Faculty of Engineering
Department of Electrical and Electronics Engineering

EE 321 PROJECT FINAL REPORT

Spring 2024

ZUMO ROBOT

Submitted by

Yunus Emre Ozan **Seyit Kubilay Uluçay**
S021369 **S029261**

APPROVAL PAGE

Faculty of Engineering
Department of Electrical and Electronics Engineering

EE 321 PROJECT FINAL REPORT

Spring 2024

Yunus Emre Ozan
Seyit Kubilay Uluçay

ZUMO ROBOT

Video Link: https://www.youtube.com/watch?v=27th-UQyDgE

ABSTRACT

This project focuses on the details of a system designed to diversify the usage of the Zumo robot, which is developed on Arduino board. The system aims to enhance the performance exhibited by the robot in specific scenarios. Towards this goal specific algorithms is incorporated to ensure the effective operation of the project. The uniqueness of the created algorithm is a notable advantage of the project over similar studies. The preferred code approach allows the system to detect, count and notify the user of desired obstacles in the desired area using an infrared sensor, LED and a buzzer that increases its functionality.

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

Focus of the Zumo robot project, aimed at increasing educational usage and providing exposure to C programming on a microcontroller, is to complete the desired tasks to observe sensor and motor interactions. The project aims to enable Zumo to count the different number of obstacles in a circular area limited by white tape and after count process it notify the user by light.

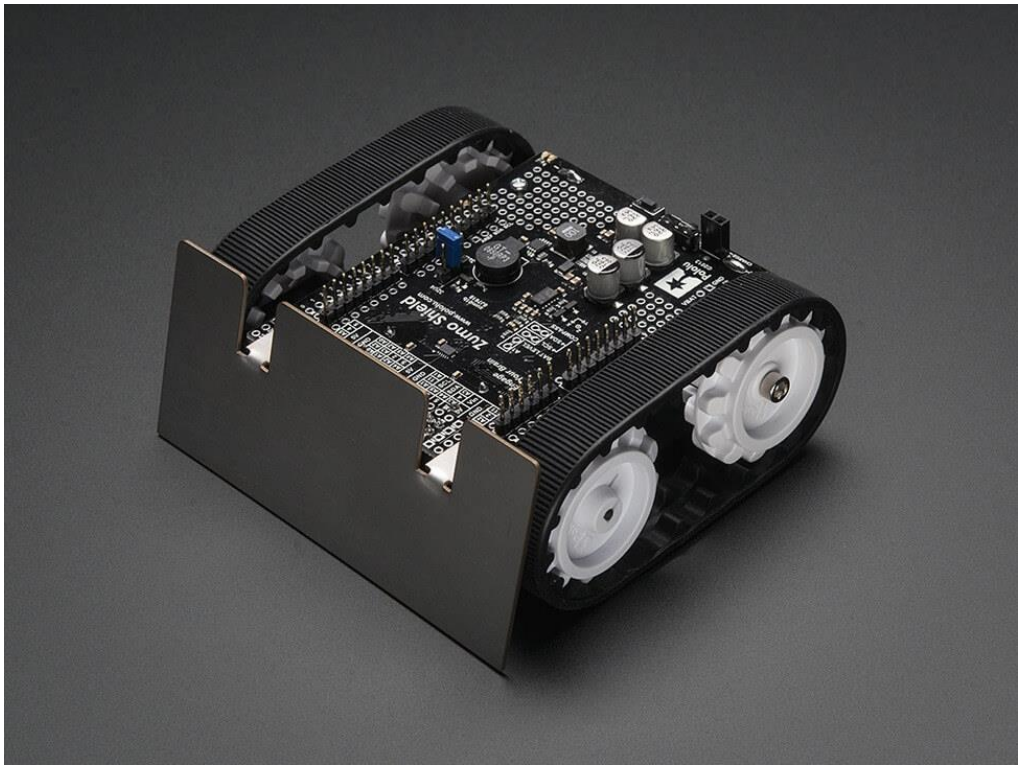

Figure 1: Zumo Robot

The main objectives of this project are as follows:

1. Software Development: In order for the planned system to work and fulfill its assigned tasks, software will be developed that includes libraries consisting of infrared sensors and reflection sensors that can detect the contrast between black and white areas. In the project where we aim to reach the limits of the sensor's possibilities, it aims to scan the area by going around the corners of the arena and strengthening the accuracy by seeing it from different angles.

2. System Integration and Testing: After completing the software development phase, our focus is to measure the performance of a system with all compatible components in possible scenarios. Here, it is planned to test objects in different numbers and placements. Tests will be recorded live in 6 different experimental setups and reported later.

2 METHODOLOGY

The project aims to develop an object counting system that allows the zumo robot to count objects when it sees them with the IR sensor and to notify the user by illuminating the obstacles with the help of LED light after making certain turns during the project. The sensor plays an important role in helping the zumo robot count obstacles while driving and determine whether they are in the area delimited by the white tape.

