EE321 PROJECT REPORT

OBJECT COUNTER ZUMO ROBOT

by

Hasan Alp Doyduk
alp.doyduk@ozu.edu.tr
S025015

Ismail Akbaş
ismail.akbas@ozu.edu.tr
S024094

Faculty of Engineering
Department of Electrical and Electronics Engineering

ÖZYEĞİN UNIVERSITY

TABLE OF CONTENTS

LIST OF FIGURES

ÖZYEĞİN UNIVERSITY

INTRODUCTION

This project aims to program a Zumo robot that scans its surroundings and indicates the number of objects around it while situated in an arena with objects and white borders. The primary objective is to program the Zumo robot to autonomously navigate the arena, detect objects within its vicinity, and indicate the number of detected objects using an LED indicator. The robot will be confined to a designated area marked by white borders, which it must recognize and avoid crossing. Utilizing its sensors, the robot will detect objects and perform a 360-degree scan to ensure comprehensive coverage of the arena.

DETAILED EXPLANATION

Stay in Arena

Zumo robot must detect the data from the reflectance sensors below to stay within the black arena surrounded by white borders and make decisions accordingly. If one of the sensors detects the white borders, the robot should change direction or move backward to stay within the arena. In our project, if one of the sensors on the right side of the robot detects the white borders, it should turn left, and if one of the sensors on the left side detects the white borders, it should turn right, thus ensuring that it has not left the arena.

Figure 1. Bottom part of the Zumo robot. Reflectance sensors.

Object Detection

The MZ-80 infrared sensor on top of the robot can detect objects at a certain distance in a single direction. The detection range of the MZ-80 can be adjusted. This sensor provides information about whether there is an object in front of the robot. However, since this sensor only detects objects in front of it, it cannot detect other objects in the arena. To overcome this

limitation, the robot must rotate around its axis. In our project, we will increment a counter in the code when the Zumo robot detects an object with the MZ-80 sensor during its rotation around its axis. We will also ensure that the robot stops when it returns to the starting point of the rotation.

LED Blinking

As mentioned in the previous section, when the MZ-80 sensor on the robot detects an object during its rotation, the object counter will be incremented. Meanwhile, the LED on the robot will blink many times equal to the number of objects detected. A waiting period (delay) will be created between the LED turning on and off to ensure that it is perceptible to humans.

THE LIBRARIES

QTRSensors.h

This is a library for an Arduino-compatible controller that interfaces with the Pololu QTR reflectance sensors.

ZumoReflectanceSensorArray.h

This library provides an interface for using a Zumo Reflectance Sensor Array connected to a Zumo robot. The library provides access to the raw sensor values as well as to high-level functions including calibration and line-tracking. You can get more detailed information about reflectance sensors with this link: http://www.pololu.com/product/1419

ZumoMotors.h

The ZumoMotors library provides functions for PWM-based speed (and direction) control of the two motors on the Zumo with the onboard DRV8835 dual motor driver.

Pushbutton.h

A simple library for Push Buttons.

Wire.h

This library allows you to communicate with I2C devices, a feature that is present on all Arduino boards. I2C is a common protocol, primarily used for reading/sending data to/from external I2C components.

LSM303.h

This is a library for an Arduino-compatible controller that interfaces with LSM303D, LSM303DLHC, LSM303DLM, and LSM303DLH 3D compass and accelerometer ICs on Pololu boards.

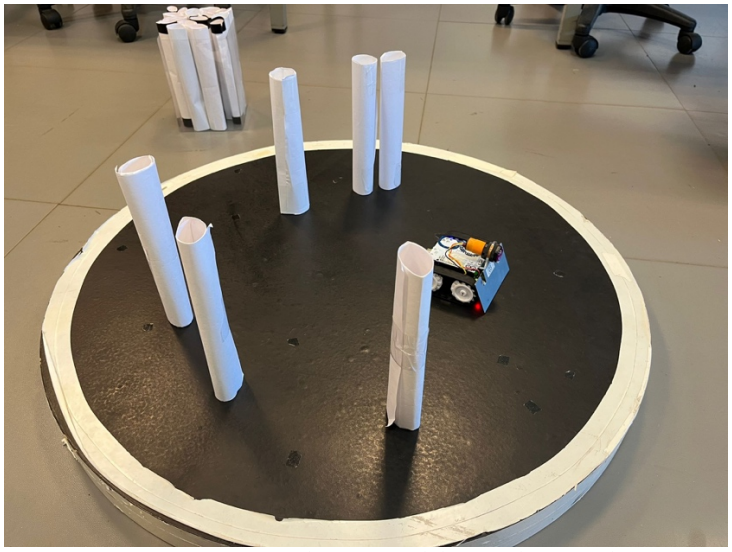


Figure 2. Zumo rotates around its axis and detects objects with its sensor.

