ROB-GY 6103: Advanced Mechatronics
Arduino Project Report
Team 25

KaiYuan Ma
Ifedayo Olusanya
Justin Ramlall

Table Of Contents

Introduction:

This project is aimed at using an Arduino UNO microcontroller to create a robot that will be able to maneuver a parking lot and park in the first empty parking space that it detects. The purpose of this project is to test our understanding of the arduino UNO and its functions in tandem with commonly used mechatronic devices. Our robot uses the Arduino UNO along with eight infrared sensors, an ultrasonic sensor, two servos, two LEDs, and a LCD screen. We have been able to create a robot that is able to use line following and an ultrasonic sensor to determine the first available space that it can park in. The robot has been tested to complete all of the required tasks and we are confident in its performance. This report will cover the required tasks of the robot, the design of the mechatronic system, the testing procedures used to ensure proper functionality of the robot, and conclude with an analysis of the robots performance, as well as future improvements that can be made.

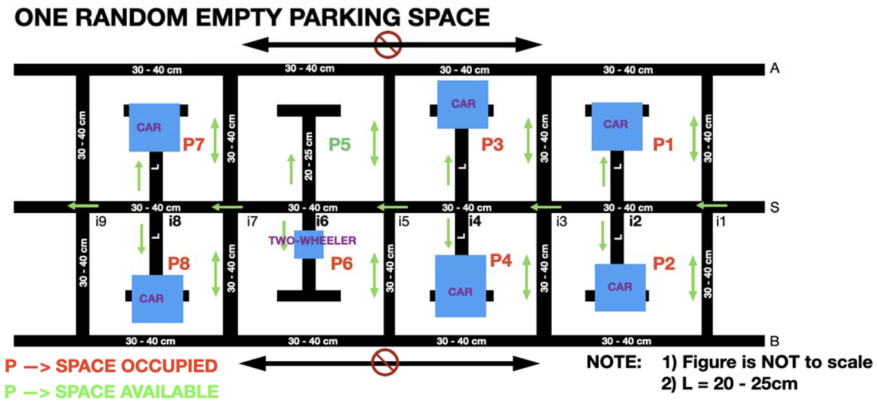

Figure 1: A diagram of the Parking Lot that the robot has been designed to traverse. Image courtesy of the ROB-GY 6103 instructional team.

Objectives:

There are two main requirements and two bonus objectives that are outlined for the robot to meet. We used these as a framework to determine the hardware that we would need to complete the objectives. The objectives of the robot are listed below. The robot must:

1. Be a line following robot, able to accurately follow the center and vertical lanes
2. Indicate when it traverses an intersection
3. Detect and indicate when there are object in a parking space
4. Park in the first available parking space
5. Display the amount of occupied spaces encountered before it parked

The two bonus objectives for the robot to complete are stated as:

1. Indicate if a space is occupied by a two wheeler or a four wheeler
2. Parking in a space that has a two wheeler outside of the “T” shaped line

In order to complete these objectives, we decided to use the robot chassis and servos used in the Boe-Bot Kit from Parallax along with a Pololu Reflectance Sensor Array, and Ultrasonic Sensor, and LCD screen. These sensors were chosen due to availability, ease of use, and they allow us to complete all of the five objectives.

System Design:

The mechatronic system that we have created can be separated into the mechanical, electrical, and arduino code designs. The mechanical design focuses on the individual hardware that was used to build the robot. It will also include any custom parts created to add to the chassis. The electrical design will include a circuit diagram that shows all electrical connections between the components. The arduino code will include a flowchart that explains the functionality of the code, while the code itself will be included in the appendix.

Mechanical Design

The main framework for the robot starts with the chassis provided by the Boe-Bot Kit from Parallax. From this we are using the chassis, the two Continuous Rotation Servo with the wheel attachments, and the ball attachment for the third rear wheel. From Parallax, we are also using the PING))) Ultrasonic Sensor (Figure 2), the Parallax 2x16 Serial LCD with Piezo Speaker (Figure 3), and a red and blue LEDs and from Pololu we are using the QTR-8RC Reflectance Sensor Array (Figure 4). In order to power the servos, we have decided to use a Breadboard Power Supply Module from Elegoo (Figure 5). All of these are controlled via an Arduino UNO Rev3 Board (Figure 6).

The Breadboard Power Supply Module is being used so that we can power the two servos on a different power supply than the rest of the sensors and the Arduino. This ensures that the servos do not pull too much current from the Arduino and there is no effect on the performance of the sensors. There was no indication that this would be a problem, however we decided to move forward with this choice anyway just to make sure that we would not encounter any issues. An important part of doing this is to ensure that the two circuits have a common ground just to help make sure that all current is flowing properly.