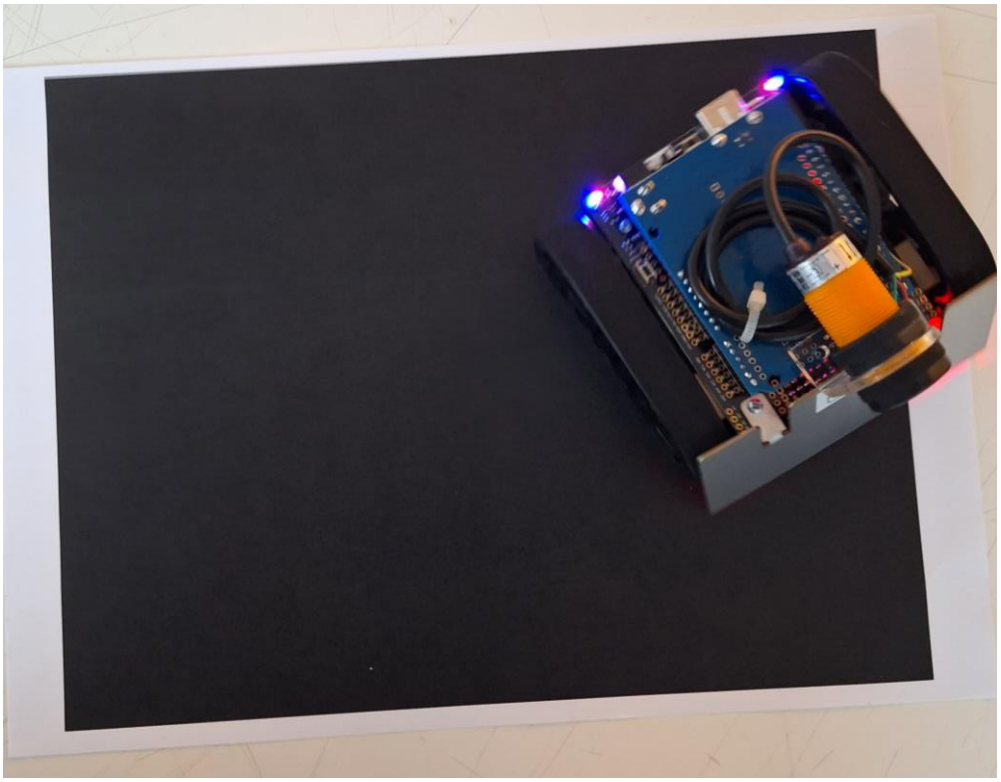

Figure 2: Team 16 Zumo Robot

2.1 Problem Formulation

In the project, it was emphasized that in case the IR sensor does not have a sufficiently distant view and has difficulty scanning rowed objects, an algorithm should be worked on to prevent this. In the first stage, it was found insufficient for the robot to count the objects by turning full

circle on its own axis in the middle of the arena. It has been observed that the code that will work in this area has the ability to see only the objects around it.

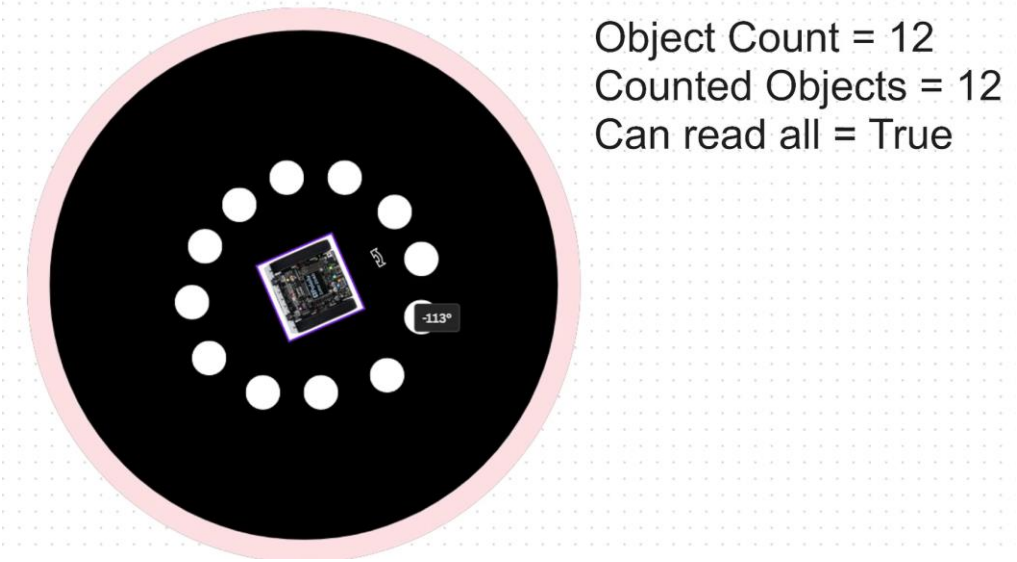

Figure 3: First Assumption

As shown in the example below, based on the simulation tests, it was decided to work on a new algorithm to ensure the robot, which has difficulty seeing consecutive objects, achieves more accurate test results.

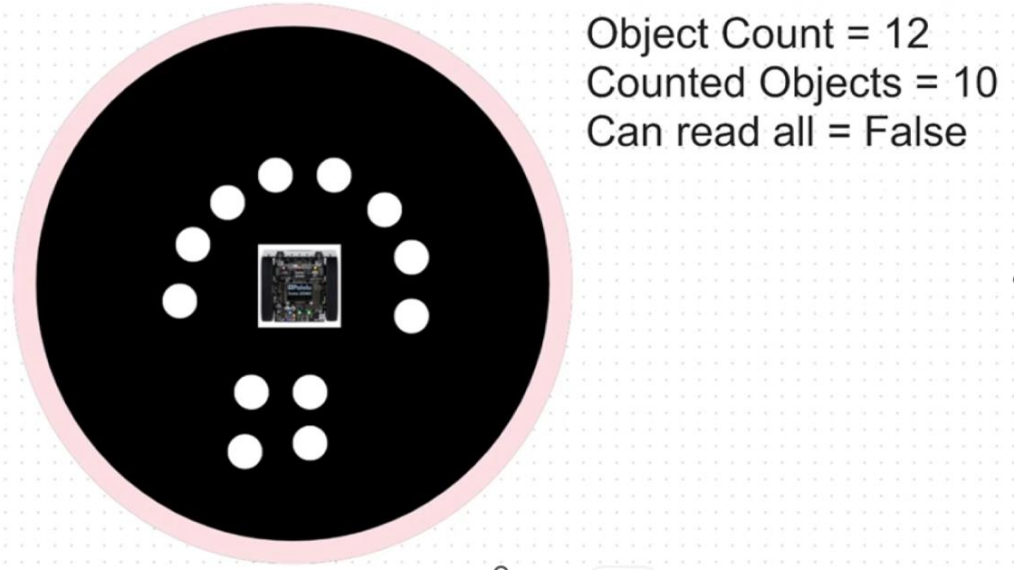

Figure 4: First Assumption Failed Case Simulation

2.2 Proposed Solution (Code Analysis)

Our algorithm's system diagram is shown below.

