EE321 PROJECT PROGRESS REPORT

Object counter Zumo Robot

ÖZYEĞİN
UNIVERSITY

Table of Contents

The Explanation of the Progress

As mentioned in the project proposal, the purpose of this project is to program a Zumo Robot that scans its environment and displays the number of objects around it without leaving the arena.

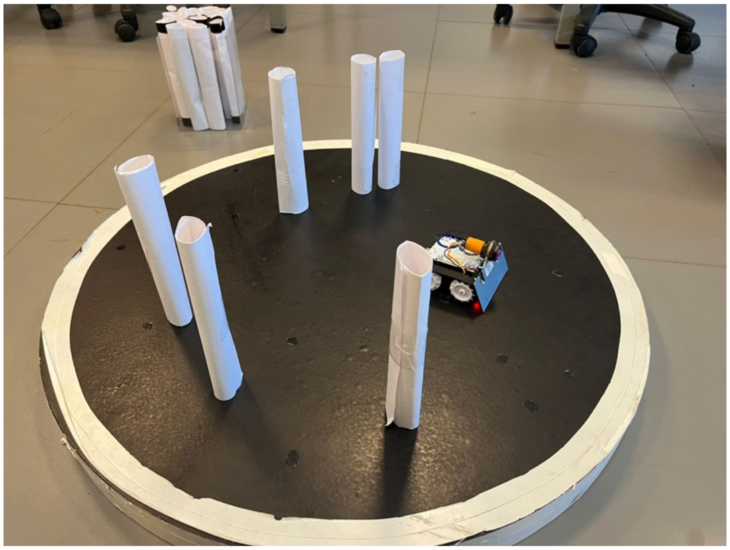

Figure 1. Zumo rotates around its own axis and detects objects with its sensor.

The code was written to achieve this purpose and the code is explained in the code section below. The code written so far aims for the Zumo robot to rotate around its own axis and detect objects with the reflectance sensor on it. The LED (LED_PIN = 13) flashes during each detection. After Zumo completes its rotation around its axis, the LED blinks for the number of objects it sees during the rotation.

The Libraries that Used

Libraries	Purpose
`QTRSensors.h`	MZ80 sensor
`ZumoReflectanceSensorArray.h`	Bottom sensors
`ZumoMotors.h`	Moving the robot
`Pushbutton.h`	Button operations
`Wire.h`	Serial communication during the tests (I2C)
`LSM303.h`	Compass operations

LSM606.h is the compass library. With this library, we measure the direction values of the head of the Zumo robot is positioned. By continuously measuring compass values, the completion of the robot's rotation around its own axis is measured. If the rotation is completed, the LED blinking section is started.

Problematic Cases and Possible Solutions

Case-1: Objects Side by Side

Zumo robot cannot detect each object while objects are side by side in the arena which is shown in Figure 2.

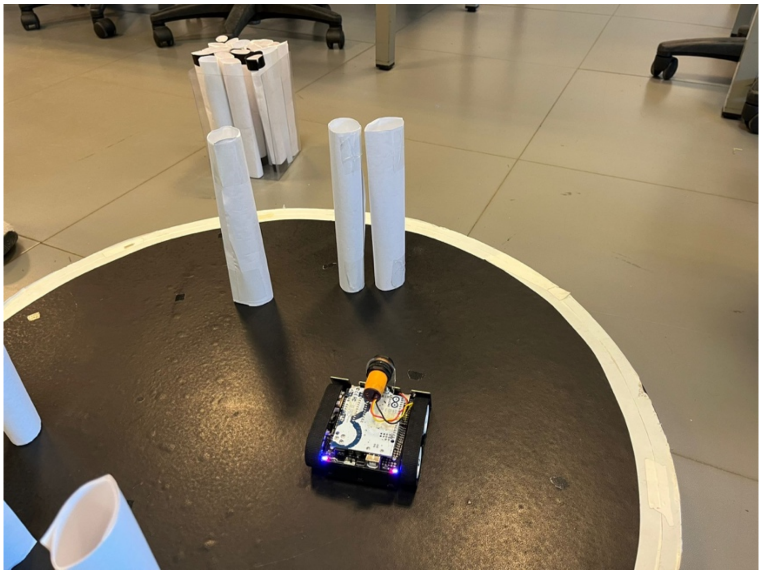

Figure 2. Zumo robot is trying to count the side by side objects.