Figure 2: The PING))) sensor from parallax. It has 3 pins for 5V power supply, Ground, and SIG. SIG gets connected to an Arduino digital pin and is responsible for sending and receiving the ultrasonic signal.

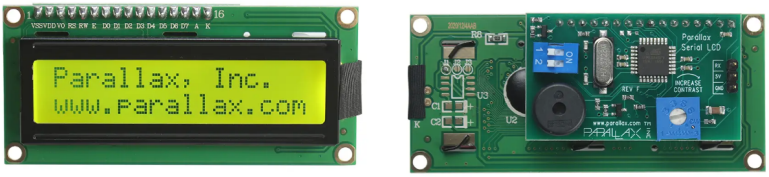

Figure 3: The Serial LCD from Parallax. It has two rows for displaying text. It has a rear header board with the (unused) piezo speaker. This allows us to communicate with it using just three pins: 5V power supply, Ground, and RX for receiving serial data.

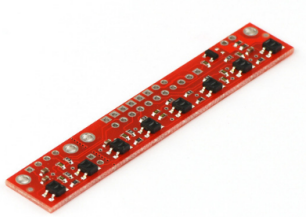

Figure 4: The Reflectance Sensor Array from Pololu. It has 8 Infrared Sensors and an(unused) LED for a light source. There is a pin for 5V power supply and Ground, and then all 8 IR sensors have their own pin as well.

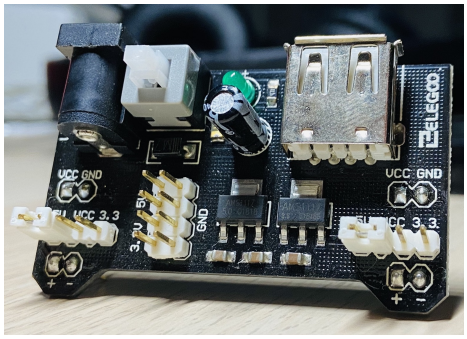

Figure 5: The Elegoo Breadboard Power Supply Module. It can supply any voltage from 12, 5, or 3.3 V. It can also be powered using a USB connector or DC connector. There is also a power switch and status LED. Pins underneath the board allow it to be connected directly to the power rails on a breadboard.

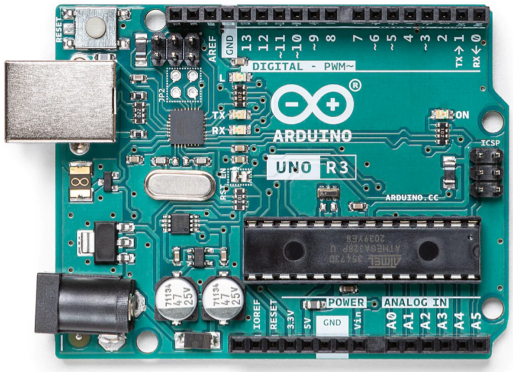

Figure 6: The Arduino UNO Board. The microcontroller in charge of everything in our system. All of the digital pins and 3 of the analog pins are used to control all of the hardware.

Electrical Design

Below we have our circuit diagram showing all of the electrical connections made between our parts. As mentioned before, the servos are powered using a different supply than the UNO. It is important that the two circuits share a common ground however to make sure that no current from one circuit flows into the other and also to make sure both circuits have the same ground reference. We are using all of the available digital IO pins as well as three of the analog pins on the UNO board. The reflectance sensor uses digital Pins 2-9, the servos use

Pins 10 and 11, and the PING))) sensor uses Pin 12. Pins 0 and 1 cannot be used since these are reserved for TX and RX in USART communication. Similarly, Pin 13 cannot be used because this is connected to the integrated LED on the UNO board. In order to use more pins, we are using analog Pins A0, A1, and A2 as digital outputs. A0 is used to blink the red LED that indicates an object is detected, A1 is used to blink the blue LED that indicates an intersection has been reached, and A2 is used to display messages on the LCD screen. Both of the LEDs are connected in series with a 220 Ohm resistor to help limit the current. Each circuit is powered with a 9V battery, and both the Power Supply Module and the UNO board bring this voltage down to a supply voltage of 5V.

