ELEC 299 2021 Final Project:

Find, Study, and Return!

Presented to Brian Frank and Emily Taylor

ELEC 299 – Group 08:

Ben

Adam

Alex

June 20, 2021

Statement of Originality

We hereby declare that this submission is our own work and to the best of our knowledge contains no materials previously published or written by another person, except where explicitly cited, nor does it contain any work by any student not in this group. Following professional engineering practice, we bear the burden of proof for original work. We have read the Policy on Academic Integrity posted on the Faculty of Engineering and Applied Science web site and confirm that this work is in accordance with the Policy.

Executive Summary

A mechatronic vehicle has been designed using Arduino hardware and software. It must test to ensure all sensors are functional before driving straight ahead to a location a known distance away. The obstacles placed about the environment were avoided. Once the target location was reached, the vehicle returned home via the most efficient path. If at either end the vehicle did not detect the floor marker where it expected to find it, the vehicle must perform a search algorithm to locate it before continuing.

The target location is approximately 3 m away. The two obstacles, approximately the size of the given cardboard box, are placed more than 1 m from the starting location and are rectangular in shape.

The electrical hardware components used on the vehicle include a battery pack, two DC motors, two encoder sensors, an ultrasonic sensor, two infrared obstacle sensors, and a phototransistor infrared sensor. No additional hardware components will be implemented.

The vehicle uses the Bug 2 algorithm to navigate the environment. If it believed it had finished its path but did not sense the floor marker, the vehicle implements a square version of a 2D outward spiral search function. Once it had found the floor sensor, it uses memory from its initial pass of the objects to determine a more efficient route home.

The vehicle's performance closely matches the expected behaviour. Some optimisations which require more fine tuning on the wheel encoder sensors could still be improved upon. After updating and narrowing the scope of the project, the team stayed on track with the original project plan and used the project extension wisely to ensure a quality submission and report. In the event of significant hardware malfunctions in the final day, the group worked together to find solutions and workarounds to the malfunctioning hardware.

Table of Contents

List of Figures

List of Tables

Project Overview

A mechatronic vehicle must be designed using Arduino hardware and software to travel to a known location approximately 3 metres away, avoiding obstacles and readjusting course with respect to such obstacles. Once the location has been reached, the vehicle must then travel back to the starting position in a more efficient (i.e., shorter) route then originally travelled. If the vehicle has returned back to what is meant to be the starting position but the starting position marker is not found, a searching algorithm must be used to find the starting position marker.

To accomplish this task, the vehicle will use the Bug 2 robotic motion algorithm [1]. Bug 2 robotic motion drives straight toward a goal location on a direct line called the m-line. If the direct route is blocked by an obstacle, the path is adjusted to travel around the object at which time it returns to the initial straight path towards the goal location [1]. An ultrasonic sensor will be used to detect incoming obstacles while infrared (IR) sensors will be used to determine when the vehicle has passed the obstacles to return to the initial path. Finally, a photoresistor IR sensor will be used to detect the location markers on the floor.

The software used to perform the driving task builds off the existing functions created in Weeks 1 to 4 of ELEC 299. The existing functions used will be to perform behaviours such as turning, PID control to drive in a straight line, and sensor polling for self-check and obstacle detection. Notable additions and adjustments include a new search function which must be created to search the vehicle's general area for location markers when required, as well as a function to determine a more efficient route back to the starting position using data collected during the initial trip to the goal marker.

Requirements and Specifications

Requirements:

The requirements for the vehicle are presented in Table 1.

Table 1: Project Requirements

Requirement:	Description:
Self-check	Perform a self-check of all sensors on your robot.
Drive to goal	Drive straight to a location a known distance away.
Obstacle avoidance	Avoid obstacles and return to the initial path.
Search pattern	Locate the starting marker using search pattern if return trip was inaccurate.
Return navigation	Return to starting point must be navigated in the given environment to shorten the travel distance compared to the initial path.
Combination of sensors (optional)	Can add additional, non-communicative sensors.

Specifications:

The specifications for the environment the vehicle will work within are presented in Table 2.

Table 2: Project Environment Specifications

Specification:	Description:
Obstacles	• Obstacles are approximately the size of the given cardboard box. • Obstacles will not be placed less than 1 m from the starting location. • Only two obstacles will exist in the environment. • Obstacles are assumed to be square/rectangular.
Distance	Target location will be located approximately 3 m away from starting location.

It should be noted that the obstacles must be placed a variety of positions for the test environment to prove the adaptability of the vehicle.

Hardware Design

The principal hardware used to implement the vehicle are an Arduino Uno microcontroller board paired with a motor shield and powered by a 9 V battery pack. Two Adafruit DC motors are used to propel the vehicle forward while wheel encoders are used to track travel distance. For obstacle detection, a combination of ultrasonic and infrared sensors is used. The ultrasonic sensor faces directly forward on the vehicle to measure incoming obstacles, while digital IR sensors aimed to the left and right of the vehicle are used to determine whether the vehicle has passed by the obstacles being avoided. Potentiometers are used on the IR sensors to change the range at which the signal is triggered. For consistency, the IR sensors are measured to trigger at 10 cm distance from obstacles. Finally, a phototransistor IR sensor aimed at the ground below the vehicle is used to locate the floor markers used for vehicle travel. The output of the sensor changes based on the amount of light is reflected to the sensor, therefore a material such as aluminum foil will be used to trigger a significant signal from the sensor. Due to time constraint and lack of available additional hardware, the vehicle will not be using additional sensors beyond the given hardware to complete the task. A visual schematic of the hardware is presented in Figure 1. The motor encoders are represented by the potentiometers in the figure. The 9V battery represents the 7.5V attached to the battery pack. Note that the motor shield is not present in the schematic, however it is presented in Figure 2 resting on top of the Arduino Uno Board.

