BOOP Challenge Report MCU

Owen Cook & Alexa Witkin

October 17, 2025

Contents

1 Introduction

The BOOP challenge is a target acquisition challenge in which each group is given an Arduino car kit which must be programmed to find a target in the least amount of time possible. The arena will be two meters in diameter and each group's car will start in the same position. The target is roughly 5 cm wide and 20cm tall. Each group gets three attempts to find the target in three different. The sum of the two shortest attempts is the score.

The main components in the car kit to be used are the DC motors attached to the wheels, the ultrasonic sensor, IR line trackers, IR distance detectors, and a mini servo. Additional modifications were made with professor permission to enhance the design.

2 Hardware List

- Arduino board
- Computer & USB cable for programming/communication
- 2 DC motors + motor control shield
- IR object sensor
- IR line sensors
- Servo motor (SG90 or similar)
- Ultrasonic distance sensor (HC-SR04 or similar)
- Various other hardware: breadboard, wires, resistors, etc.
- Mechanics: robot platform, wheels, batteries, battery holder, etc.
- MPU-6500 (IMU on module board with integrated voltage regulator)

3 Assembled Car

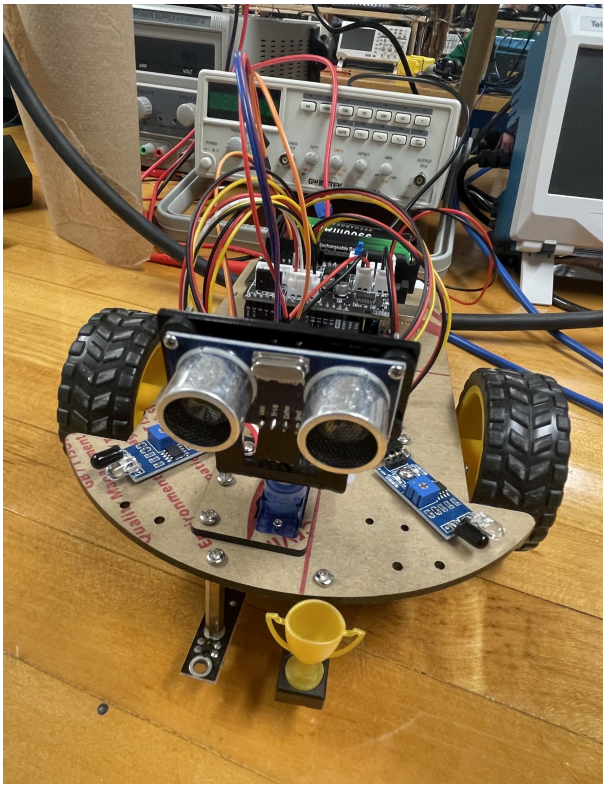

Figure 1: Car with First Place Trophy

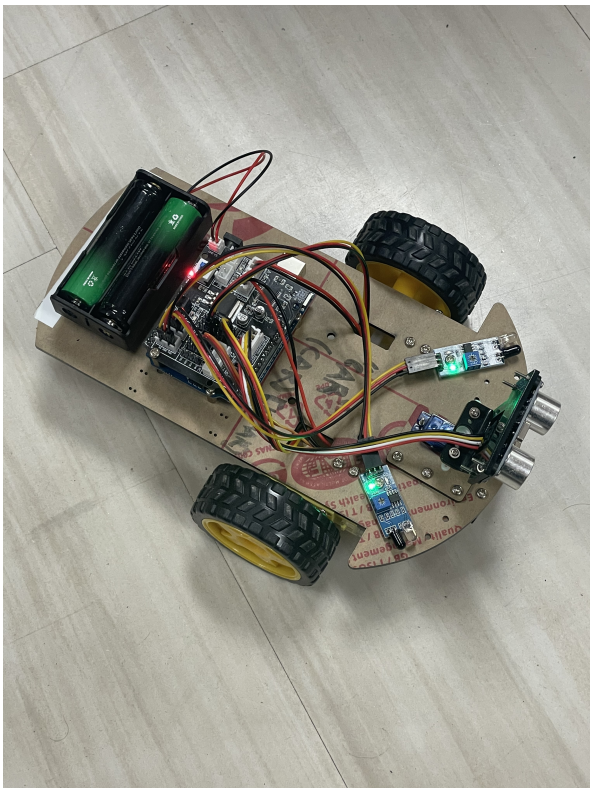

Figure 2: Top View of Car

Note: the accelerometer is not pictured, but was placed on a small proto-board in the middle of the car between the wheels

4 Design Approach

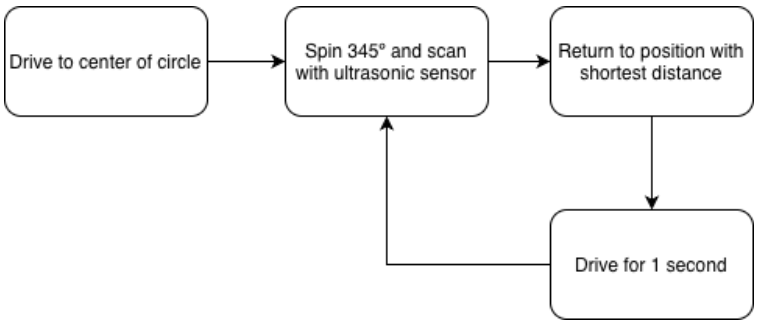

Figure 3: Block Diagram of Approach

The design approach that made this design stand out from other designs was the use of an accelerometer/gyroscope/magnetometer module. The general method of creating this project is to

turn the servo with the ultrasonic sensor attached to it to scan for the target, then move towards the target. The problem with this approach is the servo has a limited viewing angle ($0 - 180°$) and the servo turns relatively slowly.

The solution to this problem is turning the car instead of the servo when scanning for objects, which allows for faster and full 360° scanning.

