Universidade do Porto

Faculdade de Engenharia

FEUP

Development of a robotic arm

Bruno Filipe Torres Costa - up202004966

Rúben Barbosa Lopes - up202005107

Final project report carried out within the scope of the curricular unit
Arquitetura de Computação Embarcada, from the 1st year of
Electrotechnic and Computers Engineering Master's degree

04/02/2024

We declare that this work/report is our authorship and has not been previously used in another course or curricular unit, from this or another institution. References to other authors (statements, ideas, thoughts) scrupulously respect the rules of attribution, and are duly indicated in the text and in the bibliographic references, according to the referencing rules. We are aware that the practice of plagiarism and self-plagiarism constitutes an academic offense.

In "Ethical Code of Academic Conduct", art.14, University of Porto, 2017.

1 Summary

This document details the outcomes and findings stemming from the practical application of a robotic arm. It covers aspects such as the design, construction, and functionality of the robotic arm, along with the methodologies employed during implementation. The document discusses the performance of the robotic arm in executing specific tasks within a three-dimensional space. Additionally, the report delves into challenges faced during implementation, solutions devised, and potential areas for improvement or future research. Overall, the report serves as a comprehensive documentation of the implementation process and its outcomes for a robotic arm.

Keywords: Automation, Innovation, Robotic Arm, Engineering, Precision.

Table of Contents

2 Introduction

In recent years, we have witnessed a growing revolution in the automation industry, driven by the development of advanced technologies. At the heart of this progress lies the development of robotic arms, an area of research and innovation aimed at enhancing the efficiency, precision, and versatility of automated operations. This report highlights the efforts and achievements in the design and development of a robotic arm, exploring the complexities involved in the conception, manufacturing, and implementation of this vital component in robotic systems.

From the options given to us to choose, we decided to work with a robotic arm mainly due to its versatility for various applications without the need for major configuration, where they can be equipped with various end-effectors or tools, allowing it to perform different functions. They can also perform repetitive tasks with high precison and speed, leading to increased production rates and efficiency.

Throughout this report, advances in materials, sensors, control algorithms and finite state machines contributing to the evolution of the robotic arm will be discussed.

3 Robot Overview

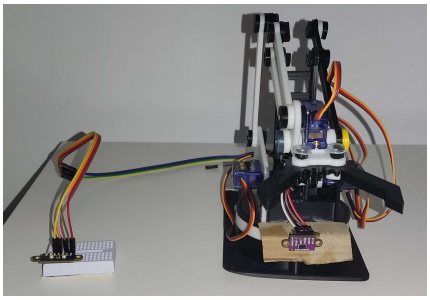


Figure 1: Robot front view

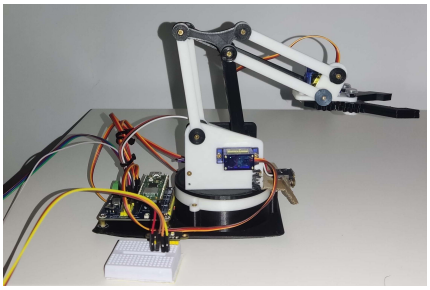


Figure 2: Robot side view

The figures 1, 2, and 3 provide distinct perspectives of the robot. This robotic system is adept at navigating three-dimensional space through the coordinated efforts of its four equipped servos. The discreetly placed base servo allows for a 180-degree rotation of the robot. Additionally, two servos attached to the upper surface of the base enable two-dimensional movement, as depicted in figure 2. The servo situated on the robot's claw is dedicated to the manipulation of objects, facilitating both grasping and releasing actions.

In tandem with the four servo motors, the robot incorporates two sensors. The TCS34725, serving as a color sensor, plays a vital role in discerning objects based on their color attributes. Meanwhile, the VL53L0X, a distance sensor, proves invaluable for detecting objects with uncertain positions.

For seamless integration and control, all sensor readings and actuations are managed by the Raspberry Pi Pico W microcontroller, mounted on a shield specifically designed for driving the servos.

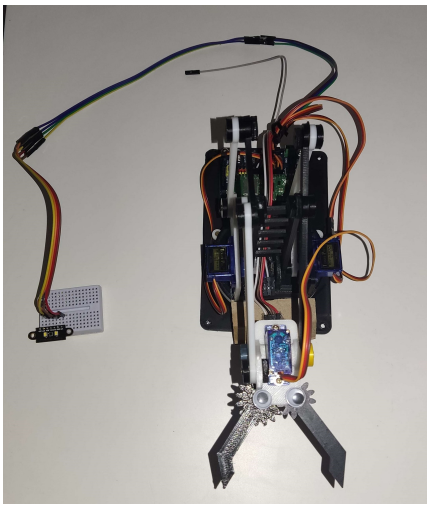


Figure 3: Robot top view

4 Servos Control

The dynamic movement and precise positioning of the robotic arm is made possible through the careful orchestration of servo motors, which act as the muscle behind their motion. This section aims to explain how smooth servos control can be achieved.

Each servo performs a corresponding angle:

- Servo1 (claw servo) - θ_1
- Servo2 (base servo) - θ_2
- Servo3 (top surface right servo) - θ_3
- Servo4 (top surface left servo) - θ_4

4.1 Servos Calibration

To ensure the peak performance of all servos in our project, we initiated a calibration process right from the project's inception. Employing the "Servo.h" library for precise servo control, our approach involves determining the corresponding PWM values needed for accurate servo positioning. The library utilizes PWM signals to articulate the desired angular positions of the servos. Through careful calibration and PWM calculation, we guarantee the precision and reliability of each servo's positioning, establishing a solid foundation for the successful execution of our robotic project.

Practically, this calibration involved measuring the minimum and maximum PWM values corresponding to 0 and 180 degrees, respectively, for each servo from Servo1 to Servo4. To efficiently access and utilize this PWM information, we opted to implement a lookup table for each servo angle movement, ensuring accessibility and speed in our control system.

4.2 Servos Objects

In conjunction with the lookup table, we opted to enhance code usability by implementing a dedicated object for each servo motor. This approach aims to streamline and organize the code, facilitating better readability and maintainability. By encapsulating the functionalities and parameters specific to each servo within individual objects, we improve the overall structure of the code. This design choice not only enhances clarity but also promotes a modular and scalable implementation, making it easier to manage and extend the functionality of each servo motor independently within our robotic project.

4.2.1 Servo1 object

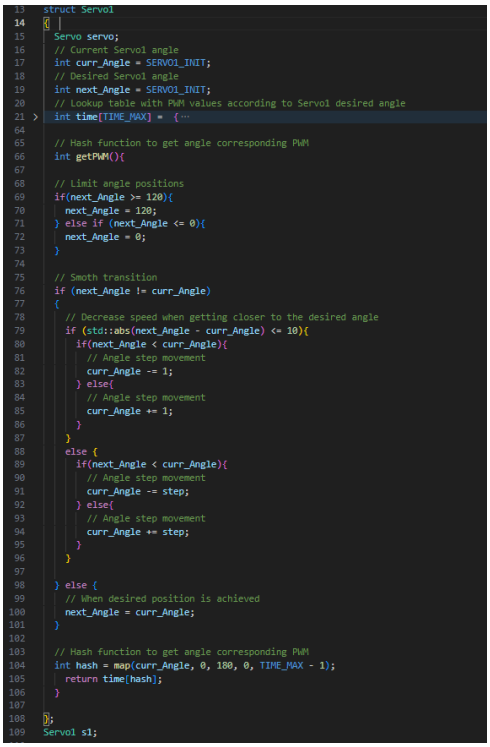


Figure 4: Servo1 object implementation

Figure 4 illustrates the implementation of the Servo1 motor object. Within this implementation, a time lookup table is complemented by the inclusion of a pivotal function known as ‘getPWM()‘. This function serves the purpose of determining the corresponding PWM value based on a given angle. Executed at intervals of every 40 ms, aligning with the program cycle period, the ‘getPWM()‘ function ensures seamless mechanical transitions. This is made possible by the algorithm implemented between lines 79 and 96 in the code.