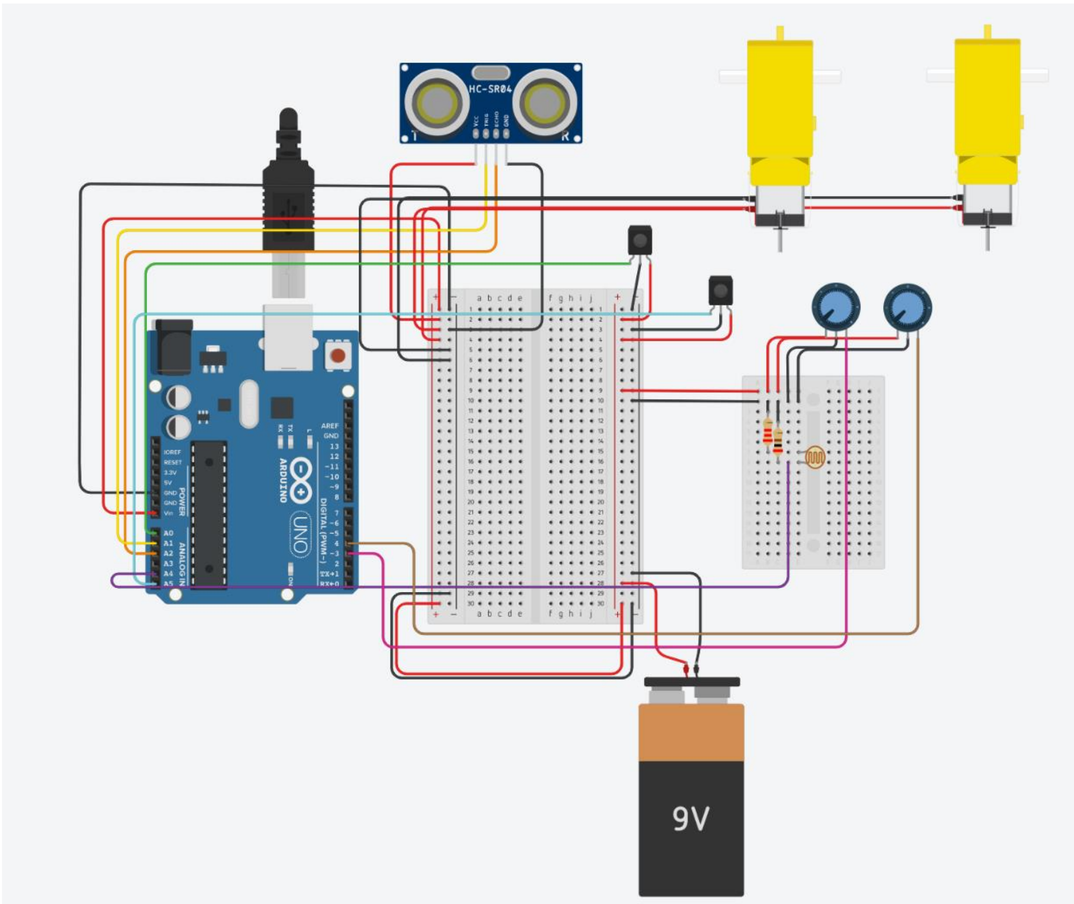


Figure 1: Visual schematic of hardware, including connections to ports.

Figure 2: Arduino Uno and motor shield.

Software Design

The full flow chart detailing the algorithm behind the robot's motion is displayed in Appendix A. The algorithm is derived from Bug 2 and uses the line pointing straight ahead toward the target located 3 m away as its m-line. The complete Arduino code is displayed in Appendix B and includes the full code with comments on the various functions enclosed.

Corkscrew Search Algorithm

If the vehicle believes it has reached the end of its path but has not detected the floor marker, it will implement a 2D search algorithm. The algorithm is a variation of an outward spiral, or corkscrew, but moves in straight line segments. The corkscrew movement pattern is displayed in Figure 3. Once the vehicle detects the marker, it re-orients itself to point away from home, to match the scenario where the marker was detected where expected. This allows for the same return algorithm to be used.

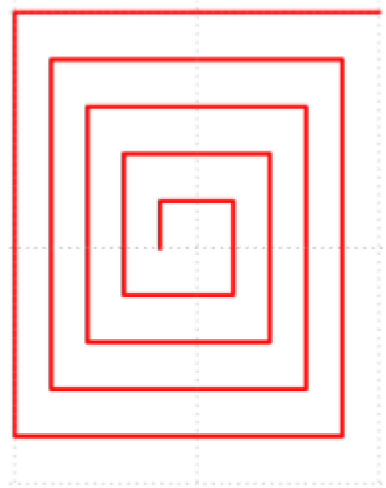


Figure 3: Visualisation of the corkscrew search algorithm. The vehicle begins at the centre and moves outward.

This algorithm is an efficient method for detecting co-terminal, concurrent, and bifurcating lines [2]. The functionality of co-terminal detection works for the current task when detecting the co-terminal 30 cm strips of tinfoil used as the floor markers.

Efficient Return Path

There are three cases the vehicle must consider when determining which return path to take. Case 1 is that no obstacles were encountered before reaching the target. Trivially, the most efficient return path is a straight line back home. Case 2 occurs when only one obstacle is encountered before reaching the target. In this case, the theoretical most efficient return path is to move diagonally. However, this saves only a small amount of time and the uncertainty in the wheel encoders make this difficult to achieve. This solution is also problematic in that the car does not know where the second obstacle is located. The car could run into the second obstacle and veer off course while taking a diagonal path. As such, for simplicity, the vehicle will take the same path home as it took to get to the target to avoid hitting the second obstacle.

Case 3 occurs when both obstacles are encountered before reaching the target. Case 3's return path is the route that can be optimized the most. By remembering the farthest distance the vehicle had to travel from the m-line, the return path can be modified to initially travel that distance sideways to avoid all obstacles on the way back home. Assuming that the vehicle encounters both obstacles in the environment, the vehicle travelling a wide berth limited by the size of the largest avoided obstacle will be shorter than the initial trip. A diagram of the more efficient travel is shown in Figure 4.

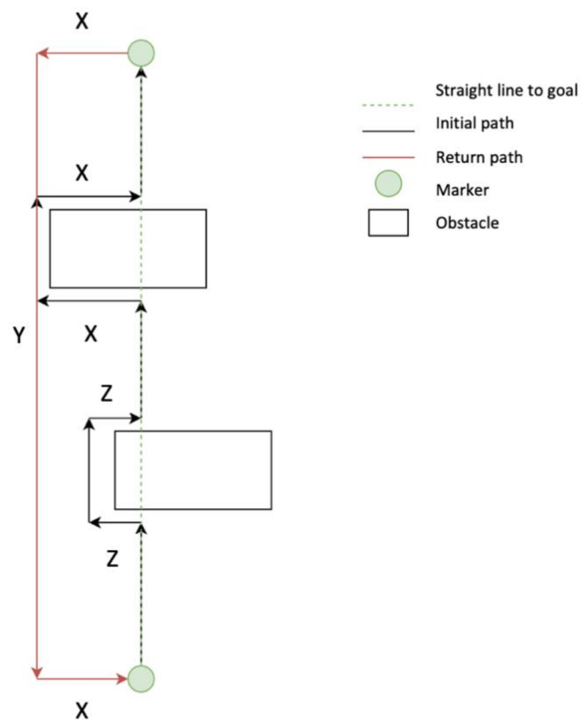


Figure 4: Efficient return trip.