THE CHALLENGES

Challenge 1: Side-by-Side Objects

When the Zumo robot is tasked with scanning its surroundings to count objects, it relies on its infrared sensor to detect objects in its path. However, a significant challenge arises when objects are placed side-by-side within the arena. This specific scenario is illustrated in Figure 2.

Problem Description:

Side-by-Side Objects: When objects are positioned closely next to each other, the infrared sensor may not effectively distinguish between individual objects. Instead, it might perceive the adjacent objects as a single, larger object.

Detection Limitations: The infrared sensor works by emitting a beam and measuring the reflected signal to determine the presence and proximity of an object. When objects are side by side, the reflected signals can overlap or merge, leading to incorrect readings or a failure to detect the separation between objects.

Technical Challenges:

Resolution of the Sensor: The sensor's ability to differentiate objects depends on its resolution and the distance between the objects. If the resolution is insufficient or the objects are too close, the sensor may not detect the gap between them.

Scan Speed and Timing: The speed at which the robot rotates and scans its surroundings also plays a role. If the robot scans too quickly, it might not capture enough data points to accurately identify multiple objects placed closely together.

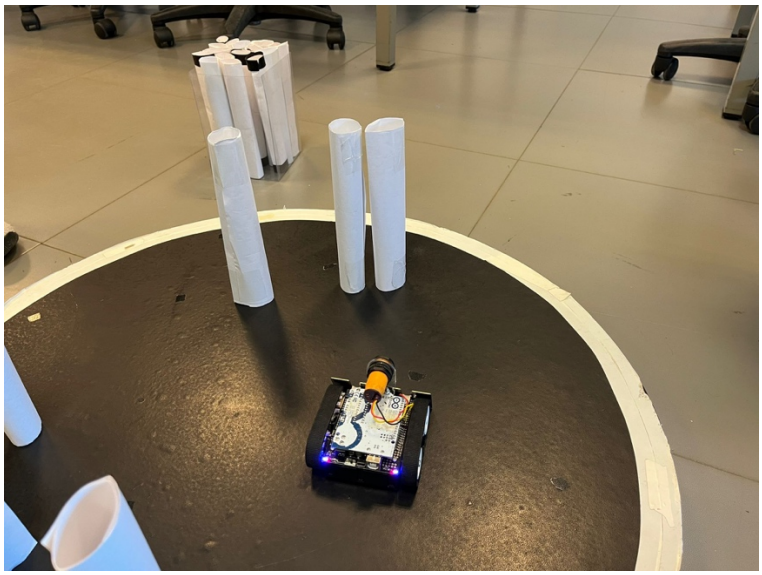


Figure 3. Zumo robot is trying to count the side-by-side objects.

Challenge 2: Objects In the Behind

In certain scenarios, objects within the arena may be positioned such that one object is partially or completely obscured by another object from the perspective of the robot's infrared sensor. This situation is depicted in Figure 3.

Problem Description:

Obstructed Objects: When an object is located behind another object, the infrared sensor may struggle to detect the obstructed object due to the blocking effect of the object in front.

Limited Visibility: The front object obstructs the sensor's field of view, preventing it from directly detecting the object behind.

Detection Interference: The infrared sensor may receive mixed or distorted signals due to the presence of the front object, leading to incorrect readings or a failure to detect the object behind.

Technical Challenges:

Sensor Angle and Coverage: The angle and coverage area of the sensor's beam affect its ability to detect objects behind other objects. If the sensor's angle is too narrow or its coverage is limited, it may not detect objects outside its direct line of sight.

Reflection and Absorption: The material properties of the objects, such as their reflectivity and absorbance of infrared light, can impact the sensor's ability to detect them. Objects that absorb or reflect infrared light differently may be harder to detect behind other objects.

