Introduction to Robotics – EMS516U
Robotic Lab Report

Hana Emma Hadidi

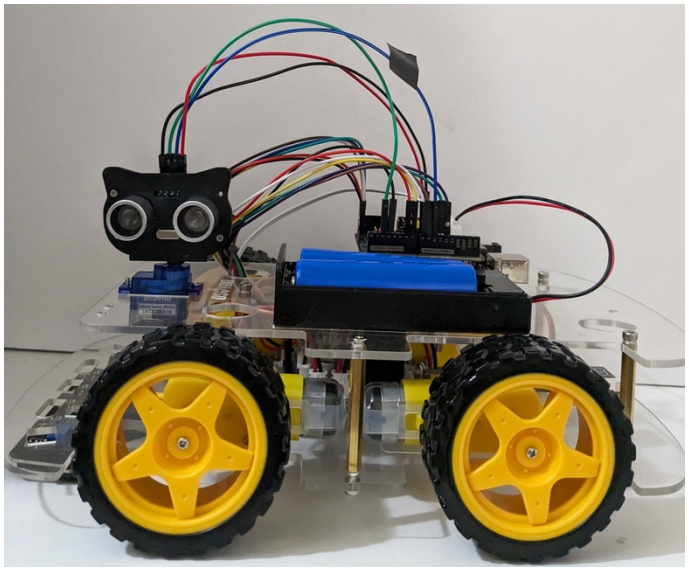

Fig 1. Osoyoo Robot V2.1

Contents:

Originality Declaration:
The work contained within this report is, to my knowledge, entirely original, except that which is stated to be otherwise. Algorithm 1 uses the obstacle avoidance code provided by Osoyoo (see report reference 6) and following algorithm 2 and 3 use parts of the Osoyoo obstacle avoidance code (see report reference 6).

Abstract:
This report analysis the ultrasound sensor and 2 algorithms used to build to a Bug 2 algorithm, which does wall following with obstacle avoidance, completed in algorithm 3. All complete codes and flowcharts are provided in the Appendix for reference.

Introduction:
The first part of the report discusses the Osoyoo Robot V2.1 hardware and software used to test and analyse the sensors and the effects it would have on the performance of the robot when executing algorithms. The aim of the algorithms provided is to provide the basis for creating a Bug 2 algorithm, by doing wall following with obstacle avoidance.

Background Theory:

Sensors and actuators:
Robots rely on sensors to quantify their environment into measurements which can be used by an algorithm, to interpret and react to its surrounds. Sensors such as the ultrasound sensor utilize sounds to measure the distance of an objects, however limitations of the sensor limit the robot's ability to perform optimally. Examples of limitations of sensors include their cost, optimal performance range means that sensor won't work accuracy in a variety of environments, the field view of sensor may be increased with servo motors however accuracy may become compromised, and size creates a limit to the number of sensors that can be placed.[1]

Bug 2 algorithm:
The aim of Bug 2 algorithms is to follow a boundary to a goal, via greedy search (going around each obstacle until it can continue its original path)[2]. The final position or goal can be initialised via global navigation such as via satellite or local navigation. The robot used was limited to local navigation and only ultrasounds and line tracking sensor, therefore the goal was defined by following a line or wall following.

Local navigation:
Benefits of local navigation over global navigation include the algorithm being easy to code and react relatively fast to objects in its path and able to easily sensor new object or adapt to changes in the environment. Limitation includes the sensor reading the environment may prevent the correct execution of code for the scenario. Greatest limitation is no guarantee that the end goal will be reached since it may fay to rediscover the line after obstacle avoidance or may fail to go back to the wall.[3]

Robot Assembly:

Fig 2. Parts used within the Osoyoo Robot[4]

Parts and its dynamic results:
The robot utilises 4 wheels for movement this provides it with stability when it performed manoeuvres, since the robot was only tested on flat non slippery surfaces, the wheels provided sufficient traction. Each wheel has its own motor and utilises differential drive to control the steering. This is set up in the code by setting the speed of each side of the motor, and selecting the motors which should be high, which allowed for backward, forward and rotational motion. This is used to achieve a slight right motion by giving the left motor a greater speed than the right but having both be high (vice versa for slight left) or right motion via giving both motors the same speed but having only the left motors high (vice versa for left turn).

The robot is powered by 2 AA batteries and utilises 2 acrylic chassis to uphold all the components, since it and all other components are rigid it allows for the utilisation of rigid kinematics. An ultrasound sensor used to measure distance of obstacles, servo motor (used to turn ultrasound sensor 170 degrees) is used to execute the Bug 2 algorithm. An issue with the robot was that the wires being sensed by the ultrasound when it turns using the servo motor which meant that it acted as though it had seen an object. This was easily overcome by taping the wires down onto the chaises.

Microcontroller:

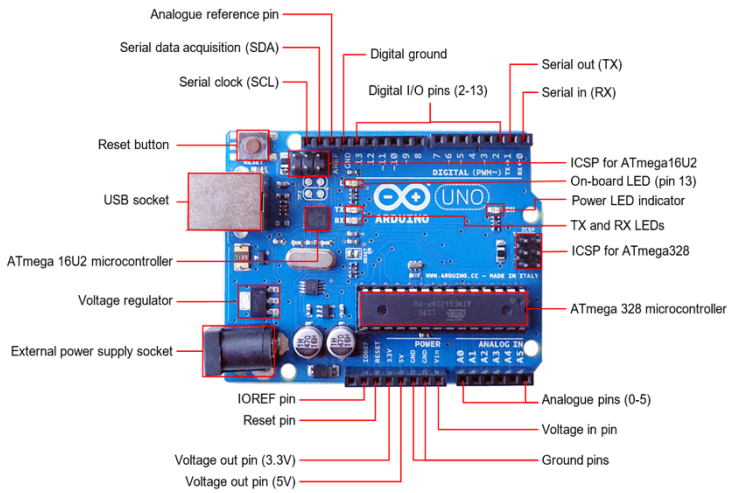

Fig 3. Arduino Uno microcontroller labelled.[5]

Hardware Interface (Arduino):
Since there are multiple ports on the Arduino Uno microcontroller allows for multiple sensors to be used simultaneously. This is via the Digital I/O pins 2-13. The code or software can be easily uploaded to the board via the USB socket and erased via the rest button. The digital I/O pins are used to connect the servo motor and the wheel motors.

Software Interface:
The language used to program an Arduino Uno is like C++, it requires all used port and speed used for motors to be defined. It can also initialise global/local integers and constant integers that in the code are used to set the distance limit and keep track of the numcycles. The main code utilises if-else statements to execute different code for different circumstances the robot comes across. While statements can also be used for recuring movements if the surrounding are in a specific condition. Functions can be created to call specific motion, such as go_Advance (see Appendix fig.1) or carry out test to check for objects within the vicinity using the ultrasound sensor, such as watch() (see Appendix fig.2). The code is used to control the speed which is important in constraint the robot's movement to prevent it crashing while executing a piece of code and allows for more opportunities for ultrasound sensor checks to the surrounding.

Ultrasound:
An ultrasound relies on sound waves to check the distance of an object, it is important to check the accuracy at which the ultrasound detects objects to configure the code accordingly.

Accuracy testing results:
Fig 4. The box lid and battery used to test distance accuracy.

It is important to test the accuracy in the different types of objects detected, depending on size and curvature to create the parameters and assumption for the testing environment. This was done via placing an object at different distance from the ultrasound, using a ruler to measure from the outer edge of the ultrasound to front of the object. The object tested includes a black lid and a round blue battery, shown in figure 4. The code compiled to check the distance is include in the Appendix Fig 3, and the ultrasound was forward facing for all tests. Figure 5 and 6 shows the accuracy testing results for each object in graphical form, the tabulated results are included in the Appendix Fig 4 and 5.

