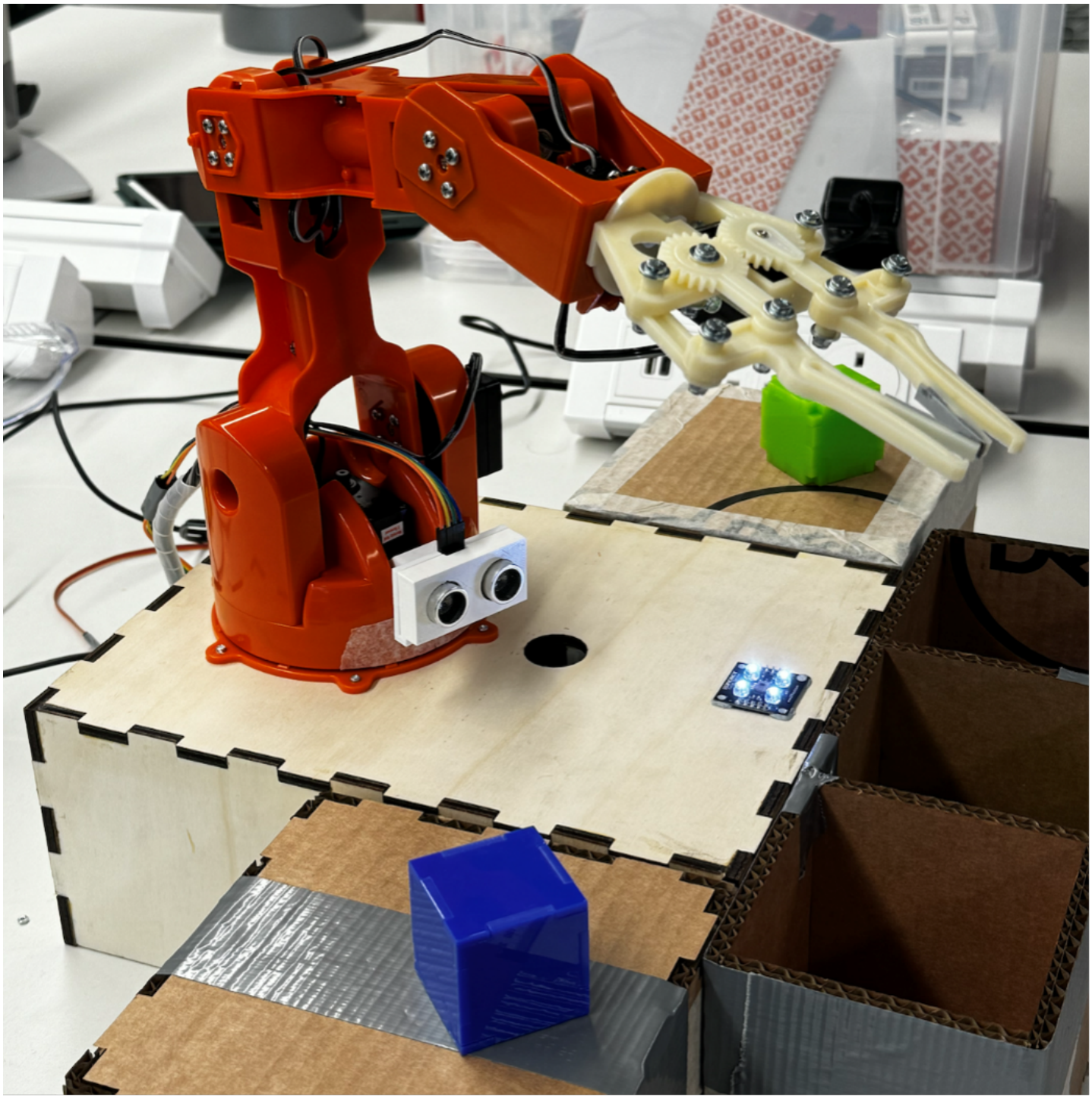

Object Detection and Sorting: A Braccio Arm Project

UCL Engineering Foundation Year: Project 5 Report

Mitchell, J. Stephen, W. Tharmapalan, D. Spasov, E.

Word Count: 1947

Table of contents

Table of figures

1 Abstract

This project focuses on the development of a robotic system capable of object detection, retrieval, and sorting based on colour. Utilising a TinkerKit Braccio Arm, an HC-SR04 ultrasonic sensor, and a TCS3200 colour sensor, the system aims to locate, grab, and relocate objects within a controlled environment. Success was achieved through a combination of structural enhancements, the integration of sensors, and careful calibration. While challenges arose due to sensor limitations and mechanical wear of the arm, the project effectively demonstrates the integration of sensing, actuation, and decision-making within a robotic context. Future work would benefit from exploring more robust robotic platforms, higher-fidelity sensors, and refined sorting algorithms for enhanced accuracy and efficiency.

2 Project Description

2.1 Context and Objectives

This project utilises a TinkerKit Braccio Arm (Figure 1 - TinkerKit Braccio Robot.) to develop a solution for the precise location, retrieval, and relocation of a designated object within a controlled workspace. The project aims to enhance skills in robotics, programming, and spatial reasoning. It addresses challenges such as sensor-guided object detection, accurate calibration, error correction, and the potential for handling varying object types. Success will be measured by the arm's ability to consistently complete the task as defined.

Figure 1 - TinkerKit Braccio Robot.

3 Solution Design and Implementation

3.1 System Architecture

The system utilises an HC-SR04 ultrasonic sensor mounted to the rotating base of the arm (Figure 2 - HCSR04 Mounted to Arm) to achieve object detection and distance measurement. This configuration simulates radar functionality, providing a 180-degree detection arc with usable range up to 250 mm.

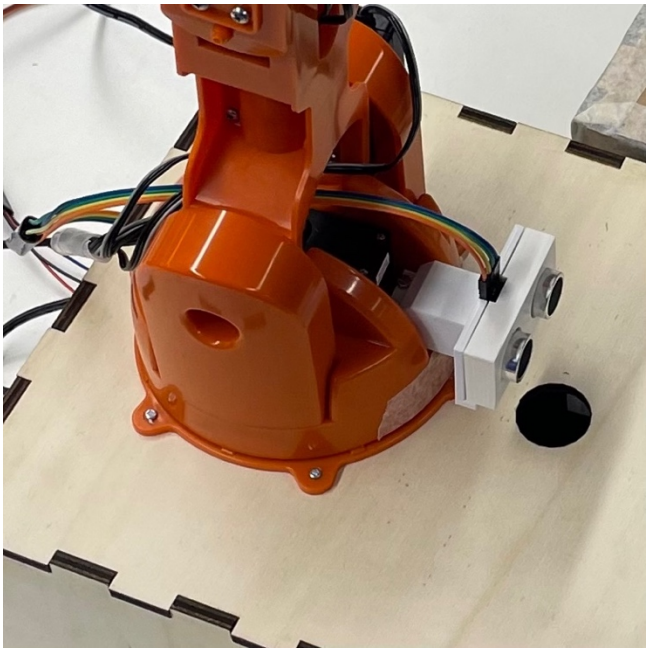

Figure 2 - HCSR04 Mounted to Arm

The system includes a TCS3200 (Figure 3 - TCS3200 Sensor) Colour sensor that will be used to determine an objects colour, it will then be sorted into allocated bins of red, green, and blue depending on the objects colour.