The sequence the car will take to find the object consists of driving to the center of the circle, doing a full rotation while simultaneously scanning with the ultrasonic sensor, storing the angle the car was at when it saw the shortest distance, turning back to the position with the shortest distance, and finally driving in this direction for one second and repeating the process. The car does not do a full 360° rotation because of the 15° viewing angle of the ultrasonic sensor, it only does 345° to omit redundancy.

The line tracker modules were not used as if our car got as far as the edge of the circle, the target would not be found. There was also trouble with integration in the code, so it was omitted, but the functions can be found in the Appendix.

5 Code

The car kit consists of many components that require complex control, so the code was split into multiple ".ino" files with functions controlling wheels, accelerometer, and ultrasonic sensor. These functions would be called in the *main.ino* file. Ideally a *Car* class would have been made to control the entire car, but time did not allow.

5.1 Ultrasonic Sensor

```
unsigned long ultrasonicDistance(byte tPin, byte ePin) {
  // initialize variables
  static float duration, distance;

  digitalWrite(tPin, LOW);
  delayMicroseconds(2);
  digitalWrite(tPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(tPin, LOW);
  // set the duration as the measurement of the echoPin sensed as HIGH
  duration = pulseIn(ePin, HIGH);
  // set the distance as a function of the duration
  distance = (duration*.0343)/2;
  // display distance
  // Serial.print("Distance: ");
  // Serial.println(distance);
  delay(5);

  return(distance);
}
```

The ultrasonic sensor module works by sending an ultrasonic sound signal out, recording the time it takes to return, and using the speed of sound to translate to distance. The above code shows a function that takes in the pins that the ultrasonic sensor are connected to, sends a pulse by toggling the trigger pin from low to high to low in quick succession, then records how long the pulse took to go return. The math to find the distance consists of multiplying the speed of sounds by the time it took to travel and dividing this value by two because the time includes the time it

took for the pulse to get to the target and return to the sensor. Printing statements were included strictly for debugging as they added delay.

