principles of HX711 Amplifiers:</u>

The HX711 is a specialized amplifier designed to work with load cells. Its primary functions include amplifying the weak signals from the load cells and converting them into a digital format. Here's how it works:

- Instrumentation Amplifier:

The HX711 includes an instrumentation amplifier, which amplifies the differential voltage output from the load cell. This amplification is crucial because the signals from load cells are usually in the millivolt range.

- Analog-to-Digital Converter (ADC):

After amplification, the HX711 digitizes the analog signal using an ADC (Analog-to-Digital Converter). The HX711 is often equipped with a high-resolution ADC, such as 24-bit.

- Digital Output:

The HX711 provides a stable and noise-resistant digital output, typically in a two's complement format. This digital signal can be read by a microcontroller (e.g., Arduino) for further processing.

- Serial Communication:

The HX711 communicates with the microcontroller using a simple serial interface, usually with two pins for data (DT) and clock (SCK). This makes it easy to interface with microcontrollers.

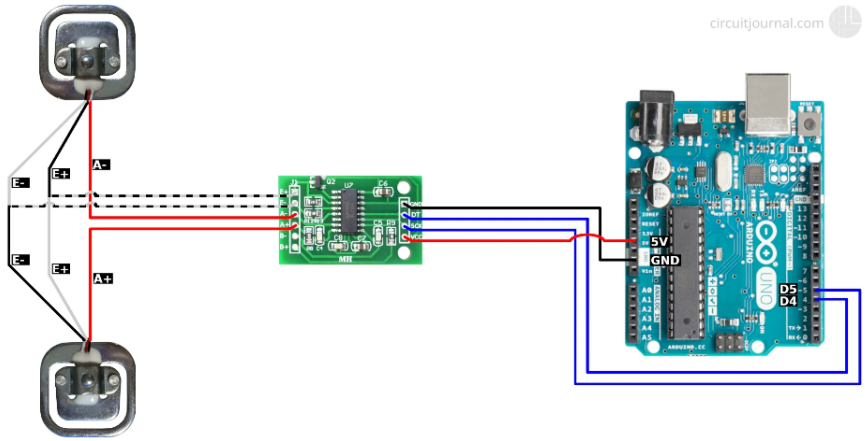
- Gain Selection:

The HX711 often includes selectable gain settings to accommodate different load cell configurations and measurement ranges. This allows for flexibility in using various load cells with different sensitivities.

- Tare Functionality:

The HX711 often has a tare function, allowing you to zero out any initial offset or load on the load cell before taking measurements.

Connection Diagram:



Combining two 3-wire load cells in order to get precise weight data, we have used the HX-711 amplifier module to send the data from load cells to the Arduino. 3-wire load cell has half of a Wheatstone bridge. To make a full bridge, we had to use at least two load cells or strain gauges.

Calibration Procedure:

Arduino Code:

```
#include "HX711.h"
#define DOUT_PIN 2    // Connect HX711 DAT pin to digital pin 2
#define SCK_PIN 3     // Connect HX711 SCK pin to digital pin 3
HX711 scale;
void setup() {
  Serial.begin(9600);
  Serial.println("Place a known weight on the scale for calibration.");
  delay(2000);
}
void loop() {
  if (scale.is_ready()) {
    float calibration_factor = 0; // Your initial calibration factor
    Serial.print("Reading: ");
    float weight = scale.get_units(10); // Get average reading over 10
samples
    Serial.print(weight);
    Serial.println(" grams");
    if (Serial.available()) {
      char command = Serial.read();
      if (command == 'c' || command == 'C') {
        Serial.println("Calibration Mode: Enter known weight (grams): ");
        while (!Serial.available()) {
          delay(100);
        }
        float knownWeight = Serial.parseFloat();
        calibration_factor = knownWeight / weight;
        scale.set_scale(calibration_factor);
        Serial.print("Calibration Factor: ");
        Serial.println(calibration_factor);
        Serial.println("Calibration complete!");
      }
    }
  } else {
    Serial.println("Error: Unable to detect HX711. Please check your
connections.");
    delay(5000);
```

```
  }
  delay(100);
}
```