Evaluating the expressions generated by the paths between the markers gives:

$$Path_{initial} = Y + X + X + Z + Z = Y + 2X + 2Z$$

$$Path_{return} = X + Y + X = Y + 2X$$

When comparing the two general equations, $Path_{initial}$ will always be greater than the more efficient $Path_{return}$ if all objects are encountered, because the value of Z will be greater than zero. This algorithm becomes more efficient with each new obstacle added to the environment.

It should be noted that this algorithm is not guaranteed to work unless every obstacle is encountered during the trip toward the target. If there are additional obstacles in the environment, they could interfere with the modified return path. Figure 5 illustrates this more problematic scenario. It is for such reason that the vehicle must decide which of the three cases it has experienced and respond appropriately.

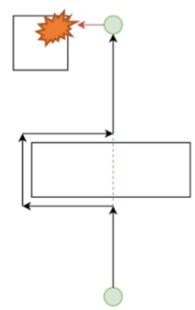


Figure 5: Example of problematic environment for efficiency algorithm.

Performance Comparison

The vehicle did not complete the most efficient return path after encountering only one obstacle (case 2). The most efficient method would be to move along the hypotenuse of a triangle where the other two sides are the m-line and the obstacle. However, the inaccuracy in the wheel encoders and the time frame for the project meant that 90-degree turns were very finely tuned, and any other rotation would be unreliable.

The vehicle was also unable to handle obstacles misaligned with the m-line. This could be theoretically solved by rotating 90 degrees, checking the IR sensor, and slowly rotating back until the object was detected. Unfortunately, since the new angle would have to be stored, this ran into the same problem where any rotations other than 90-degree turns were unpredictable.

Lastly, the error in the phototransistor made it difficult to calibrate. Even by choosing optimal resistors, the voltage swing was rarely large enough to confidently detect a difference between the floor and the tinfoil. Averaging the values helped reduce erroneous readings, but there was still a fair amount of uncertainty.

When performing final edits to the code for video recording, ultrasonic sensor of the main test vehicle stopped responding, adding to the list of hardware problems the project faced. To resolve the issue, one of the front-facing IR sensors was used in place of the ultrasonic sensor. The sensor adjustment was a downgrade, as it could not be used to anticipate obstacles as accurately as the range was closer to 10 cm compared to over 1 m that the ultrasonic sensor is capable of.

In terms of overall success, the car was successful before it broke. 6 out of 10 trials were successful in finding the marker and returning to the starting position. When comparing two different car’s results when using a similar algorithm, the results were similar. The secondary vehicle yielded 7 out of 10 successful trials. Both vehicles faced the same difficulties in turning accurately and light negatively affecting the phototransistor reading. Using extra sensors, such as a gyroscope, would make the car navigate more accurately. More accurate and expensive encoders could also be used to improve turn efficiency. The algorithm could also be improved to include non-rectangular obstacles and implement the more efficient solution to case 2.

Team Performance

During the Week 5 Lab, the team organized the final report document to assign tasks for the following week with due dates. Decisions regarding which vehicles would be filmed were based on the conditions of each vehicle. Additionally, as each member of the group had different strategies for functions and behaviours, decisions on who would do the bulk of the programming considered which members had already similar code due to time constraints of the project. Based on these metrics the group agreed to give primary programming responsibilities to two members, while the third member’s primary focus was organizing, writing, and formatting the final report. Team communication was done primarily through Facebook Messenger for casual communication. For more major meetings to discuss programming work, Zoom meetings were scheduled throughout the week. Members showed up to meetings and were active members of the project despite conflicting work schedules and outside commitments.

During the initial planning phase, brainstorming solutions to finding the most efficient return route were primarily focused on routes involving triangulation of obstacle corners with regard to the final marker. Unfortunately, calculation and implementation of specific turn angles proved to be a significant challenge due to inconsistency of the given motor encoders. The scope was eventually changed from finding the most efficient route to finding a more efficient route only. The redirect in scope lead to the final return algorithm, however significant time was wasted on triangulation that could have been used on perfecting the current efficiency algorithm. Future projects should place more attention to time constraints when designing new functionality with the vehicle.