4.2.2 Servo2 object

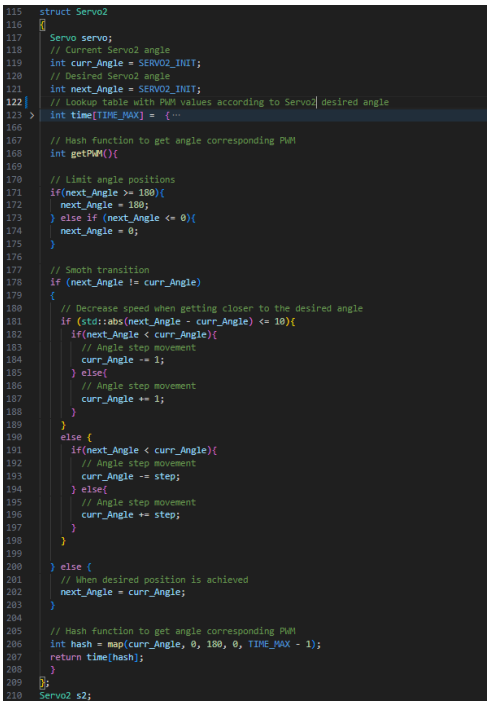
Figure 5 illustrates the implementation of the Servo2 motor object which follows the same implementation of Servo1 motor object.

4.2.3 Servo3 object

Figure 6 illustrates the implementation of the Servo3 motor object which follows the same implementation of Servo1 motor object.

4.2.4 Servo4 object

Figure 7 illustrates the implementation of the Servo4 motor object which follows the same implementation of Servo1 motor object.



```
struct Servo2
{
  Servo servo;
  // Current Servo2 angle
  int curr_Angle = SERVO2_INIT;
  // Desired Servo2 angle
  int next_Angle = SERVO2_INIT;
  // Lookup table with PWM values according to Servo2 desired angle
  int time[TIME_MAX] =  {···

  // Hash function to get angle corresponding PWM
  int getPWM(){

  // Limit angle positions
  if(next_Angle >= 180){
    next_Angle = 180;
  } else if (next_Angle <= 0){
    next_Angle = 0;
  }

  // Smoth transition
  if (next_Angle != curr_Angle)
  {
    // Decrease speed when getting closer to the desired angle
    if (std::abs(next_Angle - curr_Angle) <= 10){
      if(next_Angle < curr_Angle){
        // Angle step movement
        curr_Angle -= 1;
      } else{
        // Angle step movement
        curr_Angle += 1;
      }
    }
    else {
      if(next_Angle < curr_Angle){
        // Angle step movement
        curr_Angle -= step;
      } else{
        // Angle step movement
        curr_Angle += step;
      }
    }

  } else {
    // When desired position is achieved
    next_Angle = curr_Angle;
  }

  // Hash function to get angle corresponding PWM
  int hash = map(curr_Angle, 0, 180, 0, TIME_MAX - 1);
  return time[hash];
  }
};
Servo2 s2;
```

Figure 5: Servo2 object implementation

```
struct Servo3
{
  Servo servo;
  // Current Servo3 angle
  int curr_Angle = SERVO3_INIT;
  // Desired Servo3 angle
  int next_Angle = SERVO3_INIT;
  // Lookup table with PWM values according to Servo3 desired angle
  int time[TIME_MAX] =  {···

  // Hash function to get angle corresponding PWM
  int getPWM(){

  // Limit angle positions
  if(next_Angle >= 180){
    next_Angle = 180;
  } else if (next_Angle <= 0){
    next_Angle = 0;
  }

  // Smoth transition
  if (next_Angle != curr_Angle)
  {
    // Decrease speed when getting closer to the desired angle
    if (std::abs(next_Angle - curr_Angle) <= 10){
      if(next_Angle < curr_Angle){
        // Angle step movement
        curr_Angle -= 1;
      } else{
        // Angle step movement
        curr_Angle += 1;
      }
    }
    else {
      if(next_Angle < curr_Angle){
        // Angle step movement
        curr_Angle -= step;
      } else{
        // Angle step movement
        curr_Angle += step;
      }
    }

  } else {
    // When desired position is achieved
    next_Angle = curr_Angle;
  }

  // Hash function to get angle corresponding PWM
  int hash = map(curr_Angle, 0, 180, 0, TIME_MAX - 1);
  return time[hash];
  }

};
Servo3 s3;
```

Figure 6: Servo3 object implementation

```
struct Servo4
{
  Servo servo;
  // Current Servo4 angle
  int curr_Angle = SERVO4_INIT;
  // Desired Servo4 angle
  int next_Angle = SERVO4_INIT;
  // Lookup table with PWM values according to Servo4 desired angle
  int time[TIME_MAX] = {
  // Hash function to get angle corresponding PWM
  int getPWM(){

  // Limit angle positions
  if(next_Angle >= 140){
    next_Angle = 140;
  } else if (next_Angle <= 0){
    next_Angle = 0;
  }

  // Smoth transition
  if (next_Angle != curr_Angle)
  {
    // Decrease speed when getting closer to the desired angle
    if (std::abs(next_Angle - curr_Angle) <= 10){
      if(next_Angle < curr_Angle){
        // Angle step movement
        curr_Angle -= 1;
      } else{
        // Angle step movement
        curr_Angle += 1;
      }
    }
    else {
      if(next_Angle < curr_Angle){
        // Angle step movement
        curr_Angle -= step;
      } else{
        // Angle step movement
        curr_Angle += step;
      }
    }

  } else {
    // When desired position is achieved
    next_Angle = curr_Angle;
  }

  // Hash function to get angle corresponding PWM
  int hash = map(curr_Angle, 0, 180, 0, TIME_MAX - 1);
    return time[hash];
  }

};
Servo4 s4;
```

Figure 7: Servo4 object implementation

5 Position Mapping

In projects involving robotic arm implementation, inverse kinematics is a common technique. However, for tasks where the objective is to pick pieces of the same height, we have chosen to interpolate angles for joints θ_3 and θ_4 within the robot's range.

5.1 Mapping Methodology

Given that the object height remains consistent, the positional mapping in the three-dimensional plane can be effectively illustrated through Figures 8 and 9.

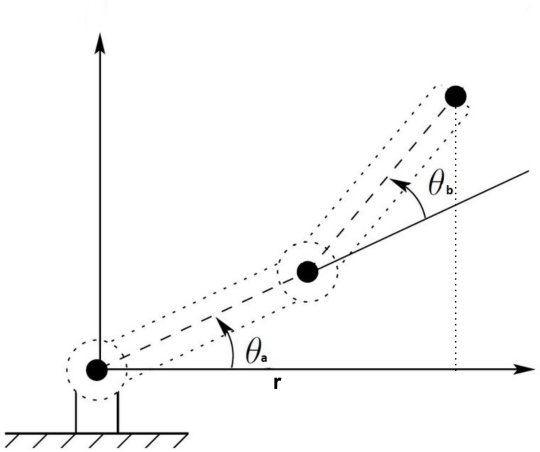


Figure 8: Robot arm vertical movement plane

As illustrated in Figure 8, interpolating the values of θ_3 and θ_4 induces corresponding movements in angles θ_a and θ_b. Executing this interpolation for each r value facilitates the transition to another position in the horizontal plane. This transition necessitates solely the rotation of the base servo, θ_2, as

demonstrated in Figure 9.