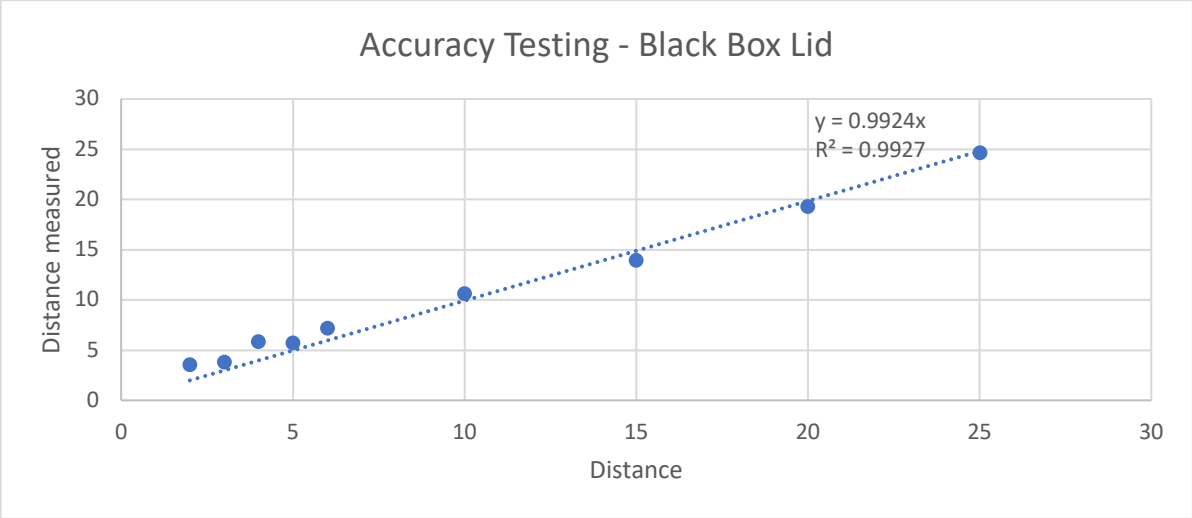

Fig 5. Results of the black box lid accuracy testing

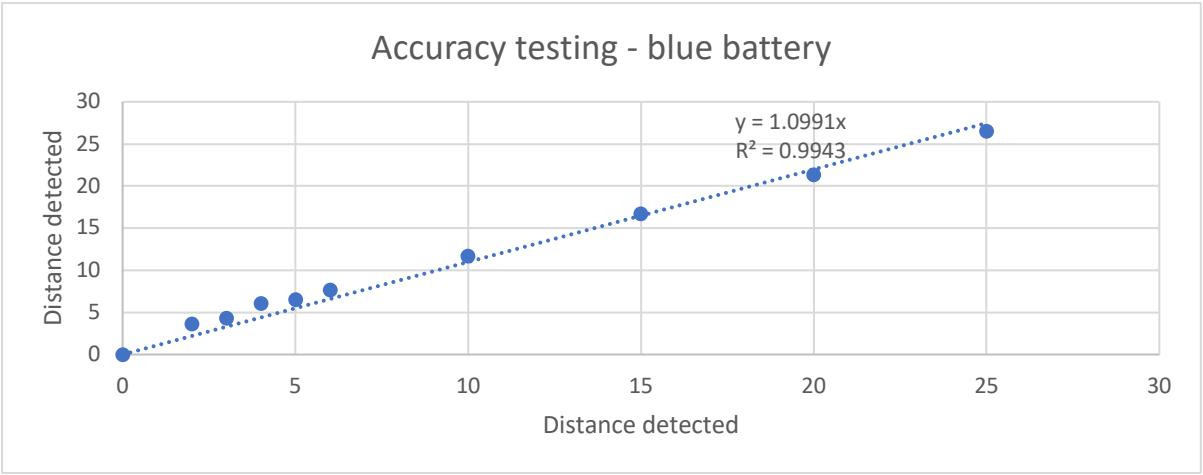

Fig 6. Results of the battery accuracy testing

It was found that objects had to be at least 4 cm tall to be detected by the ultrasound. There was found to be a systematic error of approximately 1cm due to measuring from the outer edge of the ultrasound rather from the edge connected to the board and an outlier for the box lid accuracy testing at 15cm. Putting the systematic error of 1cm aside the ultrasound measured an excess of approx.1cm for the blue battery, this is likely due to it measuring the rounded sides of the battery, rather than the closer surface area. Regarding the testing environment, it is assumed the objects the robot interacts with are flat, least 7 cm tall to ensure they can be accurately detected by the ultrasound sensor.

Servo configuration and distance limit:
Servo angles were adapted to ensure that an object at the side, front and front corners can be detected. To get an accurate distance limit results were tabulated for the distance measure by the ultrasound for the robot's perimeter depending on different servo angles. The set up used to measure the distance limit at write (), front right corner, is shown in Fig 7 and the average results are tabulated in Fig 8 (see Appendix Figure 6 for the raw data and Figure 3 for the code, angles changed using the write() function).

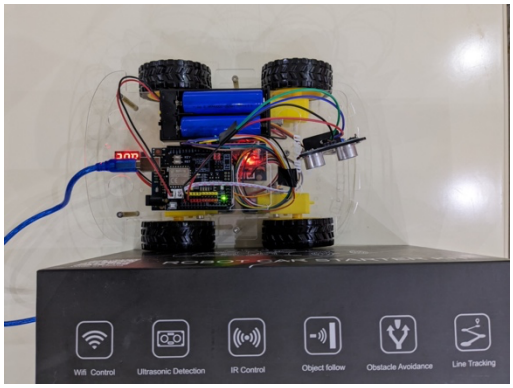

Fig 7 shows the layout used to get the distance limit for the front right corner.

Distance limit					
Servo (./write()	Right side	Upper Right Corner	Front	Upper Left Corner	Left side
0	7				
30	N/A	8	13		
70		11	4	11	
110			11	7	N/A
170					6

Fig 8. The results for the distance limit.

The results provide insight into the range in the view that can be detected at each angle, since the left and right side can only be detected at 0 or 170, to reduce error the ultrasound should check twice to ensure that the distance is measured or provide an additional angle for the servo.

Testing environment:
Flat walls vertical walls are used, without plaster at the bottom, are used for the testing environment. The walls should not be angled since the distance detected may have been deflected off the wall, giving incorrect readings. Flat floor to ensure the horizontal distance is the same to the ultrasound found distance, and since a slope can have an impact on the power consumption of the robot. Use only objects that are flat to avoid reading errors.

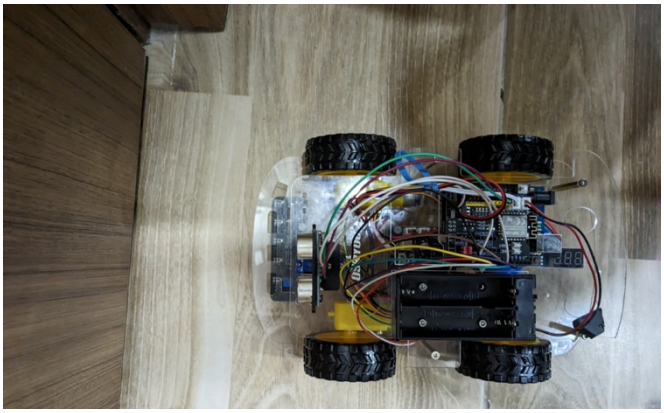

Fig 9. Testing environment example.

Algorithm testing and experimentation:

Aim:
The aim of the algorithms is to provide the basis required to perform bug 2 algorithms by testing individual aspects such as obstacle avoidance and simple boundary following algorithm, that works via turning left and right. The final aim is to use these algorithms to create a Wall following algorithm with obstacle avoidance.

Algorithms:
Obstacle avoidance (Robots' behaviour analysis):
The code utilised for obstacle avoidance is from the Osoyoo website, it uses the if else statement to turn the robot depending on its position when watch () is completed, the code and flowchart is provided in Appendix Figure 7 and 8 respectively. watch () Turns the servo to 5 different angles and makes the binary value high if there is an obstacle within the distance limit from it. The if else command checks which binary number is high

in the 00000 configuration and turns the opposite side. For example, 00001 the servo has found an obstacle on the write and therefore would turn the robot left. This is done by setting the motor speed to turn speed which allow the robot to turn quickly in its position and then calling the go_Left. The direction functions like go_Left configure the motor wheels by setting the corresponding right/left upper/lower wheel high or Low. The code uses a similar theory as in the “Robot Programming”- by J L Jones, McGraw Hill, 2004. It deals with canyoning, represented by 11011 by going right rather than straight as presented in the

Limitation of the obstacle avoidance code is that as it turns to avoid one obstacle it may crash into another, this is overcome by reducing the overall SPEED, TURN_SPEED and FAST_SPEED. This helps to prevent the robot from entering the distance limit to another object and crashing into it as it can do the watch() function more often over the same distance. another issue is round object or objects in the shape of a triangle as the servo when facing forward may not sense the close point of the triangle rather the sides, which depending on the width of the base the servo may get the distance greater than the distance limit and crash into the object. This is modelled below in Fig 10 and 11.

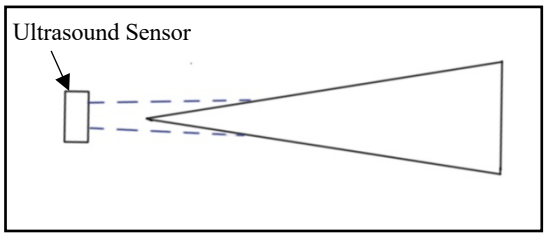

Fig 10. Shows the ultrasound not detecting the closer point of the triangle due to its narrow base, providing an incorrect measurement for distance measured.

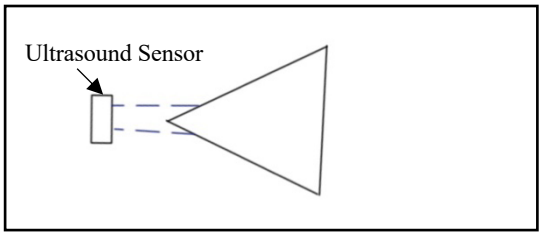

Fig 11. Shows the ultrasound detecting the closer point of the triangle due to its wider base, providing a better measurement for distance measured.

The final issue is when it comes across a box and turns past beyond its corner at an angle and finds a new object to its side. In this scenario instead of going slight left (or right) it fails to sense with the ultrasound that the box hasn’t been completely cleared. As a result, it crashes into the previously passed box, this scenario is illustrated in figure 12. This shows that the detected scenario by the ultrasound is not always affirmative of the actual environment, it can be overcome by having multiple sensors or a 360 servo so that it can survey its entire surrounding.

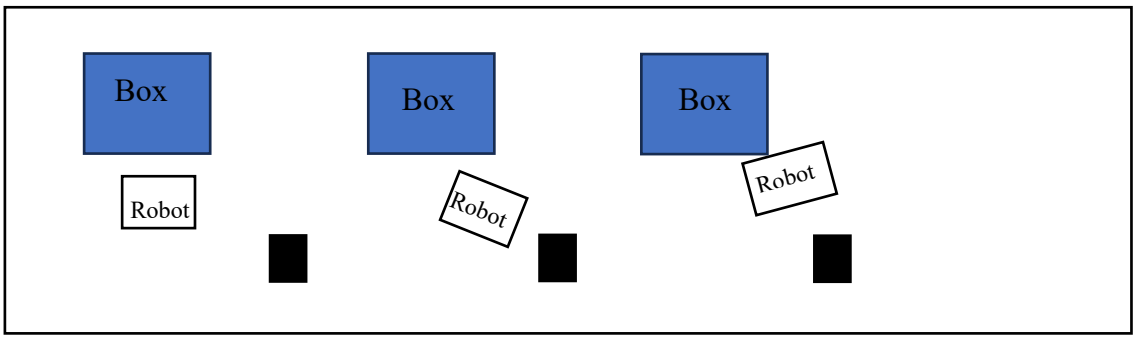

Fig 12. Models the scenario of the prepared code for the scenario being executed but resulted in a collision.