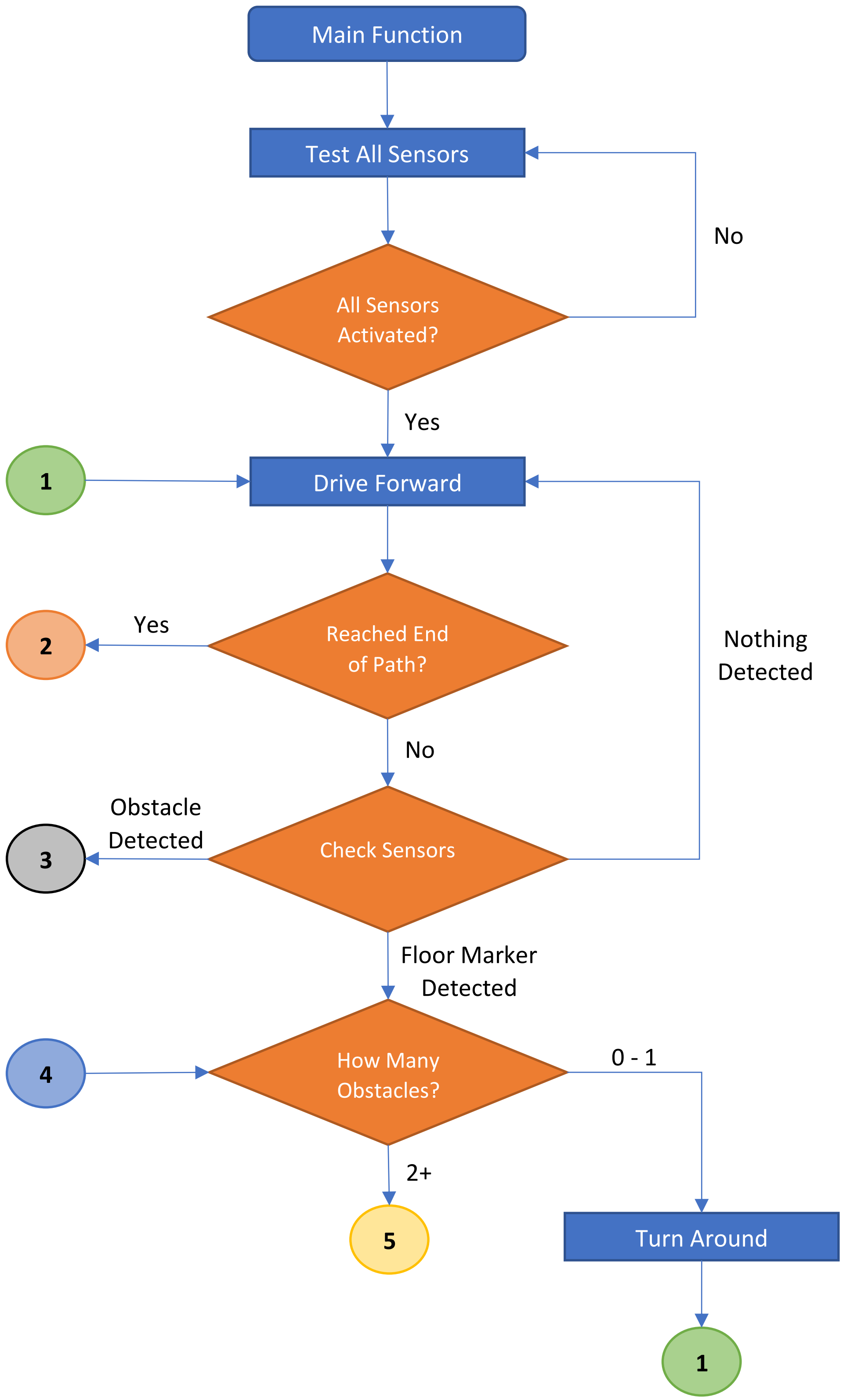
When last-minute problems with the hardware arose, solution brainstorming needed to resume to work allow for the submission of video footage of the vehicle. Specifically, Adam Bayley volunteered to work with his vehicle to update his code to use the remaining working sensors in place of the broken ultrasonic sensor. Moving forward, members of the group will be more proactive with dealing with hardware and software issues as they arise to avoid situations where their vehicle may need to be used for parts or presentation at a moment's notice.