In summary, to attain a position in the plane with an r distance from the base and an angle θ, it suffices to compute the interpolated values of θ_3 and θ_4 and subsequently rotate the base to the desired angle θ.

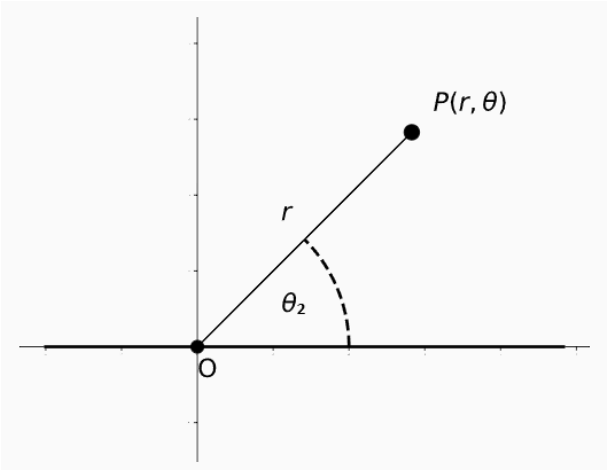


Figure 9: Robot arm horizontal movement plane

5.2 Interpolation

To interpolate the values of θ_3 and θ_4 for the same height, we conducted the measurements presented in Table 1.

Table 1: Measurement data

r (cm)	θ_3 (°)	θ_4 (°)
7.5	97	42
8.5	104	43
9	100	45
9.5	99	47
10	103	50
10.5	106	52
11	108	54
11.5	114	71
12	109	70
12.5	110	79
13	114	80
13.5	117	82
14	120	85
14.5	125	93
15	127	94
15.5	133	100
16	135	110
16.5	142	119
17	145	122
17.5	150	128

It's essential to note that the robotic arm is only capable of picking objects within the range $7.5 < r < 17.5$.

ACE_Interpolation.py

```python
import numpy as np

# Your data
r = np.array([7.5, 8.5, 9, 9.5, 10, 10.5, 11, 11.5, 12, 12.5, 13, 13.5, 14, 14.5, 15, 15.5, 16, 16.5, 17, 17.5])
o3 = np.array([97, 104, 100, 99, 103, 106, 108, 114, 109, 110, 114, 117, 120, 125, 127, 133 ,135, 142, 145, 150])
o4 = np.array([42, 43 ,45, 47, 50 ,52 ,54 ,71, 70, 79, 80, 82, 85, 93, 94, 100, 110, 119, 122, 128])

# Generate distances with a step of 0.1
distances = np.arange(8, 17.5, 0.01)

# Interpolate for each distance
o3_interpolated = np.interp(distances, r, o3)
o4_interpolated = np.interp(distances, r, o4)


# Print the table
for i in range(len(distances)):
    print(f"{distances[i]:.2f}\t\t{o3_interpolated[i]:.2f}\t\t{o4_interpolated[i]:.2f}")

```

Figure 10: Python Interpolation Implementation

To interpolate these values, we decided to implement a Python script that interpolates both θ_3 and θ_4 for the given r value, as illustrated in Figure 10.

r	o3	o4
8.00	100.50	42.50
8.01	100.57	42.51
8.02	100.64	42.52
8.03	100.71	42.53
8.04	100.78	42.54
8.05	100.85	42.55
8.06	100.92	42.56
8.07	100.99	42.57
8.08	101.06	42.58
8.09	101.13	42.59
8.10	101.20	42.60
8.11	101.27	42.61
8.12	101.34	42.62
8.13	101.41	42.63
8.14	101.48	42.64
8.15	101.55	42.65

(a) Initial obtained results

17.34	148.40	126.08
17.35	148.50	126.20
17.36	148.60	126.32
17.37	148.70	126.44
17.38	148.80	126.56
17.39	148.90	126.68
17.40	149.00	126.80
17.41	149.10	126.92
17.42	149.20	127.04
17.43	149.30	127.16
17.44	149.40	127.28
17.45	149.50	127.40
17.46	149.60	127.52
17.47	149.70	127.64
17.48	149.80	127.76
17.49	149.90	127.88

(b) Final obtained results

Figure 11: Partial Interpolation obtained results

5.3 Hybrid Search

To optimize the efficiency of our interpolation process, we have implemented a hybrid search approach. This method combines the utilization of lookup tables with binary search for each angle in our project, taking into consideration the provided distance value. The implementation in C++ is illustrated in Figure 12.

The values d, θ_3, and θ_4 are obtained from the Python script. To obtain a specific θ_3 or θ_4 angle value based on the given distance, a binary search is initially conducted to determine the corresponding index (d index). Subsequently, using this index, the θ_3 and θ_4 values are looked up in their respective tables.

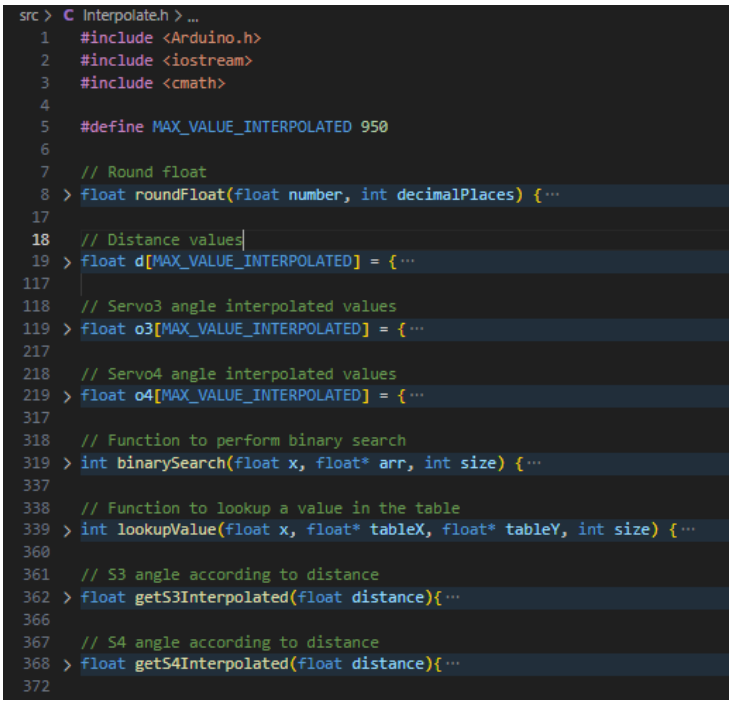


Figure 12: Interpolate.h file that implements the hybrid search

```
// Function to perform binary search
int binarySearch(float x, float* arr, int size) {
  int low = 0;
  int high = size - 1;

  while (low <= high) {
    int mid = low + (high - low) / 2;

    if (arr[mid] == x) {
      return mid; // Found exact match
    } else if (arr[mid] < x) {
      low = mid + 1;
    } else {
      high = mid - 1;
    }
  }

  return -1; // Not found
}
```

Figure 13: Function to perform binary search

```
// Function to lookup a value in the table
int lookupValue(float x, float* tableX, float* tableY, int size) {

  // Distance
  x = roundFloat(x, 2);

  // Max Distance
  if (x >= 17.50)
    x = 17.50;

  // Min Distance
  if ( x <= 7.5){
    x = 7.50;
  }

  int index = binarySearch(x, tableX, size);
  if (index != -1) {
    return tableY[index];
  }
  // Handle the case where x is not found in the table
  return 0;  // You can choose a default value or handle it differently
}
```

Figure 14: Function to lookup a value in the table

```
// S3 angle according to distance
float getS3Interpolated(float distance){
    // Lookup the value
    return lookupValue(distance, d, o3, MAX_VALUE_INTERPOLATED);
}

// S4 angle according to distance
float getS4Interpolated(float distance){
    // Lookup the value
    return lookupValue(distance, d, o4, MAX_VALUE_INTERPOLATED);
}
```

Figure 15: Servo3 and Servo4 functions that deliverer the angle according to the given distance

Figure 15 illustrates the implementation of functions designed to retrieve the values of θ_3 and θ_4 based on the specified distance from the base of the robot arm. This method enhances the speed and efficiency of accessing new required angle values.

