EMS516U – Introduction to Robotics

Robotics Lab Report

Name:	Student ID:
JATINDER DHALIWAL	220451019

Table of Contents

Introduction

The application of robots has evolved immensely over the last few years. There are now many uses for robots and their technology in any field of work, alongside being available to people for everyday use. Some examples of robots include – robot waiters in restaurants, automatic robot vacuum cleaners, and drones which can scan a whole area and plan a specific path for themselves [1]. These are only a few of an extensive range of robots which has had a significant impact on industry and everyday life.

The primary aim of this lab is to use an Arduino microcontroller and software to compute algorithms for the OSOYOO Robot Car to complete various functions, including ultrasonic sensing for distance measuring, wall following, obstacle avoidance, and line following [2]. The method to achieve this is to first assemble the car using the various parts provided and write code in the Arduino program, which can then be inputted into the OSOYOO Robot Car to complete a specific function. Specific parameters such as the speed of the car, calibration and distance measuring of the ultrasonic sensors and the delay of the car must be optimised to allow the car to complete specific functions. For instance, if the speed of the car is set too high in the code whilst following a line, there may be a turn which is too sharp for the car to make, causing it to divert from the intended path and lose track of the line completely. If the distance for ultrasonic measuring is too small, the robot may be unable to turn away from an obstacle in its path without colliding, which would disregard the original purpose. A sufficient delay time is required for the car to process its next step and perform a subsequent function, whilst adapting to its environment.

Robot Assembly

In the assembly stage of the lab, the initial step was connecting all of the hardware provided before computing any code into Arduino. The main components of the car were the OSOYOO Basic Board, the OSOYOO Uart Wi-Fi Shield, the OSOYOO Model X Motor Driver Module, a voltage meter, wheels, gear motors for each wheel and a battery pack with batteries included. All of the hardware was assembled using various screws, nuts and wires. Once the OSOYOO Robot Car was set up, a sample code from the OSOYOO website was copied into the Arduino microcontroller and loaded into the board to test the functionality of the car and to ensure that all of the components were linked correctly [3]. The setup for the robot car can be seen below in Figure 1.

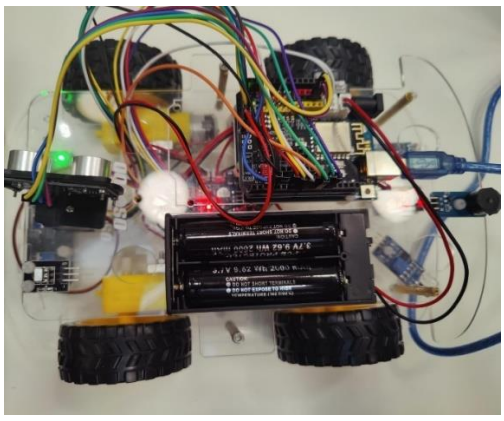

Figure 1 – Hardware Setup for the OSOYOO Robot Car

The next stage of the experiment was to implement an IR remote control system into the robot car. This was achieved using an IR receiver and an IR remote controller. As done previously, the IR receiver was assembled to the car using screws and nuts and connected to the Wi-Fi Shield using wires. The following step was to download the corresponding code for IR remote control from the OSOYOO website and load it onto the board. The code provided was relatively basic, allowing four simple movements via the IR controller – going forwards and backwards and turning left and right. The remote controller had to be aligned with the IR receiver for the robot car to carry out the instructions and within a certain distance for the receiver to pick up the signal [4]. The IR receiver and the IR remote controller are displayed in Figures 2 and 3, respectively.