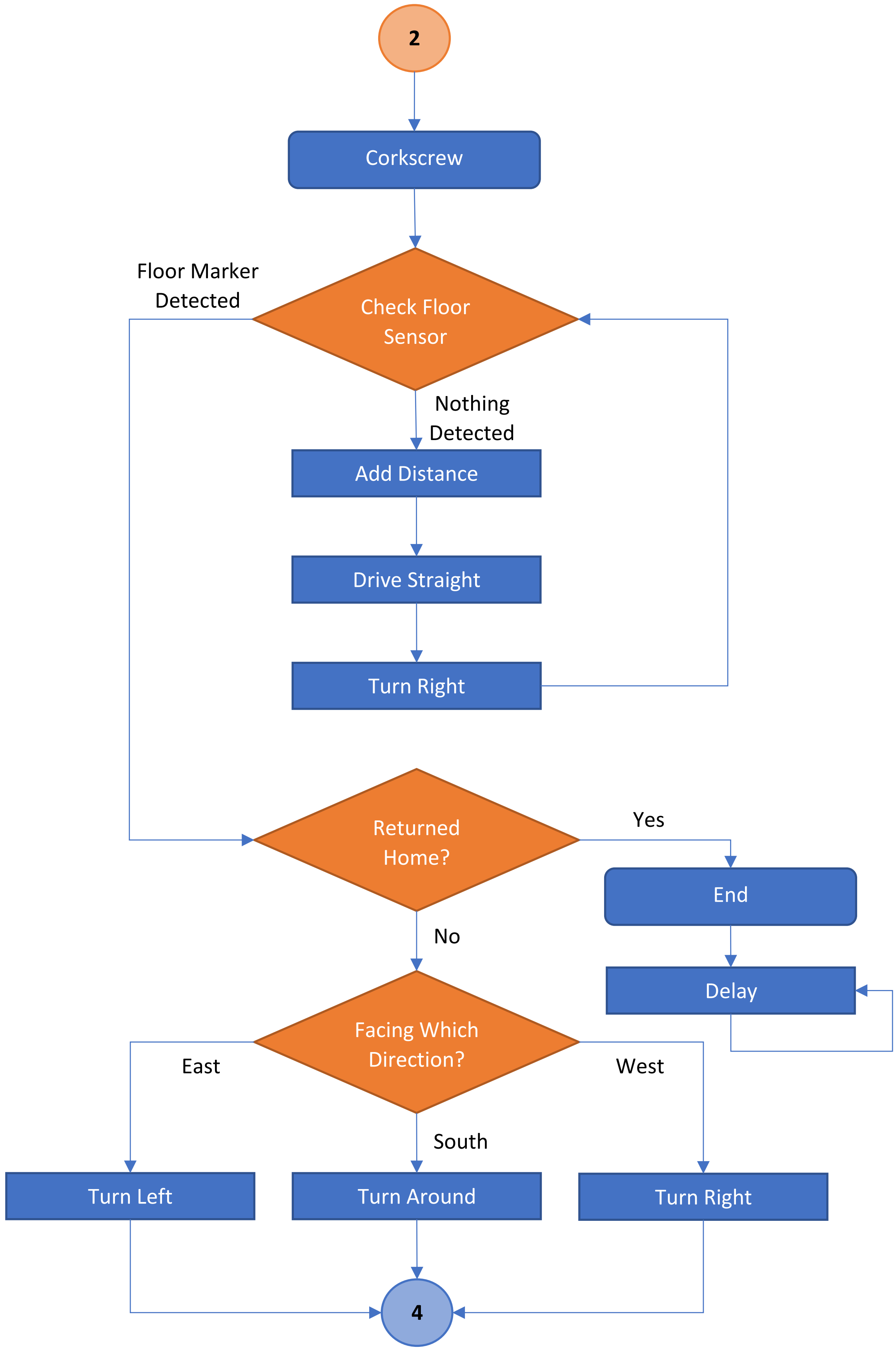
References

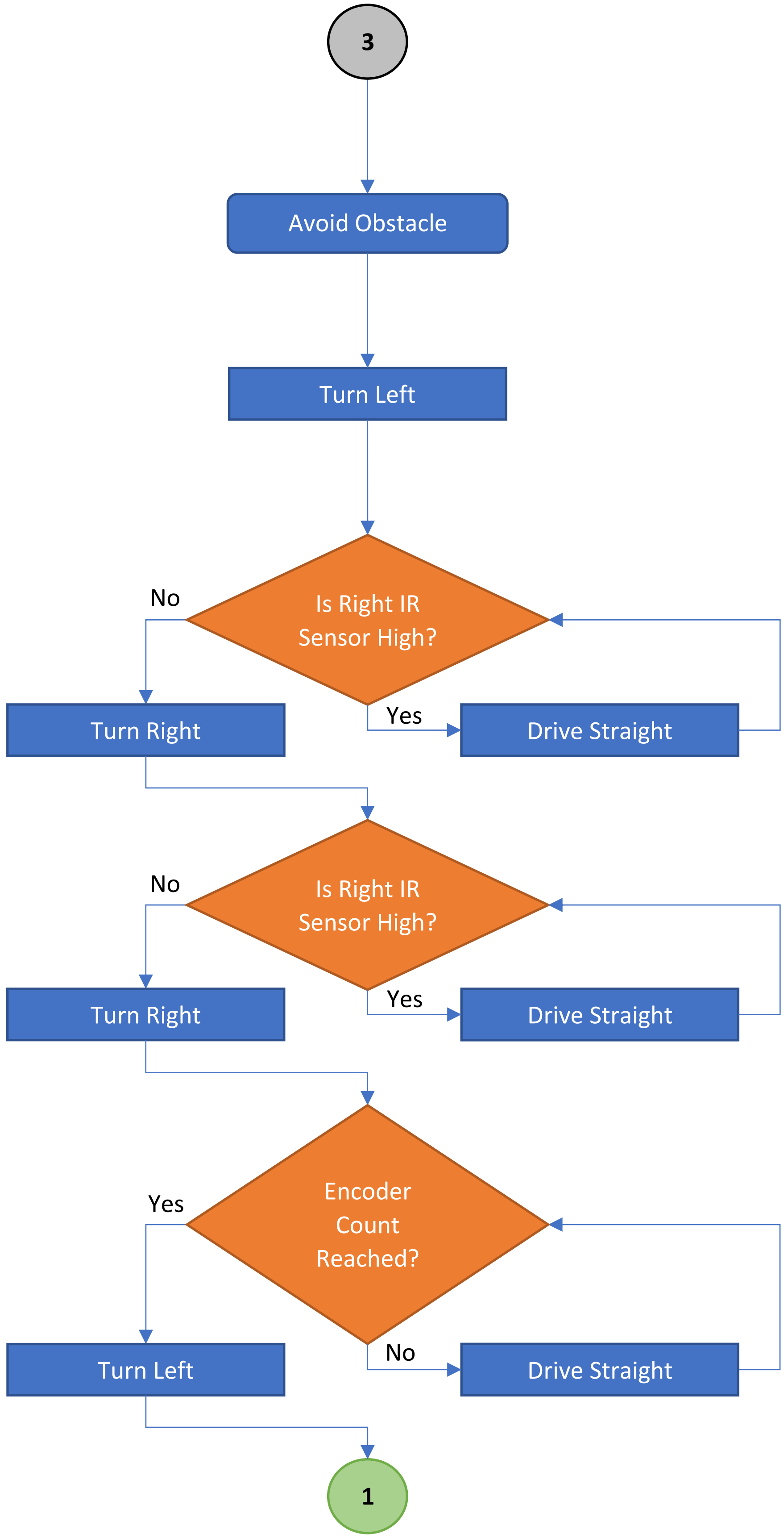
[1] H. Choset, “Robotic Motion Planning: Bug Algorithms,” p. 63.

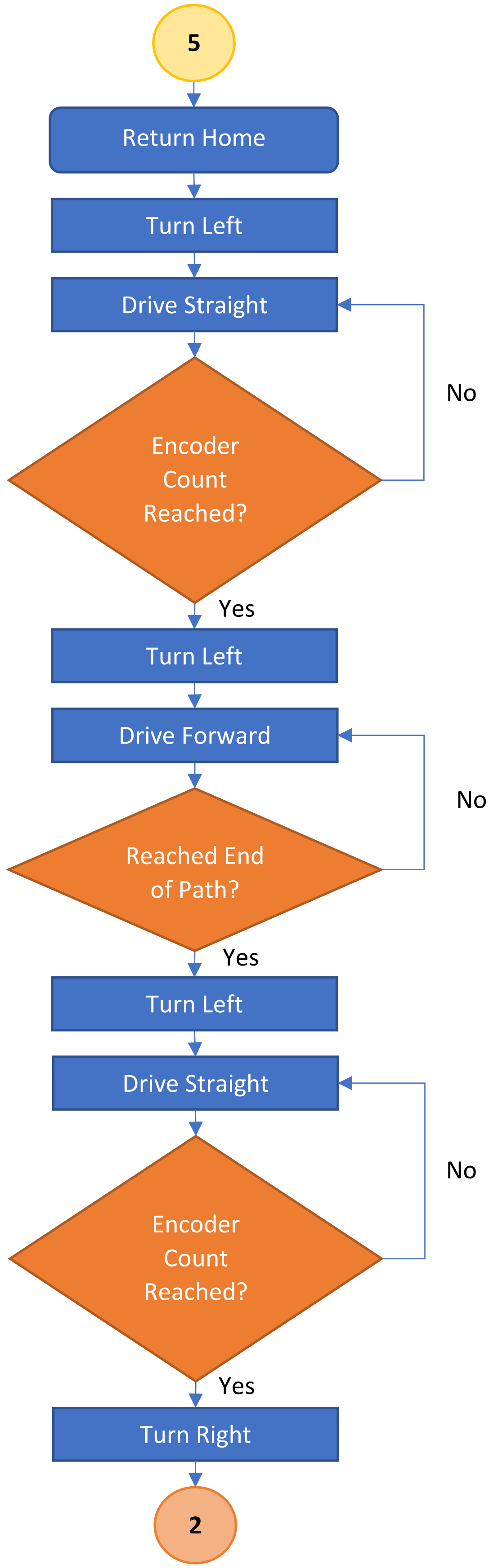
[2] S. Burlington and G. Dudek, “SPIRAL SEARCH AS AN EFFICIENT MOBILE ROBOTIC SEARCH TECHNIQUE,” p. 10.