6 State Machines

Robotic arms typically utilize state machines to regulate their operation and control. Within the framework of state machine architecture, the system's behavior is segmented into distinct states, with transitions between these states triggered by specific events or conditions. In the context of robotic arms, state machines play a crucial role in organizing and managing various functionalities, thereby improving the overall efficiency and responsiveness of the system. The typical implementation sequence follows the subsequent sections.

6.1 Inputs

Inputs in state machines serve as the stimuli or signals that influence transitions between different states. These inputs can come from various sources, such as sensors, user commands, or external events. In the context of state machines used in robotic arms, inputs play a pivotal role in determining the arm's behavior and response to changing conditions.

In this project, the inputs include two sensors, the TCS34725 color sensor and the VL53L0X distance sensor, as well as a digital web browser button. Both sensors operate using the I2C communication protocol, and the microcontroller provides two wiring options for their connection. The first option involves linking the sensors on the same I2C bus, while the second option employs two separate I2C buses. For simplicity, we opted to use two independent buses, namely I2C0 and I2C1.

6.1.1 Color Sensor

A color sensor is a device designed to detect and measure the color of an object or light source. It works by capturing the light reflected or emitted by the object and then analyzing its spectral characteristics to determine the color. Color sensors are widely used in various applications, including robotics, industrial automation, consumer electronics, and more.

The TCS34725 mentioned in the previous context is a color sensor that integrates red, green, blue, and clear light sensing elements. It provides accurate color recognition by combining the intensity of these different light channels. This sensor communicates using the I2C (Inter-Integrated Circuit) protocol, allowing it to interface with microcontrollers or other devices.

```
// Connect TCS34725 Vin to 3.3
Wire.setSDA(8);
Wire.setSCL(9);
Wire.begin();

while (!tcs.begin()) {
  Serial.println("No TCS34725 found ... check your connections");
  delay(500);
}
```

Figure 16: Color sensor setup implementation in I2C0 bus

After conducting some testing, we determined that the RGB color standard was not performing effectively in distinguishing objects by color. As a result, we made the decision to switch to using the HSV (Hue, Saturation, Value) color standard. This change allows for a more robust and accurate differentiation of colors, particularly in scenarios where RGB may fall short. The HSV color space is often preferred in applications that prioritize color perception and segmentation, making it a suitable alternative for improved performance in object recognition and color-based tasks.

```
// tcs.getRawData() does a delay(Integration_Time) after the sensor readout.
// We don't need to wait for the next integration cycle because we wait "interval" ms before requesting another read
// (make sure that interval > Integration_Time)
void getRawData_noDelay(uint16_t *r, uint16_t *g, uint16_t *b, uint16_t *c)
{
  *c = tcs.read16(TCS34725_CDATAL);
  *r = tcs.read16(TCS34725_RDATAL);
  *g = tcs.read16(TCS34725_GDATAL);
  *b = tcs.read16(TCS34725_BDATAL);
}
```

Figure 17: Function that communicates with the color sensor and gets a new RGB reading

```
107  // Structure to store HSV values
108  struct HSV {
109      float h; // Hue
110      float s; // Saturation
111      float v; // Value
112  };
113
114
115  // Function to convert RGB to HSV
116  HSV rgbToHsv(int r, int g, int b) {
117      HSV hsv;
118
119      float min_val = std::min(std::min(r, g), b);
120      float max_val = std::max(std::max(r, g), b);
121
122      // Calculate the hue
123      if (max_val == min_val) {
124          hsv.h = 0; // undefined, but for simplicity, set to 0
125      } else if (max_val == r) {
126          hsv.h = 60 * (0 + (g - b) / (max_val - min_val));
127      } else if (max_val == g) {
128          hsv.h = 60 * (2 + (b - r) / (max_val - min_val));
129      } else { // max_val == b
130          hsv.h = 60 * (4 + (r - g) / (max_val - min_val));
131      }
132
133      // Make sure hue is in the range [0, 360)
134      while (hsv.h < 0) {
135          hsv.h += 360;
136      }
137
138      // Calculate saturation
139      hsv.s = (max_val == 0) ? 0 : (1 - (min_val / max_val));
140
141      // Calculate value
142      hsv.v = max_val / 255.0;
143
144      return hsv;
145  }
```

Figure 18: Function and struct that converts RGB values to HSV values

The function depicted in Figure 19 holds significant importance as it plays a crucial role in obtaining a valid and precise reading of the object currently in the processing stage. Therefore, meticulous implementation was imperative to ensure its effectiveness and reliability in accurately capturing the object's color information.

```
159  // Function to guess color based on HSV values
160  Color getColorValue() {
161      uint16_t r, g, b;
162      getRawData_noDelay(&r, &g, &b, &c);
163
164      // Convert RGB to HSV
165      HSV hsv = rgbToHsv(r, g, b);
166
167      // Determine color based on the hue component
168      if (hsv.h >= 0 && hsv.h < 30) {
169          return RED;
170      } else if (hsv.h >= 30 && hsv.h < 90) {
171          return YELLOW;
172      } else if (hsv.h >= 90 && hsv.h < 150) {
173          return GREEN;
174      } else if (hsv.h >= 150 && hsv.h < 210) {
175          return BLUE;
176      } else if (hsv.h >= 210 && hsv.h <= 270) {
177          return BLUE;
178      } else {
179          return INVALID;
180      }
181  }
```

Figure 19: Function that delivers the current object color

6.1.2 Distance Sensor

A distance sensor is a device crafted to gauge the distance between the sensor itself and a nearby object or surface. These sensors leverage various technologies, including ultrasonic, infrared, or laser, to ascertain the distance either by measuring the time taken for a signal to travel to the object and back or by evaluating the intensity of reflected signals.

The VL53L0X, mentioned in the earlier context, is a distinct type of distance sensor. Functioning as a Time-of-Flight (ToF) sensor, it utilizes laser light for precise distance measurements. This sensor finds applications in fields such as robotics, drones, and automation, where accuracy in distance measurements is paramount.

In our project, we integrate the VL53L0X distance sensor as one of the inputs. Its implementation revolves around capturing distance data, a crucial aspect for tasks where having precise knowledge of the distance to a target is essential. This sensor communicates using the I2C (Inter-Integrated Circuit) protocol, allowing it to interface with microcontrollers or other devices.

```
391  // Connect Tof to 3.3 V
392  Wire1.setSDA(10);
393  Wire1.setSCL(11);
394  Wire1.begin();
395
396  tof.setBus(&Wire1);
397  tof.setTimeout(500);
398
399  while (!tof.init()) {
400    Serial.println(F("Failed to detect and initialize VL53L0X!"));
401    delay(100);
402  }
403  Serial.println("Found distance sensor");
404
405  // Reduce timing budget to 20 ms (default is about 33 ms)
406  tof.setMeasurementTimingBudget(20000);
407
408  // Start new distance measure
409  tof.startReadRangeMillimeters();
```

Figure 20: Distance sensor setup implementation in I2C1 bus

The function illustrated in Figure 21 is of paramount importance, as it plays a critical role in acquiring a valid and accurate reading of the object undergoing processing. Consequently, meticulous implementation was imperative to ensure the effectiveness and reliability of capturing precise distance information for the object. Subsequent testing led to corrections, including addressing issues like DRIF, to enhance the function's performance.

```
/*-------------------------------------Time of Flight-----------------------------------------*/
VL53L0X tof;
// Distance in cm
float distance, prev_distance;
// Used to calculate the average distance
float sum_distance = 0;
int count = 0;

// DRIFT in cm
#define DRIFT 0.65
// Max distance that sensor can detect objects in cm
#define DETECT_DISTANCE 16

// Get tof distance and prev_distance value in cm
void tofGetValue(){
    if (tof.readRangeAvailable()) {
      prev_distance = distance;

      distance = tof.readRangeMillimeters() * 1e-1 - DRIFT;
    }

    // Start new distance measure
    tof.startReadRangeMillimeters();
}
```

Figure 21: Function that communicates with the VL53L0X and gets the current distance reading

6.1.3 Webserver Button

The last input employed in this robotic arm project involved utilizing a webserver button. By pressing the designated button, as illustrated in section 7.1, users can switch between different operational modes.