Resolution and Sensitivity: The sensor's resolution and sensitivity determine its ability to distinguish between closely spaced objects and detect faint signals. Insufficient resolution or sensitivity may result in missed detections of objects behind others.

Figure 4. Zumo is trying to count the object behind.

THE SOLUTIONS

Solution 1

To address the challenge of detecting objects next to each other more efficiently, a modification to the Zumo robot code can be implemented to address the challenge of detecting objects next to each other more efficiently. This change aims to reduce the object detection time, enhancing the sensor's ability to accurately identify closely spaced objects.

Reduce Detection Time Interval: The time interval between successive object detection scans can be reduced. This will enable the sensor to update its readings more frequently, allowing it to

capture rapid changes in the environment and detect objects that are positioned next to each other.

Solution 2

To address the challenge of detecting objects next to each other more efficiently, a modification to the robot's code can be implemented. This change aims to reduce the object detection time, enhancing the sensor's ability to accurately identify closely spaced objects, the robot needs to move strategically within the arena.

Reduce Detection Time Interval: The time interval between successive object detection scans can be reduced. This will enable the sensor to update its readings more frequently, allowing it to capture rapid changes in the environment and detect objects that are positioned next to each other.

Integrate Strategic Movement Patterns: To overcome the issue of limited detection coverage, the robot should be programmed to move strategically within the arena. In addition to rotating around its axis, the robot needs to move forward, backward, and sideways to cover more ground.

THE CODE

The code can be examined in 4 parts: Initialization, setup function and calibrations, loop function, and other functions.

Codes using the embedded compass on the Zumo robot and the compass calibration using the LSM303 library were taken from the original GitHub page of the Pololu.
https://github.com/pololu/zumo-shield/blob/master/ZumoExamples/examples/Compass/Compass.ino

Initialization

Include Libraries:
The libraries whose purposes are given in the table above were used.

```
5   // Libraries
6   #include <QTRSensors.h>
7   #include <ZumoReflectanceSensorArray.h>
8   #include <ZumoMotors.h>
9   #include <Pushbutton.h>
10  #include <Wire.h>  // Used for I2C communication
11  #include <LSM303.h>  // Compass library
```

Pinouts:
The PINs of the buzzer MZ80 sensor and LED connected to the Arduino on the Zumo robot are defined.

```
13  // Pinouts
14  #define BUZZER_PIN 3
15  #define MZ80_PIN 6
16  #define LED_PIN 13
```

Initial Definitions:
The values of the variables to be used in the methods and calibration parts are defined. The parameters required for compass calibration are also defined.

```
18  // Initial definitions
19  #define NUM_SENSORS 6
20  #define SPEED 200
21
22  // Initial definitions for calibration of LSM303 sensor
23  #define CALIBRATION_SAMPLES 70  // Number of compass readings to take when calibrating
24  #define CRB_REG_M_2_5GAUSS 0x60  // CRB_REG_M value for magnetometer +/-2.5 gauss full
25  #define CRA_REG_M_220HZ 0x1C  // CRA_REG_M value for magnetometer 220 Hz update rate
26  // Allowed deviation (in degrees) relative to target angle that must be achieved befor
27  #define DEVIATION_THRESHOLD 5
28
29  // Create instances (like objects in classes)
30  ZumoMotors motors;
31  LSM303 compass;
32  Pushbutton button(ZUMO_BUTTON);
33  ZumoReflectanceSensorArray reflectanceSensors;
34
35  // Initial definitions
36  unsigned int sensorValues[NUM_SENSORS];
37  unsigned int positionVal = 0;
38  unsigned int counter = 0; // Object counter
39  unsigned int isObject = 0;
40  unsigned int turner = 0;
41  unsigned int isLedOn = 0;
42  bool myExit = false;
43  float myheading;
44  float myrelative_heading;
45  float mytarget_heading;
```

A ZumoMotor class object named motors and an LSM303 class object named compass are created. Then, variables are defined to display and store situations such as position, number of objects, and number of rotations.