Appendix A – Flowchart




2
Corkscrew
Floor Marker
Detected
Check Floor
Sensor
Nothing
Detected
Add Distance
Drive Straight
Turn Right
Returned
Home?
Yes
End
Delay
No
Facing Which
Direction?
East
West
South
Turn Left
Turn Around
Turn Right
4


3
Avoid Obstacle
Turn Left
Is Right IR Sensor High?
No
Yes
Turn Right
Drive Straight
Is Right IR Sensor High?
No
Yes
Turn Right
Drive Straight
Encoder Count Reached?
Yes
No
Turn Left
Drive Straight
1


5
Return Home
Turn Left
Drive Straight
Encoder Count Reached?
No
Yes
Turn Left
Drive Forward
Reached End of Path?
No
Yes
Turn Left
Drive Straight
Encoder Count Reached?
No
Yes
Turn Right
2

Appendix B – Code

final_draft_week_6_elec299

```
//final submission file Adam Bayley, group 8 of ELEC299 for Thursday 6:30-9:30 PM EST

//setup the pins on the arduino
#define LEncoderPin 3
#define REncoderPin 2
#define echo A2
#define trig A1
#define sensor A4
#define LSensor A0
#define RSensor A5

//define some values that won't change
#define TicksPerRot 80
#define MaxSpeed    255
#define MinSpeed    0
#define TargetDist  200 // 2 m = 200 cm
#define NumReadings 10
#define NumSensors  4

//include the PID and AFMotor libraries
#include <AFMotor.h>
#include <PID_v2.h>

AF_DCMotor left_motor(1, MOTOR34_1KHZ);  // left motor to M1 on motor control board
AF_DCMotor right_motor(3, MOTOR12_1KHZ); // right motor to M3 on motor control board

int placeholder = 1;
int turnOne = 0;
int turnTwo = 0;
int lastTimeL = 0;
int lastTimeR = 0;
int LeftEncoderCount = 0;
int RightEncoderCount = 0;
int DelayTime = 1000;
int PrevRightEncoderCount = 0;
int PrevLeftEncoderCount = 0;
int obstacleCount = 0;
int LeftEncoderSmall, RightEncoderSmall;\
int trackWidth = 0;
int trackLength = 0;
int Length1 = 0;
int Length2 = 0;
int turnLength = 0;
int firstPass = 0;
int RPWM = 0;
int LPWM = 0;
int LSens = 1, RSens = 1, BSens = 1, DSens = 1;

//PID control variables
double EncDelta = 0;
double OutDelta = -20.0;
double EncDeltaSet = 0;
double Kp = 0.5, Ki = 0.5, Kd = 0.1; // starting PID controller settings



double circumference = 10; //40.84; // Measured


unsigned long duration, lastTimeMovement, lastTimeSensors;


PID DiffSpeed(&EncDelta, &OutDelta, &EncDeltaSet, Kp, Ki, Kd, DIRECT);


//variables for averaging functions
const int numReadings = 10;
float readings[numReadings];      // the readings from the analog input
int readIndex = 0;              // the index of the current reading
float total = 0;                  // the running total
float average = 0;                // the average


void setup() {

  attachInterrupt(1, countLEncoder, RISING); //calls on any CHANGE on pin 3 (interrupt #1, soldered to Pin3)
  attachInterrupt(0, countREncoder, RISING); //calls on any CHANGE on pin 2 (interrupt #0, connected to header on Pin2)
  interrupts();

  Serial.begin (115200);    // set up Serial library at 115200 bps
  pinMode(LSensor, INPUT);
  pinMode(RSensor, INPUT);
```

```
  pinMode(echo, INPUT);
  pinMode(trig, OUTPUT);
  pinMode(sensor, INPUT);

  //setup for PID, including the bounds and auto straightening thing
  DiffSpeed.SetMode(AUTOMATIC);
  DiffSpeed.SetOutputLimits(-150, 150);
  DiffSpeed.SetSampleTime(70);

  //Set motor direction
  left_motor.run(RELEASE);
  right_motor.run(RELEASE);

  //setup with the array for the function
  for (int thisReading = 0; thisReading < numReadings; thisReading++) {
    readings[thisReading] = 0;
  }

  lastTimeMovement = millis();
  lastTimeSensors = millis();

  scannerSetup();
}

int flag;
void loop() {

  unsigned long myTime = millis();
  left_motor.run(BACKWARD);
  right_motor.run(BACKWARD);
  float avg = getDistance();  //variable for the ultrasonic sensor

  //while the ultrasonic sensor doesn't detect anything or there's no paper being detected, go straight.
  while (1)
  {

    //while there's no object or there's no paper, go forward, update the time
    while ( avg > 20 && (getAverage(A4) > .40 || getAverage(A4) < .30) && myTime < 24000 )
    {
      flag = 1;
      myTime = millis();
      avg = getDistance();
      adjustSpeed(90);

    }//end while

    //if you're greater than 24 sec (longest time to hit both obj's and find paper), and flag = 1, and you're  on the first route
    //find the paper and turn flag to 0 so it can never activate this again
    if (myTime > 24000 && flag == 1 && firstPass == 0)
    {
      corkscrew();
      flag = 0;
    }

    //in the case of an obstacle, avoid it
    if (avg < 20)
    {
      avoidObstacle();
    }//end if

    //but if you come across the paper, change firstPass to 1 (so you don't hit the previous if) then start the return to start function.
    if (getAverage(A4) < .40 || getAverage(A4) > .30)
    {
      firstPass = 1;
      returnToStart();
      break;
    }//end if

  }//end while 1

  while (1) {}

}//end void loop

void resetEncoders() {
  LeftEncoderCount = 0;
  LeftEncoderSmall = 0;
  RightEncoderCount = 0;
  RightEncoderSmall = 0;
}//end void reset encoders