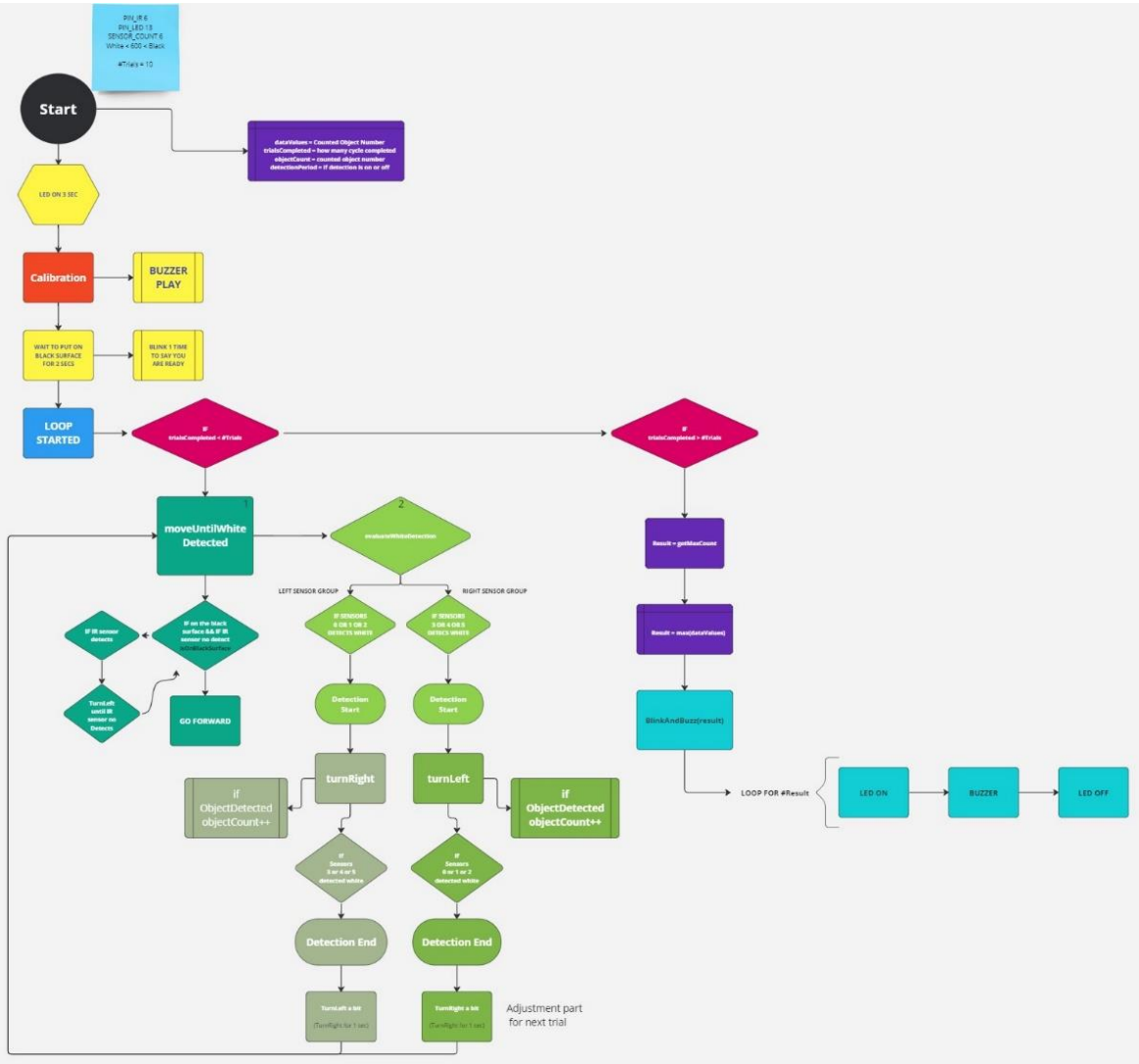

Figure 5 System Block Diagram

Our algorithm designed to work with these steps

1. Zumo starts, Led states zumo is powered on, calibration starts.
2. When calibration ends, buzzer plays music.
3. Until our desired number of trials have reached (it is 10 in our case) we call two methods, **moveUntilWhiteDetected** and **evaluateWhiteDetection.**
4. **moveUntilWhiteDetected** method leads us to reach the edge of the circle while avoiding the hitting objects.

5. **evaluateWhiteDetection** method leads us to which part of the reflectance sensors has detected the white edge, so our zumo can adjust itself with respect to the white edge detection.
6. Then **detectionPeriod** starts and zumo turns left or right with respect to the sensor detection from left or right sensor groups until detecting white edge again, while turning zumo counts objects detected by the IR sensor and increment the **objectCount** variable by one.
7. When zumo detects white edge, robot stops and stores **objectCount** in the **dataValues** variable.
8. Then zumo rotates back to original position, by turning opposite direction until detecting white, as soon as zumo detects white edge again the robot stops and adjust itself to center for small amount and increments the **trialsCompleted** by one then repeats the cycle between step 4 and 8 until our desired number of trials have reached.
9. When we reach the desired number of the trials zumo stops and we simply get maximum value of the **dataValues** and assing it to variable named resault, because we thought it would be best approach to estimate the object count.
10. Then we call the **BlinkAndBuzz** method and it basically repeats lighting up the LED and closing down with playing buzzer as many times as our result value.

```
#include <ZumoMotors.h>
#include <ZumoReflectanceSensorArray.h>
#include <ZumoBuzzer.h>

#define PIN_IR 6
#define PIN_LED 13
#define SENSORS_COUNT 6
#define THRESHOLD_WHITE 550  // Adjust based on sensor readings for white
#define NUM_TRIALS 10
```

Figure 6: Code pt.1

The initial lines of our Zumo code involve the inclusion of several libraries. The first library, #include **<ZumoMotors.h>,** is used for controlling the robot's motors. The second library, #include **<ZumoReflectanceSensorArray.h>,** allows the sensors to detect white and black

lines. Finally, #include <**ZumoBuzzer.h**> is included to control the buzzer, which will be used to signal the end of calibration.

After including the necessary libraries, we define some pins and constants. The **MZ80 IR** sensor is connected to pin 6, the LED to pin 13, and we define the number of sensors as 6. We set the threshold for detecting white at 550, and the number of desired trials to 10. Last two definitions are important because in **THRESHOLD_WHITE** we can compare which color is white or black thanks to the our reflectance sensor. **NUM_TRIALS** is for the number of trials in total of the zumo have to complete before the blinking.

```
unsigned int sensorReadings[SENSORS_COUNT];
int dataValues[NUM_TRIALS];
int trialsCompleted = 0;
int objectCount = 0;
bool detectionPeriod = false;
```

Figure 7: Code pt.2

The variables **sensorReadings, dataValues, trialsCompleted, objectCount, and detectionPeriod** are also defined to store sensor readings, object count in each trial, the count of completed trials, the number of objects detected in single trial, and whether the detection period is active.