5.2 Accelerometer

The accelerometer module was find the angle of the car. Using the raw data proved to be difficult the only angles that could be measured simply with the Pythagorean theorem were the angles between the X and Z axis and Y and Z axis. To solve this, a library was found that uses differentiation and integration of the accelerometer and gyroscope readings to measure the relative angle of the device. Documentation for this library can be found in the Appendix.

```
int restrainAngle(int angle) {

  angle = angle % 360;

  if (angle < 0) angle += 360;

  return angle;
}
```

Three functions were included in this file. The first one restrains an angle from 0-360 to keep a consistent and logical angle scheme. The input to the function is an integer angle. The input gets modded by 360 to keep it between 0-359. Next, there is a check to see if the angle is negative, if it is, 360 is added to it, so -10 would become 350 for example. This allowed for easy scaling of angles.

```
 #include "Wire.h"
 #include <MPU6050_light.h>

 void mpuInit(MPU6050 &sensor) {
  byte status = sensor.begin();
  Serial.print("MPU6050 status: ");
  Serial.println(status);

  while (status != 0) { } // stop everything if could not connect to sensor6050
  Serial.println("Calculating offsets, do not move MPU6050");
  delay(200);

  sensor.calcOffsets(); // gyro and accelero
  Serial.println("MPU Callibrated and Setup");

}
```

Next, the I2C library and the accelerometer library mentioned were included into the sketch to create a function to initialize the sensor. This function takes in an MPU sensor as an input by reference, so it would edit the actual object, and calls all of the necessary functions to initialize, calibrate, and zero the device.

```
int getYaw(MPU6050 &sensor) {
  sensor.update();
  int yaw = sensor.getAngleZ();
  yaw = yaw % 360;
  if (yaw < 0) yaw += 360;
  return yaw;
}
```

To get the yaw of the vehicle, a function was created that took in an MPU object by reference, updated the sensor, and got the yaw of the sensor. This yaw was then mapped from 0-360 and returned.

5.3 Wheels

```
#ifndef DIRECTION_H
#define DIRECTION_H

enum Direction {
  FORWARD,
  BACK,
  LEFT,
  RIGHT,
  STOP
};

#endif
```

To start, an enum was created that handled the directions of the car. Using an enum over any other type made the code more readable and helped keep everything organized. This enum was kept in a header file and this header file was included in multiple sketches. A problem occurred where declaring the enum multiple times caused compiler errors, so this solved the problem.

```
#include "Direction.h"

void moveDir(byte pinRight, byte pinSpeedRight, byte pinLeft, byte pinSpeedLeft,
    Direction dir, int speed) {
  switch (dir) {
     case FORWARD:
      // Move forward
      digitalWrite(pinLeft, HIGH);
      analogWrite(pinSpeedLeft, speed - 2);
      digitalWrite(pinRight, LOW);
      analogWrite(pinSpeedRight, speed);
      break;

    case BACK:
      // Move backward
      Serial.println("REVERSE");
      digitalWrite(pinLeft, LOW);
      analogWrite(pinSpeedLeft, speed);
      digitalWrite(pinRight, HIGH);
      analogWrite(pinSpeedRight, speed);
      break;

    case LEFT:
      // Turn left
      Serial.println("LEFT TURN");
      digitalWrite(pinLeft, LOW);
      analogWrite(pinSpeedLeft, speed);
      digitalWrite(pinRight, LOW);
      analogWrite(pinSpeedRight, speed);
      break;

    case RIGHT:
      // Turn right
      Serial.println("RIGHT TURN");
      digitalWrite(pinLeft, HIGH);
      analogWrite(pinSpeedLeft, speed);
      digitalWrite(pinRight, HIGH);
      analogWrite(pinSpeedRight, speed);
      break;
```

```
39
40      case STOP:
41        // Stop all movement
42        Serial.println("STOP");
43        analogWrite(pinSpeedLeft, 0);
44        analogWrite(pinSpeedRight, 0);
45        break;
46    }
47  }
```

The first function created in the wheels file dealt with moving the car in a specified direction at a certain speed. The wheels are controlled with a TB6612FNG motor controller that has one pin for speed and one pin for direction. The function takes in the pin number for the direction and speed of each wheel as well as the values for direction of movement and speed.