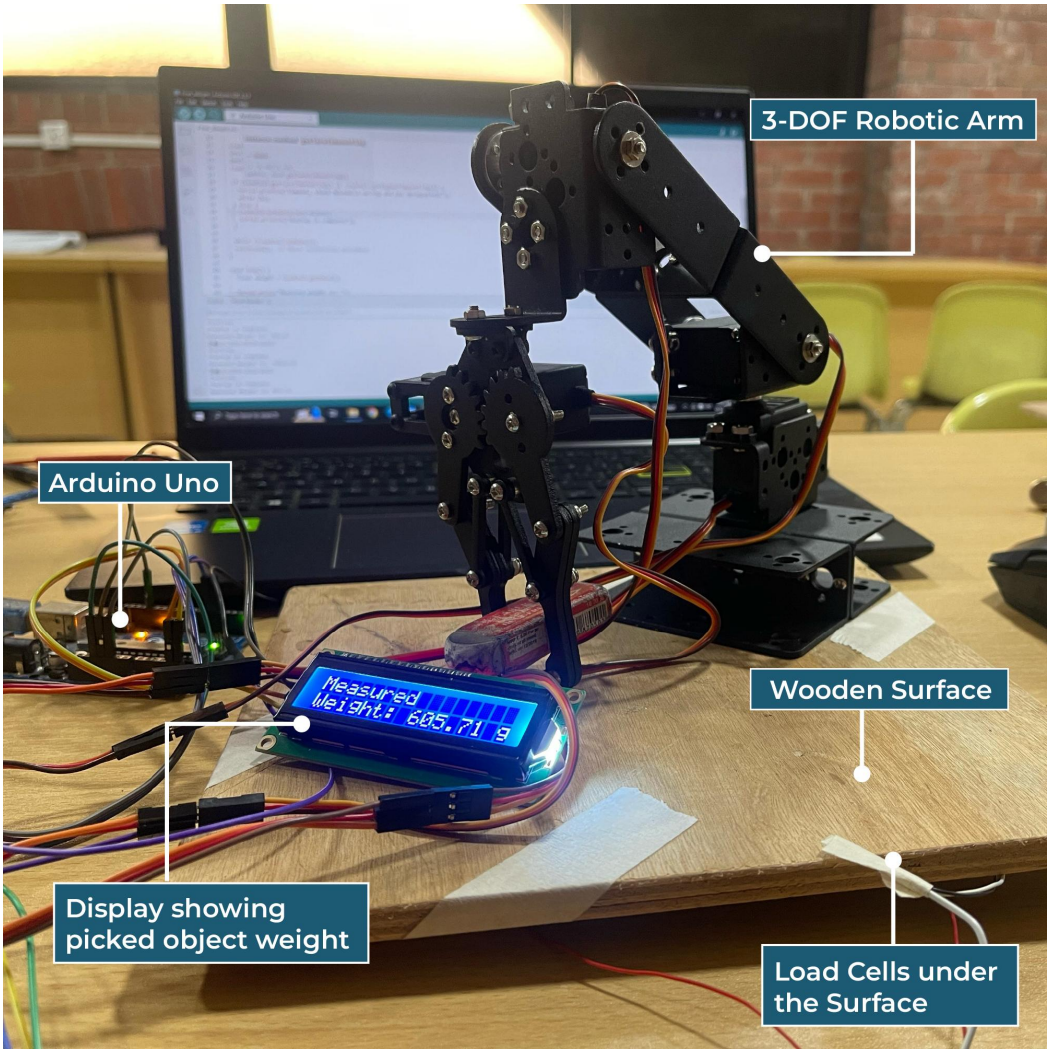
Finding out the calibration factor is very crucial in order to correctly measure the weight. This arduino code from the HX711 library was used to find out the calibration factor.

Arduino code for weight measurement:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <HX711_ADC.h>
const int HX711_dout = 4; // MCU > HX711 DOUT pin
const int HX711_sck = 5;  // MCU > HX711 SCK pin
// HX711 constructor:
HX711_ADC LoadCell(HX711_dout, HX711_sck);
// Function prototype
void calibrate();
void setup() {
  Serial.begin(57600);
  delay(10);
  Serial.println();
  Serial.println("Starting...");
  LoadCell.begin();
  unsigned long stabilizingtime = 2000;
  boolean _tare = true;
  LoadCell.start(stabilizingtime, _tare);
  if (LoadCell.getTareTimeoutFlag() || LoadCell.getSignalTimeoutFlag()) {
  Serial.println("Timeout, check MCU>HX711 wiring and pin designations");
  while (1);
  } else {
    LoadCell.setCalFactor(10400);
    Serial.println("Startup is complete");
  }
  while (!LoadCell.update());
  calibrate();  // Start calibration procedure
}
void loop() {
  float weight = LoadCell.getData();
  Serial.print("Measured weight is: ");
  Serial.println(weight);
```