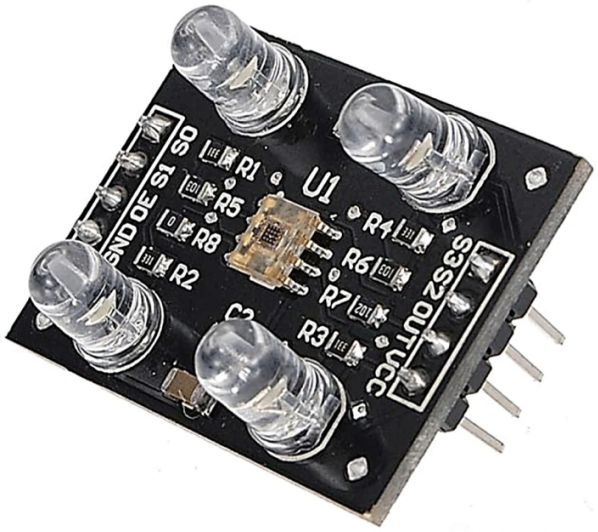

Figure 3 - TCS3200 Sensor

Figure 4 - Schematic diagram of Arduino setup (TinkerCAD) demonstrates the wiring layout that was planned for the project, made using TinkerCAD. The CAD software didn't include the TCS3200 component in its library and was substituted out to be represented by an 8-pin header with the corresponding pins labelled as how they would be setup, the 4 LEDs on the module are also connected as how they are in the modules schematic diagram.

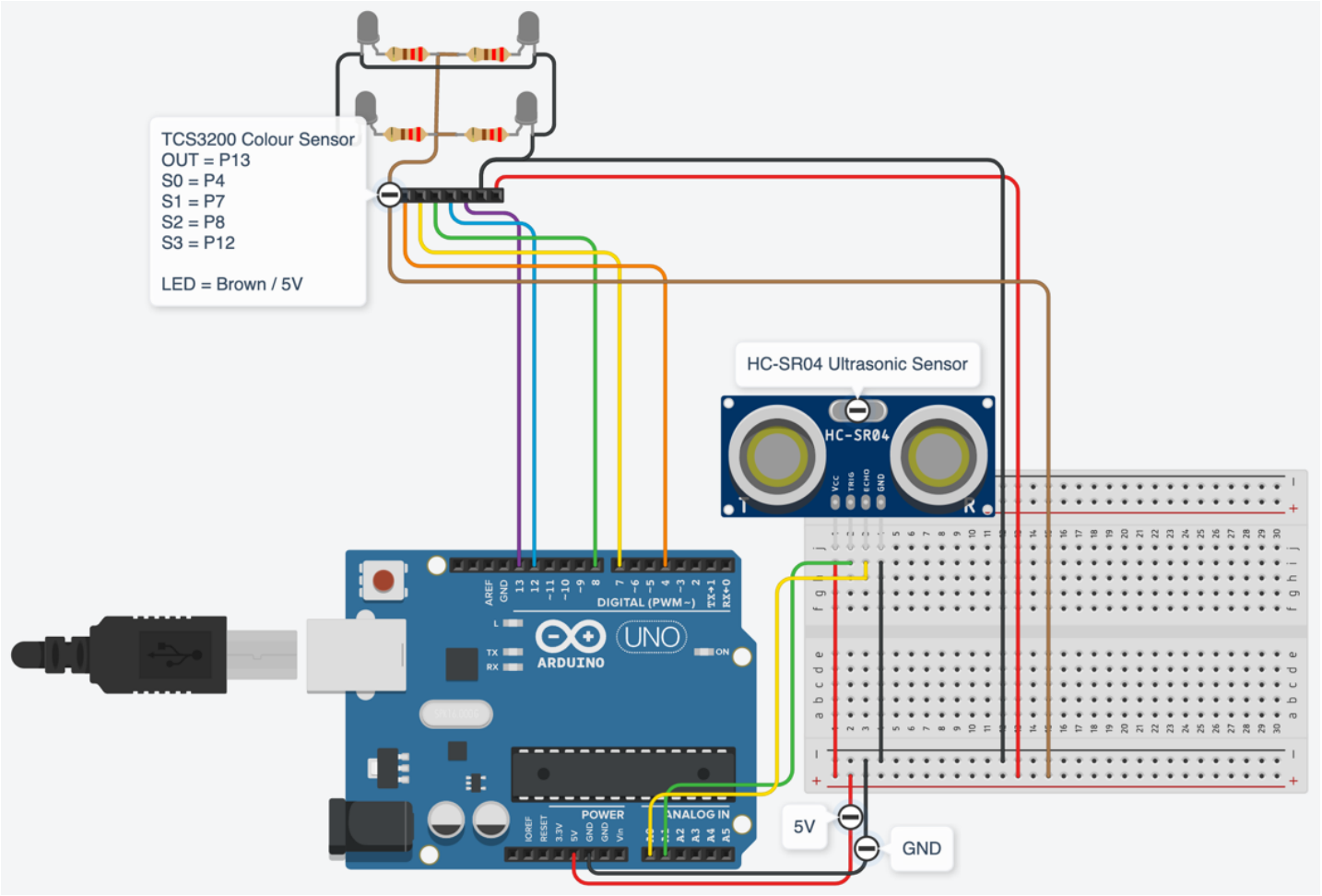

Figure 4 - Schematic diagram of Arduino setup (TinkerCAD)

3.2 Sensor Integration

The HC-SR04 was chosen for data collection on the location of the object. Once the location of the object was determined the arm could then move towards and collect the object. The module uses short burst high frequency waves emitted from the 'trigger' those sound waves then travel through the air and bounce off an object if encountered, the 'echo' sensor listens for these returning sound waves and distance is calculated based on echo time and the speed of sound.

$$Distance\ (m) = \frac{Echo\ Time\ (Seconds)}{2} \times Speed\ of\ Sound\ (\frac{m}{s})$$

The TCS3200 was chosen as a simple way to distinguish between different objects based on its colour. The sensor works by using an array of photodiodes with red, green, blue, and clear filters. Light hits the sensor and the photodiodes behind each colour filter will then generate a current proportional to the intensity of that colour. That generated current is then converted into a signal with a frequency that corresponds to the current. The colour is determined by comparing red, green and blue frequency readings, with the lowest value of the three being the detected colour.