A large switch case statement was created that contains the logic for moving the car in each specified direction at the given speed. The left and right wheels have motors mounted opposite to one another, so the pin activation is opposite to what would intuitively make sense.

The speed of the tires were offset slightly to account for skewing of the car and loose screws on the motor.

```
1  void sweepAndReturn(byte pinRight, byte pinSpeedRight, byte pinLeft, byte
       pinSpeedLeft,
2                  Direction initialDir, int speed, MPU6050 &sensor, int turnAngle,
3                  byte trig, byte echo) {
4
5    int currAngle = getYaw(sensor);
6    int shortestAngle = currAngle; // Initialize to avoid undefined behavior
7    unsigned long shortestDist = ultrasonicDistance(trig, echo);
8
9    // Determine endAngle based on requested scan direction (left or right)
10   int oprtr = (initialDir == RIGHT) ? -1 : 1;
11   int endAngle = restrainAngle(currAngle + oprtr * turnAngle);
12
13   // Begin turning
14   moveDir(pinRight, pinSpeedRight, pinLeft, pinSpeedLeft, initialDir, speed);
15
16   while (true) {
17     // Check if at each angle to see which angle saw the closest
18     // Object
19     currAngle = getYaw(sensor);
20     unsigned long distance = ultrasonicDistance(trig, echo);
21
22     // Update shortest distance if applicable
23     if (distance < shortestDist) {
24       shortestDist = distance;
25       shortestAngle = currAngle;
26     }
27
28     // Compute normalized angular difference to endAngle
29     int diff = restrainAngle(endAngle - currAngle);
30
31     // Check if we've reached the end of the sweep (  5   tolerance)
32     if (abs(diff) < 5 || abs(diff) > 355) {
33       moveDir(pinRight, pinSpeedRight, pinLeft, pinSpeedLeft, STOP, speed);
34       break;
35     }
36   }
37
```

```
38    // Determine which way to turn back to the
39    // shortest angle (one way will have a smaller angle)
40    int currYaw = getYaw(sensor);
41    int toTarget = restrainAngle(shortestAngle - currYaw);
42
43    Direction turnDir;
44    int turnAmount;
45
46    if (toTarget <= 180) {
47      turnDir = LEFT;
48      turnAmount = toTarget; // tune with -5 or something
49    } else {
50      turnDir = RIGHT;
51      turnAmount = 360 - toTarget; // tune with -5 or something
52    }
53
54    // Fine adjustment turn toward the target direction
55    moveDir(pinRight, pinSpeedRight, pinLeft, pinSpeedLeft, turnDir, speed / 2);
56    while (true) {
57      currYaw = getYaw(sensor);
58      int diff = restrainAngle(shortestAngle - currYaw);
59
60      if (abs(diff) < 5 || abs(diff) > 355) {
61        moveDir(pinRight, pinSpeedRight, pinLeft, pinSpeedLeft, STOP, speed);
62        break;
63      }
64    }
65
66  }
```

The above function combines functionality from the wheels, ultrasonic sensor, and the accelerometer. This function is the bulk of what the block diagram in Figure 3 shows. The function takes in the parameters (this would be cut down if it was a class) listed below:

- Pins to control right wheel
- Pins to control left wheel
- Direction for car to sweep
- Speed that car turns
- MPU sensor by reference
- Angle to turn (amount to sweep)
- Pins to control ultrasonic sensor

The code starts with making calculations to calculate the end angle that the car must turn to based on the input parameters. The car then begins turning in the specified direction. While the car is turning, the distance between a target is measured with the ultrasonic sensor. If a new shortest distance is found, the current angle and new shortest distance are stored. Lastly, there is a check to see if the car is within $\pm 5^\circ$ of the end angle and a command to stop it from turning if it is within this distance. The tolerance is included for error in the sensor and reading speed to make sure the car stops scanning and does not miss the end angle.