```
  lcd.init();                      // initialize the lcd
  lcd.init();
  // Print a message to the LCD.
  lcd.backlight();
  lcd.setCursor(0,0);
  lcd.print("Measured");
  lcd.setCursor(0,1);
  lcd.print("Weight: ");
  lcd.print(weight);
  lcd.print(" gm");
  delay(1000);  // Delay for stability
  while (1);
}
```

Hardware Setup:



Integration of Weight Measuring Feature with the Robotic Arm:

- Keeping the robotic arm above the wooden plate that has load cells attached beneath the surface
- Arm picking up the object
- Load cells measuring the object's weight along with the arms weight
- Finally displaying only the objects weight in LCD by subtracting the arms actual weight

Object Color Detection Using OpenCV

Introduction to Computer Vision:

Computer vision is an interdisciplinary field that enables computers to interpret and understand the visual world. It seeks to automate tasks that the human visual system can do effortlessly, such as image and video analysis, object recognition, and scene understanding. One of the fundamental aspects of computer vision is image processing, where algorithms are applied to digital images to extract meaningful information.

OpenCV (Open Source Computer Vision Library) is a popular open-source computer vision and machine learning software library. It provides a wide range of tools and functions for image and video analysis, making it a valuable resource for developing computer vision applications.

The Code :

```
import cv2
 import numpy as np
 def mouse(event, x, y, flag, param):
      global EVENT
      global xPos
      global yPos
      if event == cv2.EVENT_LBUTTONDOWN:
      EVENT = event
      xPos = x
      yPos = y
      if event == cv2.EVENT_LBUTTONUP:
      EVENT = event
      xPos = x
      yPos = y
 height = 780
 width = 1500
 H = 28
 W = (1500 / 780) * 28
 EVENT = 0
```

```
 camera = cv2.VideoCapture(1)
 cv2.namedWindow("hello")
 cv2.setMouseCallback("hello", mouse)

 while True:
       ignore, frame = camera.read()
       output_frame = cv2.resize(frame, (width, height))
       if EVENT == 1:
       black = np.zeros([250, 250, 3], dtype=np.uint8)
       # black_HSV=cv2.cvtColor(black,cv2.COLOR_BGR2HSV)
       # frame_HSV=cv2.cvtColor(output_frame,cv2.COLOR_BGR2HSV)
       color_pick = output_frame[yPos, xPos]
       black[:, :] = color_pick
       cv2.putText(black, str(color_pick), (10, 100), cv2.FONT_HERSHEY_COMPLEX,
1, (100, 50, 0), 3)
       cv2.imshow("desired color", black)
       print(xPos, " ", yPos)
       print("cordinates are : ", (abs(width * 0.5 - xPos) * (W / width),
abs(height * 0.5 - yPos) * (H / height)),
             "in cm")
       cv2.imshow("hello", output_frame)
       if cv2.waitKey(1) == ord("a"):
       break
```

Objective of the Code:

The provided code aims to detect the color of an object in a live video stream using OpenCV. Additionally, it allows the user to select a region of interest by clicking on the video window, and then displays the selected color in a separate window along with its corresponding RGB values. The code also calculates the coordinates of the selected point in centimeters, relative to the center of the video frame.

Code Breakdown:

1. Importing Libraries:

```
import cv2
 import numpy as np
```

The code begins by importing the necessary libraries. cv2 is the OpenCV library, and numpy is used for numerical operations.

2. Mouse Event Handling:

```
def mouse(event, x, y, flag, param):
       global EVENT
```

```
        global xPos
        global yPos
        if event == cv2.EVENT_LBUTTONDOWN:
        EVENT = event
        xPos = x
        yPos = y
        if event == cv2.EVENT_LBUTTONUP:
        EVENT = event
        xPos = x
        yPos = y
```

The code defines a mouse event handling function (mouse) to capture the position of a mouse click. It stores the event type, as well as the x and y coordinates of the click.

3. Setting Up Video Capture:

```
height = 780
 width = 1500
 H = 28
 W = (1500 / 780) * 28
 EVENT = 0
 camera = cv2.VideoCapture(1)
 cv2.namedWindow("hello")
 cv2.setMouseCallback("hello", mouse)
```

Here, the dimensions of the video frame (height and width) and the aspect ratio for calculating real-world coordinates (H and W) are defined. The code initializes the video capture object (camera) and sets up a window named "hello" with the mouse callback function.

4. Main Loop :

```
while True:
        ignore, frame = camera.read()
        output_frame = cv2.resize(frame, (width, height))
```

The main loop captures frames from the camera, resizes them to the specified dimensions, and stores them in the output_frame variable.

5. Object Color Detection :

```
        if EVENT == 1:
        black = np.zeros([250, 250, 3], dtype=np.uint8)
        # black_HSV=cv2.cvtColor(black,cv2.COLOR_BGR2HSV)
        # frame_HSV=cv2.cvtColor(output_frame,cv2.COLOR_BGR2HSV)
        color_pick = output_frame[yPos, xPos]
        black[:, :] = color_pick
        cv2.putText(black, str(color_pick), (10, 100), cv2.FONT_HERSHEY_COMPLEX,
```

```
1, (100, 50, 0), 3)
        cv2.imshow("desired color", black)
        print(xPos, " ", yPos)
        print("cordinates are : ", (abs(width * 0.5 - xPos) * (W / width),
abs(height * 0.5 - yPos) * (H / height)),
              "in cm")
```

When a mouse click event occurs (EVENT == 1), a black image (black) is created, and the color of the selected pixel is extracted from the current frame. The selected color is displayed in a separate window along with its RGB values. The coordinates of the selected point in centimeters are calculated and printed.

6. Display Video Frame :