```

```
//function for getting voltage
float getVoltage(int pin) {
  float voltage = 5.0 * analogRead(pin) / 1024;
  return voltage;
}

// interrupt function for left encoder
void countLEncoder() {
  if (micros() - lastTimeL > 100) {
    LeftEncoderCount++;
    LeftEncoderSmall++;
    lastTimeL = micros();
  }
}

// interrupt function for right encoder
void countREncoder() {
  if (micros() - lastTimeR > 100) {
    RightEncoderCount++;
    RightEncoderSmall++;
    lastTimeR = micros();
  }
}

//function for getting distance
float getDistance() {
  digitalWrite(trig, LOW);
  delayMicroseconds(10);
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);
  float duration = pulseIn(echo, HIGH);
  float velocity = 0.0343;
  float distance = duration * velocity / (2);
  delay(60);
  return distance;
}

//function for gettig the average in general
float getAverage(int pin) {
  total = total - readings[readIndex];
  float data = getVoltage(pin);
  readings[readIndex] = data;
  total = total + readings[readIndex];
  readIndex = readIndex + 1;


  if (readIndex >= numReadings) {
    readIndex = 0;
  }

  average = total / numReadings;
  //Serial.println(average);
  delay(1);        // delay in between reads for stability
  return average;
}//end get average

//function for getting the average distance
float getAverageDistance() {

  total = total - readings[readIndex];
  readings[readIndex] = getDistance();
  total = total + readings[readIndex];
  readIndex = readIndex + 1;


  if (readIndex >= numReadings) {
    readIndex = 0;
  }


  average = total / numReadings;

  //Serial.println(average);
  delay(1);
  return average;
}//end get average distance

//function for adjusting the speed
int adjustSpeed(int i) {

  EncDelta = (double)(1.0 * LeftEncoderCount - 1.0 * RightEncoderCount);
```

```
  DiffSpeed.Compute(); // compute the value of OutDelta using the PID controller
  RPWM = (int)round(i - OutDelta);
  LPWM = (int)round(i + OutDelta);

  left_motor.setSpeed(LPWM);
  right_motor.setSpeed(RPWM);
  return LPWM;
}//end adjust speed

//function for the scanner setup thing
void scannerSetup() {
  while (DSens == 1  || LeftEncoderCount < 20 || RightEncoderCount  < 20) {
    //RSens == 1 || BSens == 1 || LSens == 1||
    if (RightEncoderCount > 20) {
      Serial.println("Right Encoder Working");
    }
    if (LeftEncoderCount > 20) {
      Serial.println("Left Encoder Working");
    }
    if (getAverage(LSensor) < 2) {
      Serial.println("Left Sensor Working");
      LSens = 0;
    }
    if (getAverage(RSensor) < 2) {
      Serial.println("Right Sensor Working");
      RSens = 0;
    }
    if (getAverage(sensor) > 0.30 && getAverage(sensor) < 0.40) {
      Serial.println("Bottom Sensor Working");
      BSens = 0;
    }

    if (getAverageDistance() < 50) {
      Serial.println("Distance Sensor Working");
      DSens = 0;
    }
  }

  Serial.println("Sensor Test done");
  RightEncoderCount = 0;
  LeftEncoderCount = 0;
  delay(2000);
}//end of sensor test

//function for use of determining largest width needed to turn (only used in event of case 2)
int obstacleLength( int Length1, int Length2) {

  if (Length1 > Length2)
  {
    turnLength = Length1;
    return turnLength;

  }
  else if (Length2 > Length1)
  {
    turnLength = Length2;
    return turnLength;
  }
  else {
    turnLength = Length1;
    return turnLength;
  }
}//end obstacleLength

//function to stop moving
void stopMoving() {
  Serial.println("Stopping...");
  left_motor.run(RELEASE);
  right_motor.run(RELEASE);

  while (true) {
    delay(10000);
  }//end while
}

//function for the left turn
void leftTurn() {
  right_motor.run(BACKWARD);
  LeftEncoderCount = 0;
  RightEncoderCount = 0;
  left_motor.setSpeed(0);
  while (RightEncoderCount < 40) {
```

```
    right_motor.setSpeed(200);
    Serial.println(RightEncoderCount);
  }
  right_motor.run(BACKWARD);
  left_motor.run(BACKWARD);
  left_motor.setSpeed(0);
  right_motor.setSpeed(0);
  RightEncoderCount = 0;
}//end left turn

//function for the right turn
void rightTurn() {
  left_motor.run(BACKWARD);
  right_motor.run(FORWARD);
  LeftEncoderCount = 0;
  RightEncoderCount = 0;
  right_motor.setSpeed(0);
  while (LeftEncoderCount < 40) {
    left_motor.setSpeed(200);
    right_motor.setSpeed(200);
    Serial.println(LeftEncoderCount);
  }
  left_motor.run(BACKWARD);
  right_motor.run(BACKWARD);
  left_motor.setSpeed(0);
  right_motor.setSpeed(0);
  LeftEncoderCount = 0;
}//end right turn

//comment for doing the bug algorithm. assumes the left hand side is the fastest method of reaching the paper.
void avoidObstacle() {
  if (getAverageDistance() < 15) {
    obstacleCount++;
    leftTurn();
    Serial.println("Left Turn");
    /*left_motor.setSpeed(0);
      right_motor.setSpeed(0);
      delay(1000);*/

    while (getAverage(RSensor) < 2) { //while going left, track the side
      adjustSpeed(150);

    }
    while (RightEncoderCount < 70 || LeftEncoderCount < 70) {
      adjustSpeed(150);
      Serial.println(RightEncoderCount);
      Serial.println(LeftEncoderCount);
    }
    int prevRight = RightEncoderCount;
    int prevLeft = LeftEncoderCount;

    //placeholder detects which object it's on. it is initialized to 1. these 2 statements are used in case 3 to detect
    //the largest latitutidal distance the car must travel to avoid hitting the objects when returning to the start.
    if (placeholder == 1)
    {
      //first run,
      turnOne = LeftEncoderCount;
    }

    if (placeholder == 2)
    {
      turnTwo = LeftEncoderCount;
    }
    placeholder++;

    Serial.println("Right Turn");
    rightTurn();