Wall following:
The aim of this algorithm is to follow an object around via doing a zig zag motion around it, the code and

flowchart is provided in Appendix Figure 9 and 10 respectively. Using an if else statement the robot turns right when there's an obstacle in front of it, at the start of the program this can be the wall. By going right, it senses the wall and goes left. When it doesn't sense any obstacle, it goes right. This allows it to re-find and follow the wall even when it's pointing away from it. The algorithm works to an extent as it manages to move alongside the wall, however it goes left it usually end up much further than expected from an object. To handle this an int variable leftloopcycle is declared where if it occurs more than twice it turns slightly left rechecks the surroundings. To prevent canyoning, it advances after each right or left turn allowing for it to overcome the obstacles in its path. The key part of the code is shown in figure 13 below.

```
void objectDetected() {
  String obstacle_sign2 = watchsurrounding();  // 5 digits of obstacle_sign
binary value means the 5 direction obstacle status
  if (leftloopcycle > 3) {
    Serial.println("Slight right");
    set_Motorspeed(SPEED, FAST_SPEED);
    go_Advance();
    delay(turntime);
    stop_Stop();
    detectObject();
  } else if {
    (obstacle_sign2 == "11011" || obstacle_sign2 == "11101" || obstacle_sign2
== "11110" || obstacle_sign2 == "01110" || obstacle_sign2 == "11100" ||
obstacle_sign2 == "01000" || obstacle_sign2 == "11000" || obstacle_sign2 ==
"10100" || obstacle_sign2 == "01100" || obstacle_sign2 == "00100" ||
obstacle_sign2 == "01000" || obstacle_sign2 == "00010" || obstacle_sign2 ==
"00111" || obstacle_sign2 == "00011" || obstacle_sign2 == "00101" ||
obstacle_sign2 == "00110" || obstacle_sign2 == "01010" || obstacle_sign2 ==
"01111" || obstacle_sign2 == "10111" || obstacle_sign2 == "11111") {
      go_Right();
      Serial.println("Obstacle");
      set_Motorspeed(TURN_SPEED, TURN_SPEED);
      delay(turntime);
      go_Advance();
      set_Motorspeed(SPEED, SPEED);
      delay(turntime);
      stop_Stop();
      leftloopcycle = 0;
    }
    else {
      Serial.println("go left");
      go_Left();  //Turn left
      set_Motorspeed(TURN_SPEED, TURN_SPEED);
      delay(turntime);
      go_Advance();
      set_Motorspeed(SPEED, SPEED);
      delay(turntime);
        stop_Stop();
      ++leftloopcycle;
    }
    objectDetected();
  }
}
```

It is very limited as whole since if the wall is slightly slanted it would eventually hit it and its movements were erratic resulting in it hitting the wall. It wasn't effective in re-finding the wall and risked going in circles. To improve the code, instead of listing all possible configurations, it should have been else if (obstacle_sign== "00000") go left, else go right, as it would prevent missing a configuration. The speed can be reduced to increase efficiency, by allowing more surrounding distance checks.

If the robot keeps going left it goes a slight right

If there is an object go left and increase left counter by 1

If no obstacle go left

Fig. 13 Shows the key code of the objectDetected() function.

Wall Following with Obstacle Avoidance:
The aim with this algorithm is to successful wall follow by having the robot circumvent the boundary in a zig zag pattern, the code and flowchart is provided in Appendix Figure 11 and 12 respectively. A zig zag (right-left) pattern is used within an upper and lower distance limit range to move the robot across the wall. Ideally the robot would go straight at the wall, however, is the robot is at angle and the code states for it to go straight it would go straight into the wall. If it is outside of the wall following range or distance limit used for obstacle avoidance, the robot should move slightly left to boundary follow and obstacle and re-find the wall. Since it'll turn left it prevents the robot to return to the same wall but follow it from the opposite direction. However, it is still possible for it to follow a parallel wall from the opposite direction.

Issues remain concerning the ultrasound and detecting, round objects or corners. Other issues are regarding the motion of the robot when it does the right-left wall following as it may still crash into the wall if the speed is too great. The greatest difference between this algorithm and those covered in class is that it utilises a while loop to continue along the wall rather than if else. This is so that can continue along the wall without executing the obstacle avoidance code, whilst still checking there are no objects in the surrounding that are smaller than the distance limits. Key parts of the code are analysed below in Figure 14.

Fig. 14 Shows the key aspect of the checkConsistency() function that does a separate check to watchsurrounding() to check is the movement is consistent with the provided range for the distance on the right sensor.

Discussion:

Overall, this report discusses the theory required to understand the aim, the analyses the hardware and software to build a good testing environment. The final algorithm successful to an extent in executing the Bug 2 aim. Below is a more detailed analysis of each algorithm and the improvement it brought forth.

The first algorithm is useful in checking obstacle avoidance using an ultrasound, its limitations and importance of the testing environment. One such being that the program is limited to the servo's degree of freedom and the height of the ultrasound as it is unable to detect small objects in its path. Also, when the robot is stuck it lack sensors in the back of the robot to make it aware of this fact and terminate the program. Such issues are important to consider when building and analysing algorithm as it illustrates the improvements required to allow for the robot to function in more diverse surrounding. The algorithm uses the ideas put forth in Kaspar Althoefer's lectures in which he references Robot Programming"- by J L Jones, McGraw Hill, 2004, for the logic behind the code.

The second algorithm uses only left and right turns with advance to move the robot along the wall in a zig-zag pattern. It is limited in accuracy and effectives to deal with returning to the object or prevent crashing into it, like obstacle avoidance. This time however it because the robot has been set to turn left when there is no object in front of it. Though this meant it can return to the side the wall was on it fails to avoid hitting it or to perform more concise boundary following.

The final algorithm uses a while loop, which had not been introduced in the lectures, but allowed for recuring movement along the boundary of the wall. It combines obstacle avoidance to prevent crashing into the obstacles and wall, which was the greatest limitation of the second algorithm when it attempted boundary following.

All algorithms can be improved via better comments, and more specific branches for each configuration in conjunction with and additional ultrasound or 360 servos to give specific instructions. The print statement should be clearer to allow for better insight into the robots thought process when reading the serial monitor.

Conclusion:

In conclusion, a deep understanding of the Osoyoo robot is developed, which is used to build a better suited algorithm to the robot and attempt to overcome its hardware limitations via software alterations. Via analysing the algorithms a great understand of the code is required to be able to perform the adaptations required. A bug 2 algorithm is possible on the Osoyoo Robot, a version is provided in Algorithm 3, and simply allows for a boundary wall following with obstacle avoidance.

References:

1. Lecture notes "16-311 Introduction to Robotics" by Sean Pieper, Bob Grabowski, Howie Choset, Carnegie Melon (HC-05)
2. "Principles of Robot Motion" - by Choset, Lynch, Hutchinson, Kantor, Burgard, Kavraki, Thrun, The MIT Press, 2005
3. Prof Kaspar Althoefer – QMUL – Aspects of Robotics – Local Navigation
4. Osoyoo Robot Car Learning Kits (no date) OSOYOO.Store. Available at: https://osoyoo.store/collections/robot-car (Accessed: 27 December 2023).
5. Arduino Uno (no date) Arduino Uno - Internet of Things. Available at: https://bdavison.napier.ac.uk/iot/Notes/microprocessors/arduino/ (Accessed: 24 December 2023).
6. Osoyoo v2.1 robot car kit lesson 5: Obstacle avoidance robot car (no date) OSOYOO V2.1 Robot car kit Lesson 5: Obstacle Avoidance Robot Car " osoyoo.com. Available at: https://osoyoo.com/2020/05/12/osoyoo-v2-1-robot-car-kit-lesson-5-obstacle-avoidance-robot-car/ (Accessed: 22 December 2023).

Appendix:

```
/*Arduino Uno (no date) Arduino Uno – Internet of Things. Available at:
https://bdavison.napier.ac.uk/iot/Notes/microprocessors/arduino/ (Accessed: 24 December
2023). */
```

```
/*motor control*/
void go_Advance(void)  //Forward
{
  digitalWrite(RightDirectPin1, LOW);
  digitalWrite(RightDirectPin2,HIGH);
  digitalWrite(LeftDirectPin1,LOW);
  digitalWrite(LeftDirectPin2,HIGH);
}
void go_Left()  //Turn left
{
  digitalWrite(RightDirectPin1, HIGH);
  digitalWrite(RightDirectPin2,LOW);
  digitalWrite(LeftDirectPin1,LOW);
  digitalWrite(LeftDirectPin2,HIGH);
}
void go_Right()  //Turn right
{
  digitalWrite(RightDirectPin1, LOW);
  digitalWrite(RightDirectPin2,HIGH);
  digitalWrite(LeftDirectPin1,HIGH);
  digitalWrite(LeftDirectPin2,LOW);
}
void go_Back()  //Reverse
{
  digitalWrite(RightDirectPin1, HIGH);
  digitalWrite(RightDirectPin2,LOW);
  digitalWrite(LeftDirectPin1,HIGH);
  digitalWrite(LeftDirectPin2,LOW);
}
void stop_Stop()    //Stop
{
  digitalWrite(RightDirectPin1, LOW);
  digitalWrite(RightDirectPin2,LOW);
  digitalWrite(LeftDirectPin1,LOW);
  digitalWrite(LeftDirectPin2,LOW);
  set_Motorspeed(0,0);
}
```