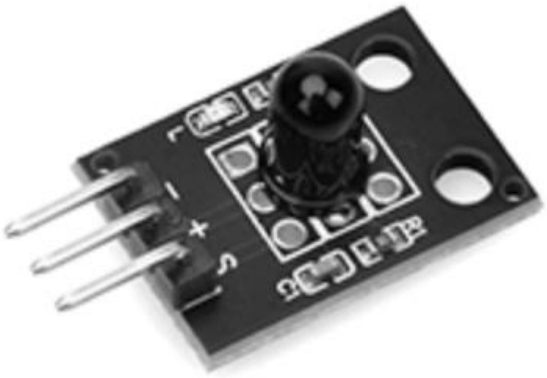

Figure 2 – IR Receiver

Figure 3 – IR Remote Controller

For the object following stage of the car assembly, the primary component was the IR distance sensors. These were connected to the back of the car to move the car forward, left or right, depending on where the object was. The sensors were connected to the Wi-Fi Shield and the code for object following was computed on the board through the Arduino microcontroller. The code was written so that any object greater than 10 cm away from the car would not signal the car to move towards it [5]. An image of the IR distance sensor is seen below in Figure 4.

Figure 4 – IR Distance Sensor

The line tracking for the OSOYOO Robot Car involved a tracking sensor module connected to the front, lower chassis of the car, to detect a line below it and be able to follow a track. Once the module was connected, the code for line following was computed into the robot car through Arduino. The sensitivity was adjusted to the best status, determined using the LED light, which would turn on when the robot recognised the line below it. The test was done on white ground and a track made of black tape with a width of 20-30 mm for a clear contrast so the sensor could detect the line. The track was made to ensure that no bend angle was greater than 90º. Otherwise, the car would have moved off the track [6]. The tracking sensor module can be seen in Figure 5 below.

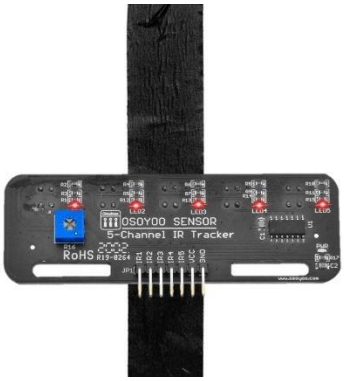

Figure 5 – Tracking Sensor Module on the track

The final stage of the assembly was to apply the obstacle avoidance feature to the OSOYOO Robot Car, which required a few key components, such as the ultrasonic sensor, a servo motor, a mount holder for the ultrasonic sensor and a buzzer sensor module. These components were then connected to the robot car, and the code from the Arduino software was loaded. To test obstacle avoidance, the ultrasonic sensor had to face directly forward before the robot started moving. A few adjustments were made to the servo to meet this requirement. The algorithm was computed so that the robot car would move forward if there was no object in the way, and if there was an object, the car would stop, and the servo head would turn left and right to determine which way to move the car to avoid the object [7]. The ultrasonic sensor was then calibrated to correct the measured distance by using measuring tape. The experiment for this can be seen below in Figure 6.

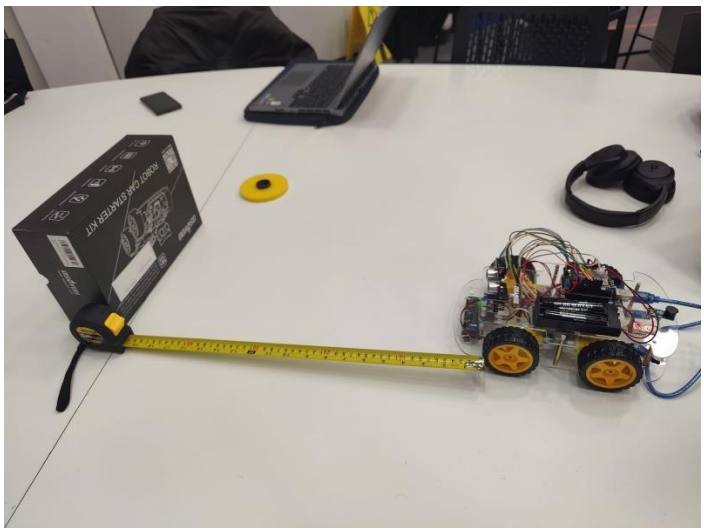

Figure 6 – Calibration of the Ultrasonic Sensor Distance

Robot Programming

Line Following Algorithm

The complete code for the line following can be found in Appendix 1. The five sensors on the tracking sensor module were used to signal if the ground below the car was black (line) or white (ground). The digitalRead() function was used for each line following sensor to return binary values, with 1 corresponding to black and 0 corresponding to white. The