Case-2: Objects in the Behind

Zumo robot cannot detect the object which is behind of another one in the arena. This case is shown in figure 2.

Figure 3. Zumo is trying to count the object in the behind.

The Solutions for Problematic Cases

Solution of Case-1: Objects Side by Side

A change will be made in the code that will reduce the object detection time. In this way, objects next to each other can be detected.

Solution of Case-2: Objects in the Behind

To overcome this issue, in addition to rotating around its own axis, the robot needs to move within the arena. During this movement, it must ensure that it does not go beyond the boundaries of the arena.

The Code

The code can be examined in 3 parts. 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.

```
// Libraries
#include <QTRSensors.h>
#include <ZumoReflectanceSensorArray.h>
#include <ZumoMotors.h>
#include <Pushbutton.h>

#include <Wire.h> // Used for I2C communication
#include <LSM303.h> // Compass library
```

Pinouts

The pin numbers of the buzzer mz80 sensor and LED connected to the Arduino on the Zumo robot are defined.

```
// Pinouts
#define BUZZER_PIN 3
#define MZ80_PIN 6
#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.

```
// Initial definitions
#define NUM_SENSORS 6
#define SPEED 200

// Initial definitions for LSM303 library
#define CALIBRATION_SAMPLES 70 // Number of compass readings to take when calibrating
#define CRB_REG_M_2_5GAUSS 0x60 // CRB_REG_M value for magnetometer +/-2.5 gauss full scale
#define CRA_REG_M_220HZ 0x1C // CRA_REG_M value for magnetometer 220 Hz update rate
// Allowed deviation (in degrees) relative to target angle that must be achieved before driving straight
#define DEVIATION_THRESHOLD 5
```

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, number of rotations.

```
ZumoMotors motors;
LSM303 compass;
Pushbutton button(ZUMO_BUTTON);
ZumoReflectanceSensorArray reflectanceSensors;

// Initial definitions
unsigned int sensorValues[NUM_SENSORS]; // Array for holding sensor values
unsigned int positionVal = 0;
unsigned int objectCounter = 0; // Number of objects
unsigned int isObject = 0; // Is there an object?
unsigned int turner = 0; // Turning value
unsigned int isLedOn = 0;

// Initial definitions for LSM303 library
float myheading;
float myrelative_heading;
float mytarget_heading;
bool exit = false;

```

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.

```
void setup() {

  pinMode(LED_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  isLedOn = 1;
  digitalWrite(LED_PIN, HIGH);
  digitalWrite(BUZZER_PIN, LOW);

  turnet = 0;

  Serial.begin(9600);  // Serial communication
  Serial.println("...starting...");
```

The reflectance sensor is calibrated and the LED flashes 3 times to indicate the end of the calibration. To do this, a for loop was created in which the incrementor (i) counts from 0 to 5 (on 3 times, off 3 times).

```

  // --------------------------- Start Of The Calibration -------------------------
  reflectanceSensors.init();
  unsigned long startTime = millis();
  while (millis() - startTime < 5000)  // make the calibration take 10 seconds
  {
    reflectanceSensors.calibrate();
  }
  // -------------------------- End Of The Calibration -----------------------------

  // At the end of the bottom sensor calibration blink LED 3 times
  isLedOn = 1;
  // 3 HIGH + 3 LOW = 6 times
  for (int i = 1; i < 6; i++) {
    if (isLedOn) {
      digitalWrite(LED_PIN, LOW);
      isLedOn = 0;
      delay(100);
    } else {
      digitalWrite(LED_PIN, HIGH);
      isLedOn = 1;
      delay(100);
    }
  }
```

The compass calibration code, which is also published on the Github page, has been written.

```
  // --------------------------- Start Of The Compass Calibration -----------------
  compassCalibration();
  // ------------------------------ End Of The Calibration -------------------------

  // At the end of the compass calibration blink LED 5 times
  isLedOn = 1;
  // 5 HIGH + 5 LOW = 10 times
  for (int i = 1; i < 10; i++) {
    if (isLedOn) {
      digitalWrite(LED_PIN, LOW);
      isLedOn = 0;
      delay(100);

    } else {
      digitalWrite(LED_PIN, HIGH);
      isLedOn = 1;
      delay(100);
    }
  }
}
```

Loop Method

The 360 degree rotation method of the zumo robot around its own axis is called.