```
cv2.imshow("hello", output_frame)
```

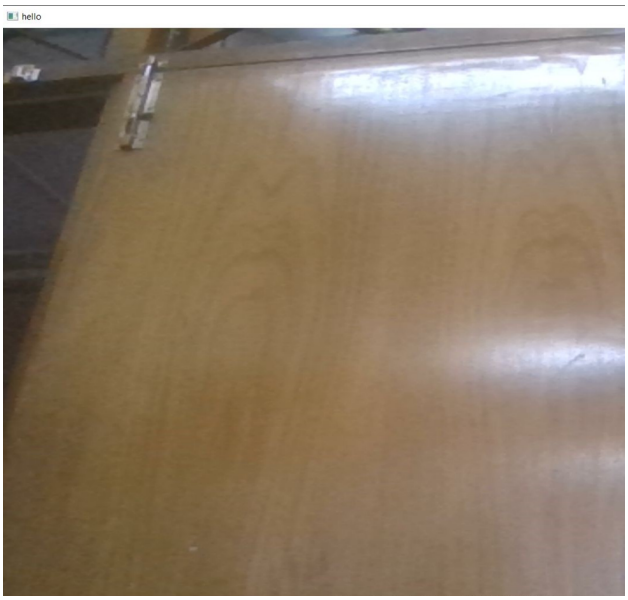
The original video frame with the selected region of interest is displayed in the "hello" window.

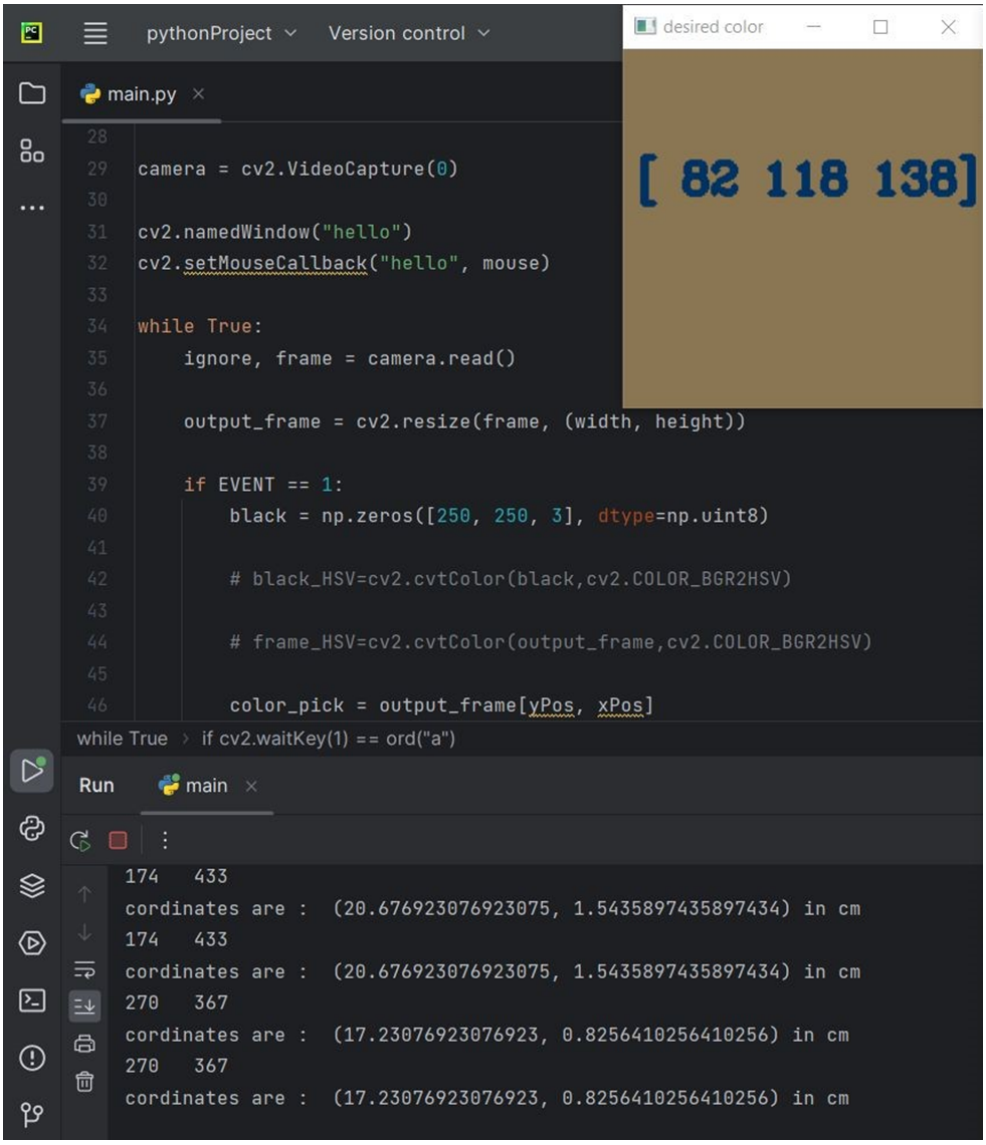
7. Exit Condition :

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

The loop continues until the user presses the 'a' key, at which point the program exits.

OUTPUT of the CODE:





As we can see it shows the color and the color code of the object in the webcam in a separate window.

In conclusion, this code serves as a practical illustration of color detection using OpenCV. The integration of a mouse event handling function allows interactive selection of a point in the video frame, and the subsequent display of color information and calculated coordinates enhances the educational value of the code. Understanding and utilizing color information in computer vision applications have wide-ranging implications, from robotics to human-computer interaction interfaces. This code provides a foundation for further exploration and development in the realm of computer vision.

Final Code:

Before Integration with the HC Sensor:

```
#include <Servo.h>
// Define servo objects for each joint (3 DOF)
Servo baseServo;
Servo shoulderServo;
Servo elbowServo;
Servo gripperServo;
const float L1 = 12.0;  // Length of the first link
const float L2 = 5.0;   // Length of the second link
float pi = 3.1416;
bool isConditionMet1 = false;
bool isconditionmet2 = false;
int potpin = A0;
int potpin1 = A1; // analog pin connected to the potentiometer
int val;  // variable to store the potentiometer value
int angle;
int val1;
int angle1;
float x ;
float y;
float theta1; // Angle of the first motor (in radian)
float theta1_deg; // Angle of the first motor (in degree)
float theta2; // Angle of the second motor (in radian)
float theta2_deg; // Angle of the second motor (in degree)
float theta3; // Angle of the second motor (in radian)
float theta3_deg; // Angle of the second motor (in degree)
int pos2;
int pos3;
void setup() {
  // Initialize servos and ultrasonic sensor
    baseServo.attach(9);
    shoulderServo.attach(10);
   elbowServo.attach(11);
   gripperServo.attach(12);
 // delay(1000);
  shoulderServo.write(80);
  delay(2000);
  elbowServo.write(0);
  delay(2000);// Initialize serial communication for debugging
  Serial.begin(9600);
}
// If the object is within a certain range, pick it up
void loop(){
  if(!isConditionMet1){
```