3.3 Design additions.

Our team felt it was appropriate to include an improved base for the robot to be fastened down to as we had an incident that damaged the arm due to it breaking off the included wooden base. The new base design (Figure 5 - Base Design Concept) allowed for the installation of the TCS3200 Colours Sensor into the base, and for the Arduino and wiring to be hidden.

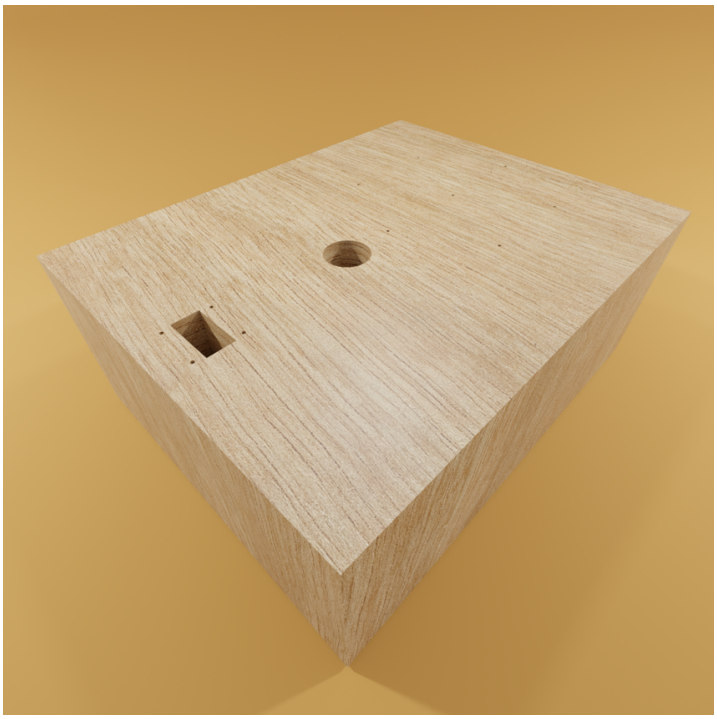

Figure 5 - Base Design Concept

A bracket for the HC-SR04 Ultrasonic sensor (Figure 6 - HC-SR04 bracket 3D model) was 3D printed. The design was created by modifying an existing bracket model (1).

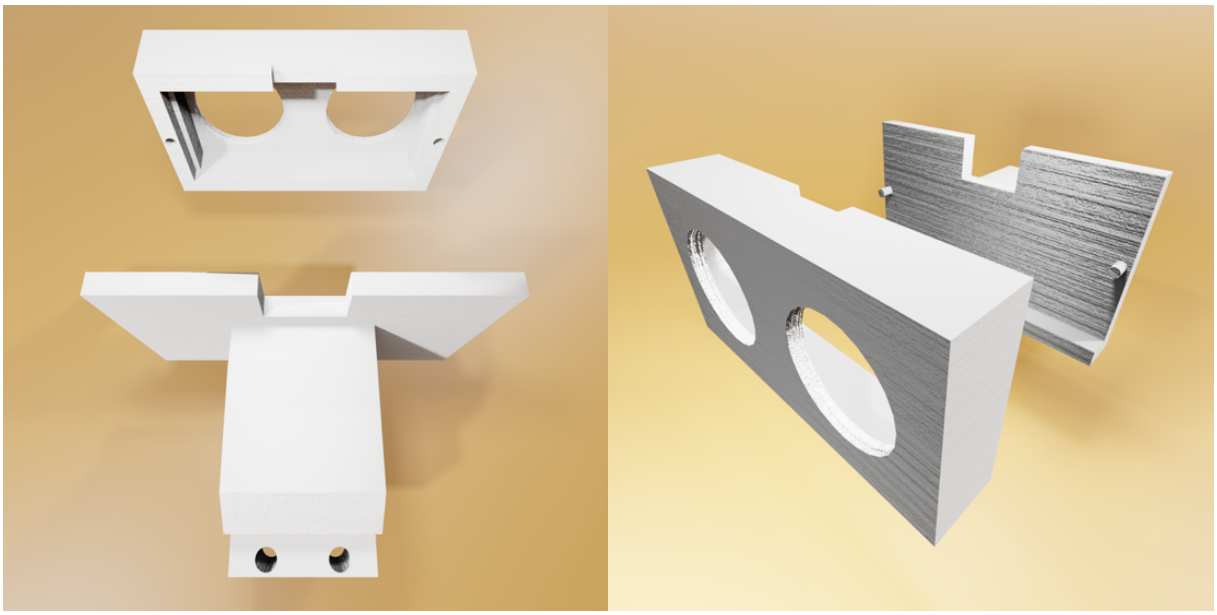

Figure 6 - HC-SR04 bracket 3D model.

3.4 Path Planning and Navigation

Given a desired end-effector location (x, y, z) in millimetre space (Figure 7 - Definition of the system of axes for calculation .), an inverse kinematics library (2) is used to calculate the required joint angles ($a_0,\ a_1, a_2, a_3$) for the arm to reach that point. This is crucial for robot programming as it allows us to define the target destination for the gripper rather than manually manipulate individual joints.