We use **detectionPeriod** because we do not always want to count objects, so with that Boolean variable we can basically adjust when we want to count objects.

```
unsigned int sensorReadings[SENSORS_COUNT];
int dataValues[NUM_TRIALS];
int trialsCompleted = 0;
int objectCount = 0;
bool detectionPeriod = false;

void setup() {
  Serial.begin(9600);
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_IR, INPUT);
  //Start with open led to show you are calibrating
  digitalWrite(PIN_LED, HIGH);
  delay(3000);
  digitalWrite(PIN_LED, LOW);
  delay(3000);

  //Calibration start
  sensors.init();
  for (int i = 0; i < 400; i++) {
    sensors.calibrate();
    if (i % 50 == 0) {
      Serial.print("Calibrating");
    }
  }
  Serial.println("\nCalibration complete.");
  buzzer.play("!L16 V10 cdegreg4"); // Signal end of calibration

  delay(2000); // Put me to the black surface...

  //Blink one time fastly to declare you are starting
  digitalWrite(PIN_LED, HIGH);
  delay(500);
  digitalWrite(PIN_LED, LOW);
}
```

Figure 8: Code pt.3

In the setup function, we define the pin modes. It sets the initial state of the Zumo robot. Serial communication is initialized and we set **PIN_LED** as output and **PIN_IR** as input. Indicates the calibration process to the user by flashing the LED. Sensors are calibrated and when calibration is completed, "Calibration completed." appears on the serial monitor. The text is

sent and a melody is played in buzzer. After a 2-second pause, the LED will flash briefly, indicating that the robot is ready for operation. These processes ensure that the robot works correctly and ready.

```
void loop() {
  if (trialsCompleted < NUM_TRIALS) {
    moveUntilWhiteDetected();
    evaluateWhiteDetection();
  } else {
    int result = getMaxCount();
    delay(2000);
    blinkAndBuzz(result);
    motors.setSpeeds(0, 0);  // Stop the robot
    while(true);  // Halt further actions
  }
```

Figure 9: Code pt.4

The loop function manages the main operating cycle of the robot. If the number of completed trials is less than **NUM_TRIALS**, it moves until white is detected with **moveUntilWhiteDetected()** and evaluates the white detection with **evaluateWhiteDetection()**. When all trials are completed, it finds the maximum number of objects with **getMaxCount(),** waits for 2 seconds, displays this number with LED and buzzer with **blinkAndBuzz(result),** stops the motors and enters an infinite loop, preventing the robot

from performing any further operations. This structure ensures that the experiments are completed and the results are delivered to the user.

```
void moveUntilWhiteDetected() {
    while (true) {
        sensors.readLine(sensorReadings);
        if (digitalRead(PIN_IR) == LOW) {          // Check if the IR sensor detects an object
            while (digitalRead(PIN_IR) == LOW) {   // Turn left until the IR sensor no longer detects anything
                motors.setSpeeds(-150, 150);       // Turn left
            }
            motors.setSpeeds(0, 0);                // Stop turning once the object is no longer detected
        }
        else if (isOnBlackSurface()) {             // Continue moving forward if on black surface and no object detected
            motors.setSpeeds(100, 100);            // Move forward
        } else {
            motors.setSpeeds(0, 0);                // Stop motors if white is detected
            break;
        }
    }
}

bool isOnBlackSurface() {
  for (int i = 0; i < SENSORS_COUNT; i++) {
    if (sensorReadings[i] < THRESHOLD_WHITE) {
      return false;  // White detected
    }
  }
  return true;  // No white detected
}

void evaluateWhiteDetection() {
  if (sensorReadings[0] < THRESHOLD_WHITE || sensorReadings[1] < THRESHOLD_WHITE || sensorReadings[2] < THRESHOLD_WHITE) {
    // Left sensors detected white, turn right
    detectionPeriod = true;
    turnRight();
  } else if (sensorReadings[3] < THRESHOLD_WHITE || sensorReadings[4] < THRESHOLD_WHITE || sensorReadings[5] < THRESHOLD_WHITE) {
    // Right sensors detected white, turn left
    detectionPeriod = true;
    turnLeft();
  }
}
```

Figure 10: Code pt.5

The `**moveUntilWhiteDetected**` function runs continuously until the white surface is detected. The IR sensor turns left when it detects an object in order to avoid hitting objects, and stops when the detection is finished. It moves forward on a black surface, and stops the motors when a white surface is detected. The `**isOnBlackSurface**` function determines whether the sensors are on a black surface by checking the white threshold value. If all sensors are above the white threshold value, it means it is on a black surface. The `**evaluateWhiteDetection**` function enables rotation to the appropriate direction when the left or right sensor groups detect a white

surface. If the left sensors detect white, it turns right, and if the right sensors detect white, it turns left. This mechanism allows the robot to move and change direction on the correct surface.

```
void turnLeft() {
    while (!isLeftGroupWhite()) {
        motors.setSpeeds(-150, 155);  // Continue turning left
        checkForObject();  // Check for objects and count if detected
    }
    motors.setSpeeds(0, 0);  // Stop turning
    delay(200);  // Small delay to ensure complete stop
    dataValues[trialsCompleted++] = objectCount;  // Store the count
    objectCount = 0;  // Reset count
    detectionPeriod = false;  // End detection period
    while (!isRightGroupWhite()){
      motors.setSpeeds(155, -150);
    }
    // Turn right for a brief period
    motors.setSpeeds(-130, 130);
    delay(200);  // Turn right for 1 second
    motors.setSpeeds(0, 0);  // Stop turning
}
```

Figure 11: Code pt.6

turnLeft and **turnRight** function operates similarly but in the opposite direction. These function makes the robot turn left/right until the left/right group of sensors detects a white line. While turning, the robot continuously checks for objects using the IR sensor and increments the object count if any are detected. Once a white line is detected by the left/right sensors, the robot stops turning, records the object count, and resets it. Then robot rotates opposite direction to reach original position, after a brief delay, the robot then makes a slight right/left turn to ensure it is correctly aligned for the next movement.

```
void checkForObject() {
  if (detectionPeriod && digitalRead(PIN_IR) == LOW) {
    objectCount++;
    delay(200);  // Debounce delay
  }
}
```

Figure 12: Code pt.7

The **checkForObject** function monitors the IR sensor during the detection period to check for objects. If the IR sensor detects an object, the object count is incremented, and a 200 ms delay is introduced to eliminate debounce effects. This function ensures accurate counting of objects detected by the robot.

```
int getMaxCount() {
  int maxCount = 0;
  for (int i = 0; i < trialsCompleted; i++) {
    if (dataValues[i] > maxCount) {
      maxCount = dataValues[i];
    }
  }
  return maxCount;
}
```

Figure 13: Code pt.8

The **getMaxCount** function iterates through the recorded data values and returns the maximum count of objects detected in any trial. This function helps in determining the trial with the highest number of detected objects.

```
void blinkAndBuzz(int count) {
  for (int i = 0; i < count; i++) {
    digitalWrite(PIN_LED, HIGH);
    buzzer.playFrequency(2000, 500, 5);
    delay(500);
    digitalWrite(PIN_LED, LOW);
    delay(500);
  }
}
```

Figure 14: Code pt.9

Lastly, the **blinkAndBuzz** function uses the LED and buzzer to signal the count of detected objects. For each count, the LED is turned on and the buzzer plays a frequency for 500 milliseconds, followed by a 500-millisecond delay with the LED turned off. This visual and auditory feedback helps in understanding the number of detected objects.