Setup Method

It is shown that LED and Buzzer pins are output pins. The isLedOn variable is set to 1, which is a boolean value of true, when the LED turns on. Then, the serial monitor is started.
Serial Monitor is a tool provided by the Arduino IDE (Integrated Development Environment) that allows you to communicate with your Arduino board via a serial connection. It is commonly used for debugging and troubleshooting purposes, as well as for sending and receiving data between the Arduino board and your computer.

```
50  void setup() {
51
52    turner = 0;
53    Serial.begin(9600);  // Serial communication
54    Serial.println("Starting.");
55    pinMode(MZ80_PIN, INPUT);
56    pinMode(LED_PIN, OUTPUT);
57    pinMode(BUZZER_PIN, OUTPUT);
58
59    digitalWrite(LED_PIN, HIGH);
60    isLedOn = 1;
61
62
63    // ---------------------- Start Of The Reflectance Calibration ----------------------
64    reflectanceSensors.init();
65    unsigned long startTime = millis();
66    while (millis() - startTime < 5000)
67    {
68      reflectanceSensors.calibrate();
69    }
70    // --------------------------- End Of The Calibration -----------------------------
71
72    // At the end of the bottom sensor calibration blink LED 3 times
73    // 3 HIGH + 3 LOW (H -> LHLHL 5 times)
74    isLedOn = 1;
75
76    for (int i = 1; i < 6; i++) {
77      if (isLedOn) {
78        digitalWrite(LED_PIN, LOW);
79        isLedOn = 0;
80        delay(100);
81      }
82      else {
83        digitalWrite(LED_PIN, HIGH);
84        isLedOn = 1;
85        delay(100);
86      }
87    }
88
```

To calibrate the reflectance sensor, the LED flashes three times to indicate the end of the process. This is done using a for loop where the counter (i) goes from 0 to 5, turning the LED on and off three times. Additionally, the compass calibration code has been written and published on the project's GitHub page. This code ensures accurate directional readings and complements the reflectance sensor calibration, improving the Zumo robot's navigation and object detection.

```
89
90    // ------------------------ Start Of The Compass Calibration ----------------------
91    CompassCalibration();
92    // ------------------------------ End Of The Calibration ----------------------------
93
94    // At the end of the compass calibration blink LED 5 times
95    // 5 HIGH + 5 LOW (H -> LHLHLHLHL 9 times)
96    digitalWrite(LED_PIN, HIGH);
97    isLedOn = 1;
98
99    for (int i = 1; i < 8; i++) {
100     if (isLedOn) {
101       digitalWrite(LED_PIN, LOW);
102       isLedOn = 0;
103       delay(100);
104     }
105     else {
106       digitalWrite(LED_PIN, HIGH);
107       isLedOn = 1;
108       delay(100);
109     }
110   }
111 }
112
```

Loop Method

In this code, the Zumo robot is programmed to perform a 360-degree rotation and detect objects during its rotation. The loop starts by setting the target and current heading using the averageHeading function and calculates the relative heading. The robot continues rotating as long as myExit is false and the turner variable is less than 360. The motors are set to rotate the robot clockwise, followed by a short delay, after which the motors stop and the heading is re-evaluated. This process repeats, incrementing the turner variable each time to track the rotation amount. After rotating more than 10 steps, if the relative heading is within 5 degrees of the target, the robot stops rotating, sets turner to 5000, calls exitFunc(counter) to handle exit procedures, and sets myExit to true, exiting the loop. Additionally, the robot uses reflectance sensors to read position values and a detection sensor (MZ80_PIN) to identify objects. If an object is detected, the LED is turned on and, if it is a new object, the object counter is incremented and displayed. If no object is detected, the LED is turned off and the detection state is reset.

```
116  void loop() {
117
118    // Target vector is the actual vector for turning its axis
119    mytarget_heading = averageHeading();
120    // Heading is given in degrees away from the magnetic vector, increasing clockwise
121    myheading = averageHeading();
122    // This gives us the relative heading with respect to the target angle
123    myrelative_heading = relativeHeading(myheading, mytarget_heading);
124
125    while (myExit == false) {
126      if (turner < 360) {
127        motors.setLeftSpeed(100);
128        motors.setRightSpeed(-100);
129        delay(50);
130        motors.setLeftSpeed(0);
131        motors.setRightSpeed(0);
132        delay(100);
133        myheading = averageHeading();
134        myrelative_heading = relativeHeading(myheading, mytarget_heading);
135        turner = turner + 1 ;
136
137        if (turner > 10) {
138          if (abs(myrelative_heading) < 5) {
139            Serial.println("Checking for exit.");
140            turner = 5000;  // Dead
141            exitFunc(counter);
142            myExit = true;
143            break;
144          }
145        }
146      }
147      positionVal = reflectanceSensors.readLine(sensorValues);
148
149      if (!digitalRead(MZ80_PIN)) {
150        digitalWrite(LED_PIN, HIGH);
151
152        if (isObject == 0) {
153          counter = counter + 1;
154          isObject = 1;
155          Serial.print("Number of Objects:");
156          Serial.println(counter);
157        }
158      }
159      else {
160        isObject = 0;
161        digitalWrite(LED_PIN, LOW);
162      }
163    }
164  }
```