A.Fig 1. The code utilised to get different direction or motions. please see reference 6 for the code)

```
/*Arduino Uno (no date) Arduino Uno – Internet of Things. Available at:
https://bdavison.napier.ac.uk/iot/Notes/microprocessors/arduino/ (Accessed: 24 December
2023). */

/*detection of ultrasonic distance*/
int watch() {
  long echo_distance;
  digitalWrite(Trig_PIN, LOW);
  delayMicroseconds(5);
  digitalWrite(Trig_PIN, HIGH);
  delayMicroseconds(15);
```

```
  digitalWrite(Trig_PIN, LOW);
  echo_distance = pulseIn(Echo_PIN, HIGH);
  echo_distance = echo_distance * 0.01657;  //how far away is the object in cm
  //Serial.println((int)echo_distance);
  return round(echo_distance);
}
```

A.Fig 2. The code utilised to get the distance to object (please see report reference 6 for the code)

```
#include <Servo.h>

#define SERVO_PIN 9  //servo connect to D9
#define Echo_PIN 2   // Ultrasonic Echo pin connect to D11
#define Trig_PIN 10  // Ultrasonic Trig pin connect to D12
int thereis;
Servo head;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}
void loop() {
  // put your main code here, to run repeatedly:
  head.write(70);
  delay(250);
  long echo_distance;
  digitalWrite(Trig_PIN,LOW);
  delayMicroseconds(5);
  digitalWrite(Trig_PIN,HIGH);
  delayMicroseconds(5);
  digitalWrite(Trig_PIN,LOW);
  echo_distance=pulseIn(Echo_PIN,HIGH);
  echo_distance=echo_distance*0.01657; //how far away is the object in cm
  Serial.println(round(echo_distance));

  Serial.print("Distance = ");
}
```

A.Fig 3. The code utilised to get the distance detected at different servo degrees(write).[4]

Object: Osyoyoo black box lid				
Distance (cm)	Distance detected (cm)			
	Test 1:	Test 2:	Test 3:	AVG
2	3.42	3.67	3.52	3.536667
3	3.77	3.79	3.83	3.796667
4	5.73	6.1	5.68	5.836667
5	6	5.53	5.67	5.733333
6	6.72	7.12	7.67	7.17
10	10.47	11.24	10.1	10.60333

15	14.13	13.6	14.01	13.91333
20	19.41	19.43	19.01	19.28333
25	24.36	24.55	24.91	24.60667

A.Fig 4. Raw data for accuracy testing distance for black box lid objected.

Object: Osyoyoo blue battery				
Distance (cm)	Distance detected (cm)			
	Test 1:	Test 2:	Test 3:	AVG
2	3.7	3.85	3.32	3.623333
3	4.21	4.5	4.32	4.343333
4	5.9	5.71	6.58	6.063333
5	6.1	7.23	6.33	6.553333
6	7.54	7.98	7.51	7.676667
10	11.53	11.39	12.11	11.67667
15	17.23	16.32	16.63	16.72667
20	21.05	21.67	21.45	21.39
25	26.98	26.45	26.23	26.55333

A.Fig 5. Raw data for accuracy testing distance for black box lid objected.

Perimeter of the robot found by using box lid so that these are parameters for not hitting object when going around it			
SERVO FACING FORWARD write(70)			
	Front detected (cm)	Right front corner	Left front corner
	4	11	10
	4	10	11
	4	11	11
	4	10	11
Max value	4	11	11
SERVO AT write(0)			
	min distance to right side		
	7		
	7		
	7		
	7		
Max value	7		
write 30 looked at right corner edge			
	min to RIGHT edge \	Straight ahead	right side wall
	7	13	968
	7	11	968
	8	13	968

	8	13	969
Max value	8	13	sensor not working
write 110 looked at left corner edge			
	min to left edge \	Straight ahead	left side wall
	7	11	146
	7	11	144
	7	11	145
	7	9	150
Max value	7	11	sensor not working
write 170 looked at left side			
	min to side		
	6		
	6		
	6		
	5		
Max value	6		

Fig 6. The raw data to get the correct servo angles and the detected perimeter.

```
/*  ___   ___  ___  _   _  ___   ___   ____ ___  ____
 * / _ \ /___)/ _ \| | | |/ _ \ / _ \ / ___) _ \|    \
 *| |_| |___ | |_| | |_| | |_| | |_| ( (__| |_| | | | |
 * \___/(___/ \___/ \__  |\___/ \___(_)____)___/|_|_|_|
 *                  (____/
 * Arduino Smart Car Tutorial Lesson 5
 * Tutorial URL http://osoyoo.com/2018/12/19/osoyoo-robot-car-kit-lesson-4-
obstacle-avoidance-robot-car/
 * CopyRight www.osoyoo.com

 * This project will show you how to make Osoyoo robot car in auto drive mode and
avoid obstacles
 *
 *
 */
#include <Servo.h>
/*Declare L298N Dual H-Bridge Motor Controller directly since there is not a
library to load.*/
//Define L298N Dual H-Bridge Motor Controller Pins
#define speedPinR 3          // RIGHT PWM pin connect MODEL-X ENA
#define RightDirectPin1 12   //  Right Motor direction pin 1 to MODEL-X IN1
#define RightDirectPin2 11   // Right Motor direction pin 2 to MODEL-X IN2
#define speedPinL 6          //  Left PWM pin connect MODEL-X ENB
#define LeftDirectPin1 7     // Left Motor direction pin 1 to MODEL-X IN3
#define LeftDirectPin2 8     ///Left Motor direction pin 1 to MODEL-X IN4
#define LPT 2                // scan loop coumter

#define SERVO_PIN 9  //servo connect to D9

#define Echo_PIN 2   // Ultrasonic Echo pin connect to D11
```

```
#define Trig_PIN 10  // Ultrasonic Trig pin connect to D12

#define BUZZ_PIN 13
#define FAST_SPEED 250   //both sides of the motor speed
#define SPEED 120        //both sides of the motor speed
#define TURN_SPEED 200   //both sides of the motor speed
#define BACK_SPEED1 255  //back speed
#define BACK_SPEED2 90   //back speed

int leftscanval, centerscanval, rightscanval, ldiagonalscanval, rdiagonalscanval;
const int distancelimit = 10;      //distance limit for obstacles in front
const int sidedistancelimit = 10;  //minimum distance in cm to obstacles at both
sides (the car will allow a shorter distance sideways)
int distance;
int numcycles = 0;
const int turntime = 250;  //Time the robot spends turning (miliseconds)
const int backtime = 300;  //Time the robot spends turning (miliseconds)

int thereis;
Servo head;
/*motor control*/
void go_Advance(void)  //Forward
{
  digitalWrite(RightDirectPin1, LOW);
  digitalWrite(RightDirectPin2, HIGH);
  digitalWrite(LeftDirectPin1, LOW);
  digitalWrite(LeftDirectPin2, HIGH);
}
void go_Left()  //Turn left
{
  digitalWrite(RightDirectPin1, HIGH);
  digitalWrite(RightDirectPin2, LOW);
  digitalWrite(LeftDirectPin1, LOW);
  digitalWrite(LeftDirectPin2, HIGH);
}
void go_Right()  //Turn right
{
  digitalWrite(RightDirectPin1, LOW);
  digitalWrite(RightDirectPin2, HIGH);
  digitalWrite(LeftDirectPin1, HIGH);
  digitalWrite(LeftDirectPin2, LOW);
}
void go_Back()  //Reverse
{
  digitalWrite(RightDirectPin1, HIGH);
  digitalWrite(RightDirectPin2, LOW);
  digitalWrite(LeftDirectPin1, HIGH);
  digitalWrite(LeftDirectPin2, LOW);
}
void stop_Stop()  //Stop
{
```

```
  digitalWrite(RightDirectPin1, LOW);
  digitalWrite(RightDirectPin2, LOW);
  digitalWrite(LeftDirectPin1, LOW);
  digitalWrite(LeftDirectPin2, LOW);
  set_Motorspeed(0, 0);
}

/*set motor speed */
void set_Motorspeed(int speed_L, int speed_R) {
  analogWrite(speedPinL, speed_L);
  analogWrite(speedPinR, speed_R);
}

void buzz_ON()  //open buzzer
{

  for (int i = 0; i < 100; i++) {
    digitalWrite(BUZZ_PIN, LOW);
    delay(2);  //wait for 1ms
    digitalWrite(BUZZ_PIN, HIGH);
    delay(2);  //wait for 1ms
  }
}
void buzz_OFF()  //close buzzer
{
  digitalWrite(BUZZ_PIN, HIGH);
}
void alarm() {
  buzz_ON();

  buzz_OFF();
}

/*detection of ultrasonic distance*/
int watch() {
  long echo_distance;
  digitalWrite(Trig_PIN, LOW);
  delayMicroseconds(5);
  digitalWrite(Trig_PIN, HIGH);
  delayMicroseconds(15);
  digitalWrite(Trig_PIN, LOW);
  echo_distance = pulseIn(Echo_PIN, HIGH);
  echo_distance = echo_distance * 0.01657;  //how far away is the object in cm
  //Serial.println((int)echo_distance);
  return round(echo_distance);
}
//Meassures distances to the right, left, front, left diagonal, right diagonal and
asign them in cm to the variables rightscanval,
//leftscanval, centerscanval, ldiagonalscanval and rdiagonalscanval (there are 5
points for distance testing)
String watchsurrounding() {
```

```
  /*  obstacle_status is a binary integer, its last 5 digits stands for if there is
any obstacles in 5 directions,
 *   for example B101000 last 5 digits is 01000, which stands for Left front has
obstacle, B100111 means front, right front and right ha
 */

  int obstacle_status = B100000;
  centerscanval = watch();
  if (centerscanval < distancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B100;
  }
  head.write(120);
  delay(100);
  ldiagonalscanval = watch();
  if (ldiagonalscanval < distancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B1000;
  }
  head.write(170);  //Didn't use 180 degrees because my servo is not able to take
this angle
  delay(300);
  leftscanval = watch();
  if (leftscanval < sidedistancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B10000;
  }

  head.write(90);  //use 90 degrees if you are moving your servo through the whole
180 degrees
  delay(100);
  centerscanval = watch();
  if (centerscanval < distancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B100;
  }
  head.write(40);
  delay(100);
  rdiagonalscanval = watch();
  if (rdiagonalscanval < distancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B10;
  }
  head.write(0);
  delay(100);
  rightscanval = watch();
```

```
  if (rightscanval < sidedistancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | 1;
  }
  head.write(90);  //Finish looking around (look forward again)
  delay(300);
  String obstacle_str = String(obstacle_status, BIN);
  obstacle_str = obstacle_str.substring(1, 6);

  return obstacle_str;  //return 5-character string standing for 5 direction
obstacle status
}

void auto_avoidance() {

  ++numcycles;
  if (numcycles >= LPT) {  //Watch if something is around every LPT loops while
moving forward
    stop_Stop();
    String obstacle_sign = watchsurrounding();  // 5 digits of obstacle_sign binary
value means the 5 direction obstacle status
    Serial.print("begin str=");
    Serial.println(obstacle_sign);
    if (obstacle_sign == "10000") {
      Serial.println("SLIT right");
      set_Motorspeed(FAST_SPEED, SPEED);
      go_Advance();

      delay(turntime);
      stop_Stop();
    } else if (obstacle_sign == "00001") {
      Serial.println("SLIT LEFT");
      set_Motorspeed(SPEED, FAST_SPEED);
      go_Advance();

      delay(turntime);
      stop_Stop();
    } else if (obstacle_sign == "11100" || obstacle_sign == "01000" ||
obstacle_sign == "11000" || obstacle_sign == "10100" || obstacle_sign == "01100" ||
obstacle_sign == "00100" || obstacle_sign == "01000") {
      Serial.println("hand right");
      go_Right();
      set_Motorspeed(TURN_SPEED, TURN_SPEED);
      delay(turntime);
      stop_Stop();
    } else if (obstacle_sign == "00010" || obstacle_sign == "00111" ||
obstacle_sign == "00011" || obstacle_sign == "00101" || obstacle_sign == "00110" ||
obstacle_sign == "01010") {
      Serial.println("hand left");
      go_Left();  //Turn left
```

```
      set_Motorspeed(TURN_SPEED, TURN_SPEED);
      delay(turntime);
      stop_Stop();
    }

    else if (obstacle_sign == "01111" || obstacle_sign == "10111" || obstacle_sign
== "11111") {
      Serial.println("hand back right");
      go_Left();
      set_Motorspeed(FAST_SPEED, SPEED);
      delay(backtime);
      stop_Stop();
    } else if (obstacle_sign == "11011" || obstacle_sign == "11101" ||
obstacle_sign == "11110" || obstacle_sign == "01110") {
      Serial.println("hand back left");
      go_Right();
      set_Motorspeed(SPEED, FAST_SPEED);
      delay(backtime);
      stop_Stop();
    }

    else Serial.println("no handle");
    numcycles = 0;  //Restart count of cycles
  } else {
    set_Motorspeed(SPEED, SPEED);
    go_Advance();  // if nothing is wrong go forward using go() function above.
    delay(backtime);
    stop_Stop();
  }

  //else  Serial.println(numcycles);

  distance = watch();              // use the watch() function to see if anything
is ahead (when the robot is just moving forward and not looking around it will test
the distance in front)
  if (distance < distancelimit) {  // The robot will just stop if it is completely
sure there's an obstacle ahead (must test 25 times) (needed to ignore ultrasonic
sensor's false signals)
    Serial.println("final go back");
    go_Right();
    set_Motorspeed(SPEED, FAST_SPEED);
    delay(backtime * 3 / 2);
    ++thereis;
  }
  if (distance > distancelimit) {
    thereis = 0;
  }  //Count is restarted
  if (thereis > 25) {
    Serial.println("final stop");
    stop_Stop();  // Since something is ahead, stop moving.
    thereis = 0;
```

```
  }
}

void setup() {
  /*setup L298N pin mode*/
  pinMode(RightDirectPin1, OUTPUT);
  pinMode(RightDirectPin2, OUTPUT);
  pinMode(speedPinL, OUTPUT);
  pinMode(LeftDirectPin1, OUTPUT);
  pinMode(LeftDirectPin2, OUTPUT);
  pinMode(speedPinR, OUTPUT);
  stop_Stop();  //stop move
  /*init HC-SR04*/
  pinMode(Trig_PIN, OUTPUT);
  pinMode(Echo_PIN, INPUT);
  /*init buzzer*/
  pinMode(BUZZ_PIN, OUTPUT);
  digitalWrite(BUZZ_PIN, HIGH);
  buzz_OFF();

  digitalWrite(Trig_PIN, LOW);
  /*init servo*/
  head.attach(SERVO_PIN);
  head.write(90);
  delay(2000);

  Serial.begin(9600);
}

void loop() {
  auto_avoidance();
}
```