6.2 State Machine Processing

This section covers the explanation and implementation of state machine state transitions.

6.2.1 FSM 0 - Control State Machine

In the Control State Machine's initial state, the system commences in the HOLD mode. Upon a user pressing the button in the webserver, it transitions to a mode distinct from the previous one. The HOLD mode is only used when the system starts.

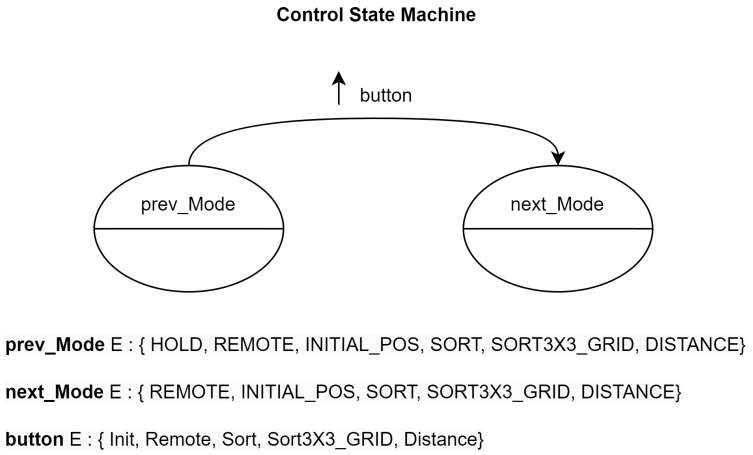


Figure 22: Control State Machine

```
// Control State Machine
if(mode == REMOTE || mode == INITIAL_POS){
  fsm2.new_state = 0;
  fsm3.new_state = 0;
  fsm4.new_state = 0;
  line = 1;
  column = 1;
}

if(mode == SORT){
  fsm3.new_state = 0;
  fsm4.new_state = 0;
  line = 1;
  column = 1;
}

if(mode == DISTANCE){
  fsm2.new_state = 0;
  fsm4.new_state = 0;
  line = 1;
  column = 1;
}

if(mode == SORT3X3_GRID){
  fsm2.new_state = 0;
  fsm3.new_state = 0;
}
```

Figure 23: Control State Machine Implementation

6.2.2 FSM 1 - LED State Machine

This section introduces a debug-focused state machine. If the LED stays illuminated for 0.5 seconds or more, it switches off. Transitioning into a cyclic pattern, when the LED remains off for 0.5 seconds or more, it reverts to the ON state.

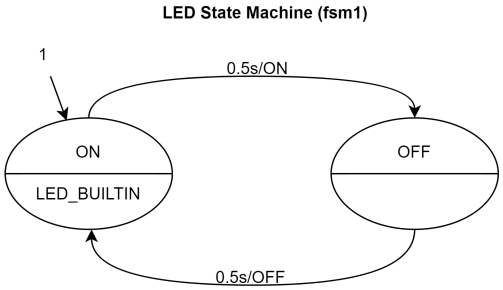


Figure 24: LED State Machine

```
// fsm1, LED state machine
if(fsm1.state == 0 && fsm1.tis >= 500){
  fsm1.new_state = 1;
} else if (fsm1.state == 1 && fsm1.tis >= 500){
  fsm1.new_state = 0;
}
```

Figure 25: LED State Machine Implementation

6.2.3 FSM 2 - SORT State Machine

In the SORT mode, the color piece begins in a default position. When servo 3's current angle matches the next angle, along with the same condition for servo 4, the robot seizes the piece. Once servo 1's current angle aligns with the next angle, the robot ascends. Subsequently, when servos 3 and 4 reach their next angles, the robot rotates toward the color sensor. Upon the sensor recognizing the piece color, and with servos 3 and 4 at their next angles, and the system in that state for 2 seconds, the robot ascends. Now, if servos 3 and 4 angles match their next angles, and the mode is either RED, GREEN, YELLOW, or BLUE, the robot rotates to their respective color positions. When servos 2, 3, and 4 angles are in sync with their next angles, the robot deposits the piece in its corresponding color position. Subsequently, servos 3 and 4 enable the robot to ascend, and when servos 3 and 4 align with their next angles, the robot returns to its initial position.