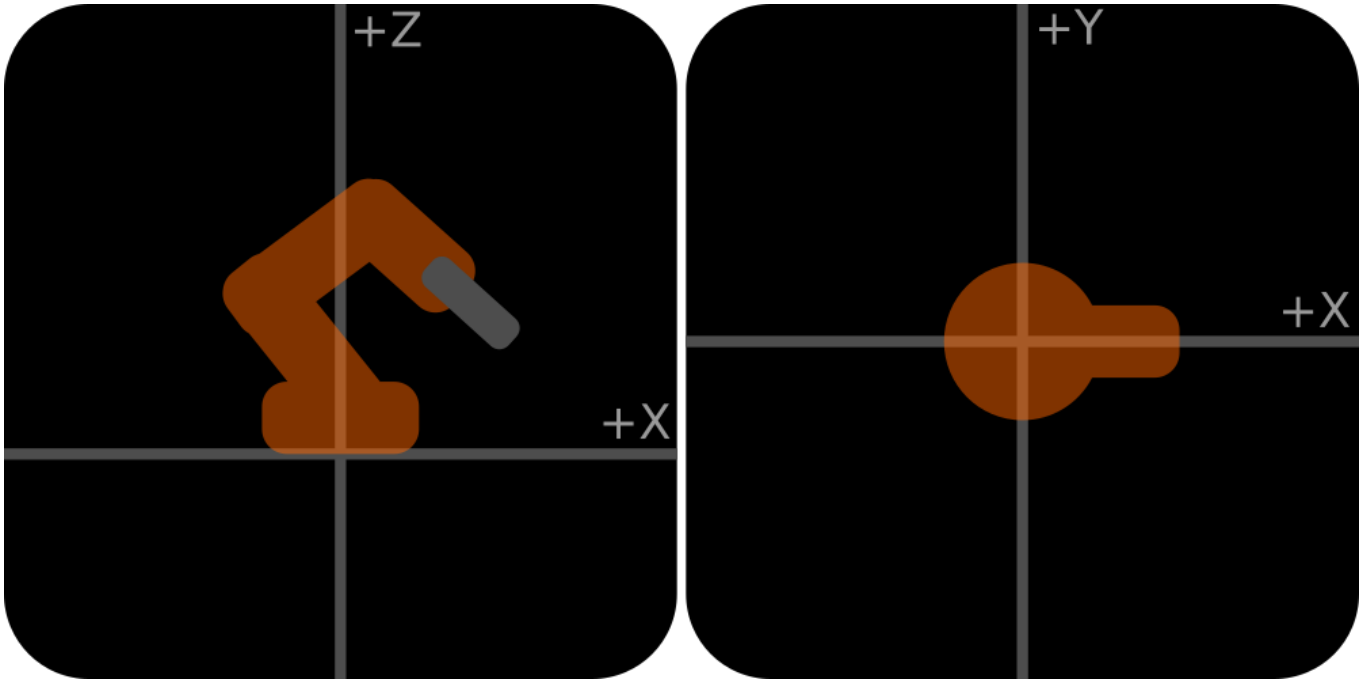

Figure 7 - Definition of the system of axes for calculation (2).

The library provides a solver that uses the rules of cosine to determine the appropriate angles needed to move the end effector (gripper) to a specific point on the x, y, z mm co-ordinate grid. To calculate the rotation of the base the cartesian coordinates are transformed to polar coordinates.

3.4.1 Inverse Kinematics with Cosine Rule

The Braccio arm, despite its simple design, requires solving inverse kinematics to control it. This means determining the motor angles (joint angles) needed for the end-effector to reach a desired position in space.

The Cosine Rule Solver:

The cosine rule, also known as the law of cosines, is a powerful tool in geometry for relating the sides and angles of a triangle. In the context of a 3-link Braccio arm, the cosine rule is strategically applied to two different triangles formed by the arm links to solve for the individual joint angles.

Braccio Arm Parameters:

For navigation the robot uses inverse kinematics to calculate the length and angles of the links $a_0,\ a_1, a_2, a_3$ depending on the position desired.

- The base, $a_0$to $a_1$relates to a length of 71.5 mm (L_0).
- The lower arm, a_1 to a_2 relates to a length of 125 mm (L_1).
- The upper arm, a_2 to a_3 relates to a length of 125 mm (L_2).
- The hand, a_4 to the end effector, relates to a length of 192 mm (L_3).

The definition of the links can be seen visually in Figure 8 - Braccio joint definitions.

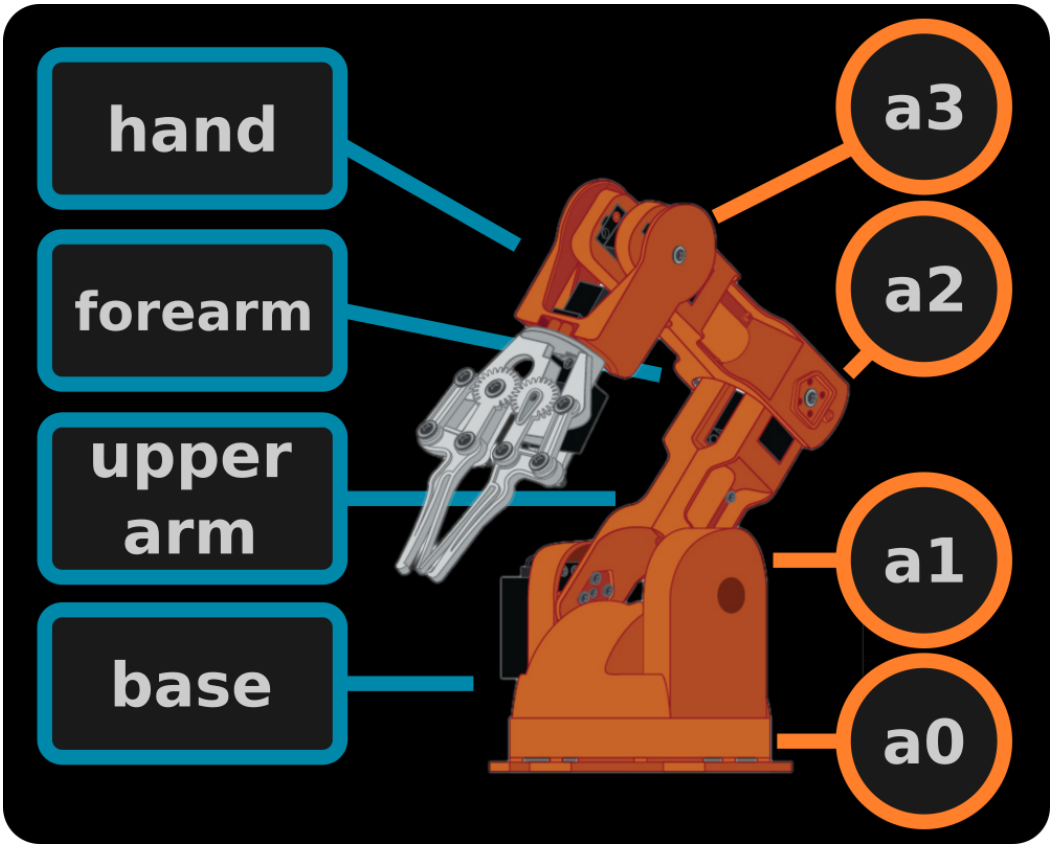

Figure 8 - Braccio joint definitions (2).

Note: Arduino documentation refers to these joints as M1 through to M6, with a0 = M1, a1 = M2, etc.

The 'a' notation is used by the Inverse Kinematics library, where 'M' notation is used to refer to the joints throughout the code.

Cosine Rule for Triangles:

The cosine rule (law of cosines) relates the sides and angles of a triangle. Given two side lengths (a, b) and the angle (θ) between them, we can solve for the third side (c) using the formula:

$$Cosine\ rule\colon\ c^2 = (a^2 + b^2 - 2ab) \cdot cos\emptyset$$

Define Arm Parameters:

Assign values to the lengths of the upper arm (L_1), forearm (L_2), and hand (L_3). These are typically known from the robot's specifications.

Calculate Intermediate Values:

1. Calculate the distance (d_0) from the base to the desired end-effector position in the X-Y plane using the distance formula (Pythagoras Theorem).

$$d_0 = \sqrt{x^2 + y^2}$$

2. Calculate the distance (d_1) between the desired Z-position of the end-effector and the hand link length (L_3).

$$d_1 = z + hand\ length$$

Reachability Check:

The desired position is reachable only if d_0 is less than the sum of the upper arm and forearm lengths (L_1+ L_2) and $d_1$is greater than zero. If either condition isn't met, there's no solution for the arm to reach that position.