3 RESULTS AND DISCUSSIONS

Thanks to unique detection algorithm code, we can count objects from different perspectives and get more precise results and zumo robot is used to analyze and calculate the number of objects in area with different scenarios from the IR sensor. The zumo robot works as a result of the calibration and continues to approach the white border until it completes 10 attempts and then continues this by scanning objects. In the image below, one of the possible routes of the robot is given. A more detailed explanation along with demo videos is available in the video link.

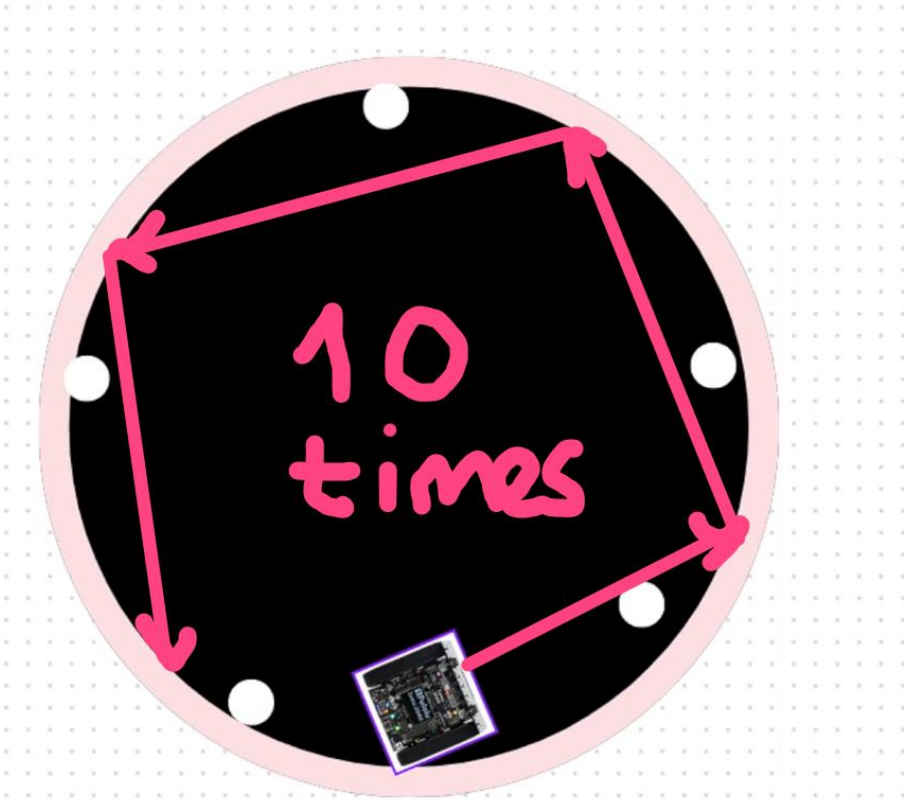

Figure 15: Algorithm Simulated Case

3.1 Results

To provide a comprehensive evaluation of the functionality and performance of the object detection system, we designed a series of experiments covering a variety of scenarios. These

experiments cover a number of factors, including changes in the number of objects, angles from the center, distances, and corner cases. We also prioritized the reproducibility of experiments to ensure consistent and reliable results. For experiments, we will design objects of different angles and different numbers. Each scenario will involve a different combination of these factors to evaluate the robustness and accuracy of the system. By conducting these experiments, we will gain valuable information regarding the system's ability to detect objects placed at different angles and different sizes. This rigorous testing approach will help us verify the performance of the object detection system.

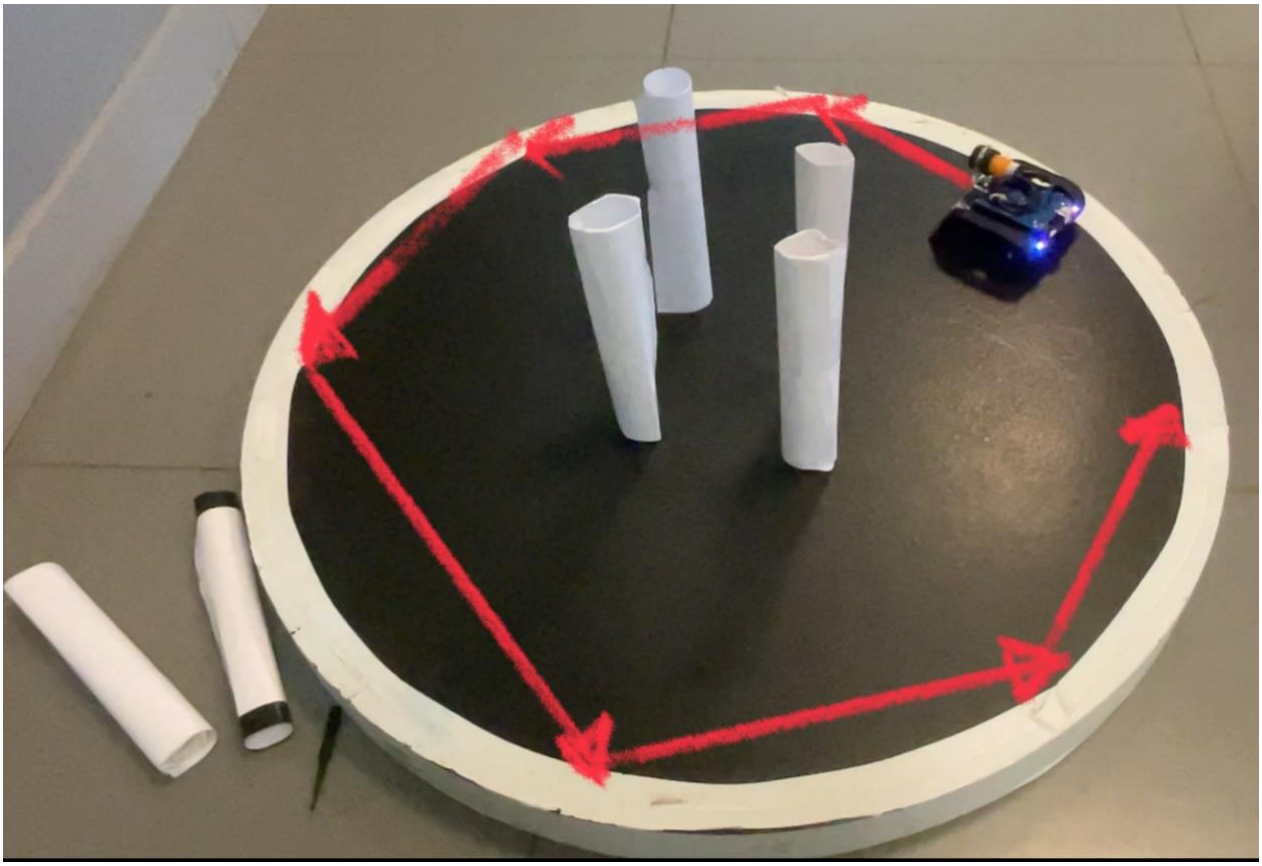

Figure 16: Real Life Experiment

EXPERIMENT 1 (3 object): 3 objects counted correctly.