read_sensor_values() function converted the sensor data into a binary string. The portion of the code using the digitalRead() function can be seen below in Figure 7.

```
char sensor[5];
/*read sensor value string, 1 stands for black, 0 stands for white, i.e 10
String read_sensor_values()
{   int sensorvalue=32;
  sensor[0]= !digitalRead(LFSensor_0);

  sensor[1]=!digitalRead(LFSensor_1);

  sensor[2]=!digitalRead(LFSensor_2);

  sensor[3]=!digitalRead(LFSensor_3);

  sensor[4]=!digitalRead(LFSensor_4);
  sensorvalue +=sensor[0]*16+sensor[1]*8+sensor[2]*4+sensor[3]*2+sensor[4];
```

Figure 7 – Code which returns binary values for each sensor

Regarding the speed settings, three speed levels were predefined for the robot car – fast, medium and slow. The analogWrite() function used these speed values to control the car's movement by changing the pulse width modulation signals.

Finally, the auto_tracking() function allowed the robot to adjust its movement based on the values provided by the sensors. For example, if a line was detected on the right, it would turn left and vice versa.

Obstacle Avoidance Algorithm

The complete code for obstacle avoidance can be found in Appendix 1. For the OSOYOO Robot Car to avoid an obstacle, the distance between the object and the car was measured using the watch() function, which utilised the ultrasonic sensor. To initially locate the obstacle, the watchsurrounding() function was implemented into the code, which rotated the servo in five directions – left, left-diagonal, front, right-diagonal and right, and the distance in each direction was measured. Similar to the line following code, a binary string was created, which indicated where the obstacle was relative to the robot car.

For this application, the speed was controlled by the set_motorspeed() function, where the left and right motor speeds were individually controlled to turn the car left or right, depending on the car's position relative to the obstacle.

The specific code for avoidance included the auto_avoidance() function, which analysed the binary string to determine the robot car's response. A threshold for the front and side distances was determined with the distancelimit() and sidedistancelimit() functions. For example, the car was programmed to turn left in response to an object on its right at a distance below the threshold for the ultrasonic sensor, and vice versa. The section of code for this is displayed in Figure 8.

```
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();
```

Figure 8 – Code for the distance threshold in each direction

A function for the sounding of the buzzer, alarm(), was also added to the code to activate in response to detecting an obstacle. However, this function was not used once it was tested.

To conclude the obstacle avoidance code, there was an implementation where the robot car would prioritise a path without obstacles by turning towards the clear path in response to an obstacle on the other path. If both paths contained obstacles, the robot car was programmed to reverse and reevaluate its surroundings before making another movement.

Ultrasonic Sensor Distance Measurement

The complete code for the ultrasonic sensor calibration can be found in Appendix 1. The ultrasonic sensor on the OSOYOO Robot Car required calibration to increase the accuracy of the measured distance, as the original distances determined by the sensor had errors of up to 5 cm, which was relatively large. Using the results of the uncalibrated distances given by the ultrasonic sensor, an equation was made to calibrate and correct the distances. The main section of this code was the getDistance() function, which calculated the distance by assessing the time it takes for a sound wave to hit the object and return. Therefore, an equation was made by using the speed of sound and halving it, as the sound wave travelled to the object and then back to the sensor. This portion of the code can be seen below in Figure 9.

```
int getDistance() {
  long echoDuration;

  // Emit Pulse
  digitalWrite(Trigger, LOW);
  delayMicroseconds(5);
  digitalWrite(Trigger, HIGH);
  delayMicroseconds(15);
  digitalWrite(Trigger, LOW);

  echoDuration = pulseIn(Echo, HIGH);
  int distance = echoDuration * 0.034 / 2;

  return distance;
}
```