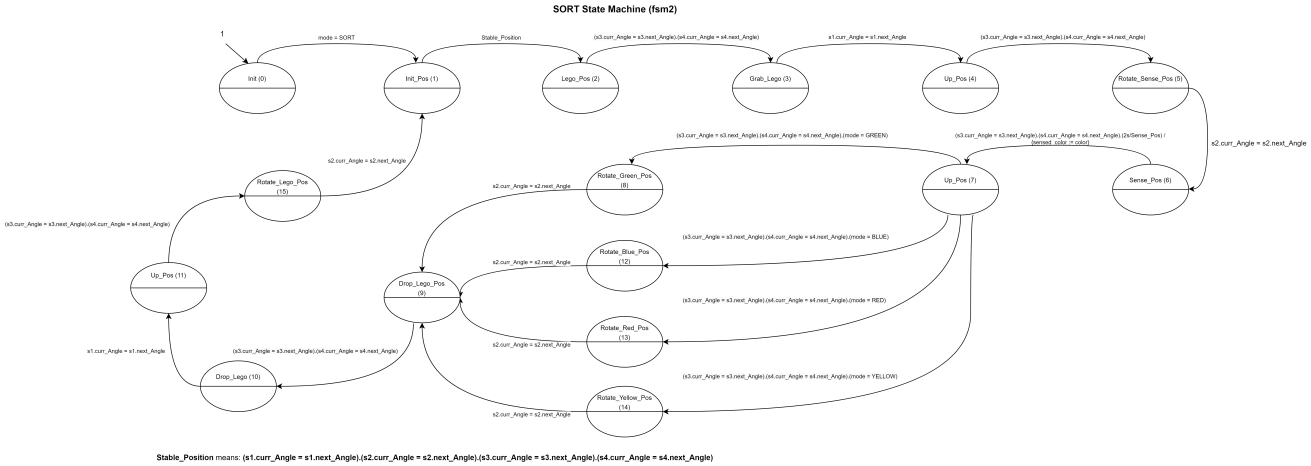


Figure 26: SORT State Machine

```
// fsm2, Sort state machine
if(fsm2.state == 0 && mode == SORT){
  // Go to init Pos
  fsm2.new_state = 1;
  angle1_aux = SERVO1_INIT;
  rotatePosSORT(sort_angle);
  angle3_aux = SERVO3_INIT;
  angle4_aux = SERVO4_INIT;
} else if(fsm2.state == 1 && s1.curr_Angle == s1.next_Angle && s2.curr_Angle == s2.next_Angle && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle){
  // Go to lego positon
  fsm2.new_state = 2;
  pickPosSORT(sort_distance);
} else if (fsm2.state == 2 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle){
  // Grab lego
  fsm2.new_state = 3;
  grabLego();
} else if (fsm2.state == 3 && s1.curr_Angle == s1.next_Angle){
  // Up Pos
  fsm2.new_state = 4;
  upPos();
} else if (fsm2.state == 4 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle){
  // Rotate Lego to sensing color pos
  fsm2.new_state = 5;
  rotateSensePos();
} else if (fsm2.state == 5 && s2.curr_Angle == s2.next_Angle){
  // Find lego color
  fsm2.new_state = 6;
  sensePos();
} else if (fsm2.state == 6 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle && fsm2.tis >= 2000){
  // Up Pos
  fsm2.new_state = 7;
  upPos();
}else if (fsm2.state == 7 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle && sensed_color == GREEN){
  // Rotate Green
  fsm2.new_state = 8;
  rotateGreenPos();
} else if (fsm2.state == 7 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle && sensed_color == BLUE){
  // Rotate Blue
  fsm2.new_state = 12;
  rotateBluePos();
} else if (fsm2.state == 7 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle && sensed_color == RED){
  // Rotate Red
  fsm2.new_state = 13;
  rotateRedPos();
} else if (fsm2.state == 7 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle && sensed_color == YELLOW){
  // Rotate Yellow
  fsm2.new_state = 14;
  rotateYellowPos();
} else if ( (fsm2.state == 8 || fsm2.state == 12 || fsm2.state == 13 || fsm2.state == 14) && s2.curr_Angle == s2.next_Angle){
  // Go to lego drop position
  fsm2.new_state = 9;
  pickPos();
} else if (fsm2.state == 9 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle){
  // Drop lego
  fsm2.new_state = 10;
  dropLego();
} else if (fsm2.state == 10 && s1.curr_Angle == s1.next_Angle && fsm2.tis >= 500){
  // Go to up position
  fsm2.new_state = 11;
  upPos();
} else if (fsm2.state == 11 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle){
  // Rotate to Lego Pos
  fsm2.new_state = 15;
  rotatePosSORT(sort_angle);
} else if (fsm2.state == 15 && s2.curr_Angle == s2.next_Angle){
  // Go to new Lego position
  fsm2.new_state = 1;
}
```

Figure 27: SORT State Machine Implementation

6.2.4 FSM 3 - DISTANCE State Machine

In the DISTANCE mode, when the robot is at the initial position, it begins sweeping to locate the object. If the distance is less than or equal to DETECT DISTANCE and higher than the previous measurement, the robot starts calculating average distances. After half a second, the robot moves to the average distance and seizes the piece when the current angles of servos 1, 3, and 4 match their next angles. Subsequently, servos 3 and 4 facilitate the robot's ascent, and servos 2, 3, and 4 reposition the robot for dropping. When servo 1 reaches its next angle, indicating the claw is open, the piece is released. Once more, servos 3 and 4 raise the robot. Upon the current angles of servos 3 and 4 matching their next angles and remaining in the same state for 1 second, the robot returns to its initial position.

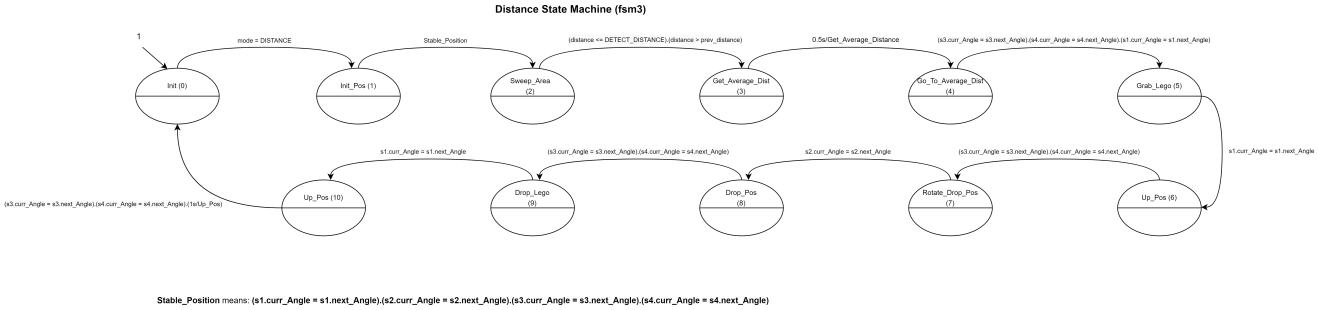


Figure 28: DISTANCE State Machine

```
// fsm3, Distance state machine
if(fsm3.state == 0 && mode == DISTANCE){
  // Go to init Pos
  fsm3.new_state = 1;
  angle1_aux = SERVO1_INIT;
  angle2_aux = SERVO2_INIT;
  angle3_aux = SERVO3_INIT;
  angle4_aux = SERVO4_INIT;
  count = 0;
  sum_distance = 0;
} else if(fsm3.state == 1 && s1.curr_Angle == s1.next_Angle && s2.curr_Angle == s2.next_Angle && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle){
  // Start Area Sweep
  fsm3.new_state = 2;
} else if(fsm3.state == 2 && distance <= DETECT_DISTANCE && distance > prev_distance){
  // Measure average distance
  fsm3.new_state = 3;
} else if (fsm3.state == 3 && fsm3.tis >= 500){
  // Go to average_distance Pos
  fsm3.new_state = 4;
  angle3_aux = getS3Interpolated(sum_distance/count);
  angle4_aux = getS4Interpolated(sum_distance/count);
  angle1_aux = 0;
} else if (fsm3.new_state == 4 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle && s1.curr_Angle == s1.next_Angle){
  // Grab Lego
  fsm3.new_state = 5;
  grabLego();
} else if (fsm3.state == 5 && s1.curr_Angle == s1.next_Angle){
  // Robot Up
  fsm3.new_state = 6;
  upPos();
 } else if(fsm3.state == 6 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle && s4.next_Angle){
   // Rotate to Drop Pos
   fsm3.new_state = 7;
   angle2_aux = 180;
 } else if(fsm3.state == 7 && s2.curr_Angle == s2.next_Angle){
   // Drop Pos
   fsm3.new_state = 8;
   pickPos();
 } else if(fsm3.state == 8 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle){
   // Drop Lego
   fsm3.new_state = 9;
   dropLego();
 } else if(fsm3.state == 9 && s1.curr_Angle == s1.next_Angle){
   // Robot Up
   fsm3.new_state = 10;
   upPos();
 }else if(fsm3.state == 10 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle && fsm3.tis >= 1000){
   // Go to init
   fsm3.new_state = 0;
}
```

Figure 29: DISTANCE State Machine Implementation

6.2.5 FSM 4 - SORT3x3GRID State Machine

In the SORT3x3GRID mode, featuring a 3x3 grid, the robot is tasked with picking up pieces from each available position in the grid. Starting from the initial position, the robot rotates to the first position in the first line and column. When servo 3's current angle matches the next angle, along with the same condition for servo 4, the robot seizes the piece. Once servo 1's current angle aligns with the next angle, the robot ascends. Subsequently, when servos 3 and 4 reach their next angles, the robot rotates toward the color sensor. Upon the sensor recognizing the piece color, and with servos 3 and 4 at their next angles, and the system in that state for 2 seconds, the robot ascends. If servos 3 and 4 angles match their next angles, and the mode is either RED, GREEN, YELLOW, or BLUE, the robot rotates to their respective color positions. When servos 2, 3, and 4 angles are in sync with their next angles, the robot deposits the piece in its corresponding color position. Subsequently, servos 3 and 4 enable the robot to ascend, and when servos 3 and 4 angles align with their next angles, the robot returns to its initial position. During this last transition, the column is increased, indicating that the next piece processed is in the next position on the 3x3 grid.