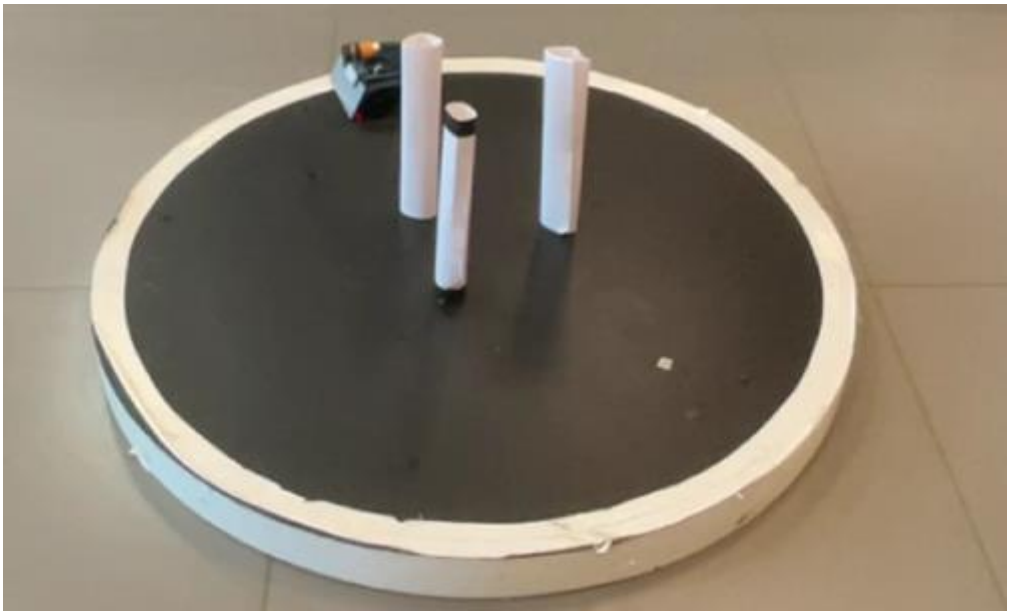

Figure 17: Experiment 1

EXPERIMENT 2 (4 object): 4 objects counted correctly.

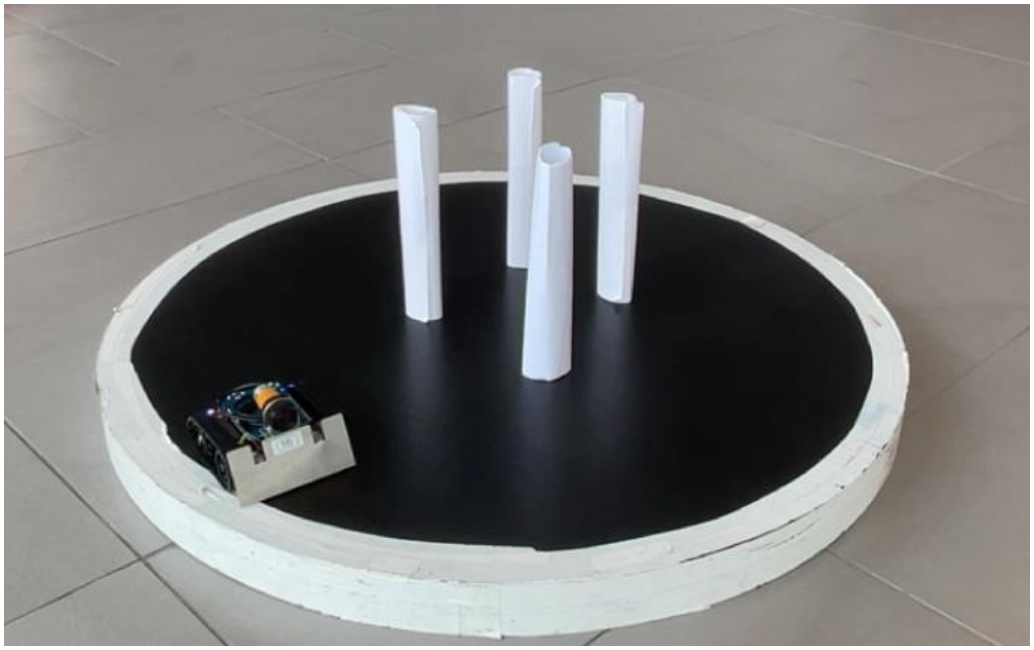

Figure 18: : Experiment 2

EXPERIMENT 3 (6 object – domino example -): 6 objects counted correctly.

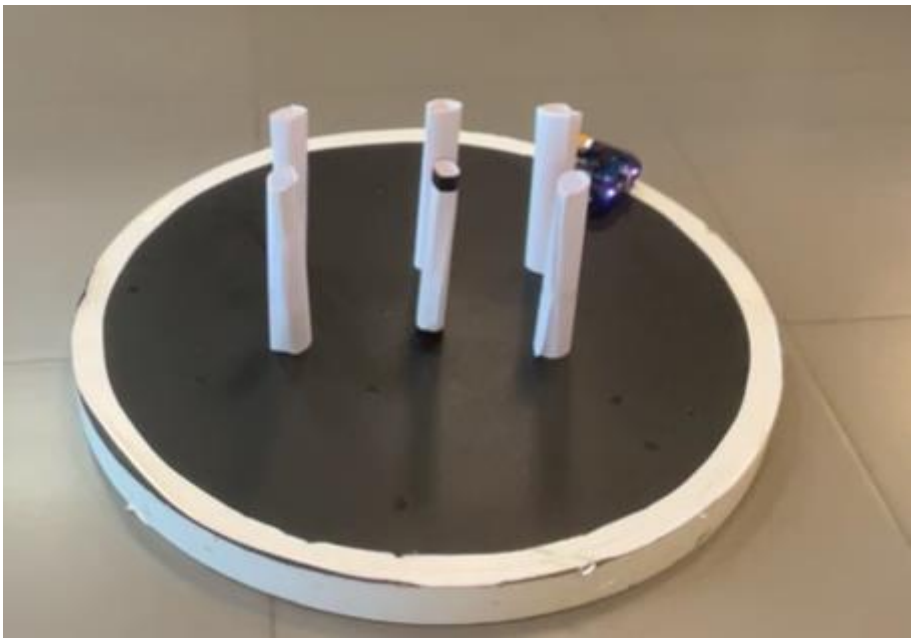

Figure 19: Experiment 3

EXPERIMENT 4 (5 object -star example-): 5 objects counted correctly.