Fig 7. Obstacle avoidance code from Osoyoo (see report reference 6)

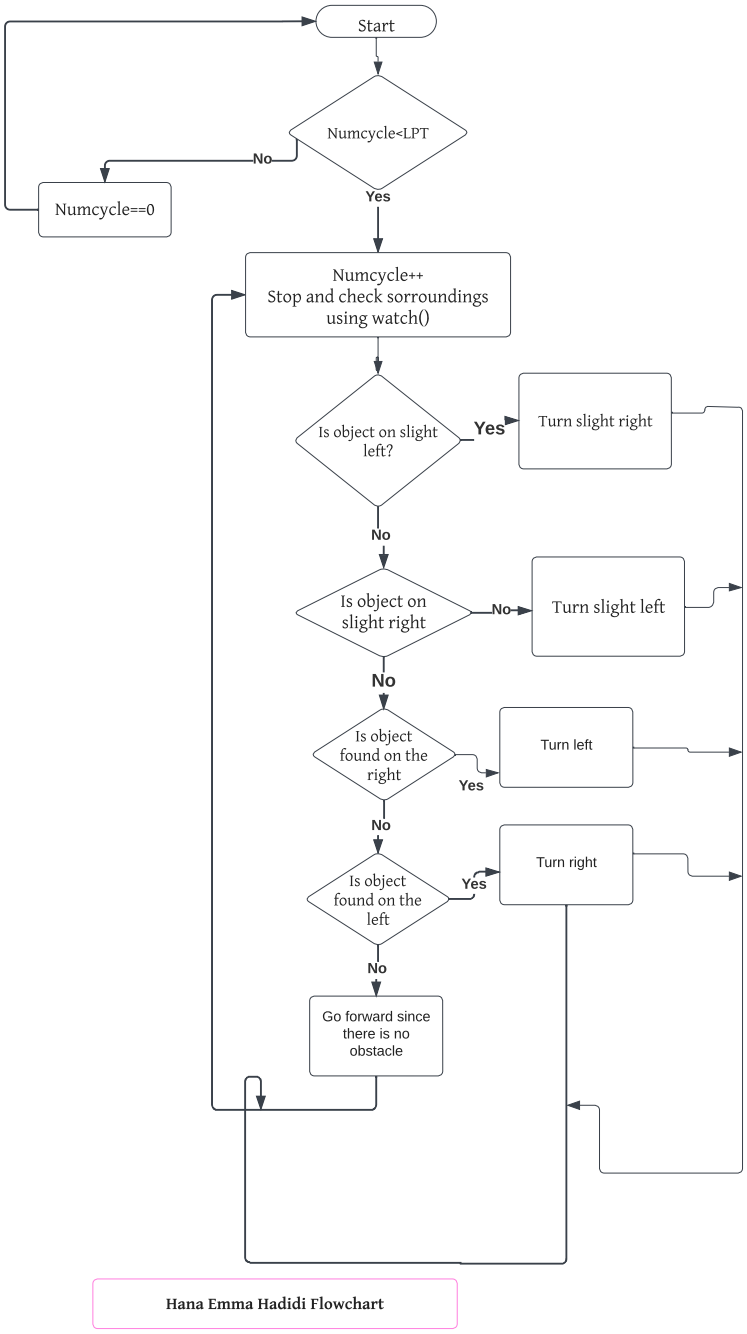

Fig 8 Flowchart for Obstacle Avoidance Code

```
//Zig zag wall following
/*Arduino Uno (no date) Arduino Uno – Internet of Things. Available at:
https://bdavison.napier.ac.uk/iot/Notes/microprocessors/arduino/ (Accessed: 24
December 2023). */
#include <Servo.h>
//Declare L298N Dual H-Bridge Motor Controller directly since there is not a
library to load./
//Define L298N Dual H-Bridge Motor Controller Pins
#define speedPinR 3         // RIGHT PWM pin connect MODEL-X ENA
#define RightDirectPin1 12  //  Right Motor direction pin 1 to MODEL-X IN1
```

```
#define RightDirectPin2 11  // Right Motor direction pin 2 to MODEL-X IN2
#define speedPinL 6         //  Left PWM pin connect MODEL-X ENB
#define LeftDirectPin1 7    // Left Motor direction pin 1 to MODEL-X IN3
#define LeftDirectPin2 8    ///Left Motor direction pin 1 to MODEL-X IN4
#define LPT 2               // scan loop coumter

#define SERVO_PIN 9  //servo connect to D9

#define Echo_PIN 2  // Ultrasonic Echo pin connect to D11
#define Trig_PIN 1  // Ultrasonic Trig pin connect to D12

#define BUZZ_PIN 13
#define FAST_SPEED 250   //both sides of the motor speed
#define SPEED 120        //both sides of the motor speed
#define TURN_SPEED 200   //both sides of the motor speed
#define BACK_SPEED1 255  //back speed
#define BACK_SPEED2 90   //back speed

int leftscanval, centerscanval, rightscanval, ldiagonalscanval, rdiagonalscanval;
const int distancelimit = 15;      //distance limit for obstacles in front
const int sidedistancelimit = 15;  //minimum distance in cm to obstacles at both
sides (the car will allow a shorter distance sideways)
int distance;
int numcycles = 0;
const int turntime = 250;     //Time the robot spends turning (miliseconds) 250
const int backtime = 300;     //Time the robot spends turning (miliseconds) 300
const int advancetime = 300;  //Time the robot spends turning (miliseconds) 300
int leftloopcycle;

int thereis;
Servo head;
//motor control//
void go_Advance(void)  //Forward
{
  digitalWrite(RightDirectPin1, LOW);
  digitalWrite(RightDirectPin2, HIGH);
  digitalWrite(LeftDirectPin1, LOW);
  digitalWrite(LeftDirectPin2, HIGH);
}
void go_Left()  //Turn left
{
  digitalWrite(RightDirectPin1, HIGH);
  digitalWrite(RightDirectPin2, LOW);
  digitalWrite(LeftDirectPin1, LOW);
  digitalWrite(LeftDirectPin2, HIGH);
}
void go_Right()  //Turn right
{
  digitalWrite(RightDirectPin1, LOW);
  digitalWrite(RightDirectPin2, HIGH);
  digitalWrite(LeftDirectPin1, HIGH);
```

```
  digitalWrite(LeftDirectPin2, LOW);
}
void go_Back()  //Reverse
{
  digitalWrite(RightDirectPin1, HIGH);
  digitalWrite(RightDirectPin2, LOW);
  digitalWrite(LeftDirectPin1, HIGH);
  digitalWrite(LeftDirectPin2, LOW);
}
void stop_Stop()  //Stop
{
  digitalWrite(RightDirectPin1, LOW);
  digitalWrite(RightDirectPin2, LOW);
  digitalWrite(LeftDirectPin1, LOW);
  digitalWrite(LeftDirectPin2, LOW);
  set_Motorspeed(0, 0);
}

/*set motor speed */
void set_Motorspeed(int speed_L, int speed_R) {
  analogWrite(speedPinL, speed_L);
  analogWrite(speedPinR, speed_R);
}

void buzz_ON()  //open buzzer
{

  for (int i = 0; i < 100; i++) {
    digitalWrite(BUZZ_PIN, LOW);
    delay(2);  //wait for 1ms
    digitalWrite(BUZZ_PIN, HIGH);
    delay(2);  //wait for 1ms
  }
}
void buzz_OFF()  //close buzzer
{
  digitalWrite(BUZZ_PIN, HIGH);
}
void alarm() {
  buzz_ON();

  buzz_OFF();
}

//detection of ultrasonic distance//
int watch() {
  long echo_distance;
  digitalWrite(Trig_PIN, LOW);
  delayMicroseconds(5);
  digitalWrite(Trig_PIN, HIGH);
  delayMicroseconds(15);
```

```
  digitalWrite(Trig_PIN, LOW);
  echo_distance = pulseIn(Echo_PIN, HIGH);
  echo_distance = echo_distance * 0.01657;  //how far away is the object in cm
  //Serial.println((int)echo_distance);
  return round(echo_distance);
}
//Meassures distances to the right, left, front, left diagonal, right diagonal and
asign them in cm to the variables rightscanval,
//leftscanval, centerscanval, ldiagonalscanval and rdiagonalscanval (there are 5
points for distance testing)
String watchsurrounding() {
  /*  obstacle_status is a binary integer, its last 5 digits stands for if there is
any obstacles in 5 directions,
 *   for example B101000 last 5 digits is 01000, which stands for Left front has
obstacle, B100111 means front, right front and right ha
 */

  int obstacle_status = B100000;
  centerscanval = watch();
  if (centerscanval < distancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B100;
  }
  head.write(110);
  delay(100);
  ldiagonalscanval = watch();
  if (ldiagonalscanval < distancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B1000;
  }
  head.write(170);  //Didn't use 180 degrees because my servo is not able to take
this angle
  delay(300);
  leftscanval = watch();
  if (leftscanval < sidedistancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B10000;
  }

  head.write(70);  //use 90 degrees if you are moving your servo through the whole
180 degrees
  delay(100);
  centerscanval = watch();
  if (centerscanval < distancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B100;
  }
```

```
  head.write(30);
  delay(100);
  rdiagonalscanval = watch();
  if (rdiagonalscanval < distancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B10;
  }
  head.write(0);
  delay(100);
  rightscanval = watch();
  if (rightscanval < sidedistancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | 1;
  }
  head.write(70);  //Finish looking around (look forward again)
  delay(300);
  String obstacle_str = String(obstacle_status, BIN);
  obstacle_str = obstacle_str.substring(1, 6);

  return obstacle_str;  //return 5-character string standing for 5 direction
obstacle status
}
void objectDetected() {

  String obstacle_sign2 = watchsurrounding();  // 5 digits of obstacle_sign binary
value means the 5 direction obstacle status
  if (leftloopcycle > 3) {
    Serial.println("Slight right");
    set_Motorspeed(SPEED, FAST_SPEED);
    go_Advance();
    delay(turntime);
    stop_Stop();
    detectObject();
  } else {
    if (obstacle_sign2 == "11011" || obstacle_sign2 == "11101" || obstacle_sign2 ==
"11110" || obstacle_sign2 == "01110" || obstacle_sign2 == "11100" || obstacle_sign2
== "01000" || obstacle_sign2 == "11000" || obstacle_sign2 == "10100" ||
obstacle_sign2 == "01100" || obstacle_sign2 == "00100" || obstacle_sign2 == "01000"
|| obstacle_sign2 == "00010" || obstacle_sign2 == "00111" || obstacle_sign2 ==
"00011" || obstacle_sign2 == "00101" || obstacle_sign2 == "00110" || obstacle_sign2
== "01010" || obstacle_sign2 == "01111" || obstacle_sign2 == "10111" ||
obstacle_sign2 == "11111") {
      go_Right();
      Serial.println("Obstacle");
      set_Motorspeed(TURN_SPEED, TURN_SPEED);
      delay(turntime);
      go_Advance();
      set_Motorspeed(SPEED, SPEED);
      delay(turntime);
```

```
      stop_Stop();
      leftloopcycle = 0;
    } else {
      Serial.println("go left");
      go_Left();  //Turn left
      set_Motorspeed(TURN_SPEED, TURN_SPEED);
      delay(turntime);
      go_Advance();
      set_Motorspeed(SPEED, SPEED);
      delay(turntime);
      stop_Stop();
      ++leftloopcycle;
    }
    objectDetected();
  }
}

void detectObject() {
  leftloopcycle == 0;
  ++numcycles;
  if (numcycles >= LPT) {  //Watch if something is around every LPT loops while
moving forward
    stop_Stop();
    String obstacle_sign = watchsurrounding();  // 5 digits of obstacle_sign binary
value means the 5 direction obstacle status
    Serial.print("begin str=");
    Serial.println(obstacle_sign);
    //Check if there is an object
    if (obstacle_sign == "11011" || obstacle_sign == "11101" || obstacle_sign ==
"11110" || obstacle_sign == "01110" || obstacle_sign == "11100" || obstacle_sign ==
"01000" || obstacle_sign == "11000" || obstacle_sign == "10100" || obstacle_sign ==
"01100" || obstacle_sign == "00100" || obstacle_sign == "01000" || obstacle_sign ==
"00010" || obstacle_sign == "00111" || obstacle_sign == "00011" || obstacle_sign ==
"00101" || obstacle_sign == "00110" || obstacle_sign == "01010" || obstacle_sign ==
"01111" || obstacle_sign == "10111" || obstacle_sign == "11111") {
      Serial.println("Object found");
      objectDetected();
    }
    //If no object go straight
    else if (obstacle_sign == "00001" || obstacle_sign == "10000" || obstacle_sign
== "00000" || obstacle_sign == "00011") {
      Serial.println("Ignore go straight");
      set_Motorspeed(SPEED, SPEED);
      go_Advance();
      set_Motorspeed(SPEED, SPEED);
      delay(turntime);
      stop_Stop();
    }

    else Serial.println("no handle");
    numcycles = 0;  //Restart count of cycles
```

```
  }

  //else  Serial.println(numcycles);
  distance = watch();              // use the watch() function to see if anything
is ahead (when the robot is just moving forward and not looking around it will test
the distance in front)
  if (distance < distancelimit) {  // The robot will just stop if it is completely
sure there's an obstacle ahead (must test 25 times) (needed to ignore ultrasonic
sensor's false signals)
    Serial.println("final go back");
    go_Right();
    set_Motorspeed(SPEED, FAST_SPEED);
    delay(backtime * 3 / 2);
    ++thereis;
  }
  if (distance > distancelimit) {
    thereis = 0;
  }  //Count is restarted
  if (thereis > 25) {
    Serial.println("final stop");
    stop_Stop();  // Since something is ahead, stop moving.
    thereis = 0;
  }
}

void setup() {
  //setup L298N pin mode/
  pinMode(RightDirectPin1, OUTPUT);
  pinMode(RightDirectPin2, OUTPUT);
  pinMode(speedPinL, OUTPUT);
  pinMode(LeftDirectPin1, OUTPUT);
  pinMode(LeftDirectPin2, OUTPUT);
  pinMode(speedPinR, OUTPUT);
  stop_Stop();  //stop move
  //init HC-SR04/
  pinMode(Trig_PIN, OUTPUT);
  pinMode(Echo_PIN, INPUT);
  //init buzzer/
  pinMode(BUZZ_PIN, OUTPUT);
  digitalWrite(BUZZ_PIN, HIGH);
  buzz_OFF();

  digitalWrite(Trig_PIN, LOW);
  //init servo/
  head.attach(SERVO_PIN);
  head.write(90);
  delay(2000);

  Serial.begin(9600);
}
```

```
void loop() {
  detectObject();
}
```