Other Methods

Exit Function:
In the exitFunc function, the Zumo robot handles the exit procedures after completing its rotation. The function begins by turning off the LED and buzzer, followed by a 1-second delay. A message "Exit turning." is printed to the serial monitor, and another 1-second delay occurs. The isLedOn flag is set to 1, indicating the LED is currently on. A for loop runs for counter * 2 + 1 iterations, where counter represents the number of detected objects. During each iteration, the state of the LED and buzzer alternates. If isLedOn is true, the LED and buzzer are turned off, and isLedOn is set to 0, followed by a 1-second delay. Otherwise, the LED is turned on, isLedOn is set to 1, and a shorter 100-millisecond delay occurs. This blinking pattern provides visual and audible feedback based on the number of detected objects.

```
168
169  void exitFunc(int counter) {
170
171    digitalWrite(LED_PIN, LOW);
172    digitalWrite(BUZZER_PIN, LOW);
173    delay(1000);
174
175    Serial.println("Exit turning.");
176    delay(1000);
177
178    isLedOn = 1;
179
180    for (int i = 1; i < counter * 2 + 1; i++) {
181
182      if (isLedOn) {
183        digitalWrite(LED_PIN, LOW);
184        digitalWrite(BUZZER_PIN, LOW);
185        isLedOn = 0;
186        delay(1000);
187
188      }
189      else {
190        digitalWrite(LED_PIN, HIGH);
191        isLedOn = 1;
192        delay(100);
193      }
194    }
195  }
196
```

heading Function:
The heading function converts the x and y components of a vector to a heading in degrees. This custom function is used instead of LSM303::heading() to avoid tilt compensation errors caused by the Zumo's acceleration, assuming the Zumo is always level. The function scales the x and y components of the vector v using the minimum and maximum compass readings, resulting in x_scaled and y_scaled values between -1 and 1. The atan2 function calculates the angle in radians between the scaled y and x components, converting it to degrees. If the resulting angle is negative, 360 degrees is added to ensure a positive heading.

relativeHeading Function:
The relativeHeading function calculates the angular difference between two headings in degrees. Given heading_from and heading_to, the function computes the difference and adjusts it to fall within the range of -180 to 180 degrees. This adjustment ensures the relative heading represents the shortest angular path between the two headings, correcting any wrap-around issues when crossing the 0/360 degree boundary.

averageHeading Function:
The averageHeading function averages ten magnetic vector readings to obtain a more accurate and stable heading measurement, reducing interference from the Zumo's motors. It initializes an avg vector to accumulate the readings, then reads the compass ten times, summing the x and y components. The averages are computed by dividing the sums by 10. The function then calculates and returns the heading from the averaged vector using the custom heading function.