Figure 9 – Code for the ultrasonic sensor distance measurement

Wall Following Algorithm

The complete code for wall following can be found in Appendix 1. In this code, the desired distance was set to 10 cm from the wall, allowing a deviation of ± 5 cm. There was a threshold in that if the robot was more than 30 cm away from the wall, the signal for a wall would be lost. This code assumed that the wall was to the left of the robot; hence, the servo head turned left to measure the distance from the wall. If the distance was too close (<10 cm), the robot car would adjust to the right to the desired distance and would adjust to the left if the distance was too far (>10cm). If the distance of the wall was greater than 30 cm, the robot was programmed to turn left to search for the wall again and maintain the desired distance away from the wall. Anything behind the robot car was not considered, so the only movements of the robot included going forward when the desired distance was reached and adjusting left and right with the set_motorspeed function to maintain the desired distance. The main part of the wall following code can be seen in Figure 10.

```
void wall_following() {
 int distanceLeft;  // Distance measured to the left
 // Look left to measure distance to the object
 head.write(170);  // Servo points left
 delay(75);      // Allow time for the servo to stabilize
 distanceLeft = watch();
 // Check if the object is still within range
 if (distanceLeft > lostThreshold) {
   go_Left();     // Turn left to search for the object
   delay(300);      // Adjust timing for smoother turns
   return;          // Skip the rest of the loop logic
 }
 // Check if the robot is too close, too far, or at the desired distance
 if (distanceLeft < (desiredDistance - distanceTolerance)) {
   // Too close, move slightly right
   go_Advance();
   set_Motorspeed(TURN_SPEED, FORWARD_SPEED);
   delay(250);  // Adjust delay to correct position
 } else if (distanceLeft > (desiredDistance + distanceTolerance)) {
   // Too far, move slightly left
   go_Advance();
   set_Motorspeed(FORWARD_SPEED, TURN_SPEED);
   delay(250);  // Adjust delay to correct position
 } else  {
   // At the correct distance, move forward
   go_Advance();
 }
 delay(100);  // Small delay for stability
}
```

Figure 10 – Code for the wall following algorithm

Test Procedure and Experimental Results

Line Following

To test the line following algorithm, a simple track was made for the OSOYOO Robot Car to follow, which was performed with little difficulty. At some points on the track, the robot car would veer away, possibly due to the angle of the bend being too high. The track was adjusted at these points, and the robot car followed the track as intended.

Obstacle Avoidance

When testing the obstacle avoidance algorithm, a hand was placed in front of the ultrasonic sensor to observe the response. As expected, the robot car reversed and turned away from the hand to continue moving. Next, the response to an obstacle to the side of the robot car was

tested by placing a hand on either side of the car. The ultrasonic sensor was able to detect the hand and turn away from it on both sides. Therefore, the test for the obstacle avoidance of the OSOYOO Robot Car was successful.

Ultrasonic Sensor Calibration

The setup shown in Figure 6 was ultimately used to measure both distance measurements of the ultrasonic sensor. Using a measuring tape, the original, uncalibrated distances recorded by the sensor were taken from 10-150 cm, increasing the distance by 20 cm at each interval. A graph of distance measured against distance expected was plotted using this data, which is displayed in Figure 11 below.