    /*left_motor.setSpeed(0);
      right_motor.setSpeed(0);
      delay(1000);*/
    while (getAverage(RSensor) < 2) { //while going forward, track the side
      adjustSpeed(150);
    }
    while (RightEncoderCount < 100 || LeftEncoderCount < 100) { //little bit of extra distance to make sure
      //the back of car doesnt clip during the turn
      adjustSpeed(150);
      Serial.println(RightEncoderCount);
      Serial.println(LeftEncoderCount);
    }
    rightTurn();
    //delay(1000);
    while (LeftEncoderCount < prevLeft || RightEncoderCount < prevRight) {
```

```
      adjustSpeed(200);
      Serial.println(RightEncoderCount);
      Serial.println(LeftEncoderCount);
    }
    leftTurn();
    /*left_motor.setSpeed(0);
      right_motor.setSpeed(0);
      delay(1000);*/
  }

} //end avoid obstacle

void avoidObstacleReturn() { //same function as avoidObstacle but in reverse; (left turn becomes right, etc.)
  //function used for case 2 returning to start position.
  if (getAverageDistance() < 15) {
    rightTurn();
    Serial.println("Right Turn");
    /*left_motor.setSpeed(0);
      right_motor.setSpeed(0);
      delay(1000);*/
    while (getAverage(LSensor) < 2) {
      adjustSpeed(150);
    }
    while (RightEncoderCount < 70 || LeftEncoderCount < 70) {
      adjustSpeed(150);
      Serial.println(RightEncoderCount);
      Serial.println(LeftEncoderCount);
    }
    int prevRight = RightEncoderCount;
    int prevLeft = LeftEncoderCount;
    Serial.println("Left Turn");
    leftTurn();
    /*left_motor.setSpeed(0);
      right_motor.setSpeed(0);
      delay(1000);*/
    while (getAverage(LSensor) < 2) {
      adjustSpeed(150);
    }
    while (RightEncoderCount < 100 || LeftEncoderCount < 100) {
      adjustSpeed(150);
      Serial.println(RightEncoderCount);
      Serial.println(LeftEncoderCount);
    }
    leftTurn();
    //delay(1000);
    while (LeftEncoderCount < prevLeft || RightEncoderCount < prevRight) {
      adjustSpeed(200);
      Serial.println(RightEncoderCount);
      Serial.println(LeftEncoderCount);
    }
    rightTurn();
    /*left_motor.setSpeed(0);
      right_motor.setSpeed(0);
      delay(1000);*/
  }
  Serial.println("Done");
  left_motor.run(BACKWARD);
  right_motor.run(BACKWARD);
} //end avoid obstacle return


//corkscrew function for finding object if marker isn't found.
void corkscrew() {
  Serial.println("Reached end of path but no marker detected");
  Serial.println("Beginning corkscrew search");

  int corkDist = 50;
  int pointing = 0;
  resetEncoders();

  while (getAverage(A4) > .40 || getAverage(A4) < .30) {
    // Drive straight while nothing has been detected (paper not detected yet)
    if (millis() - lastTimeMovement > 500) {
      adjustSpeed(150);
      lastTimeMovement = millis();
    }

    // Rotate and increase square radius.
    if (RightEncoderCount > corkDist) {
      corkDist += 0;
      resetEncoders();
      rightTurn();
      pointing++;
```

```
      if (pointing > 3) pointing = 0;
      resetEncoders();
    }
  }

  Serial.println("Floor marker detected");
  //first pass is used to represent the first time finding the marker.
  if (firstPass == 0) {
    // Rotate to point away from home.
    switch (pointing) {
      case 2:
        leftTurn();
      case 1:
        leftTurn();
        break;
      case 3:
        rightTurn();
        break;
    }
  } else {
    // Reached home. Stop.
    stopMoving();
  }//close else
}//close void corkscrew

void returnToStart()
{

  if (obstacleCount == 0) //case 1: 2 obstacles on course but doesn't detect either
  {
    leftTurn();
    leftTurn();

    while (getAverage(A4) > .40 || getAverage(A4) < .30)
    {
      adjustSpeed(100);
    }//end while

    while (getAverage(A4) < .40 || getAverage(A4) > .30)
    {
      left_motor.setSpeed(0);
      right_motor.setSpeed(0);
      left_motor.run(BACKWARD);
      right_motor.run(BACKWARD);
      delay(10000);
    }//end while
  }//end if obstacle count == 0

  if (obstacleCount == 1) //case 2: 2 obstacles on course, detects 1 obstacle
  {
    left_motor.run(BACKWARD);
    right_motor.run(BACKWARD);
    float avg = getDistance();
    if (avg > 17) {
      while (avg > 17) {
        avg = getDistance();
        adjustSpeed(90);
      }//end while (avg > 17)
    }//end if (avg >17)

    avoidObstacleReturn();
    while (getAverage(A4) > .40 || getAverage(A4) < .30)
    {
      adjustSpeed(100);
    }//end while
    while (getAverage(A4) < .40 || getAverage(A4) > .30)
    {
      left_motor.setSpeed(0);
      right_motor.setSpeed(0);
      left_motor.run(BACKWARD);
      right_motor.run(BACKWARD);
      delay(10000);
    }//end while at marker
  }//end if obstacle count == 1


  if (obstacleCount == 2) // CASE 3: 2 obstacles, detects both
  {
    leftTurn();

    obstacleLength(turnOne, turnTwo);

```

```
    LeftEncoderCount = 0;
    RightEncoderCount = 0;
    while (LeftEncoderCount < turnLength || RightEncoderCount < turnLength)
    {
      adjustSpeed(150);
    }
    leftTurn();
    LeftEncoderCount = 0;
    RightEncoderCount = 0;
    while (LeftEncoderCount < trackLength || RightEncoderCount < trackLength)
    {
      adjustSpeed(150);
    }

    leftTurn();
    LeftEncoderCount = 0;
    RightEncoderCount = 0;

    while (getAverage(A4) > .40 || getAverage(A4) < .30)
    {
      adjustSpeed(100);
    }//end while
    while (getAverage(A4) < .40 || getAverage(A4) > .30)
    {
      left_motor.setSpeed(0);
      right_motor.setSpeed(0);
      left_motor.run(BACKWARD);
      right_motor.run(BACKWARD);
      delay(10000);
    }//end while at marker

  }//end case 3: two obstacles

}//close return to start
```