Figure 5. LSM303 module

```
// Converts x and y components of a vector to a heading in degrees.
// This function is used instead of LSM303::heading() because we don't
// want the acceleration of the Zumo to factor spuriously into the
// tilt compensation that LSM303::heading() performs. This calculation
// assumes that the Zumo is always level.
template<typename T> float heading(LSM303::vector<T> v) {
  float x_scaled = 2.0 * (float)(v.x - compass.m_min.x) / (compass.m_max.x
  - compass.m_min.x) - 1.0;
  float y_scaled = 2.0 * (float)(v.y - compass.m_min.y) / (compass.m_max.y
  - compass.m_min.y) - 1.0;

  float angle = atan2(y_scaled, x_scaled) * 180 / M_PI;
  if (angle < 0)
    angle += 360;
  return angle;
}


// Yields the angle difference in degrees between two headings
float relativeHeading(float heading_from, float heading_to) {
  float relative_heading = heading_to - heading_from;

  // constrain to -180 to 180 degree range
  if (relative_heading > 180)
    relative_heading -= 360;
  if (relative_heading < -180)
    relative_heading += 360;

  return relative_heading;
}


// Average 10 vectors to get a better measurement and help smooth out
// the motors' magnetic interference.
float averageHeading() {
  LSM303::vector<int32_t> avg = { 0, 0, 0 };

  for (int i = 0; i < 10; i++) {
    compass.read();
    avg.x += compass.m.x;
    avg.y += compass.m.y;
  }
  avg.x /= 10.0;
  avg.y /= 10.0;

  // avg is the average measure of the magnetic vector.
  return heading(avg);
}
```

CompassCalibration Function:

The CompassCalibration function calibrates the Zumo robot's magnetometer by determining the maximum and minimum magnetic vector readings. Initially, the highest possible magnetic values (running_min) are set to 32767, and the lowest possible values (running_max) are set to -32767. The Wire library is initiated to join the I2C bus, and the LSM303 compass module is initialized and enabled with default settings. The compass sensitivity is set to +/- 2.5 gauss to avoid overflow, and the update rate is set to 220 Hz. The robot starts rotating with its left motor set to SPEED and the right motor set to -SPEED.

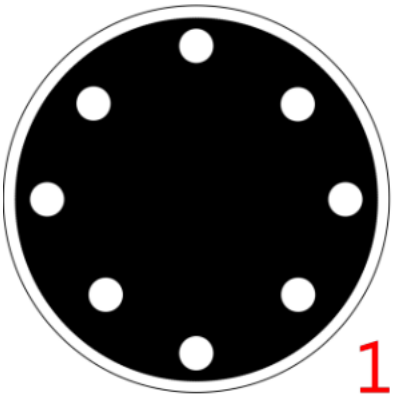
During the calibration loop, which runs for CALIBRATION_SAMPLES iterations, the compass reads the current magnetic vector, updating the running_min and running_max values with the minimum and maximum x and y readings. These values are printed to the serial monitor for each iteration. After completing the calibration samples, the motors stop, and the final maximum and minimum x and y values are printed to the serial monitor. These values are then assigned to compass.m_max and compass.m_min to correct for offsets in the magnetometer data, completing the calibration process.

```
303  void CompassCalibration() {
304    // The highest possible magnetic value to read in any direction is 2047
305    // The lowest possible magnetic value to read in any direction is -2047
306    LSM303::vector<int16_t> running_min = { 32767, 32767, 32767 },
307    running_max = { -32767, -32767, -32767 };
308
309    unsigned char index;
310
311    // Serial.begin(9600);
312
313    // Initiate the Wire library and join the I2C bus as a master
314    Wire.begin();
315
316    // Initiate LSM303
317    compass.init();
318
319    // Enables accelerometer and magnetometer
320    compass.enableDefault();
321
322    compass.writeReg(LSM303::CRB_REG_M, CRB_REG_M_2_5GAUSS);  // +/- 2.5 gauss sensiti
323    compass.writeReg(LSM303::CRA_REG_M, CRA_REG_M_220HZ);     // 220 Hz compass update
324
325    // button.waitForButton();
326
327    Serial.println("starting calibration");
```