The next part of the function deals with turning back to the position where the shortest distance was seen. First, to optimize speed, the turn direction is specified by checking which direction has

the shortest turning distance. Next, the turn amount is calculated based on whether there is a left or right turn. Finally, logic is implemented to turn the car and constantly check if the car is within $\pm 5^\circ$ of the end angle.

At this point in the block diagram, the car would ideally be facing the target.

5.4 Main File

```
#include <MPU6050\_light.h>
#include <IRremote.h>
#include "Wire.h"
#include "Direction.h"

// ======== TIRES ======== //
const byte pinLeftTire  = 2;
const byte pinLeftSpeed = 5;

const byte pinRightTire  = 4;
const byte pinRightSpeed = 6;

// ======== MPU ======== //
MPU6050 mpu(Wire);

// ======== ULTRASONIC ======== //
const int pinTrigger = 12;
const int pinEcho = 13;

// ======== IR REMOTE ======== //
const byte pinIR = 3;
IRrecv irrecv(pinIR);
bool start = false;
```

The main file begins with defining the pins and variables for each sensor component. In addition to the aforementioned sensors, the IR receiver module is also used to begin the driving sequence which is started when the *start* variable is changed to true.

```
// ======== SETUP ======== //
void setup() {

  // Start I2C
  Wire.begin();

  // Tires pins as output
  pinMode(pinLeftTire, OUTPUT);
  pinMode(pinLeftSpeed, OUTPUT);
  pinMode(pinRightTire, OUTPUT);
  pinMode(pinRightSpeed, OUTPUT);

  // Ultrasonic sensor pins
  pinMode(pinTrigger, OUTPUT);
  pinMode(pinEcho, INPUT);

  // IR receiver
  irrecv.begin(pinIR);

  // Initialize MPU
  mpuInit(mpu);

  // Wait to receive any IR signal before starting
```

```
  while(!start) {
    if (irrecv.decode()){
      start = true;
      irrecv.resume();
    }
  }

  // Drive to the middle of the circle
  // Found via trial and error
  moveDir(pinRightTire, pinRightSpeed, pinLeftTire, pinLeftSpeed, FORWARD, 200);
  delay(1350);

}
```

The setup consists of pin setups, for each respective device and special initialization of other devices. The bottom of the setup handles waiting until a startup signal is detected to start the sequence. The IR receiver waits for any IR signal from the remote, and when one is received, the sequence begins. The sequence consists of driving straight for 1.35 seconds which gets the car roughly to the middle of the circle.

```
// ======== LOOP ======== //
void loop() {
  // Sweep to find direction of nearest object (faster 180   sweep)
  sweepAndReturn(pinRightTire, pinRightSpeed, pinLeftTire, pinLeftSpeed,
                 RIGHT, 60, mpu, 345, pinTrigger, pinEcho);

  // Delay to make sure car stopped moving
  delay(10);

  // Drive forward toward target for 1 second at 120 speed
  moveDir(pinRightTire, pinRightSpeed, pinLeftTire, pinLeftSpeed, FORWARD, 120);
  delay(1000);

}
```

The loop code is simple because of most of the code being handles in the wheels file. The *sweepAndReturn* function is called to turn 345° at a speed of 60 so the ultrasonic sensor can keep up with the speed that the car turns. Then there is a small delay to make sure the car stopped moving and is set to drive towards the target. Finally the car is set to move towards the target for 1 second which is about 3/4 of the way to the outside of the circle. This process is repeated until the target is booped.

6 Performance

Attempt #	1	2	3	Total Score
Time (s)	5.5	5.0	12.1	10.5

Table 1: Score for Competition

The competition went well, the second round performed the best likely because the target was closest in angle to the starting spin angle. The first trial was similar, placing the target close to the starting angle of scanning. The third trial, the target was out of range of one iteration of the sequence which caused adding extra time.