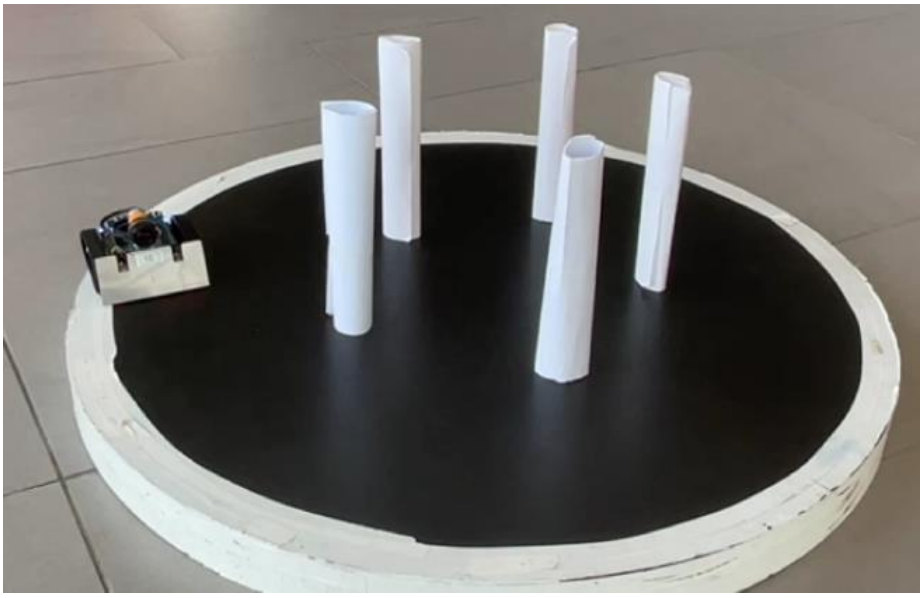

Figure 20: Experiment 4

EXPERIMENT 5 (5 object -cross example-): 5 objects counted correctly.

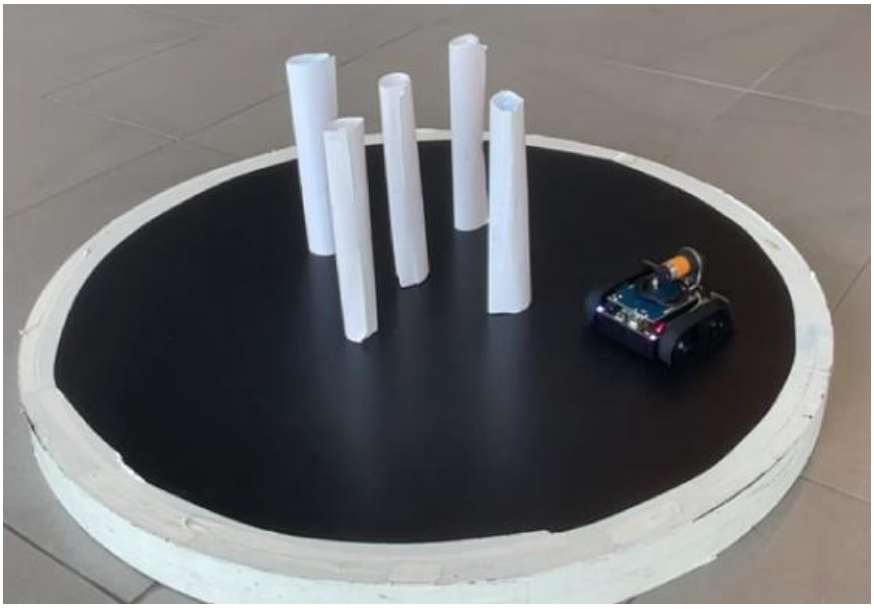

Figure 21: Experiment 5

EXPERIMENT 6 (edge example): FAILED

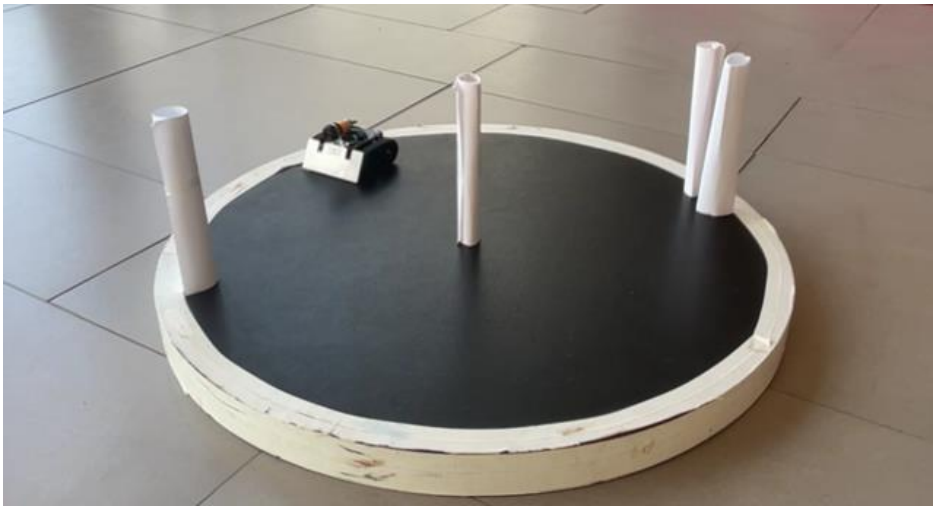

Figure 22: Experiment 6

3.2 Discussion

We see that we get correct results after testing the robot with a different number of different obstacles. We achieved this thanks to the different code we created. Different examples such as 3 pieces, 4 pieces, 5 pieces (star and cross), 6 pieces (domino stones) and obstacles at the ends were tried in real life. Since the basis of the code is to go to the corners of the circle and scan, it can sometimes count consecutive objects as missing even after 10 scans. This situation is normal with sufficient sensors and facilities.

The last experiment failed because the objects on the edge were next to the white line that determined the boundaries of the arena and the robot had to move forward until it saw the white line, so it did not see that object.