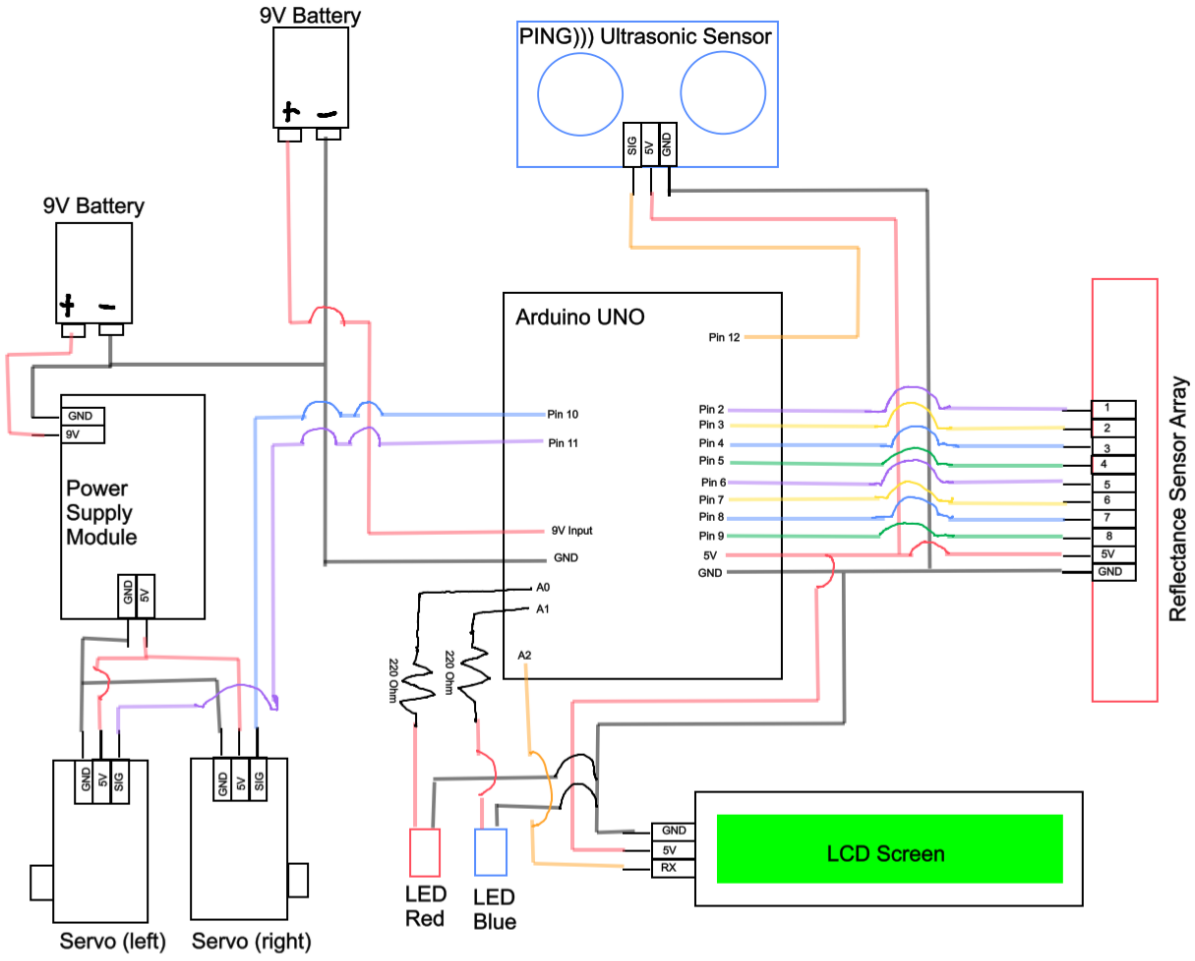

Figure 7: Circuit Diagram with all electrical connections. The reflectance sensor array takes up a total of 8 of the digital pins. The arduino featured in the center is in charge of controlling everything. We can see that the servos are powered by a separate circuit but still rely on the Arduino for instructions.

Arduino Code

The Arduino code starts off with a calibration so that the reflectance sensor is able to determine where the line to follow is at all times. From here, the robot is constantly looping between three modes; line following, main intersections, and border intersections. Referring back to Figure 1, the main intersections are i1, i3, i5, i7, and i9 and the border intersections are all of the ones along rows A and B. The robot will always be line following until it reaches an intersection. From there, it uses the reflectance sensor to determine what kind of intersection it is at, then it knows what to do based on different conditions. While all of this happens, the robot also uses the LEDs and the LCD to indicate where there are obstacles, when it reaches an intersection, and to let us know that it has parked and how many obstacles there were.