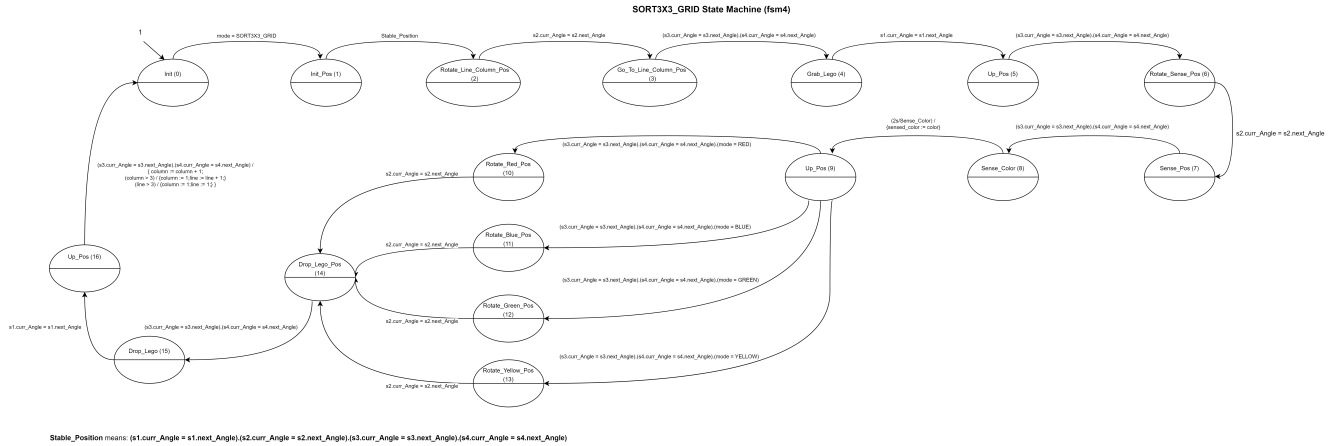


Figure 30: SORT3x3_GRID State Machine

```
// fsm4, Sort3X3_GRID state machine
if(fsm4.state == 0 && mode == SORT3X3_GRID){
  // Go to init Pos
  fsm4.new_state = 1;
  angle1_aux = SERVO1_INIT;
  angle2_aux = SERVO2_INIT;
  angle3_aux = SERVO3_INIT;
  angle4_aux = SERVO4_INIT;
} else if(fsm4.state == 1 && s1.curr_Angle == s1.next_Angle && s2.curr_Angle == s2.next_Angle && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle){
  // Rotate to line and column pos
  fsm4.new_state = 2;
  pickSORT3X3_2(line, column);
} else if(fsm4.state == 2 && s2.curr_Angle == s2.next_Angle){
  // Go to lego line and column pos
  fsm4.new_state = 3;
  pickSORT3X3_34(line, column);
} else if(fsm4.state == 3 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle){
  // Grab lego
  fsm4.new_state = 4;
  grabLego();
} else if(fsm4.state == 4 && s1.curr_Angle == s1.next_Angle){
  // Up Pos
  fsm4.new_state = 5;
  upPos();
} else if(fsm4.state == 5  && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle){
  // Rotate sensing pos
  fsm4.new_state = 6;
  rotateSensePos();
} else if(fsm4.state == 6  && s2.curr_Angle==s2.next_Angle){
  // Sensing Pos
  fsm4.new_state = 7;
  sensePos();
} else if(fsm4.state == 7  && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle){
  // Sensing color
  fsm4.new_state = 8;
} else if(fsm4.state == 8 && fsm4.tis > 2000){
  // Up Pos
  fsm4.new_state = 9;
  upPos();
} else if (fsm4.state == 9 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle && sensed_color == RED){
  // Rotate RED
  fsm4.new_state = 10;
  rotateRedPos();
} else if (fsm4.state == 9 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle && sensed_color == BLUE){
  // Rotate BLUE
  fsm4.new_state = 11;
  rotateBluePos();
} else if (fsm4.state == 9 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle && sensed_color == GREEN){
  // Rotate GREEN
  fsm4.new_state = 12;
  rotateGreenPos();
} else if (fsm4.state == 9 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle && sensed_color == YELLOW){
  // Rotate YELLOW
  fsm4.new_state = 13;
  rotateYellowPos();
} else if ( (fsm4.state == 10 || fsm4.state == 11 || fsm4.state == 12 || fsm4.state == 13 ) && s2.curr_Angle == s2.next_Angle){
  // Drop Pos
  fsm4.new_state = 14;
  pickPos();
} else if ( fsm4.state == 14 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle){
  // Drop lego
  fsm4.new_state = 15;
  dropLego();
} else if ( fsm4.state == 15 && s1.curr_Angle == s1.next_Angle){
  // Up Pos
  fsm4.new_state = 16;
  upPos();
} else if (fsm4.state == 16 && s3.curr_Angle == s3.next_Angle && s4.curr_Angle == s4.next_Angle){
  // Go to init
  fsm4.new_state = 0;
  // Increase column
  column+=1;
  // If column ends
  if(column > 3){
    column = 1;
    line+=1;
  }
  // If grid ends
  if(line > 3 ){
    column = 1;
    line = 1;
  }
}
```

Figure 31: SORT3x3_GRID State Machine Implementation

6.3 Outputs

In this section, the outputs of the state machines are initiated or activated based on the current state.

6.3.1 FSM 0 - Control State Machine

```
724   // fsm0 outputs
725   if (mode == REMOTE || mode == SORT || mode == DISTANCE || mode == SORT3X3_GRID)
726   {
727     // State machines change angles
728     s1.next_Angle = angle1_aux;
729     s2.next_Angle = angle2_aux;
730     s3.next_Angle = angle3_aux;
731     s4.next_Angle = angle4_aux;
732   }
733
734   if (mode == INITIAL_POS)
735   {
736     // Go to Initial Position
737     angle1_aux = SERVO1_INIT;
738     angle2_aux = SERVO2_INIT;
739     angle3_aux = SERVO3_INIT;
740     angle4_aux = SERVO4_INIT;
741
742     s1.next_Angle = SERVO1_INIT;
743     s2.next_Angle = SERVO2_INIT;
744     s3.next_Angle = SERVO3_INIT;
745     s4.next_Angle = SERVO4_INIT;
746
747   }
```

Figure 32: FSM0 State Machine outputs

According to the code snippet present in figure 32 if the mode is set to REMOTE, SORT, DISTANCE, or SORT3X3_GRID, the state machines change the angles of four servos (s1, s2, s3, and s4) based on auxiliary angle variables (angle1_aux, angle2_aux, angle3_aux, and angle4_aux).

On the other hand, if the mode is set to INITIAL_POS, the code sets the auxiliary angle variables to their respective initial values (SERVO1_INIT, SERVO2_INIT, SERVO3_INIT, and SERVO4_INIT), effectively positioning the servos at the initial configuration.

6.3.2 FSM 1 - LED State Machine

```
749   // fsm1 outputs
750   if (fsm1.state == 0){
751     digitalWrite(LED_BUILTIN, HIGH);
752   } else if (fsm1.state == 1){
753     digitalWrite(LED_BUILTIN, LOW);
754   }
```

Figure 33: FSM1 State Machine outputs

In this snippet of code available in figure 33, the outputs of the state machine, denoted as FSM1, depend on its current state. When FSM1 is in state 0, the built-in LED is activated and set to a HIGH state through the command **digitalWrite(LED_BUILTIN, HIGH)**. Conversely, if FSM1 transitions to state 1, the LED is deactivated and switched to a LOW state using **digitalWrite(LED_BUILTIN, LOW)**. This piece of code serves the purpose of controlling the LED output for debugging, dynamically adjusting the LED's illumination based on the state of FSM1.