Fig 9. Code for basic wall following that uses Zig Zag pattern, uses aspect of the Osoyoo obstacle avoidance code (see Report Reference 6)

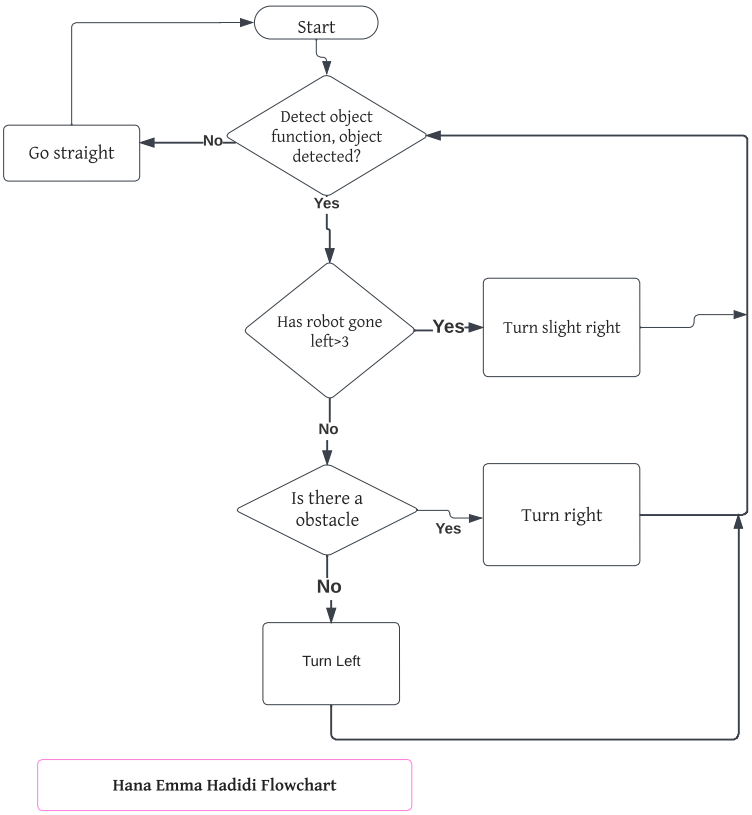

Fig 10. The flowchart for the basic wall following code that uses a zig zag pattern.

```
///Wallfollowing with obstacle avoidance
#include <Servo.h>
//Declare L298N Dual H-Bridge Motor Controller directly since there is not a
library to load./
//Define L298N Dual H-Bridge Motor Controller Pins
#define speedPinR 3          // RIGHT PWM pin connect MODEL-X ENA
#define RightDirectPin1 12   //  Right Motor direction pin 1 to MODEL-X IN1
#define RightDirectPin2 11   // Right Motor direction pin 2 to MODEL-X IN2
#define speedPinL 6          //  Left PWM pin connect MODEL-X ENB
#define LeftDirectPin1 7     // Left Motor direction pin 1 to MODEL-X IN3
#define LeftDirectPin2 8     ///Left Motor direction pin 1 to MODEL-X IN4
#define LPT 2                // scan loop coumter

#define SERVO_PIN 9  //servo connect to D9

#define Echo_PIN 2   // Ultrasonic Echo pin connect to D11
#define Trig_PIN 10  // Ultrasonic Trig pin connect to D12
```

```
#define BUZZ_PIN 13
#define FAST_SPEED 250  //both sides of the motor speed
#define SPEED 120       //both sides of the motor speed
#define SLOW_SPEED 60   //both sides of the motor speed
#define TURN_SPEED 200  //both sides of the motor speed
#define WALL_SPEED 150  //Wall speed

int leftscanval, centerscanval, rightscanval, ldiagonalscanval, rdiagonalscanval;
const int distancelimit = 10;      //distance limit for obstacles in front
const int sidedistancelimit = 15;  //minimum distance in cm to obstacles at both
sides (the car will allow a shorter distance sideways)
const int uppersidedistancelim = 20;
const int lowersidedistancelim = 15;

int distance;
int numcycles = 0;
const int turntime = 250;  //Time the robot spends turning (miliseconds)
const int backtime = 300;  //Time the robot spends turning (miliseconds)

int thereis;
Servo head;
//motor control/
void go_Advance(void)  //Forward
{
  digitalWrite(RightDirectPin1, LOW);
  digitalWrite(RightDirectPin2, HIGH);
  digitalWrite(LeftDirectPin1, LOW);
  digitalWrite(LeftDirectPin2, HIGH);
}
void go_Left()  //Turn left
{
  digitalWrite(RightDirectPin1, HIGH);
  digitalWrite(RightDirectPin2, LOW);
  digitalWrite(LeftDirectPin1, LOW);
  digitalWrite(LeftDirectPin2, HIGH);
}
void go_Right()  //Turn right
{
  digitalWrite(RightDirectPin1, LOW);
  digitalWrite(RightDirectPin2, HIGH);
  digitalWrite(LeftDirectPin1, HIGH);
  digitalWrite(LeftDirectPin2, LOW);
}
void go_Back()  //Reverse
{
  digitalWrite(RightDirectPin1, HIGH);
  digitalWrite(RightDirectPin2, LOW);
  digitalWrite(LeftDirectPin1, HIGH);
  digitalWrite(LeftDirectPin2, LOW);
}
void stop_Stop()  //Stop
```

```
{
  digitalWrite(RightDirectPin1, LOW);
  digitalWrite(RightDirectPin2, LOW);
  digitalWrite(LeftDirectPin1, LOW);
  digitalWrite(LeftDirectPin2, LOW);
  set_Motorspeed(0, 0);
}

/*set motor speed */
void set_Motorspeed(int speed_L, int speed_R) {
  analogWrite(speedPinL, speed_L);
  analogWrite(speedPinR, speed_R);
}

void buzz_ON()  //open buzzer
{

  for (int i = 0; i < 100; i++) {
    digitalWrite(BUZZ_PIN, LOW);
    delay(2);  //wait for 1ms
    digitalWrite(BUZZ_PIN, HIGH);
    delay(2);  //wait for 1ms
  }
}
void buzz_OFF()  //close buzzer
{
  digitalWrite(BUZZ_PIN, HIGH);
}
void alarm() {
  buzz_ON();

  buzz_OFF();
}

//detection of ultrasonic distance/
int watch() {
  long echo_distance;
  digitalWrite(Trig_PIN, LOW);
  delayMicroseconds(5);
  digitalWrite(Trig_PIN, HIGH);
  delayMicroseconds(15);
  digitalWrite(Trig_PIN, LOW);
  echo_distance = pulseIn(Echo_PIN, HIGH);
  echo_distance = echo_distance * 0.01657;  //how far away is the object in cm
  //Serial.println((int)echo_distance);
  return round(echo_distance);
}
//Meassures distances to the right, left, front, left diagonal, right diagonal and
asign them in cm to the variables rightscanval,
//leftscanval, centerscanval, ldiagonalscanval and rdiagonalscanval (there are 5
points for distance testing)
```

```
String watchsurrounding() {
  /*  obstacle_status is a binary integer, its last 5 digits stands for if there is
any obstacles in 5 directions,
 *   for example B101000 last 5 digits is 01000, which stands for Left front has
obstacle, B100111 means front, right front and right ha
 */

  int obstacle_status = B100000;
  centerscanval = watch();
  if (centerscanval < distancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B100;
  }
  head.write(110);
  delay(100);
  ldiagonalscanval = watch();
  if (ldiagonalscanval < distancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B1000;
  }
  head.write(170);  //Didn't use 180 degrees because my servo is not able to take
this angle
  delay(300);
  leftscanval = watch();
  if (leftscanval < sidedistancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B10000;
  }

  head.write(70);  //use 90 degrees if you are moving your servo through the whole
180 degrees
  delay(100);
  centerscanval = watch();
  if (centerscanval < distancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B100;
  }
  head.write(30);
  delay(100);
  rdiagonalscanval = watch();
  if (rdiagonalscanval < distancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B10;
  }
  head.write(0);
  delay(100);
```

```
  rightscanval = watch();
  if (rightscanval < sidedistancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | 1;
  }
  head.write(70);  //Finish looking around (look forward again)
  delay(300);
  String obstacle_str = String(obstacle_status, BIN);
  obstacle_str = obstacle_str.substring(1, 6);

  return obstacle_str;  //return 5-character string standing for 5 direction
obstacle status
}

String checkConsistency() {
  /*  obstacle_status is a binary integer, its last 5 digits stands for if there is
any obstacles in 5 directions,
 *   for example B101000 last 5 digits is 01000, which stands for Left front has
obstacle, B100111 means front, right front and right ha