The author's group got first place, winning a tiny gold trophy as a award.

7 Conclusion

The BOOP challenge brought forth a complex challenge in finding a target fully autonomously. The target would be placed at random into a 2m diameter circle in which the car would start on the outside. Each group had three attempts, and the sum of the two lowest times gave the score. The hardware provided included an Arduino car kit with many components including: DC motors attached to wheels, DC motor drivers, a mini servo, an ultrasonic sensor, and an IR sensor. Modification of the vehicle was also allowed with instructor permission.

The approach taken was to drive to the center of the circle, do a 360° spin while scanning with the ultrasonic sensor, drive towards the target for one second, and repeat the scanning step. This process was implemented by creating functions for each component separated into separate Arduino files. Files were created for the wheels, ultrasonic sensor, line tracker, and an accelerometer/gyroscope that was added onto the vehicle to keep track of the angle. These functions were all combined in the main file that handled performing the sequence starting with a button press.

This approach balances both speed an accuracy, and for this reason it ended up winning first place in the competition.

8 Sources

Example Code for Measuring Yaw of Car

https://iotprojectsideas.com/measure-pitch-roll-and-yaw-angles-using-mpu6050-and-arduino/
https://github.com/rfetick/MPU6050_light/tree/master

9 Appendix/Full Code

9.1 main.ino

```
/*
  Owen Cook and Alexa Witkin
  ==========================
  Boop project main file
*/


#include <MPU6050_light.h>
#include <IRremote.h>
#include "Wire.h"
#include "Direction.h"

// ======== TIRES ======== //
const byte pinLeftTire  = 2;
const byte pinLeftSpeed = 5;

const byte pinRightTire  = 4;
const byte pinRightSpeed = 6;

// ======== MPU ======== //
MPU6050 mpu(Wire);

// ======== ULTRASONIC ======== //
const int pinTrigger = 12;
const int pinEcho = 13;

// ======== IR REMOTE ======== //
const byte pinIR = 3;
IRrecv irrecv(pinIR);
bool start = false;

// ======== SETUP ======== //
void setup() {

  // Start I2C
  Wire.begin();

  // Tires pins as output
  pinMode(pinLeftTire, OUTPUT);
  pinMode(pinLeftSpeed, OUTPUT);
  pinMode(pinRightTire, OUTPUT);
  pinMode(pinRightSpeed, OUTPUT);

  // Ultrasonic sensor pins
  pinMode(pinTrigger, OUTPUT);
  pinMode(pinEcho, INPUT);
```

```
47
48    // IR receiver
49    irrecv.begin(pinIR);
50
51    // Initialize MPU
52    mpuInit(mpu);
53
54    // Wait to receive any IR signal before starting
55    while(!start) {
56      if (irrecv.decode()){
57        start = true;
58        irrecv.resume();
59      }
60    }
61
62    // Drive to the middle of the circle
63    // Found via trial and error
64    moveDir(pinRightTire, pinRightSpeed, pinLeftTire, pinLeftSpeed, FORWARD, 200);
65    delay(1350);
66
67  }
68
69  // ======== LOOP ======== //
70  void loop() {
71    // Sweep to find direction of nearest object (faster 180   sweep)
72    sweepAndReturn(pinRightTire, pinRightSpeed, pinLeftTire, pinLeftSpeed,
73                   RIGHT, 60, mpu, 345, pinTrigger, pinEcho);
74
75    // Delay to make sure car stopped moving
76    delay(10);
77
78    // Drive forward toward target for 1 second at 120 speed
79    moveDir(pinRightTire, pinRightSpeed, pinLeftTire, pinLeftSpeed, FORWARD, 120);
80    delay(1000);
81
82  }
```