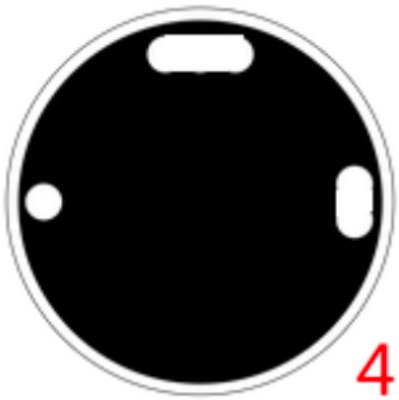
```
  // To calibrate the magnetometer, the Zumo spins to find the max/min
  // magnetic vectors. This information is used to correct for offsets
  // in the magnetometer data.
  motors.setLeftSpeed(SPEED);
  motors.setRightSpeed(-SPEED);

  for (index = 0; index < CALIBRATION_SAMPLES; index++) {
    // Take a reading of the magnetic vector and store it in compass.m
    compass.read();

    running_min.x = min(running_min.x, compass.m.x);
    running_min.y = min(running_min.y, compass.m.y);

    running_max.x = max(running_max.x, compass.m.x);
    running_max.y = max(running_max.y, compass.m.y);

    Serial.println(index);

    delay(50);
  }

  motors.setLeftSpeed(0);
  motors.setRightSpeed(0);

  Serial.print("max.x   ");
  Serial.print(running_max.x);
  Serial.println();
  Serial.print("max.y   ");
  Serial.print(running_max.y);
  Serial.println();
  Serial.print("min.x   ");
  Serial.print(running_min.x);
  Serial.println();
  Serial.print("min.y   ");
  Serial.print(running_min.y);
  Serial.println();

  // Set calibrated values to compass.m_max and compass.m_min
  compass.m_max.x = running_max.x;
  compass.m_max.y = running_max.y;
  compass.m_min.x = running_min.x;
  compass.m_min.y = running_min.y;

  //button.waitForButton();
}
```

THE CASES

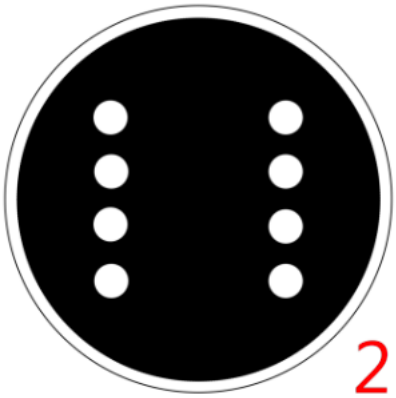
Case 1



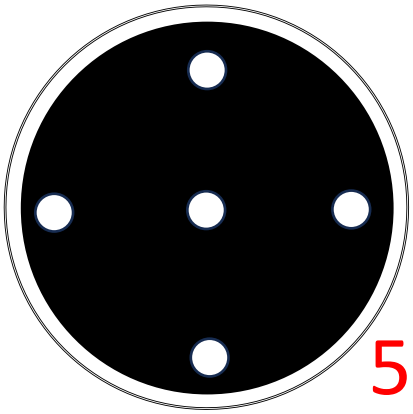
Case 4



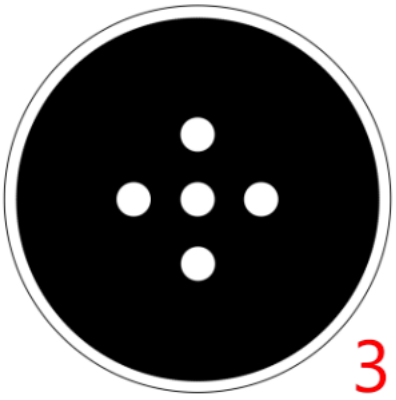
Case 2



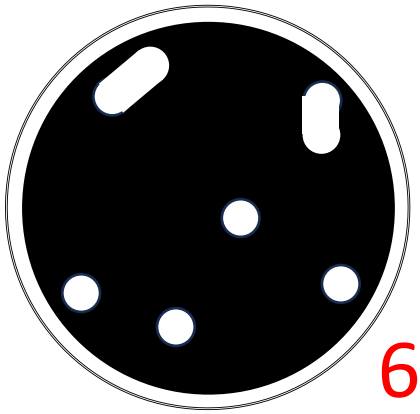
Case 5



Case 3



Case 6



ÖZYEĞİN UNIVERSITY

THE CONCLUSION

In this project, we successfully programmed a Zumo robot to autonomously navigate an arena, detect objects, and indicate the number of detected objects using an LED indicator. By utilizing reflectance sensors for boundary detection, an MZ-80 infrared sensor for object detection, and implementing strategic movement patterns, the robot effectively handled challenges such as detecting closely spaced and obscured objects. The comprehensive calibration of sensors and integration of precise feedback mechanisms ensured reliable performance, demonstrating the robot's capability to perform complex navigation and detection tasks within a confined environment.

YOUTUBE LINK