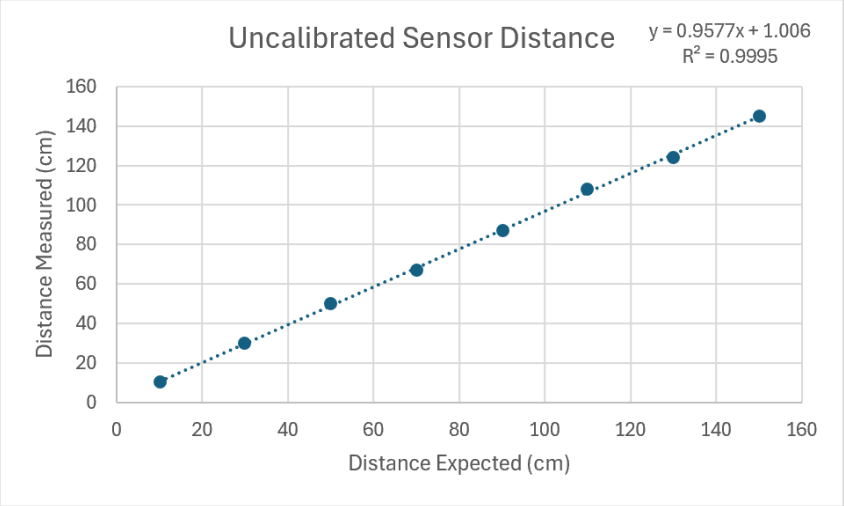

Figure 11 – Graph of uncalibrated sensor distance

Using the R^2 value and the gradient of the graph, an equation was made to calibrate the distances the ultrasonic sensor provided. The calibrated data was plotted into another graph, shown below in Figure 12.

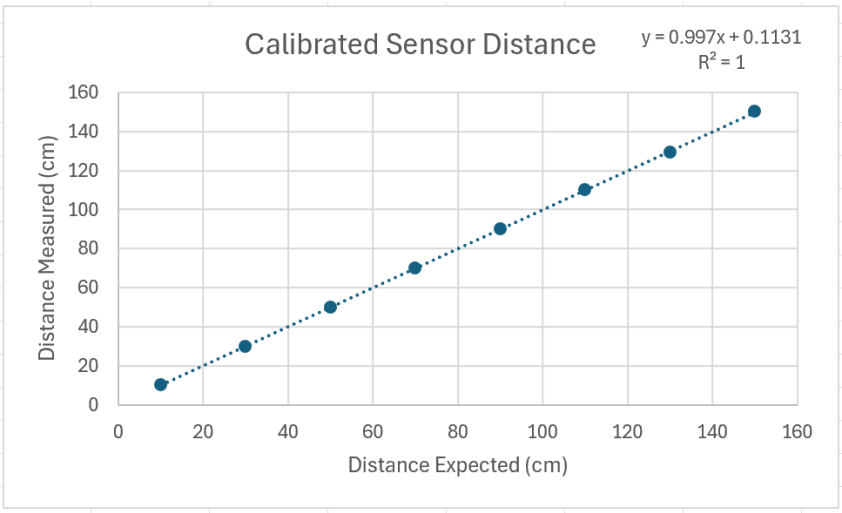

Figure 12 – Graph of calibrated sensor distance

This graph indicates that the calibration was successful due to the R^2 value of 1, as opposed to the previous R^2 value of 0.9995.

Wall Following

For the wall following algorithm, the robot car was placed at a distance that was barely outside the threshold for the signal of a wall in the area (30 cm) and was switched on to observe its movement. The robot car could locate the wall and reach the desired distance of 10 cm from the wall before turning parallel to the wall and following it. The robot adjusted side-to-side for some of its path because it was programmed to do so if it was too close or far from the wall. As a result, the wall following algorithm was coded and carried out successfully.

Bug 2 Algorithm

Testing the Bug 2 algorithm involved combining each code into one complete code, which would allow the car to follow a line, come across an obstacle in its path, and have the ability to turn and follow the obstacle's path through to the other side, where the track continued. The robot car was able to follow the line as intended. However, once it reached the obstacle, it turned 180^{o} and started following the line in the other direction rather than travelling around the obstacle using the wall following algorithm. The code was rechecked and reloaded into the robot car and was tested again. This time, the obstacle was not detected by the ultrasonic sensor, resulting in a collision with the obstacle. The reason for these outcomes may have been that the loop in the code crashed at some point, meaning that the code could not execute the desired outcome.

Discussion

Links to Lecture Material

The lecture material mentioned the two types of bug algorithms – Bug 1 and Bug 2. The Bug 1 algorithm involves a robot detecting an obstacle in its path and tracing its perimeter, finding the point closest to the goal destination. If another obstacle is in its path towards the goal, it will repeat the same process until there is a straight line towards the end destination. The Bug 2 algorithm involves what was tested in the lab. A robot is designated a goal destination and follows a line that leads to the goal from the start position. If an obstacle is encountered, it will travel around the perimeter until it reaches the line on the other side of the obstacle, closer to the goal, where it continues to follow the line until it reaches its destination. Both bugs use a threshold-based system, where a range of values is inputted into the microcontroller to allow the robot to function. Though this is an acceptable way to code the robot, there may be more precise and accurate methods that could be implemented, such as utilising memory to recognise the shortest path to the goal and store it or using a global path planning approach, where the robot scans the entire environment and determines the most efficient path to take to the destination [7].