Solving for a_0, a_1 and a_2:

We can identify two triangles formed by the arm links:

Triangle 1: Base - Upper Arm - Forearm. Here, we know two sides (L_1 and L_2) and the distance between them (d_0, calculated earlier). The cosine rule helps us solve a_1, the angle between the base and upper arm.

$$cos\ a_1 = \frac{(L_1^2 + L_2^2 - d_0^2)}{2 \cdot L_1 \cdot L_2}$$

To simplify the calculations:

$$a_0 = a_1$$

Triangle 2: Upper Arm - Forearm - Desired End-effector position (represented by d_0). Like triangle 1, we know two sides (L1 and L2) and their distance (d_0). This time, we solve for a_2, the angle between the upper and forearm.

$$cos\ a_2 = \frac{(d_0^2 + L_1^2 - L_2^2)}{2 \cdot d_0 \cdot L_1}$$

Solving for a_3:

Once a_1 and a_2 is determined, we can use the law of cosines again to solve for a_3, the angle between the forearm and hand. This step involves trigonometric calculations using the previously obtained theta values, arm link lengths (L_1, L_2, L_3), and the calculated d1 value.

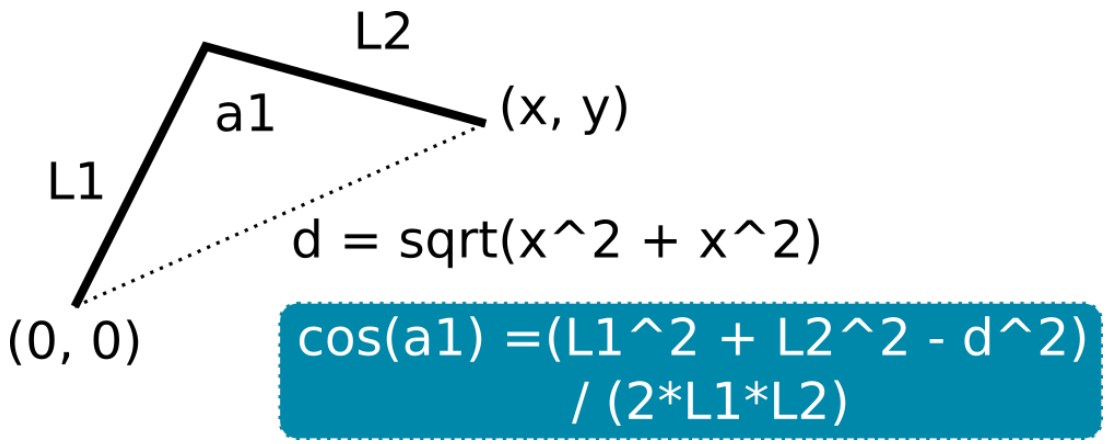

Figure 9 - Cosine solver diagram (2).

If all calculations are successful, you'll obtain the three joint angles (a_0, a_1, a_2, a_3) that correspond to the desired end-effector position.

3.5 Program Implementation

Figure 10 - Program Flow Chart visualises the loop performed by the code. Firstly, the *Find()* Function is called which performs a radar sweep and calculates the (x, y) co-ordinates of any object detected within a set range. Secondly, those co-ordinates are passed to the *Grab()* function, which utilises the Inverse Kinematics library to translate the (x, y) co-ordinates calculated into the servo rotations required to pick up the object at a fixed height of z = 0 mm. Finally, the *Sort()* function is called with *Colour_Scan()* as an argument, meaning the object is brought to the colour scanner, is scanned, and that reading is passed into the *Sort()* function, which drops the object into the corresponding colour bin.

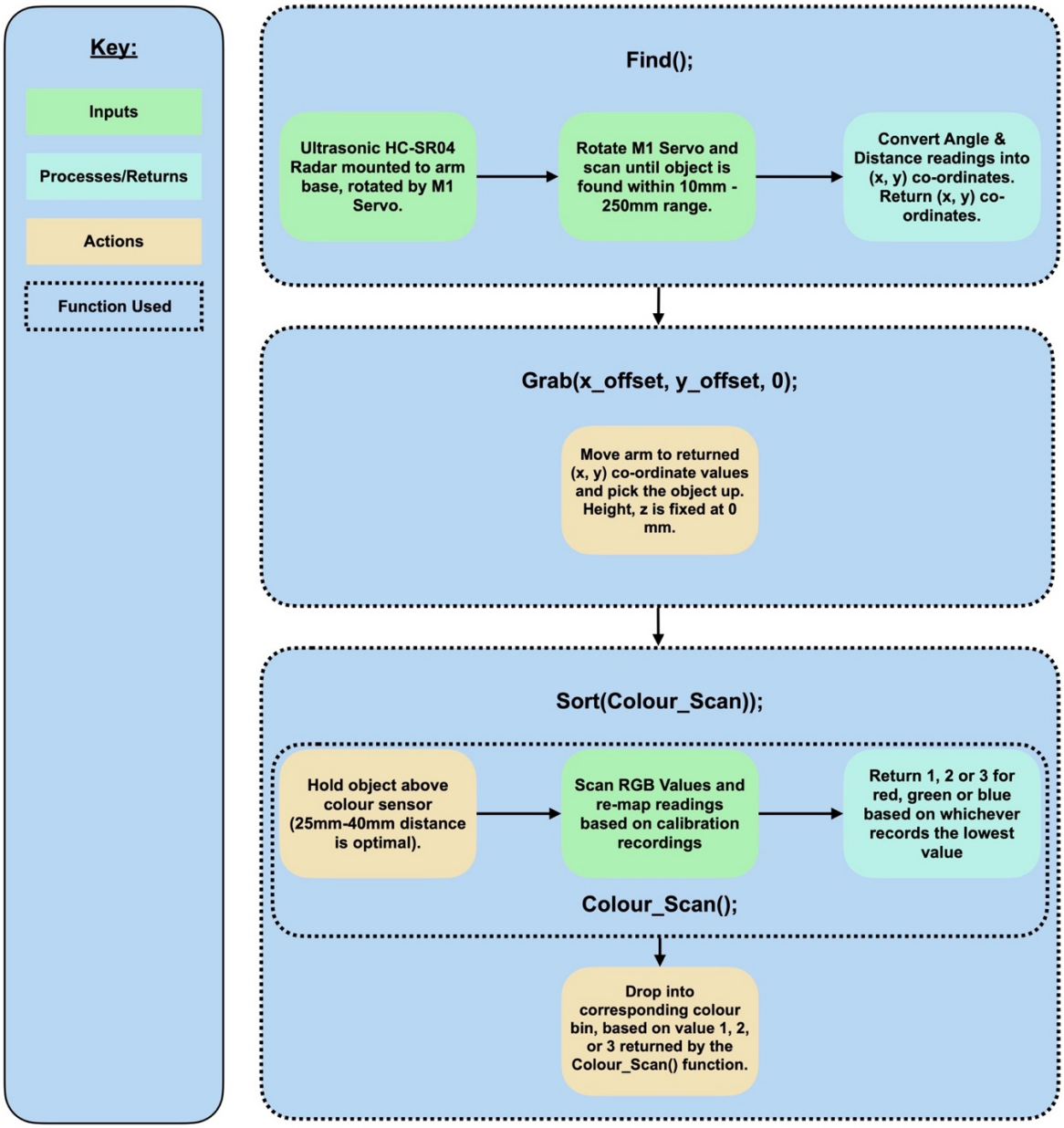

Figure 10 - Program Flow Chart

4 Results and Evaluation

4.1 Outcomes

The project aimed to augment a Braccio robotic arm with the capacity to locate, grasp, colour-detect, and sort acrylic blocks into designated bins. This goal was successfully achieved through the implementation of several key additions:

- **Structural Enhancements:** A custom-designed base and bins provided secure housing for electronics, improved cable management, and enhanced the overall system aesthetic. This significantly streamlined the workspace and contributed to a more professional final product.
- **Sensor Integration:** The addition of an HC-SR04 ultrasonic sensor, mounted on a custom-designed bracket, enabled non-intrusive distance measurement for object localisation. The bracket ensured both sensor stability and the preservation of the robotic arm's components.
- **End-Effector Modification:** A rubber grip was added to the Braccio's claw mechanism, dramatically enhancing its ability to grasp and retain blocks. This modification proved critical for reliable object manipulation, especially when block orientations were not ideal.

The complete product and these additions are shown in Figure 9, 10 & 11.

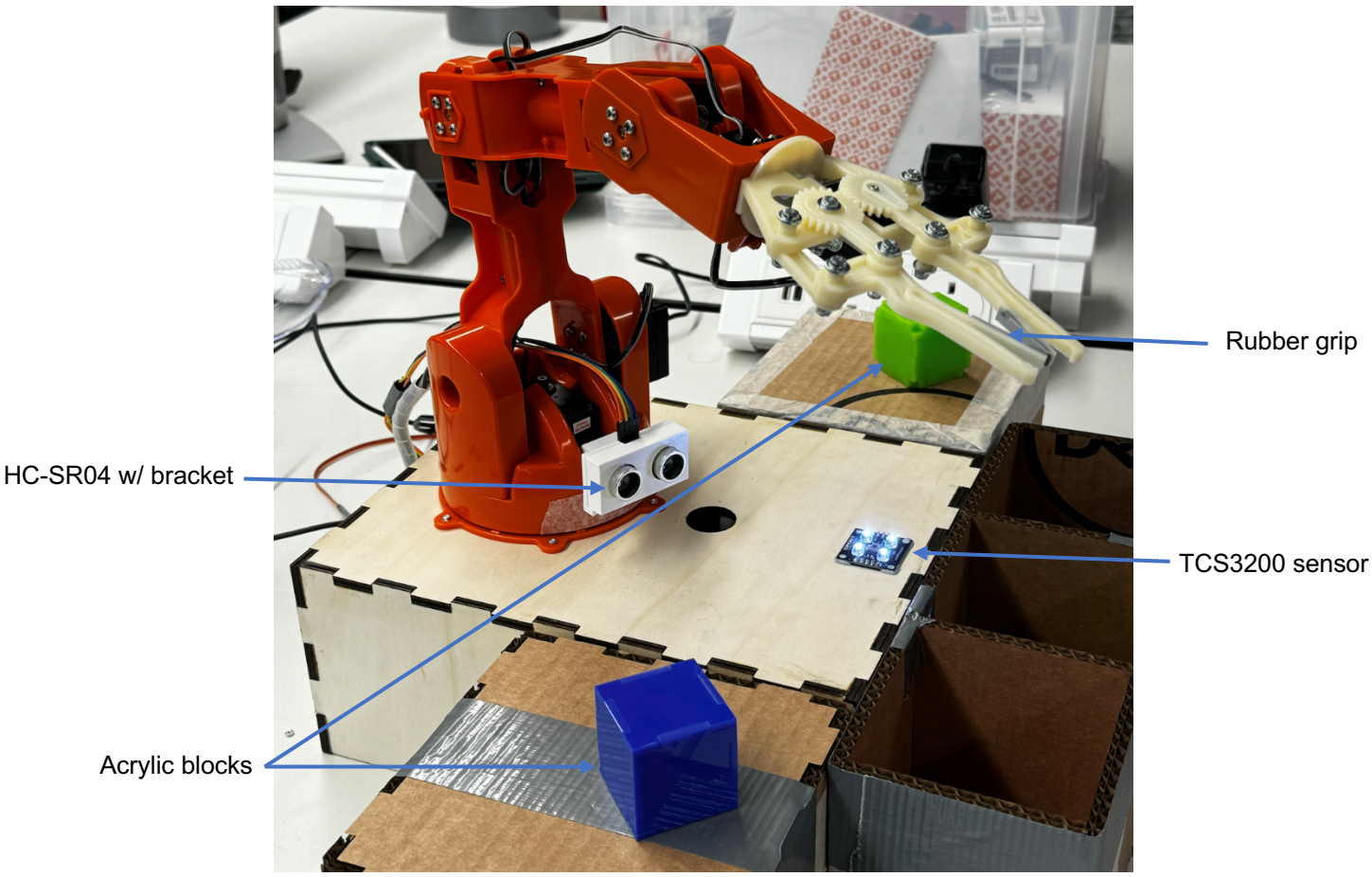

Figure 11 - Complete system.

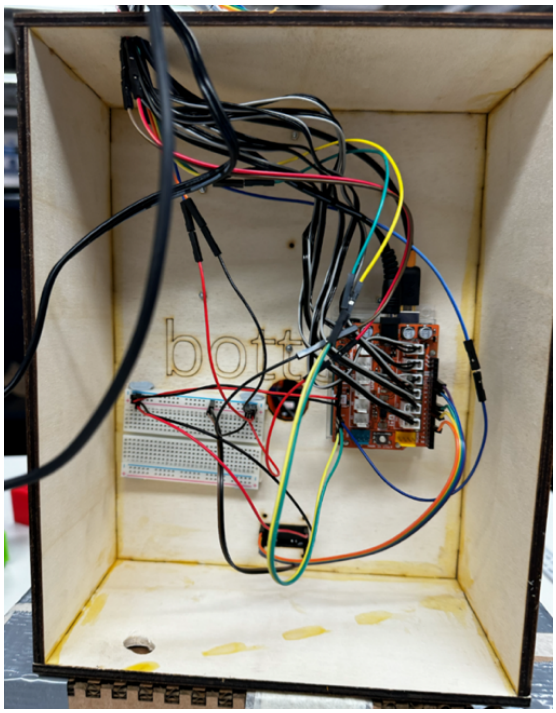

Figure 12 - Wiring.