 */
  int obstacle_status = B100000;
  centerscanval = watch();
  if (centerscanval < distancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B100;
  }
  head.write(110);
  delay(100);
  ldiagonalscanval = watch();
  if (ldiagonalscanval < distancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B1000;
  }
  head.write(170);  //Didn't use 180 degrees because my servo is not able to take
this angle
  delay(300);
  leftscanval = watch();
  if (leftscanval < sidedistancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B10000;
  }

  head.write(70);  //use 90 degrees if you are moving your servo through the whole
180 degrees
  delay(100);
  centerscanval = watch();
  if (centerscanval < distancelimit) {
```

```
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B100;
  }
  head.write(30);
  delay(100);
  rdiagonalscanval = watch();
  if (rdiagonalscanval < distancelimit) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B10;
  }
  head.write(0);
  delay(100);
  rightscanval = watch();
  if (lowersidedistancelim <= distancelimit <= uppersidedistancelim) {
    stop_Stop();
    alarm();
    obstacle_status = obstacle_status | B1;
  }
  head.write(70);  //Finish looking around (look forward again)
  delay(300);
  String obstacle_str = String(obstacle_status, BIN);
  obstacle_str = obstacle_str.substring(1, 6);

  return obstacle_str;  //return 5-character string standing for 5 direction
obstacle status
}

void wallfollowing_obstacle_avoidance() {

  ++numcycles;
  if (numcycles >= LPT) {  //Watch if something is around every LPT loops while
moving forward
    stop_Stop();
    String obstacle_sign = watchsurrounding();  // 5 digits of obstacle_sign binary
value means the 5 direction obstacle status
    Serial.print("begin str=");
    Serial.println(obstacle_sign);
    if (obstacle_sign == "10000") {
      Serial.println("Slight right");
      set_Motorspeed(SPEED, FAST_SPEED);
      go_Advance();
      delay(turntime);
      stop_Stop();
    } else if (obstacle_sign == "00001") {
      Serial.println("Go left");
      go_Left();  //Turn left
      set_Motorspeed(TURN_SPEED, TURN_SPEED);
      delay(turntime);
      stop_Stop();
```

```
    } else if (obstacle_sign == "11100" || obstacle_sign == "01000" ||
obstacle_sign == "11000" || obstacle_sign == "10100" || obstacle_sign == "01100" ||
obstacle_sign == "00100" || obstacle_sign == "01000") {
      Serial.println("Turn left");
      go_Left();  //Turn left
      set_Motorspeed(TURN_SPEED, TURN_SPEED);
      delay(turntime);
      stop_Stop();
    } else if (obstacle_sign == "00010" || obstacle_sign == "00111" ||
obstacle_sign == "00011" || obstacle_sign == "00101" || obstacle_sign == "00110" ||
obstacle_sign == "01010") {
      Serial.println("Turn robot left");
      go_Left();  //Turn left
      set_Motorspeed(TURN_SPEED, TURN_SPEED);
      delay(turntime);
      stop_Stop();
    } else if (obstacle_sign == "01111" || obstacle_sign == "10111") {
      Serial.println("Move left");
      go_Left();  //Turn left
      set_Motorspeed(TURN_SPEED, TURN_SPEED);
      delay(turntime);
      stop_Stop();
    } else if (obstacle_sign == "01111" || obstacle_sign == "10111") {
      Serial.println("Back up");
      go_Left();  //Turn left
      set_Motorspeed(TURN_SPEED, TURN_SPEED);
      delay(turntime);
      stop_Stop();
    } else if (obstacle_sign == "11011" || obstacle_sign == "11101" ||
obstacle_sign == "11110" || obstacle_sign == "01110") {
      Serial.println("Go right");
      go_Right();
      set_Motorspeed(SPEED, FAST_SPEED);
      delay(backtime);
      stop_Stop();
    } else {
      numcycles = 0;  //Restart count of cycles
      Serial.println("Go left to find wall");
      go_Left();
      set_Motorspeed(SPEED, SPEED);
      delay(backtime);
      go_Advance();
      set_Motorspeed(SPEED, SPEED);
      delay(backtime);
      stop_Stop();
    }
  } else {
    stop_Stop();
  }
```

```
  String obstacle_sign2 = checkConsistency();  // 5 digits of obstacle_sign binary
value means the 5 direction obstacle status
  if (obstacle_sign2 == "00001") {
    while (obstacle_sign2 == "00001") {
      Serial.println("FOLLOW THE WALL with short zig zag motion");
      go_Right();  //Turn right
      set_Motorspeed(WALL_SPEED, WALL_SPEED);
      go_Advance();
      set_Motorspeed(SLOW_SPEED, SLOW_SPEED);
      go_Left();  //Turn left
      set_Motorspeed(WALL_SPEED, WALL_SPEED);
      go_Advance();
      set_Motorspeed(SLOW_SPEED, SLOW_SPEED);
      String obstacle_sign2 = checkConsistency();  // 5 digits of obstacle_sign
binary value means the 5 direction obstacle status
    }
  }

  distance = watch();              // use the watch() function to see if anything
is ahead (when the robot is just moving forward and not looking around it will test
the distance in front)
  if (distance < distancelimit) {  // The robot will just stop if it is completely
sure there's an obstacle ahead (must test 25 times) (needed to ignore ultrasonic
sensor's false signals)
    Serial.println("final go back");
    go_Right();
    set_Motorspeed(SPEED, FAST_SPEED);
    delay(backtime * 3 / 2);
    ++thereis;
  }
  if (distance > distancelimit) {
    thereis = 0;
  }  //Count is restarted
  if (thereis > 25) {
    Serial.println("final stop");
    stop_Stop();  // Since something is ahead, stop moving.
    thereis = 0;
  }
}

void setup() {
  //setup L298N pin mode/
  pinMode(RightDirectPin1, OUTPUT);
  pinMode(RightDirectPin2, OUTPUT);
  pinMode(speedPinL, OUTPUT);
  pinMode(LeftDirectPin1, OUTPUT);
  pinMode(LeftDirectPin2, OUTPUT);
  pinMode(speedPinR, OUTPUT);
  stop_Stop();  //stop move
  //init HC-SR04/
  pinMode(Trig_PIN, OUTPUT);
```

```
  pinMode(Echo_PIN, INPUT);
  //init buzzer/
  pinMode(BUZZ_PIN, OUTPUT);
  digitalWrite(BUZZ_PIN, HIGH);
  buzz_OFF();

  digitalWrite(Trig_PIN, LOW);
  //init servo/
  head.attach(SERVO_PIN);
  head.write(90);
  delay(2000);

  Serial.begin(9600);
}

void loop() {
  wallfollowing_obstacle_avoidance();
}
```