Limitations of the Lab Experiment

A limitation found in the line following algorithm was its inability to follow a line if the bend angle was greater than 90^{0}. This may have been due to the limited number of sensors on the tracking sensor module, meaning that a signal could not be picked up for the line. A method to amend this setback could be to implement a module that has more sensors to account for more orientations of the line.

A minor challenge with the obstacle avoidance algorithm was that there was no specific code for if an obstacle was behind the robot car. While this situation is unlikely, there may be a chance that the car reverses to avoid an obstacle in front of it but ends up colliding with an obstacle behind it, with no way to detect it. This could be fixed by simply adding a sensor on the rear end of the car and coding it to detect an obstacle behind it.

Regarding the ultrasonic sensor calibration, the only error in the experiment would have been a parallax error when setting up the measuring tape, where the person measuring may not have had the exact distance. This could be fixed by simply implementing digital equipment, such as laser sensors, to measure the distance instead.

The wall following algorithm itself didn't have any issues. However, when combined with the rest of the algorithms to create Bug 2, it did not function correctly, potentially due to the loop in the code crashing, as mentioned previously. To fix this, debugging tools could be used to identify the problem with the loop and solve it accordingly.

Conclusion

In conclusion, this experiment successfully created algorithms for the OSOYOO Robot Car for line following, obstacle avoidance, ultrasonic distance sensing and wall following individually. However, the Bug 2 algorithm could not be implemented into the system. As a consequence of this, more effective integration of techniques that increase the precision and accuracy of the sensors, such as global path planning and utilising memory, could be applied in the future to optimise robots in general. The OSOYOO Robot Car allowed for a simple yet beneficial understanding of the components of robots and how they are typically assembled, the different responses to stimuli in an environment, and the use of various methods of computing algorithms in a robot to perform a specific function. By adapting these strategies into standard practice, the robotics industry will inevitably continue to make progress.

References

[1] Lam, C. (2022). *6 Examples of Robots in everyday life and its benefits to humans*. [online] InApps Technology. Available at: https://www.inapps.net/examples-of-robots-in-everyday-life/.

[2] osoyoo.com. (n.d.). *OSOYOO V2.1 Robot Car Kit for Arduino: Introduction Model#2019012400 «osoyoo.com*. [online] Available at: https://osoyoo.com/2020/05/12/v2-1-robot-car-kit-for-arduino-tutorial-introduction/.

[3] osoyoo.com. (n.d.). *OSOYOO V2.1 Robot car kit Lesson 1: Basic Robot car «osoyoo.com*. [online] Available at: https://osoyoo.com/2020/05/12/osoyoo-v2-1-robot-car-kit-lesson-1-basic-robot-car/.

[4] osoyoo.com. (n.d.). *OSOYOO V2.1 Robot car kit Lesson 2: IR Remote Control Robot Car «osoyoo.com*. [online] Available at: https://osoyoo.com/2020/05/12/osoyoo-v2-1-robot-car-kit-lesson-2-ir-remote-control-robot-car/.

[5] osoyoo.com. (n.d.). *OSOYOO V2.1 Robot car kit Lesson 3: Object follow Robot car «osoyoo.com*. [online] Available at: https://osoyoo.com/2020/05/12/osoyoo-v2-1-robot-car-kit-lesson-3-object-follow-robot-car/.

[6] osoyoo.com. (n.d.). *OSOYOO V2.1 Robot car kit Lesson 4: Tracking Line Robot Car «osoyoo.com*. [online] Available at: https://osoyoo.com/2020/05/12/osoyoo-v2-1-robot-car-kit-lesson-4-tracking-line-robot-car/.