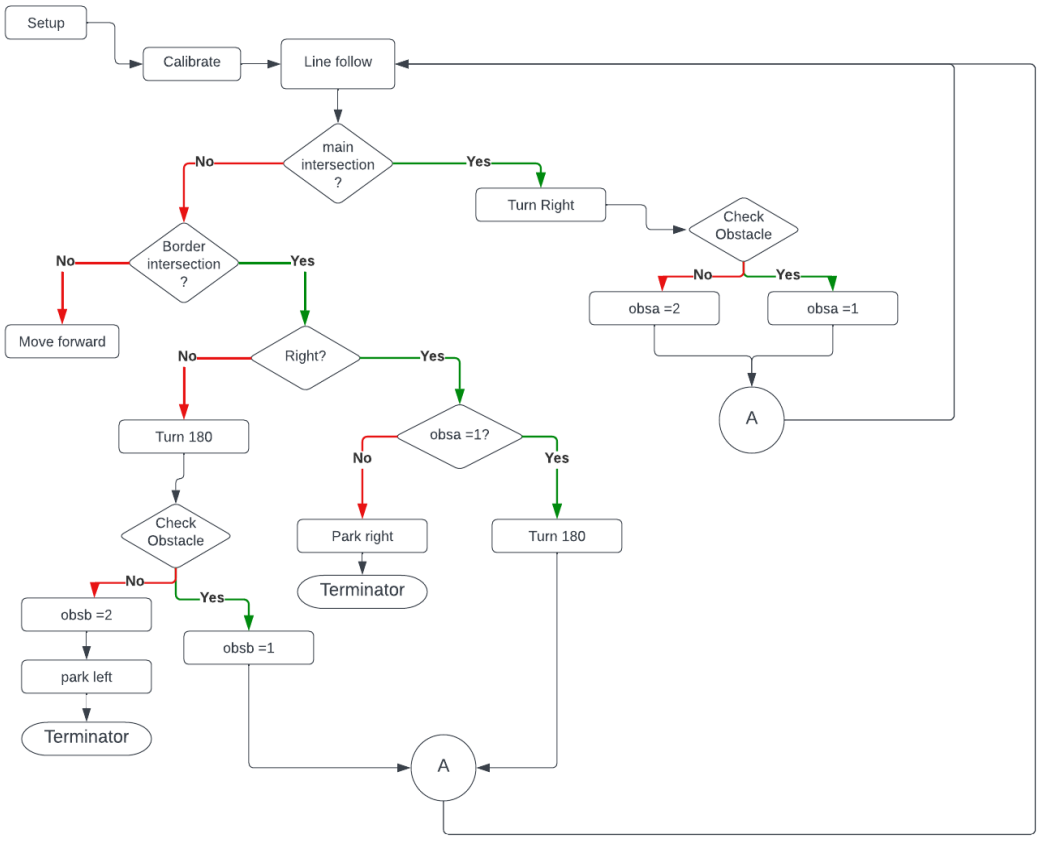

Figure 8: This flowchart details the control scheme behind our arduino code. The park left or right branches indicate where the code ends. The other ending branches all loop back to the first “Line Follow” box after calibration. While this flowchart does not include the indications of the LEDs or the LCD, it describes the physical behavior of the robot based on where on the map it is.

System Testing:

To test the system, we first broke the system up into parts and used an iterative method in order to ensure complete functionality of all of the components. We first started by assembling the chassis with the servos and wheels included with the Boe Bot Kit. We then connected the reflectance sensor and the Arduino UNO to the chassis and made the necessary electrical connections. We begin with testing the line following capabilities of the robot. First the reflectance sensor needs to be calibrated before we can begin. This is done by running the calibration code and having the robot move so that the reflectance sensor can calibrate the white and black value of the test field. Then, we determine the path that the robot will follow. This path is determined by the flow chart shown in Figure 8.

Once the line following was working, we then moved onto adding the ultrasonic sensor to the chassis using a custom bracket. To test the functionality of the ultrasonic sensor, the ultrasonic sensor would scan the parking spot as the robot drove past it. If it detected an object it would move to the next spot and scan that. If it did not detect anything, it would park. We built this part of the code by slowly going from spot to spot and making sure the robot would either park or not park based on where we placed the obstacles. This process also allowed us to refine the conditions for the line following. Once both of these were working, then we were halfway done with the objectives.