```
  move(16,2);
  }
 delay (1000);
  val = analogRead(potpin);
  // Map the potentiometer value to servo angle
  angle = map(val, 0, 1023, 0, 180);
  gripperServo.write(angle);  // set the angle for the servo
  // Print servo angle to the serial monitor
  Serial.print("Angle: ");
  Serial.println(angle);
  delay(1000);
 val1 = analogRead(potpin1);
  // Map the potentiometer value to servo angle
  angle1 = map(val1, 0, 1023, 0, 180);
  baseServo.write(angle1);  // set the angle for the servo
  // Print servo angle to the serial monitor
  Serial.print("Angle1: ");
  Serial.println(angle1);
  delay(100);
}
 void move(float x, float y) {
  theta3 = 3.1416 -  acos(- (pow(0.01*L1, 2) + pow(0.01*L2, 2)-
pow(0.01*x, 2) - pow(0.01*y, 2) ) / (2 *0.01* L1 *0.01* L2));
  theta2 = atan(y/x)+ atan((L2*sin(theta3))/(L1+L2*cos(theta3)));
  theta2_deg = 90 + theta2 * 180 / pi;
  theta3_deg =  theta3 * 180 / pi;
  Serial.print("theta2_deg: ");
  Serial.println(theta2_deg);
  Serial.print("theta3_deg: ");
  Serial.println(theta3_deg);
  for (int pos2 = 80; pos2 <= theta2_deg; pos2 += 1) {
    // STEPS FOR 1 degree
    shoulderServo.write(pos2);
    Serial.print("pos2: ");
    Serial.println(pos2);
    delay(15);  // Adjust the delay as needed
  }
  for (int pos3 = 0; pos3 <= theta3_deg; pos3 += 1) {
    // STEPS FOR 1 degree
    elbowServo.write(pos3);
    Serial.print("pos3: ");
    Serial.println(pos3);
    delay(15);  // Adjust the delay as needed
  }
  delay(1000);
 isConditionMet1= true;
}
```

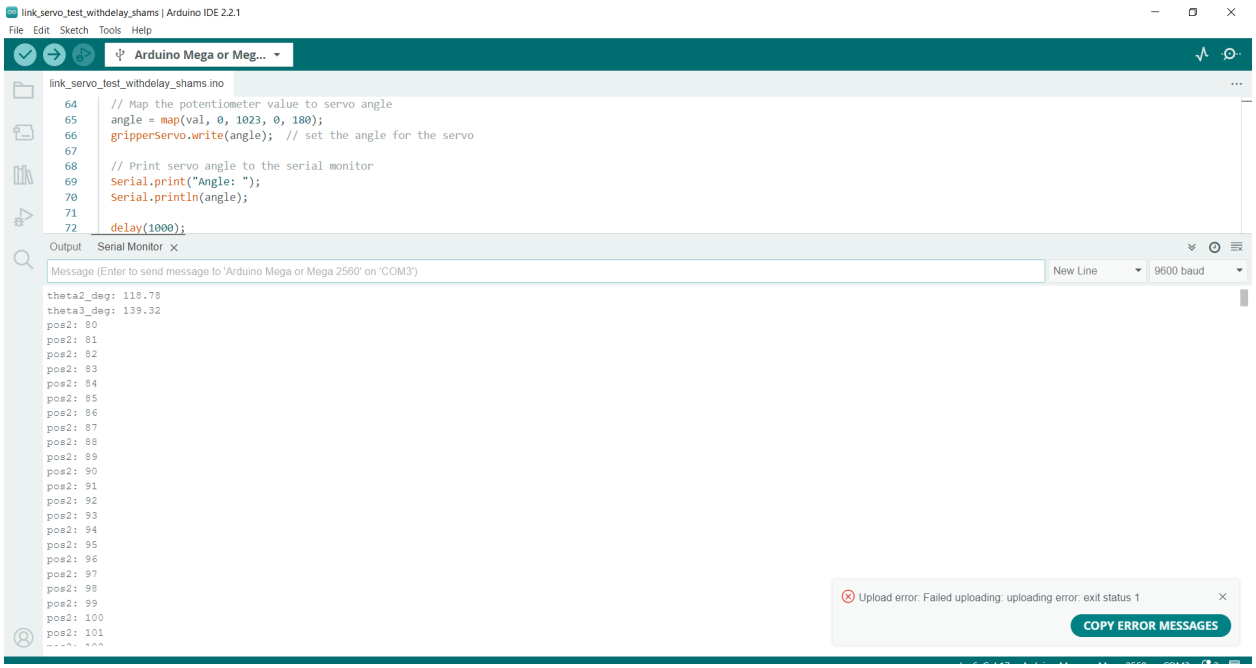
After Integration with the HC Sensor:

```
#define trigPin 7
#define echoPin 8
float p;
bool objectDetected = false;

#include <Servo.h>
// Define servo objects for each joint (3 DOF)
Servo baseServo;
Servo shoulderServo;
Servo elbowServo;
Servo gripperServo;

const float L1 = 12.0;  // Length of the first link
const float L2 = 5.0;  // Length of the second link
float pi = 3.1416;
bool isConditionMet1 = false;
bool isconditionmet2 = false;
int potpin = A0;
int potpin1 = A1;
int potpin2= A2; // analog pin connected to the potentiometer
int val;          // variable to store the potentiometer value
int angle;
int val1;
int angle1;
int val2;
int angle2;
float y;
float theta1;     // Angle of the first motor (in radian)
float theta1_deg; // Angle of the first motor (in degree)
float theta2;     // Angle of the second motor (in radian)
float theta2_deg; // Angle of the second motor (in degree)
float theta3;     // Angle of the second motor (in radian)
float theta3_deg; // Angle of the second motor (in degree)
int pos2;
int pos3;
void setup() {
  // Initialize servos and ultrasonic sensor
  baseServo.attach(9);
  shoulderServo.attach(10);
  elbowServo.attach(11);
  gripperServo.attach(12);
  Serial.begin(9600); // Initialize serial communication
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  // Delay for sensor stability
  delay(1000);
```

```
}
// If the object is within a certain range, pick it up
void loop() {
  if (!objectDetected) {
    long duration, distance;
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    duration = pulseIn(echoPin, HIGH);
    distance = (duration / 2) / 29.1; // Convert to centimeters

    Serial.print("Distance: ");
    Serial.print(distance);
    Serial.println(" cm");
    p = 25 - distance; // Adjust as needed based on your setup
    objectDetected = true; // Set flag to indicate object detected
  }
  if (!isConditionMet1) {
    move(p, 2);
  }
  delay(1000);
  val = analogRead(potpin);
  angle = map(val, 0, 1023, 0, 180);
  gripperServo.write(angle);
  Serial.print("Angle: ");
  Serial.println(angle);
  delay(1000);
  val1 = analogRead(potpin1);
  angle1 = map(val1, 0, 1023, 0, 180);
  baseServo.write(angle1);
  Serial.print("Angle1: ");
  Serial.println(angle1);
  delay(100);
   val2 = analogRead(potpin2);
  angle2 = map(val2, 0, 1023, 0, 180);
  baseServo.write(angle2);
  Serial.print("Angle2: ");
  Serial.println(angle2);
  delay(100);
}
void move(float x, float y) {
  theta3 = acos(-(pow(0.01 * L1, 2) + pow(0.01 * L2, 2) - pow(0.01 * x, 2)
- pow(0.01 * y, 2)) / (2 * 0.01 * L1 * 0.01 * L2));
  theta2 = atan(y / x) + atan((L2 * sin(theta3)) / (L1 + L2 *
cos(theta3)));
```