6.3.3 FSM 2 - SORT and FSM 4 - SORT3x3GRID State Machines

```
756   // fsm2 and fsm4 outputs
757   if (fsm2.state == 6 || fsm4.state == 8)
758   {
759     sensed_color = color;
760   }
```

Figure 34: FSM2 and FSM4 State Machine outputs

The outputs of both FSM2 and FSM4 are influenced by the conditions of their respective states. If FSM2 is in state 6 or FSM4 is in state 8, the variable sensed_color is assigned the value of the variable color. This variable color represents the corresponding color reading of the TCS34725 color sensor.

6.3.4 FSM 3 - DISTANCE State Machines

```
762   // fsm3 outputs
763   if (fsm3.state == 2){
764     // Sweep_Area
765     if(clockwise){
766       angle2_aux -= 1;
767     } else if(!clockwise){
768       angle2_aux +=1;
769     }
770
771     // Change Direction
772     if (s2.curr_Angle == 50)
773       clockwise = false;
774     else if (s2.curr_Angle == 180)
775       clockwise = true;
776   }
777
778   // Measure average distance
779   if (fsm3.state == 3){
780     sum_distance += distance;
781     count +=1;
782   }
```

Figure 35: FSM3 State Machine outputs

In this code segment present in figure 32, the outputs of the state machine FSM3 are determined by its current state. When in state 2, the code executes a **Sweep_Area** functionality. Depending on the boolean variable clockwise, the auxiliary angle angle2_aux is adjusted either by decrementing it by 1 if clockwise is true or incrementing it by 1 if clockwise is false. Additionally, the direction is modified based on the current angle of servo s2. If s2 is at an angle of 50 degrees, the direction is switched to counterclockwise (clockwise set to false). Conversely, if s2 reaches an angle of 180 degrees, the direction changes to clockwise (clockwise set to true). This algorithm enables the robotic arm to search for unknown objects within this range.

In the subsequent state, state 3, the code measures the average distance by accumulating the distance values in the variable sum_distance and incrementing the count variable. This algorithm empowers the robot to compute precise distance measurements by calculating an average distance among its readings.

7 WebServer

The implemented server serves as a pivotal component in the project, acting as a central hub for seamless communication between the robotic system and external entities, including users and connected devices, all achieved through WiFi. Its paramount importance lies in providing an intuitive web interface, allowing users to effortlessly control and monitor the robot's behavior in real-time.

Through this server, users have the capability to smoothly transition between various operational modes, initiate specific actions, and make real-time adjustments to critical parameters, all facilitated over WiFi. The user-friendly interface empowers individuals to interact with the robotic system in a straightforward manner, utilizing wireless connectivity for enhanced convenience.

The server's dynamic interaction capabilities over WiFi enable users to set angles, fine-tune step values, and exert control over sorting distances and angles. This flexibility proves essential for tailoring the robot's functionality to diverse scenarios and tasks, all achieved through wireless communication. Moreover, the integration of JavaScript within the HTML responses enhances the overall user experience by enabling responsive updates that adapt to user input, delivered seamlessly over WiFi.

The remote adjustment of operational modes and parameters via the web interface, facilitated through WiFi, streamlines the entire control process. Whether initializing the system, remotely directing its movements, or refining sorting parameters, the server ensures an accessible and user-friendly means of interaction, contributing significantly to the project's overall functionality and wireless user experience.

7.1 Interface

7.1.1 Init

Figure 36: INITIAL_POS control mode interface

The illustration labeled as 36 depicts the robot operating in control mode, specifically the INITIAL_POS mode. During this mode, the robot arm is directed to its initial, secure position. Additionally, users have the capability to adjust the current step linked to arm velocity movement. Moreover, they can modify the default object position in the SORT control mode by altering the distance to the robot base and the object angle.

7.1.2 Remote

Figure 37: REMOTE control mode interface

In the REMOTE control mode illustrated in figure 37, users possess the ability to manipulate the robotic arm remotely through WiFi. By individually adjusting the angles of each servo, users can navigate the robotic arm to their desired positions.

7.1.3 Sort

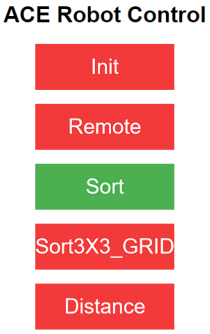


Figure 38: SORT control mode interface

Figure 38 illustrates the robot operating in SORT control mode. In this autonomous and repetitive mode, the robot retrieves a piece from a location specified in the INITIAL_POS mode and proceeds to sort it based on color.

7.1.4 Sort_3X3_GRID

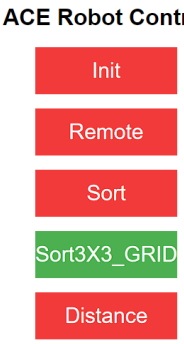


Figure 39: SORT_3X3_GRID control mode interface

The figure 39 depicts the robot functioning in SORT_3X3_GRID control mode. In this autonomous and repetitive mode, the robot retrieves a piece from a 3X3 grid and then proceeds to sort it based on its color.

7.1.5 Distance

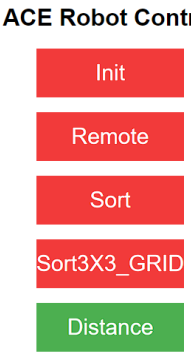


Figure 40: DISTANCE control mode interface

The figure 40 depicts the robot functioning in DISTANCE control mode. In this autonomous and repetitive mode, the robot employs a distance sensor to scan the area for unknown objects and then places the retrieved item in a predefined location.

7.2 Implementation

The implemented web-based control system for the robotic arm involves the development of a C++ web server, employing the raspberry pi pico WiFi module for efficient communication between the web interface and the robotic system. The key function, ‘getClientInfo‘, processes client requests, identifying different operation modes such as REMOTE, INITIAL_POS, SORT, DISTANCE, and SORT3X3_GRID based on HTTP request content. The dynamically generated HTML content caters to each mode, providing users with an intuitive interface. JavaScript ensures dynamic updates as users interact with input fields, allowing for real-time adjustments of servo angles, step values, sort distance, and sort angle. The system offers a responsive web design, accommodating various devices for an enhanced user experience. The implementation not only allows remote control of the robotic arm but also facilitates mode-specific functionalities, making it a versatile and user-friendly solution. This implementation was based on this article [1].

8 Conclusions

In conclusion, this robotic arm project represents a significant achievement in the realm of robotics. By implementing state machines and carefully orchestrating the interaction between various components, the project has successfully endowed the robotic arm with intelligent and adaptive behavior. The utilization of state machine , as demonstrated in the code snippets, allows the arm to perform diverse tasks such as color sensing, sweeping an area, and precise distance measurements. Overall, this project underscores the efficacy of employing state machines for versatile and dynamic control in robotic applications, paving the way for further advancements in the field of robotic manipulation and automation.

9 Appendix

9.1 Code Repository

Project code available on GitHub: GitHub Link.

9.2 Video

Video describing the robots main functionalities: YouTube Link.

References

[1] DroneBot Workshop, *PicoW Arduino - An Introduction to Raspberry Pi Pico with Arduino IDE*, `https://dronebotworkshop.com/picow-arduino/`;

[2] ChatGPT, *ChatGPT: OpenAI's Language Model*, OpenAI, San Francisco, CA, `https://openai.com/`;