The next two objectives were simple to complete. These were to just indicate when the robot traversed an intersection and when an object was detected. This was done by adding a blue and red LED to the breadboard on the robot. We then added just a few more lines of code and tested again. This time we wanted to verify that the red LED would blink when the robot detected an object and that the blue LED would blink when the robot traversed an intersection. Once we ensured that this was consistent for the entire test area, we moved to the final objective. This was just to add an LCD screen to the robot in order to indicate when the robot was parking and how many occupied spaces it detected. After again making sure that this was working for the entire test field, our testing procedure was done.

Our entire testing procedures spanned about 12 hours total. By the end of it, the robot had demonstrated the ability to run the entire test course and park in every spot while showing the proper indications of obstacles and intersections.

Conclusion:

This project was aimed at using an Arduino UNO microcontroller to create a robot that will be able to maneuver a parking lot and park in the first empty parking space that it detects. The robot has been tested to complete all of the required tasks and we are confident in its performance. This was done by combining several different forms of hardware into a single

robotic package that is robust to perform in any configuration of available parking spaces indicated on the map in Figure 1. The Mechanical and Electrical Design coupled with the Arduino Code is the most effective use of available hardware and resulted in a system that is not too complex and able to meet all of the objectives that were laid out for our team.

Appendix:

```
//infrared sensor setup
#include <QTRSensors.h>                 //Include library to control the
QTRRC reflectance sensor
QTRSensors qtr;                         //set up reflectance sensor instance
const uint8_t SensorCount = 8;          // declare number of IRsensors
```

```
uint16_t sensorValues[SensorCount];  //declare array to hold sensor values

//ultrasonic sensor setup
#include <NewPing.h>  // Include Library for the ultrasonic Sensor
// Define pins for ultrasonic sensor
#define trigPin 12                              //set pin for ultrasonic
sensor trigger
#define echoPin 12                              //set pin for ultrasonic
sensor echo
int maxdistance = 20;                           // set maximum distance the
ultrasonic sensor can read to 20cm
NewPing sonar(trigPin, echoPin, maxdistance);  // Create NewPing object
with maximum distance of 20cm

//continious motor setup
#include <Servo.h>  //Include library to contol the 2 continious servo
motors
Servo myservo1;     //Setup first servo instance
Servo myservo2;     //Setup second servo instance
int motor1 = 11;    // define pin for left wheel
int motor2 = 10;    // define pin for right wheel

//lcd display setup
#include <SoftwareSerial.h>
SoftwareSerial lcd = SoftwareSerial(255, A2);

//general variables

int movecount;        // declare variable to control PWM of motors
int position;         // declare variable to hold the position of black
line
int maininter = 0;    // delare variable to count main intersections
int searchinter = 0;  //Vairable to count all intersections
int distanceA = 0;    // declare variable to determine how far an obstacle
is for all parking spots on the right
int distanceB = 0;    // declare variable to determine how far an obstacle
is for all parking spots on the left
int obs1a = 0;        // variable to hold parking spot 1 on the right
availability (0 - not checked, 1- checked & obstacle present, 2- checked &
no obstacle)
```

```
int obs1b = 0;         // variable to hold parking spot 1 on the left
availability (0 - not checked, 1- checked & obstacle present, 2- checked &
no obstacle)
int obs2a = 0;         // variable to hold parking spot 2 on the right
availability (0 - not checked, 1- checked & obstacle present, 2- checked &
no obstacle)
int obs2b = 0;         // variable to hold parking spot 2 on the left
availability (0 - not checked, 1- checked & obstacle present, 2- checked &
no obstacle)
int obs3a = 0;         // variable to hold parking spot 3 on the right
availability (0 - not checked, 1- checked & obstacle present, 2- checked &
no obstacle)
int obs3b = 0;         // variable to hold parking spot 3 on the left
availability (0 - not checked, 1- checked & obstacle present, 2- checked &
no obstacle)
int obs4a = 0;         // variable to hold parking spot 4 on the right