```
void loop() {

  // Turn 360 degrees
  mytarget_heading = averageHeading();
  // Heading is given in degrees away from the magnetic vector, increasing clockwise
  myheading = averageHeading();
  // This gives us the relative heading with respect to the target angle
  myrelative_heading = relativeHeading(myheading, mytarget_heading);
```

The code has been written for the Zumo robot to continue rotating if it does not complete its scanning rotation around its own axis. The amount of rotation is controlled. If the quantity is completed, the LED will turn on and the object quantity will be displayed.

```
while (Exit == false) {
  if (turner < 360) {
    motors.setLeftSpeed(100);
    motors.setRightSpeed(-100);
    delay(100);
    myheading = averageHeading();
    myrelative_heading = relativeHeading(myheading, mytarget_heading);
    turner = turner + 1;

    if (turner > 10) {
      if (abs(myrelative_heading) < 5) {
        Serial.println("Bearing check true Exit");
        turner = 5000; // Dead

        digitalWrite(LED_PIN, LOW);
        delay(1000);
        Serial.println("exit işlemleri");
        delay(1000);
        digitalWrite(BUZZER_PIN, LOW);

        isLedOn = 1;
        for (int i = 1; i < objectCounter*2; i++) {
          if (isLedOn) {
            digitalWrite(LED_PIN, LOW);
            isLedOn = 0;
            delay(1000);

          }
          else {
            digitalWrite(LED_PIN, HIGH);
            isLedOn = 1;
            delay(100);
          }
        }
        //break();
        Exit = true;
        break;
      }

    }
  }
```

```
    positionVal = reflectanceSensors.readLine(sensorValues);

    if (!digitalRead(MZ80_PIN) && Exit==false) {

     digitalWrite(LED_PIN, HIGH);

      if (isObject == 0) {
        objectCounter = objectCounter + 1  ;
        isObject = 1; // Do not count the same object

        Serial.print("Object Counter:");
        Serial.println(objectCounter);
        Serial.println(digitalRead(MZ80_PIN));
      }
    }
    else {
      isObject = 0;
      digitalWrite(LED_PIN, LOW);
    }

  }
}
```

```cpp
// 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);
}
```

```
unsigned int mostLeftSensor() {
  if (sensorValues[0] < 600)
    return 1;
  else
    return 0;
}

unsigned int leftSensor() {
  if (sensorValues[1] < 600)
    return 1;
  else
    return 0;
}

unsigned int midLeftSensor() {
  if (sensorValues[2] < 600)
    return 1;
  else
    return 0;
}

unsigned int midRightSensor() {
  if (sensorValues[3] < 600)
    return 1;
  else
    return 0;
}

unsigned int rightSensor() {
  if (sensorValues[4] < 600)
    return 1;
  else
    return 0;
}

unsigned int mostRightSensor() {
  if (sensorValues[5] < 600)
    return 1;
  else
    return 0;
}

void turnRight() {
  motors.setSpeeds(200, -200);
}

void go() {
  motors.setSpeeds(300, 300);
}
```

Compass calibration method is shown below.

```
// Compass calibration method
void CompassCalibration() {
  // The highest possible magnetic value to read in any direction is 2047
  // The lowest possible magnetic value to read in any direction is -2047
  LSM303::vector<int16_t> running_min = {32767, 32767, 32767}, running_max = {-32767, -32767, -32767};
  unsigned char index;
  // Serial.begin(9600);
  // Initiate the Wire library and join the I2C bus as a master
  Wire.begin();
  // Initiate LSM303
  compass.init();
  // Enables accelerometer and magnetometer
  compass.enableDefault();
  compass.writeReg(LSM303::CRB_REG_M, CRB_REG_M_2_5GAUSS);  // +/- 2.5 gauss sensitivity to hopefully avoid overflow problems
  compass.writeReg(LSM303::CRA_REG_M, CRA_REG_M_220HZ);     // 220 Hz compass update rate
  // button.waitForButton();
  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();
}
```