9.2 wheels.ino

```
1  // top view
2  // Left pos
3  // right neg
4  #include "Direction.h"
5
6  void moveDir(byte pinRight, byte pinSpeedRight, byte pinLeft, byte pinSpeedLeft,
       Direction dir, int speed) {
7    switch (dir) {
8       case FORWARD:
9        // Move forward
10       digitalWrite(pinLeft, HIGH);
11       analogWrite(pinSpeedLeft, speed - 2);
12       digitalWrite(pinRight, LOW);
13       analogWrite(pinSpeedRight, speed);
14       break;
15
16     case BACK:
17       // Move backward
18       Serial.println("REVERSE");
```

```
      digitalWrite(pinLeft, LOW);
      analogWrite(pinSpeedLeft, speed);
      digitalWrite(pinRight, HIGH);
      analogWrite(pinSpeedRight, speed);
      break;

    case LEFT:
      // Turn left
      Serial.println("LEFT TURN");
      digitalWrite(pinLeft, LOW);
      analogWrite(pinSpeedLeft, speed);
      digitalWrite(pinRight, LOW);
      analogWrite(pinSpeedRight, speed);
      break;

    case RIGHT:
      // Turn right
      Serial.println("RIGHT TURN");
      digitalWrite(pinLeft, HIGH);
      analogWrite(pinSpeedLeft, speed);
      digitalWrite(pinRight, HIGH);
      analogWrite(pinSpeedRight, speed);
      break;

    case STOP:
      // Stop all movement
      Serial.println("STOP");
      analogWrite(pinSpeedLeft, 0);
      analogWrite(pinSpeedRight, 0);
      break;
  }
}

void sweepAndReturn(byte pinRight, byte pinSpeedRight, byte pinLeft, byte
    pinSpeedLeft,
                Direction initialDir, int speed, MPU6050 &sensor, int turnAngle,
                byte trig, byte echo) {

  int currAngle = getYaw(sensor);
  int shortestAngle = currAngle; // Initialize to avoid undefined behavior
  unsigned long shortestDist = ultrasonicDistance(trig, echo);

  // Determine endAngle based on requested scan direction (left or right)
  int oprtr = (initialDir == RIGHT) ? -1 : 1;
  int endAngle = restrainAngle(currAngle + oprtr * turnAngle);

  // Begin turning
  moveDir(pinRight, pinSpeedRight, pinLeft, pinSpeedLeft, initialDir, speed);

  while (true) {
    // Check if at each angle to see which angle saw the closest
    // Object
    currAngle = getYaw(sensor);
    unsigned long distance = ultrasonicDistance(trig, echo);

    // Update shortest distance if applicable
    if (distance < shortestDist) {
      shortestDist = distance;
      shortestAngle = currAngle;
```

```
    }

    // Compute normalized angular difference to endAngle
    int diff = restrainAngle(endAngle - currAngle);

    // Check if we've reached the end of the sweep (  5   tolerance)
    if (abs(diff) < 5 || abs(diff) > 355) {
      moveDir(pinRight, pinSpeedRight, pinLeft, pinSpeedLeft, STOP, speed);
      break;
    }
  }

  // Determine which way to turn back to the
  // shortest angle (one way will have a smaller angle)
  int currYaw = getYaw(sensor);
  int toTarget = restrainAngle(shortestAngle - currYaw);

  Direction turnDir;
  int turnAmount;

  if (toTarget <= 180) {
    turnDir = LEFT;
    turnAmount = toTarget; // tune with -5 or something
  } else {
    turnDir = RIGHT;
    turnAmount = 360 - toTarget; // tune with -5 or something
  }

  // Fine adjustment turn toward the target direction
  moveDir(pinRight, pinSpeedRight, pinLeft, pinSpeedLeft, turnDir, speed / 2);
  while (true) {
    currYaw = getYaw(sensor);
    int diff = restrainAngle(shortestAngle - currYaw);

    if (abs(diff) < 5 || abs(diff) > 355) {
      moveDir(pinRight, pinSpeedRight, pinLeft, pinSpeedLeft, STOP, speed);
      break;
    }
  }

}
```