```
  theta2_deg = 90 + theta2 * 180 / pi;
  theta3_deg = theta3 * 180 / pi;
  Serial.print("theta2_deg: ");
  Serial.println(theta2_deg);
  Serial.print("theta3_deg: ");
  Serial.println(theta3_deg);
  for (int pos2 = 80; pos2 <= theta2_deg; pos2 += 1) {
    shoulderServo.write(pos2);
    Serial.print("pos2: ");
    Serial.println(pos2);
    delay(15);
  }
  for (int pos3 = 0; pos3 <= theta3_deg; pos3 += 1) {
    elbowServo.write(pos3);
    Serial.print("pos3: ");
    Serial.println(pos3);
    delay(15);
  }
  delay(1000);
  isConditionMet1 = true;
}
```

Output:



Overall Cost:

Component	Price
Arm Kit	4150
Arduino Mega	2000
Servo Motor	1880+1740
Breadboard	80
Jumper Wires	400
Load Cell Voltage Amplifier	175
Buck Converter	325
Load Cell Sensor	170
Potentiometer	50
Ultrasonic Sonar Sensor	130
Total	**11100**

Applications of 3-DOF Robotic Arms:

Educational Robotics: 3-DOF robotic arms are commonly used in educational settings to teach basic principles of robotics and automation. Their simplicity allows students to understand fundamental concepts such as kinematics and control.

Small Parts Assembly: In manufacturing and assembly lines, 3-DOF robotic arms are employed for assembling small components. Their compact size and precise movement make them suitable for tasks that require accurate manipulation of parts.

Quality Inspection: Robotic arms with three degrees of freedom are used for inspecting products on assembly lines. They can carry sensors or cameras to examine items for defects or ensure they meet quality standards.

Medical Applications: Surgical Assistance: In minimally invasive surgeries, 3-DOF robotic arms can assist surgeons by holding and manipulating specialized tools. Their precision and small form factor are advantageous in delicate medical procedures.

Sample Handling: In laboratories, 3-DOF robotic arms are employed for sample handling tasks. They can transport test tubes, pipette fluids, and perform other routine tasks, enhancing the efficiency and accuracy of laboratory processes.

Remote Handling: In hazardous environments or situations where direct human intervention is challenging, 3-DOF robotic arms can be remotely operated. This is useful in scenarios such as handling radioactive materials or exploring confined spaces.

Character Animation: In the entertainment industry, 3-DOF robotic arms find applications in animatronics and character animation. They can replicate simple human-like movements for entertainment purposes in theme parks, movies, and shows.

Training Environments: 3-DOF robotic arms are used in simulators for training purposes. For example, they can simulate the movement of control sticks in aviation training or replicate the manipulation of tools in industrial training.

Electronic Component Placement: In the manufacturing of consumer electronics, 3-DOF robotic arms are employed for placing and soldering electronic components onto circuit boards. Their precision is crucial for ensuring the functionality of electronic devices.

Automated Packaging: 3-DOF robotic arms are used in automated packaging systems for picking products from a conveyor belt and placing them into packaging containers. This streamlines the packaging process and improves efficiency.

Research and Development:Researchers and engineers use 3-DOF robotic arms for prototyping and experimenting with different robotic applications. Their simplicity allows for quick testing and iteration in the development of more complex robotic systems.

Challenges faced during the implementation of the project:

- The main challenge we had faced was the wrong choice of Servo Motor. We had chosen MG996R - a continuous 360 degree motor instead of a MG995R (180 degree) . As a result our motor was not taking any feedback and thus, we were not being able to move the joints accordingly.
- The weight sensor was not working properly due to different obstacles.
- Since we had chosen the arm kit design of a 6 dof Robotic arm , assembling that kit to make a 3 Dof Arm was very complicated and troublesome.
- Integrating the different parts of our project- measuring the distance with HC sensor, object detection with python and controlling the motors - to a single code was very challenging.

Learning Outcomes of the Robotic Arm Project:

- Throughout the project we have worked with servo motors. We have been able to understand and control the behavior of servo motors which would be very useful in any type of future projects.
- Implementation of Inverse Kinematics in the robotic arm
- Characteristics and behavior of different sensors like HC sensor and Load cell sensor.
- Object detection procedure OpenCV in python.

Overall, this project has been a great medium for us to develop our technical skills in controlling different electronic devices, programming skills and most importantly soft skills like co-operation , coordination and collaboration within and as a team.

Reference:

1. https://www.ijcaonline.org/archives/volume179/number37/farman-2018-ijca-916848.pdf

2. https://www.irjet.net/archives/V9/i6/IRJET-V9I6190.pdf