4 CONCLUSIONS AND FUTURE WORKS

In this project, we successfully implemented a Zumo robot capable of navigating its circle environment by detecting white objects and avoiding them. The robot uses a combination of reflectance sensors and an IR sensor to determine its area and detect objects. The primary objectives of moving forward until a white line is detected, turning in the appropriate direction based on sensor readings, and counting detected objects were achieved. The calibration process and the signalling via the buzzer and LED provided effective feedback mechanisms for ensuring the robot's functionality. The robot also records the number of objects detected during each trial, providing useful data for analysis. Additionally, the use of LEDs and buzzer signals enhances user interaction and debugging by allowing easy monitoring of the robot's status and actions.

5 APPENDICES

```
#include <ZumoMotors.h>
#include <ZumoReflectanceSensorArray.h>
#include <ZumoBuzzer.h>

#define PIN_IR 6
#define PIN_LED 13
#define SENSORS_COUNT 6
#define THRESHOLD_WHITE 550  // Adjust based on sensor readings for white
#define NUM_TRIALS 10
ZumoMotors motors;
ZumoReflectanceSensorArray sensors;
ZumoBuzzer buzzer;

unsigned int sensorReadings[SENSORS_COUNT];
int dataValues[NUM_TRIALS];
int trialsCompleted = 0;
int objectCount = 0;
bool detectionPeriod = false;

void setup() {
  Serial.begin(9600);
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_IR, INPUT);
  //Start with open led to show you are calibrating
  digitalWrite(PIN_LED, HIGH);
```

```
  delay(3000);
  digitalWrite(PIN_LED, LOW);
  delay(3000);

  //Calibration start
  sensors.init();
  for (int i = 0; i < 400; i++) {
    sensors.calibrate();
    if (i % 50 == 0) {
      Serial.print("Calibrating");
    }
  }
  Serial.println("\nCalibration complete.");
  buzzer.play("!L16 V10 cdegreg4"); // Signal end of calibration

  delay(2000); // Put me to the black surface...

  //Blink one time fastly to declare you are starting
  digitalWrite(PIN_LED, HIGH);
  delay(500);
  digitalWrite(PIN_LED, LOW);
}

void loop() {
  if (trialsCompleted < NUM_TRIALS) {
    moveUntilWhiteDetected();
```

```
    evaluateWhiteDetection();
  } else {
    int result = getMaxCount();
    delay(2000);
    blinkAndBuzz(result);
    motors.setSpeeds(0, 0);  // Stop the robot
    while(true);  // Halt further actions
  }
}

void moveUntilWhiteDetected() {
    while (true) {
        sensors.readLine(sensorReadings);
        if (digitalRead(PIN_IR) == LOW) {          // Check if the IR sensor detects an object
            while (digitalRead(PIN_IR) == LOW) {     // Turn left until the IR sensor no longer detects anything
                motors.setSpeeds(-150, 150);       // Turn left
            }
            motors.setSpeeds(0, 0);               // Stop turning once the object is no longer detected
        }
        else if (isOnBlackSurface()) {            // Continue moving forward if on black surface and no object detected
            motors.setSpeeds(100, 100);           // Move forward
        } else {
            motors.setSpeeds(0, 0);               // Stop motors if white is detected
            break;
        }
```

```
    }
}

bool isOnBlackSurface() {
  for (int i = 0; i < SENSORS_COUNT; i++) {
    if (sensorReadings[i] < THRESHOLD_WHITE) {
      return false;  // White detected
    }
  }
  return true;  // No white detected
}

void evaluateWhiteDetection() {
  if (sensorReadings[0] < THRESHOLD_WHITE || sensorReadings[1] < THRESHOLD_WHITE || sensorReadings[2] < THRESHOLD_WHITE) {
    // Left sensors detected white, turn right
    detectionPeriod = true;
    turnRight();
  } else if (sensorReadings[3] < THRESHOLD_WHITE || sensorReadings[4] < THRESHOLD_WHITE || sensorReadings[5] < THRESHOLD_WHITE) {
    // Right sensors detected white, turn left
    detectionPeriod = true;
    turnLeft();
  }
}

void turnLeft() {
```

```
    while (!isLeftGroupWhite()) {
        motors.setSpeeds(-150, 155);  // Continue turning left
        checkForObject();  // Check for objects and count if detected
    }
    motors.setSpeeds(0, 0);  // Stop turning
    delay(200);  // Small delay to ensure complete stop
    dataValues[trialsCompleted++] = objectCount;  // Store the count
    objectCount = 0;  // Reset count
    detectionPeriod = false;  // End detection period
    while (!isRightGroupWhite()){
      motors.setSpeeds(155, -150);
    }
    // Turn right for a brief period
    motors.setSpeeds(-130, 130);
    delay(200);  // Turn right for 1 second
    motors.setSpeeds(0, 0);  // Stop turning
}

void turnRight() {
    detectionPeriod = true;
    while (!isRightGroupWhite()) {
        motors.setSpeeds(155, -150);  // Continue turning right
        checkForObject();  // Check for objects and count if detected
    }
    motors.setSpeeds(0, 0);  // Stop turning
    delay(1000);  // Small delay to ensure complete stop
```

```
    dataValues[trialsCompleted++] = objectCount;  // Store the count
    objectCount = 0;  // Reset count
    detectionPeriod = false;  // End detection period
    while (!isLeftGroupWhite()){
      motors.setSpeeds(-150, 155);
    }
    // Turn left for a brief period
    motors.setSpeeds(130, -130);
    delay(200);  // Turn left for 1 second
    motors.setSpeeds(0, 0);  // Stop turning
}



bool isLeftGroupWhite() {
  sensors.readLine(sensorReadings);
  return sensorReadings[0] < THRESHOLD_WHITE || sensorReadings[1] < THRESHOLD_WHITE || sensorReadings[2] < THRESHOLD_WHITE;
}


bool isRightGroupWhite() {
  sensors.readLine(sensorReadings);
  return sensorReadings[3] < THRESHOLD_WHITE || sensorReadings[4] < THRESHOLD_WHITE || sensorReadings[5] < THRESHOLD_WHITE;
}


void checkForObject() {
  if (detectionPeriod && digitalRead(PIN_IR) == LOW) {
```

```
    objectCount++;
    delay(200);  // Debounce delay
  }
}

int getMaxCount() {
  int maxCount = 0;
  for (int i = 0; i < trialsCompleted; i++) {
    if (dataValues[i] > maxCount) {
      maxCount = dataValues[i];
    }
  }
  return maxCount;
}

void blinkAndBuzz(int count) {
  for (int i = 0; i < count; i++) {
    digitalWrite(PIN_LED, HIGH);
    buzzer.playFrequency(2000, 500, 5);
    delay(500);
    digitalWrite(PIN_LED, LOW);
    delay(500);
  }
}
```