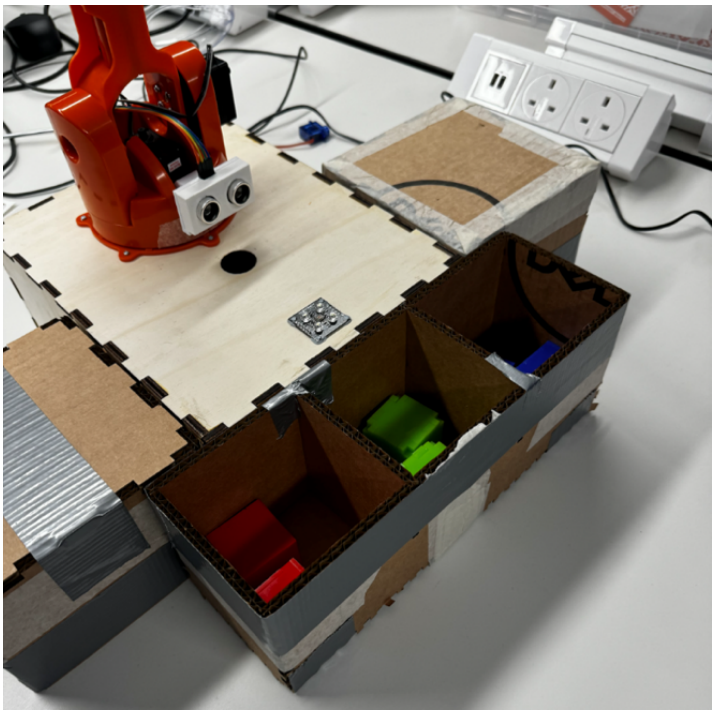

Figure 13 - Bin Configuration.

4.1.1 Limitations and Challenges

- **Sensor Constraints**: The HC-SR04 sensor demonstrated limitations in precision when determining block location. To achieve an acceptable success rate, considerable calibration was required. Additionally, block orientation and distance from the sensor significantly impacted system accuracy. Further exploration of alternative sensing technologies, such as computer vision, could potentially yield greater precision and flexibility in object detection.
- **Mechanical Reliability:** A substantial portion of the project timeline was dedicated to troubleshooting and repairing mechanical failures within the Braccio arm. Frequent servo malfunctions and general component wear were largely outside the team's control. These issues detracted from time that could have been focused on optimizing the object detection and sorting algorithms.
- **Time Constraints:** The combination of mechanical issues and sensor calibration resulted in limited time for fine-tuning the colour-sorting logic. Consequently, while successful, the accuracy and efficiency of the sorting process could have been further enhanced with additional development time.

4.2 Conclusions

Despite the challenges encountered, the project successfully demonstrated the integration of sensing, actuation, and basic decision-making within a robotic system. Key lessons were learned regarding the importance of hardware robustness and the need for ample calibration when deploying sensors in real-world applications.

Future work could explore the use of more robust robotic platforms, the integration of higher-fidelity sensors (potentially a vision-based system), and the refinement of sorting algorithms to increase both the speed and overall accuracy of the system.

5 Bibliography

1. **ArthurSh.** HC-SR04 Holder. *thingiverse.* [Online] June 2020. https://www.thingiverse.com/thing:4458193.

2. **Onchi, Eiji.** Inverse Kinematic Library for Arduino. *Github.* [Online] https://github.com/cgxeiji/CGx-InverseK.