9.3 ultrasonic_function.ino

```
// Alexa Witkin
// EE3815 - Boop Project

// initialize pins
// const int trigPin = 12;
// const int echoPin = 13;

unsigned long ultrasonicDistance(byte tPin, byte ePin) {
  // initialize variables
  static float duration, distance;

  digitalWrite(tPin, LOW);
  delayMicroseconds(2);
  digitalWrite(tPin, HIGH);
```

```
  delayMicroseconds(10);
  digitalWrite(tPin, LOW);
  // set the duration as the measurement of the echoPin sensed as HIGH
  duration = pulseIn(ePin, HIGH);
  // set the distance as a function of the duration
  distance = (duration*.0343)/2;
  // display distance
  // Serial.print("Distance: ");
  // Serial.println(distance);
  delay(5);

  return(distance);
}
```

9.4 yaw_function.ino

```
/* Get tilt angles on X and Y, and rotation angle on Z
    Angles are given in degrees
 License: MIT

 // https://iotprojectsideas.com/measure-pitch-roll-and-yaw-angles-using-mpu6050-and-arduino/
 */
 #include "Wire.h"
 #include <MPU6050_light.h>

//  MPU6050 mpu(Wire);
//  unsigned long timer = 0;

int restrainAngle(int angle) {

  angle = angle % 360;

  if (angle < 0) angle += 360;

  return angle;
}

void mpuInit(MPU6050 &sensor) {
  byte status = sensor.begin();
  Serial.print("MPU6050 status: ");
  Serial.println(status);

  while (status != 0) { } // stop everything if could not connect to sensor6050
  Serial.println("Calculating offsets, do not move MPU6050");
  delay(200);

  sensor.calcOffsets(); // gyro and accelero
  Serial.println("MPU Callibrated and Setup");

}

int getYaw(MPU6050 &sensor) {
  sensor.update();
  int yaw = sensor.getAngleZ();
  yaw = yaw % 360;
  if (yaw < 0) yaw += 360;
  return yaw;
```

```
}
```

9.5 line_avoider_function.ino

```
// Alexa Witkin
// EE3815 - Boop Project

#include "Wire.h"
#include "Direction.h"

// initialize pins
int s_R = 8;   // sensor right
int s_L = 7;   // sensor left

void lineAvoidance(int r_sensor, int l_sensor) {
  static int sval_R = 0; // sensor value left
  static int sval_L = 0; // sensor value right

  sval_R = digitalRead(r_sensor);
  sval_L = digitalRead(l_sensor);

  if(sval_L == LOW && sval_R == LOW) {
    //forward(); // if the no line detected, call function to move forward
    Serial.println("NO LINE -- GO FORWARD");
    return;
  }

  if(sval_L == HIGH && sval_R == LOW) {
    //right(); // if the left sensor sees line, call function to move right
    Serial.println("LINE ON LEFT -- GO RIGHT");
    moveDir(pinRightTire, pinRightSpeed, pinLeftTire, pinLeftSpeed, RIGHT, 100);
  }

  if(sval_L == LOW && sval_R == HIGH) {
    //left(); // if the right sensor sees line, call function to move left
    Serial.println("LINE ON RIGHT -- GO LEFT");
    moveDir(pinRightTire, pinRightSpeed, pinLeftTire, pinLeftSpeed, LEFT, 100);
  }

  if(sval_L == HIGH && sval_R == HIGH) {
    //right(); // if both sensors see line, default to turn right
    Serial.println("LINE CROSSED -- GO RIGHT (180)");
    moveDir(pinRightTire, pinRightSpeed, pinLeftTire, pinLeftSpeed, RIGHT, 100);
  }

}
```

9.6 Direction.h

```
#ifndef DIRECTION_H
#define DIRECTION_H

enum Direction {
  FORWARD,
  BACK,
  LEFT,
```

```
8     RIGHT,
9     STOP
10 };
11
12 #endif
```