Fig 11. The code for wall following with obstacle avoidance. Uses aspects of Osoyoo obstacle avoidance code (see report reference 6)

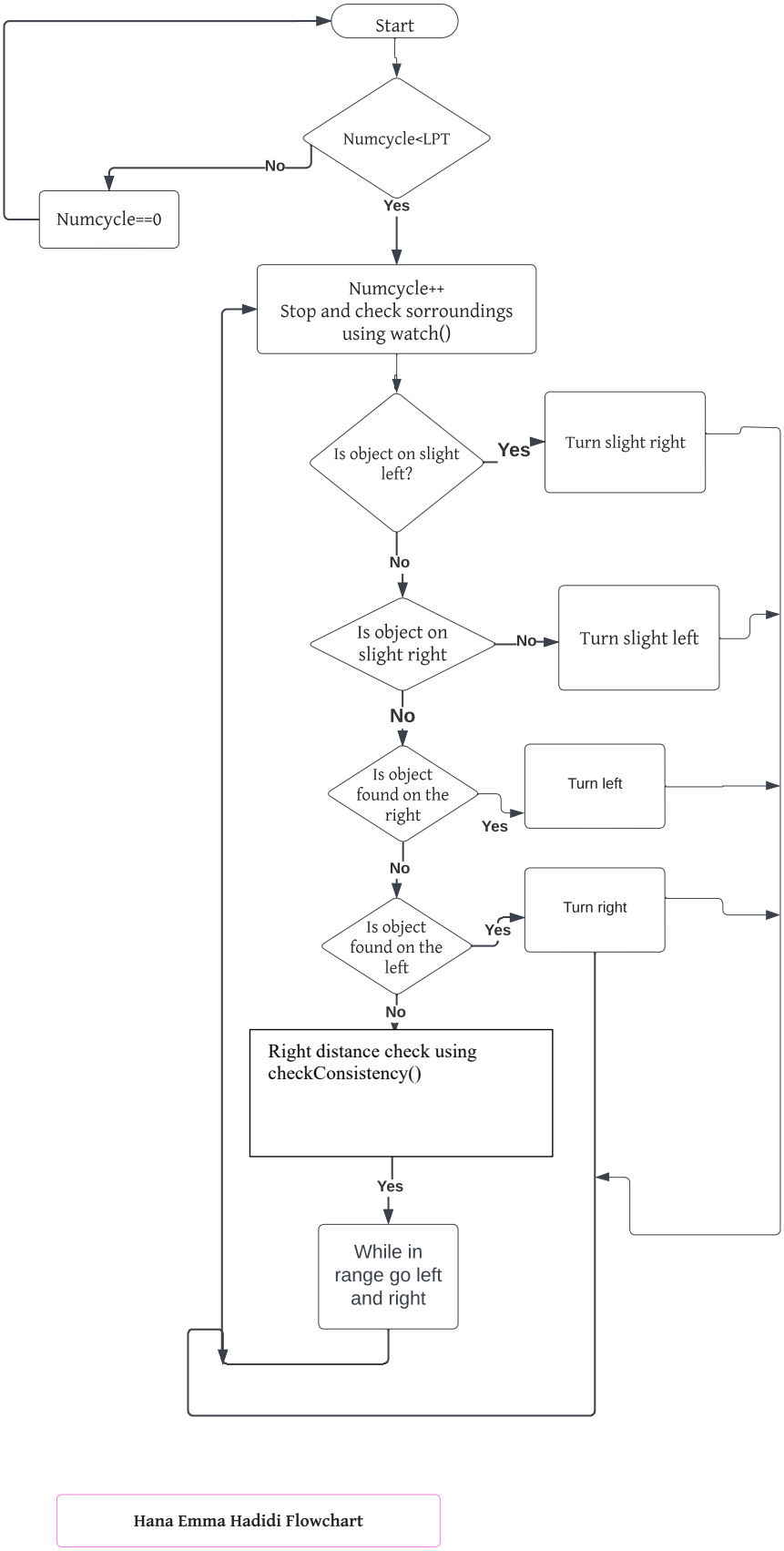

Fig 12. Flowchart for wall following with obstacle avoidance.