6 Appendix

6.1 Code

```
/*
This code is designed to control a robotic arm (Braccio) equipped with an ultrasonic sensor and a
colour sensor. The main functionality of the code is to scan the surrounding area for objects,
detect their colour, and then sort them into respective bins based on their colour (red, green, or
blue).

The code utilizes the Braccio library for controlling the robotic arm's servos, the InverseK
library for inverse kinematics calculations, and includes functions for ultrasonic distance
measurement, colour detection, and various arm movements (such as grabbing, dropping, and sorting
objects).

The arm can scan a 180-degree area in front of it, detect the closest object within a specified
distance range, move towards the object, grab it, detect its colour, and then drop it into the
appropriate colour-coded bin. The process is repeated in a continuous loop.

Servo descriptions:

M1 = Arm Base.
M2 = Shoulder.
M3 = Elbow.
M4 = Wrist Vertical.
M5 = Wrist Rotation.
M6 = Gripper Control.
*/

#include <Braccio.h>
#include <Servo.h>
#include <InverseK.h>
#include <math.h>

// Servo Class Declarations

  Servo base;
  Servo shoulder;
  Servo elbow;
  Servo wrist_rot;
  Servo wrist_ver;
  Servo gripper;
  Servo Radar;

//---GLOBAL DECLARATIONS---

  //---Radar/Location Variables---
```

```
  //I/O

  const int trigPin = A1;
  const int echoPin = A2;

  // Location data

  float duration, distance;
  int angle_M1 = 0;
  float angle_M7 = 0;
  float x, y, z;

  // Struct definition for OffsetValues
  struct OffsetValues {
  float x_offset, y_offset;
  };

  //---Colour Sensor----
  //I/O

  const int Colour_Pin_S0 = 4;
  const int Colour_Pin_S1 = 7;
  const int Colour_Pin_S2 = 8;
  const int Colour_Pin_S3 = 12;
  const int Colour_Sensor_Out = 13;

  //Colour data

  int blue=0;
  int red=0;
  int green=0;
  int frequency = 0;

  // Colour Calibration Parameters
  #define Red_Lower_Bound 20
  #define Red_Upper_Bound 70
  #define Green_Lower_Bound 70
  #define Green_Upper_Bound 100
  #define Blue_Lower_Bound 60
  #define Blue_Upper_Bound 100

// Setup function
void setup() {

  // Braccio Startup function
  Braccio.begin();

  // InverseK library Setup
  Link base, upperarm, forearm, hand;

  base.init(71.5, Degrees_to_Radians(0.0), Degrees_to_Radians(180.0));
  upperarm.init(125, Degrees_to_Radians(15.0), Degrees_to_Radians(165.0));
```

```
  forearm.init(125, Degrees_to_Radians(0.0), Degrees_to_Radians(180.0));
  hand.init(192, Degrees_to_Radians(0.0), Degrees_to_Radians(180.0));

  InverseK.attach(base, upperarm, forearm, hand);

  // HC-SR04 IO Setup
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  // Colour Sensor IO Setup
  pinMode(Colour_Pin_S0, OUTPUT);
  pinMode(Colour_Pin_S1, OUTPUT);
  pinMode(Colour_Pin_S2, OUTPUT);
  pinMode(Colour_Pin_S3, OUTPUT);
  pinMode(Colour_Sensor_Out, INPUT);
  digitalWrite(Colour_Pin_S0, HIGH);
  digitalWrite(Colour_Pin_S1, LOW);

  // Open Serial Comms
  Serial.begin(9600);

  // Set arm to upright stance
  ResetArm();

  //---Setup Complete---
  }

// Resets arm to calibrated upright state
void ResetArm() {
  // Calibration:             M1, M2 + 3, M3 - 3, M4 - 5, M5 - 10
  Braccio.ServoMovement(30,   90, 90 + 3, 90 - 3, 90 - 5, 90 - 10, 10);
  }

// Conversion from the degrees to radians
float Degrees_to_Radians(float Degrees){
  return Degrees / 180.0 * PI - HALF_PI;
  }

// Conversion from radians to the degrees
float Radians_to_Degrees(float Radians) {
  return (Radians + HALF_PI) * 180 / PI;
  }

// Moves object to colour detector, scans and returns 1 for red, 2 for green, 3 for blue.
int Colour_Scan(){

  // Move to 40 mm above colour sensor located at (-160, 0) with gripper closed
  Go_to(-160, 0, 40, 73);

  //---Scan Red---
  digitalWrite(Colour_Pin_S2, LOW);
  digitalWrite(Colour_Pin_S3, LOW);
```

```
  frequency = pulseIn(Colour_Sensor_Out, LOW);

  // Re-map reading based on calibration
  frequency = map(frequency , Red_Upper_Bound, Red_Lower_Bound, 255, 0);
  red = frequency;

  //---Scan Green---
  digitalWrite(Colour_Pin_S2, HIGH);
  digitalWrite(Colour_Pin_S3, HIGH);
  frequency = pulseIn(Colour_Sensor_Out, LOW);

  // Re-map reading based on calibration
  frequency = map(frequency, Green_Upper_Bound, Green_Lower_Bound,  255, 0);
  green = frequency;

  //---Scan Blue---
  digitalWrite(Colour_Pin_S2, LOW);
  digitalWrite(Colour_Pin_S3, HIGH);
  frequency = pulseIn(Colour_Sensor_Out, LOW);

  // Re-map reading based on calibration
  frequency = map(frequency, Blue_Upper_Bound, Blue_Lower_Bound, 255, 0);
  blue = frequency;

  // ---Scanning Complete---
  // Select Lowest Colour Value & return 1, 2 or 3 for Red, Green or Blue

  if(red<blue && red<green){
    Serial.println("RED");
    return 1;
  }

  if(green<blue && green<red){
    Serial.println("GREEN");
    return 2;
  }

  if(blue<red && blue<green){
   Serial.println("BLUE");
   return 3;
  }
  }

// Measures ultrasonic distance in MM and returns it.
float Ultrasonic_Scan(){

  // Read echo time
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
```

```
  duration = pulseIn(echoPin, HIGH);

  // Convert time reading to distance, prints & returns distance.
  distance = 10*(duration*.0343)/2;
  Serial.println(distance);
  return distance;
  }

// Scans for object, returns detected x/y co-ordinates
OffsetValues Find() {

  // ---Local Variables---
  OffsetValues offsets = {0.0f, 0.0f};
  int radius = 40;

  // Loops through M1 base servo from 0 degrees => 85 at 2 degree intervals
  for (int M1 = 0; M1 <= 85; M1 += 2) {

    //Rotate Base
    Braccio.ServoMovement(25, M1, 93, 87, 85, 80, 10);
    delay(200);

    //Scan Distance, returns distance in MM
    distance = Ultrasonic_Scan();

    //Range Check, if < 250mm update angle and distance variables
    if (distance <= 250 && distance >= 10) {
      angle_M1 = M1 + 20;

      //Calculates and returns x/y co-ordinates
      offsets.x_offset = - (distance + radius) * cos(Degrees_to_Radians(angle_M1));
      offsets.y_offset = - (distance + radius) * sin(Degrees_to_Radians(angle_M1));
      return offsets;
    }
  }

  // Loops through M1 base servo from 110 degrees => 180 at 2 degree intervals
  for (int M1 = 110; M1 <= 180; M1 += 2) {

    //Rotate Base
    Braccio.ServoMovement(25, M1, 93, 87, 85, 80, 10);
    delay(200);

    //Scan Distance, returns distance in MM
    distance = Ultrasonic_Scan();

    //Range Check, if < 250mm update angle and distance variables
    if (distance <= 250 && distance >= 10) {
      angle_M1 = M1 + 20;

      //Calculates and returns x/y co-ordinates
      offsets.x_offset = - (distance + radius) * cos(Degrees_to_Radians(angle_M1));
```

```
      offsets.y_offset = - (distance + radius) * sin(Degrees_to_Radians(angle_M1));
      return offsets;
    }
  }
 }

// Uses InverseK library to move arm to (x, y, z, claw state)
void Go_to(float x, float y, float z, int claw){
  // ---Local Variables---
  float a0, a1, a2, a3;

  // Solve for given x, y, z co-ordinates, if it is possible to approach from 45 degrees
  if (InverseK.solve(x, y, z, a0, a1, a2, a3, Degrees_to_Radians(45))) {

    // Move Servos to solved angles
    float M1 = Radians_to_Degrees(a0);
    float M2 = Radians_to_Degrees(a1);
    float M3 = Radians_to_Degrees(a2);
    float M4 = Radians_to_Degrees(a3);

    // Calibration:                    M1, M2 + 3, M3 - 3, M4 - 5, M5 - 10
    Braccio.ServoMovement(30,          M1, M2 + 3, M3 - 3, M4 - 5, 90 - 10, claw);
  }

  // Approach from any angle possible if 45-degree approach is not possible
  else {
    // Solve for given x, y, z co-ordinates
    InverseK.solve(x, y, z, a0, a1, a2, a3);

    // Move Servos to solved angles
    float M1 = Radians_to_Degrees(a0);
    float M2 = Radians_to_Degrees(a1);
    float M3 = Radians_to_Degrees(a2);
    float M4 = Radians_to_Degrees(a3);

    // Calibration:                    M1, M2 + 3, M3 - 3, M4 - 5, M5 - 10
    Braccio.ServoMovement(30,          M1, M2 + 3, M3 - 3, M4 - 5, 90 - 10, claw);
  }
  }

// Grabs object at x/y/z co - ordinates
void Grab(float x, float y, float z){
  Go_to(x, y, z, 10);
  Go_to(x, y, z, 73);
  Go_to(x, y, z + 200, 73);
  }

// Drops object at x/y/z co - ordinates
void Drop(float x, float y, float z){
  Go_to(x, y, z + 200, 73);
  Go_to(x, y, z, 73);
  Go_to(x, y, z, 10);
```

```
  }

// Drops into bins 1, 2, or 3 a.k.a Red, Green, Blue
void Sort(int colour){

  // Red bin
  if (colour == 1){
  Drop(-240, -140, 0);
  }

  // Green Bin
  if (colour == 2){
  Drop(-254, 0, 0);
  }

  // Blue Bin
  if (colour == 3){
  Drop(-254, 100, 0);
  }
  }

//Main Loop
void loop() {
  ResetArm();

  //Find object, returns x/y co-ordinates
  OffsetValues offsets = Find();

  // Grab found object
  Grab(offsets.x_offset, offsets.y_offset, 0);

  // Sort into colour bins based on Colour_Scan
  Sort(Colour_Scan());

  }
```