[7] osoyoo.com. (n.d.). *OSOYOO V2.1 Robot car kit Lesson 5: Obstacle Avoidance Robot Car «osoyoo.com*. [online] Available at: https://osoyoo.com/2020/05/12/osoyoo-v2-1-robot-car-kit-lesson-5-obstacle-avoidance-robot-car/.

[8] Qmul.ac.uk. (2024). *EMS516U - 2024/25 | MyQMUL*. [online] Available at: https://qmplus.qmul.ac.uk/pluginfile.php/4208854/mod_label/intro/Wk3%20-%20EMS516U.pdf?time=1728629618915.

Appendix

```
#include <Servo.h>
/*From left to right, connect to D3,A1-A3 ,D10*/
#define LFSensor_0 A0  //OLD D3
#define LFSensor_1 A1
#define LFSensor_2 A2
#define LFSensor_3 A3
#define LFSensor_4 A4  //OLD D10

#define FAST_SPEED 140
#define MID_SPEED 130
#define SLOW_SPEED  120     //back speed
#define speedPinR 6    //  RIGHT PWM pin connect MODEL-X ENA
#define RightMotorDirPin1  12    //Right Motor direction pin 1 to MODEL-X IN1
#define RightMotorDirPin2  11    //Right Motor direction pin 2 to MODEL-X IN2
#define speedPinL 3    // Left PWM pin connect MODEL-X ENB
#define LeftMotorDirPin1  7    //Left Motor direction pin 1 to MODEL-X IN3
#define LeftMotorDirPin2  8   //Left Motor direction pin 1 to MODEL-X IN4

#define SERVO_PIN 9
#define Echo_PIN 2
#define Trig_PIN 10

const int desiredDistance = 20; // Desired distance from the wall in cm
const int distanceTolerance = 10; // Allowable deviation from the wall distance
const int lostThreshold = 30; // Obstacle avoidance distance in front


Servo head;  // Servo to look around
int distanceSide, distanceFront;

/*motor control*/
void go_Advance(void)  //Forward
{
  digitalWrite(RightMotorDirPin1, HIGH);
  digitalWrite(RightMotorDirPin2,LOW);
  digitalWrite(LeftMotorDirPin1,HIGH);
  digitalWrite(LeftMotorDirPin2,LOW);
  analogWrite(speedPinL,200);
  analogWrite(speedPinR,200);
}
void go_Right(int t=0)  //Turn right
{
  digitalWrite(RightMotorDirPin1, HIGH);
  digitalWrite(RightMotorDirPin2,LOW);
  digitalWrite(LeftMotorDirPin1,LOW);
  digitalWrite(LeftMotorDirPin2,HIGH);
  analogWrite(speedPinL,200);
  analogWrite(speedPinR,200);
  delay(t);
}
void go_Left(int t=0)  //Turn left
{
  digitalWrite(RightMotorDirPin1, LOW);
  digitalWrite(RightMotorDirPin2,HIGH);
  digitalWrite(LeftMotorDirPin1,HIGH);
  digitalWrite(LeftMotorDirPin2,LOW);
  analogWrite(speedPinL,200);
  analogWrite(speedPinR,200);
  delay(t);
}
void go_Back(int t=0)  //Reverse
{
  digitalWrite(RightMotorDirPin1, LOW);
  digitalWrite(RightMotorDirPin2,HIGH);
  digitalWrite(LeftMotorDirPin1,LOW);
  digitalWrite(LeftMotorDirPin2,HIGH);
  analogWrite(speedPinL,200);
  analogWrite(speedPinR,200);
  delay(t);
}
```

```
void stop_Stop()    //Stop
{
  digitalWrite(RightMotorDirPin1, LOW);
  digitalWrite(RightMotorDirPin2,LOW);
  digitalWrite(LeftMotorDirPin1,LOW);
  digitalWrite(LeftMotorDirPin2,LOW);
}
/*set motor speed */
void set_Motorspeed(int speed_L,int speed_R)
{
  analogWrite(speedPinL,speed_L);
  analogWrite(speedPinR,speed_R);
}

// Measure distance using the ultrasonic sensor
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; // Convert time to cm
 return round(echo_distance);
}


void wall_following() {


 int distanceLeft;  // Distance measured to the left


 // Look left to measure distance to the object