availability (0 - not checked, 1- checked & obstacle present, 2- checked &
no obstacle)
int obs4b = 0;         // variable to hold parking spot 4 on the left
availability (0 - not checked, 1- checked & obstacle present, 2- checked &
no obstacle)
int rightpark = 0;     // variable to control parking on the right
int leftpark = 0;      // variable to control parking on the left
int i = 0;             // varible for the for loop counter in the first odd
intersection
int j = 0;             // varible for the for loop counter in the second
odd intersection
int k = 0;             // varible for the for loop counter in the third odd
intersection
int l = 0;             // varible for the for loop counter in the forth odd
intersection
int totalobstacle;     // variable to count total number of obstacles seen
before parking


void setup() {
  // put your setup code here, to run once:
  //Reflectance Sensor Setup
  qtr.setTypeRC();
```

```
  qtr.setSensorPins((const uint8_t[]){ 2, 3, 4, 5, 6, 7, 8, 9 },
SensorCount);  //determine arduino pins for the reflectance sensor

  pinMode(A0, OUTPUT);      //set Red LED to output
  pinMode(A1, OUTPUT);      // set blue LED to output
  pinMode(A2, OUTPUT);      // set LCD to output
  digitalWrite(A2, HIGH);   // make the LCD pin high
  lcd.begin(9600);          // start serial communication with lcd
  lcd.write(12);            // clear lcd
  lcd.write(17);            // turn on back light
  Serial.begin(9600);       // begin serial monitoring with computer

  //attach servo motors
  myservo1.attach(motor1);  // attached left servo
  myservo2.attach(motor2);  // attach right servo


  //Calibration Sequence
  // spin left and callibrate sensor
  for (int i = 0; i < 15; i++) {
    movecount = 1;
    qtr.calibrate();
    spinleft();
    delay(20);
  }
  // spin right and callibrate sensor
  for (int i = 0; i < 30; i++) {
    movecount = 1;
    qtr.calibrate();
    spinright();
    delay(20);
  }
  //spinleft and callibrate sensors

  for (int i = 0; i < 15; i++) {
    movecount = 1;
    qtr.calibrate();
    spinleft();
    delay(20);
  }
```

```
  // turn left
  movecount = 10;
  turnleft();
}

void loop() {
  // put your main code here, to run repeatedly:
  position = qtr.readLineBlack(sensorValues);  // read position value

  // run this block of code whenever we are on the track and not at an
intersection. i.e whenever the first and last sensor are not on the black
line
  if ((sensorValues[0] < 500) && (sensorValues[7] < 500)) {
    do {
      //behaviour at each mainintersection (i.e odd intersection on the
map)
      switch (maininter) {
        case 0:  // when main intersection is zero, simply line follow
          linefollow(); //perform corrective measures to ensure robot is
centered on black line
          break;

        case 1:
          if (searchinter == 1) {

            distanceA += sonar.ping_cm(); // check if there is an obstacle
            linefollow(); //perform corrective measures to ensure robot is
centered on black line

            if (distanceA > 0) { // if there is an obstacle in the first
parking lot, make obs1a 1
              obstacleindication();// indicate an obstacle
              obs1a = 1;


            } else if (distanceA == 0) {// if there is an obstacle in the
first parking lot, make obs1a 2

              obs1a = 2;
```

```
            } else {
              linefollow(); //perform corrective measures to ensure robot
is centered on black line
            }
          } else if (searchinter == 4) {

            distanceB += sonar.ping_cm();
            linefollow(); //perform corrective measures to ensure robot is
centered on black line



            if (distanceB > 0) { // if there is an obstacle in the first
parking lot, make obs1b 1
              obstacleindication(); // indicate obstacle
              obs1b = 1;

            } else if (distanceB == 0) { // if there is an obstacle in the
first parking lot, make obs1b 2
              obs1b = 2;

            } else {

              linefollow(); //perform corrective measures to ensure robot
is centered on black line
            }

          } else {

            linefollow(); //perform corrective measures to ensure robot is
centered on black line
          }
          break;

        case 2:
          linefollow(); //perform corrective measures to ensure robot is
centered on black line
          distanceA = 0;  // take distance readings to zero
          distanceB = 0;  // take distance readings to zero
          break;
```

```
// perform similar sequence as case 1
case 3:
  if (searchinter == 7) {

    distanceA += sonar.ping_cm();
    linefollow();


    if (distanceA > 0) {
      obstacleindication();
      obs2a = 1;

    } else if (distanceA == 0) {
      obs2a = 2;

    } else {
      linefollow();
    }
  } else if (searchinter == 10) {


    distanceB += sonar.ping_cm();
    linefollow();

    if (distanceB > 0) {
      obstacleindication();
      obs2b = 1;

    } else if (distanceB == 0) {
      obs2b = 2;

    } else {

      linefollow();
    }

  } else {

    linefollow();
  }
```

```
  break;
// perform similar sequence as case 2
case 4:
  linefollow();
  distanceA = 0;
  distanceB = 0;
  break;
//perform similar sequence as case 1
case 5:
  if (searchinter == 13) {

    distanceA += sonar.ping_cm();
    linefollow();


    if (distanceA > 0) {
      obstacleindication();
      obs3a = 1;

    } else if (distanceA == 0) {
      obs3a = 2;

    } else {
      linefollow();
    }
  } else if (searchinter == 16) {


    distanceB += sonar.ping_cm();
    linefollow();


    if (distanceB > 0) {
      obstacleindication();
      obs3b = 1;

    } else if (distanceB == 0) {
      obs3b = 2;

    } else {
```

```
      linefollow();
    }

  } else {

    linefollow();
  }
  break;
// perform similar sequence as case 3
case 6:
  linefollow();
  distanceA = 0;
  distanceB = 0;
  break;
// perform similar sequence as case 1
case 7:
  if (searchinter == 19) {

    distanceA += sonar.ping_cm();
    linefollow();

    if (distanceA > 0) {
      obstacleindication();
      obs4a = 1;

    } else if (distanceA == 0) {
      obs4a = 2;

    } else {
      linefollow();
    }
  } else if (searchinter == 22) {


    distanceB += sonar.ping_cm();
    linefollow();


    if (distanceB > 0) {
```

```
              obstacleindication();
              obs4b = 1;

            } else if (distanceB == 0) {
              obs4b = 2;

            } else {

              linefollow();
            }

          } else {

            linefollow();
          }
          break;
      }
    } while ((sensorValues[0] < 500) && (sensorValues[1] < 500) &&
(sensorValues[6] < 500) && (sensorValues[7] < 500));
  }

//code sequence at every main intersection
  else if (((sensorValues[0] > 900) && (sensorValues[1] > 900) &&
(sensorValues[2] > 900) && (sensorValues[5] > 900) && (sensorValues[6] >
900) && (sensorValues[7] > 900)) && (maininter == 0 || (maininter == 2 &&
searchinter == 6) || (maininter == 4 && searchinter == 12) || (maininter
== 6 && searchinter == 18))) {
    intersectionindication(); // indicate the presence of an intersection
    mainintersection(); // perform main intersection sequence

  }
//code sequence at other intersections that are not the main intersection
  else if ((sensorValues[0] > 900) && (sensorValues[1] > 900) &&
(sensorValues[2] > 900) && (sensorValues[5] > 900) && (sensorValues[6] >
900) && (sensorValues[7] > 900)) {
    if ((searchinter == 1 && obs1a == 1) || searchinter == 3 ||
(searchinter == 7 && obs2a == 1) || searchinter == 9 || (searchinter == 13
&& obs3a == 1) || searchinter == 15 || (searchinter == 19 && obs4a == 1)
|| searchinter == 21) {
      movecount = 20;
```

```
      intersectionindication(); // indicate the presence of an
intersection
      spinright(); // spin right

      delay(900); // delay to ensure a 180 turn
      searchinter += 1; // add 1 to the all intersection counter
    // here we say if an obstacle is present, move on to the next spot
    } else if ((searchinter == 4 && obs1b == 1) || (searchinter == 10 &&
obs2b == 1) || (searchinter == 16 && obs3b == 1) || (searchinter == 22 &&
obs4b == 1)) {
      movecount = 20;
      intersectionindication(); //indicate the presence of an intersection
      turnleft(); // turn left
      delay(700); // delay to ensure 90 degrees turn
      stopmotors(); // stop motors
      delay(5); // delay
      searchinter += 1; // add 1 to the all intersection counter
      maininter += 1;
    // here if an obstacle is not present on any of the right lots we go
park
    } else if ((searchinter == 1 && obs1a == 2) || (searchinter == 7 &&
obs2a == 2) || (searchinter == 13 && obs3a == 2) || (searchinter == 19 &&
obs4a == 2)) {
      intersectionindication();
      parkright();
// here if an obstacle is not present of any of the left lots we go park
    } else if (searchinter == 4 && obs1b == 2 || (searchinter == 10 &&
obs2b == 2) || (searchinter == 16 && obs3b == 2) || (searchinter == 22 &&
obs4b == 2)) {
      intersectionindication();
      parkleft();
// at other intersections that are not key intersections, we move forward