 head.write(170);  // Servo points left
 delay(75);      // Allow time for the servo to stabilize
```

```
 distanceLeft = watch();


 // Check if the object is still within range
 if (distanceLeft > lostThreshold) {
   go_Left();     // Turn left to search for the object
   delay(300);      // Adjust timing for smoother turns
   return;          // Skip the rest of the loop logic
 }
 // Check if the robot is too close, too far, or at the desired distance
 if (distanceLeft < (desiredDistance - distanceTolerance)) {
   // Too close, move slightly right
   go_Advance();
   set_Motorspeed(SLOW_SPEED, FAST_SPEED);
   delay(250);  // Adjust delay to correct position
 } else if (distanceLeft > (desiredDistance + distanceTolerance)) {
   // Too far, move slightly left
   go_Advance();
   set_Motorspeed(FAST_SPEED, SLOW_SPEED);
   delay(250);  // Adjust delay to correct position
 } else  {
   // At the correct distance, move forward
   go_Advance();
 }
 delay(100);  // Small delay for stability
}

void setup()
{
  pinMode(RightMotorDirPin1, OUTPUT);
  pinMode(RightMotorDirPin2, OUTPUT);
  pinMode(speedPinL, OUTPUT);

  pinMode(LeftMotorDirPin1, OUTPUT);
  pinMode(LeftMotorDirPin2, OUTPUT);
  pinMode(speedPinR, OUTPUT);
  stop_Stop();//stop move
```

```
  Serial.begin(9600);   // initialize serial for debugging

}

boolean flag=false;
void loop()
{
 int distance = watch();
 if (distance<desiredDistance){
  wall_following();
 }
 auto_tracking();
} //end of loop

char sensor[5];
 /*read sensor value string, 1 stands for black, 0 starnds for white, i.e 16
String read_sensor_values()
{   int sensorvalue=32;
  sensor[0]= !digitalRead(LFSensor_0);

  sensor[1]=!digitalRead(LFSensor_1);

  sensor[2]=!digitalRead(LFSensor_2);

  sensor[3]=!digitalRead(LFSensor_3);

  sensor[4]=!digitalRead(LFSensor_4);
  sensorvalue +=sensor[0]*16+sensor[1]*8+sensor[2]*4+sensor[3]*2+sensor[4];

  String senstr= String(sensorvalue,BIN);
  senstr=senstr.substring(1,6);


  return senstr;
}

void auto_tracking(){
String sensorval= read_sensor_values();
 Serial.println(sensorval);
 if (sensorval=="10000"  || sensorval=="11000")
 {
   //The black line is in the left of the car, need left turn
   go_Left();  //Turn left
   set_Motorspeed(FAST_SPEED,FAST_SPEED);
 }
 if (sensorval=="10100"  || sensorval=="01100" || sensorval=="11100"  || sensorval=="10010" || sensorval=="11010")
 {
   go_Advance();  //Turn slight left
   set_Motorspeed(0,MID_SPEED);
 }
 if (sensorval=="00001" || sensorval=="00011"){ //The black line is  on the right of the car, need right turn
   go_Right();  //Turn right
   set_Motorspeed(FAST_SPEED,FAST_SPEED);
 }
 if (sensorval=="00110" || sensorval=="00111" ||sensorval=="00101" || sensorval=="01011" || sensorval=="01101")
 {
   go_Advance();  //Turn slight right
   set_Motorspeed(MID_SPEED, 0);
 }

 if (sensorval=="01110" || sensorval=="11111" ||sensorval=="00100" || sensorval=="01010")
 {
   go_Advance();  //Go forward
   set_Motorspeed(FAST_SPEED,FAST_SPEED);
 }

 if (sensorval=="11111"){
   go_Advance();   //The car front touch stop line, need stop
   set_Motorspeed(SLOW_SPEED,SLOW_SPEED);
 }
}
```

Appendix 1 – Total completed code