and add 1 to all intersection counter
    } else {
      movecount = 20;
      intersectionindication(); // indicate intersection
      forward();
      searchinter += 1;

      delay(50);
```

```
    }
    position = qtr.readLineBlack(sensorValues);
  }
}

//--------------------------------------------------------------------------
--------------------------------------------------------------------------
----------------
// Function Code Section

// led code to indicate parking. turn both blue and red leds on and off
void parkingindication() {
  digitalWrite(A0, HIGH);
  digitalWrite(A1, HIGH);
  delay(500);
  digitalWrite(A0, LOW);
  digitalWrite(A1, LOW);
  delay(500);
  digitalWrite(A0, HIGH);
  digitalWrite(A1, HIGH);
  delay(500);
  digitalWrite(A0, LOW);
  digitalWrite(A1, LOW);
}

// led code to turn red led on and off
void obstacleindication() {
  digitalWrite(A0, HIGH);
  delay(50);
  digitalWrite(A0, LOW);
}

// led code to turn blue led on
void intersectionindication() {
  digitalWrite(A1, HIGH);
  delay(50);
  digitalWrite(A1, LOW);
}
```

```
// main intersection code that turn the robot right and adds 1 to both
main intersection counter and all intersection counter
void mainintersection() {
  position = qtr.readLineBlack(sensorValues);
  movecount = 20;
  turnright();
  delay(700);
  maininter += 1;
  searchinter += 1;
}

// code sequence to park right
void parkleft() {
  if (leftpark == 0) {
    lcd.print("Parking.........."); // indicate parking on the LCD
    movecount = 20;
    turnleft(); // turn left 90 degrees
    delay(700);
    leftpark += 1; // add one to left intersection counter
  } else if (leftpark == 1) {
    movecount = 20;
    turnleft(); // turn left 90 degrees again
    delay(700);
    leftpark += 1; // add one to left intersection counter
  } else {
    stopmotors(); // stop motors at T zone
    delay(5000);
    lcd.write(12); // clear lcd
    totalobstacle = obs1a + obs1b + obs2a + obs2b + obs3a + obs3b + obs4a
+ obs4b - 2; // sum up all obstacle reading and subtract 2 because a free
spot will have an obstacle counter of 2
    lcd.print("Total Obstacles =");
    lcd.print(totalobstacle); // display total number of obstacles on lcd
    parkingindication(); // indicate parking lights
    delay(7000);
  }
}

// code sequence to park right, similar to parking left. We turn right
here.
```

```
void parkright() {
  if (rightpark == 0) {
    lcd.print("Parking.....");
    movecount = 20;
    spinright();
    delay(1000);
    rightpark += 1;
  } else if (rightpark == 1) {
    movecount = 20;
    turnright();
    delay(700);
    rightpark += 1;
  } else if (rightpark == 2) {
    movecount = 20;
    turnright();
    delay(700);
    rightpark += 1;
  } else {
    stopmotors();
    delay(5000);
    totalobstacle = obs1a + obs1b + obs2a + obs2b + obs3a + obs3b + obs4a
+ obs4b - 2;
    lcd.write(12);
    lcd.print("Total Obstacles =");
    lcd.print(totalobstacle);
    parkingindication();
    delay(7000);
  }
}

// line follow code
void linefollow() {
  position = qtr.readLineBlack(sensorValues);
  // ensure the postion of the black line is between 3200 and 3800. a well
centered blackline will be at 3500
  if (position < 3200) {
    do {
      movecount = 1;
      turnright();
      position = qtr.readLineBlack(sensorValues);
```

```
    } while (position < 3200);
  } else if (position > 3800)
    do {
      movecount = 1;
      turnleft();
      position = qtr.readLineBlack(sensorValues);
    } while (position > 3800);
  movecount = 5;
  forward();
  position = qtr.readLineBlack(sensorValues);
}
// code to move robot forward
void forward() {
  for (int i = 0; i < movecount; i++) {
    myservo1.write(170);
    myservo2.write(10);
    delay(5);
  }
}

// code to stop motors
void stopmotors() {
  for (int i = 0; i < movecount; i++) {
    myservo1.write(90);
    myservo2.write(90);
    delay(20);
  }
}

// code to turn right with one wheel on
void turnright() {
  for (int i = 0; i < movecount; i++) {
    myservo1.write(170);
    myservo2.write(90);
    delay(15);
  }
}
// code to turn left with one wheel on
void turnleft() {
  for (int i = 0; i < movecount; i++) {
```

```
    myservo1.write(90);
    myservo2.write(10);
    delay(15);
  }
}

//code to turn right with both wheels on
void spinright() {
  for (int i = 0; i < movecount; i++) {
    myservo1.write(170);
    myservo2.write(170);
    delay(10);
  }
}
//code to turn left with both wheels on
void spinleft() {
  for (int i = 0; i < movecount; i++) {
    myservo1.write(10);
    myservo2.write(10);
    delay(10);
  }
}
```