E11 Final Report

Gemini

Section #2

Aditya Khant and Daphne Poon

1. Abstract

We used 2 robots: A wall-hugger side robot that tackled the beacons on the side that needed to be bumped, and a center robot that flashed gold codes. We stacked the robots on top of each other and used a backward-slanting ramp to launch the center robot off the side robot. The side robot had a wall-hugger wheel to navigate the walls without any sensors. The center robot had 2 LEDs on each side to flash gold codes, as well as a tire bumper to absorb shocks. In the final competition, we reached the finals but placed second.

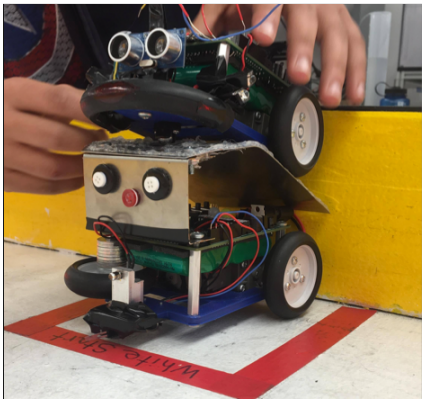

Figure 1 The starting configuration. The center robot sits on top of the side robot.

2. Introduction

Our mission was to mine as much of the rare material *Epsilonium* as possible, using the autonomous vehicles we built and coded. To win a round, we could need to claim more beacons than our opponents, with each beacon representing a station that contained *Epsilonium*. The game board (Figure 2) shows the starting state of the board. Each team (green or white) started in a square on the short-edge of the board. Beacons 1,2,3 and 4 were claimed by bumping a sensor on the front of the beacon. Beacons 5,6,7,8 and 9 required gold codes, a sequence of 1s

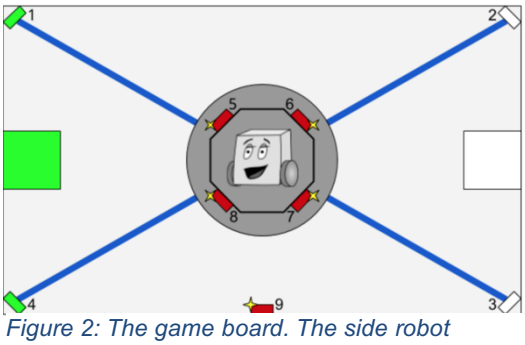

Figure 2: The game board. The side robot claimed beacons 1,2,3 and 4; the center robot claimed 5,6,7,8 and the elusive beacon 9.

and 0s, to be flashed towards the board, with the two team flashing difference codes. Teams scored points based on how many beacons they claimed at the end of 70 seconds, and whether they managed to return to the square they started in.

In this report, we will detail the physical modifications on our robots, the algorithms we used, how we performed in the competition and what we gained from this experience.

3. Physical modifications

3.1 Wall-hugger / Side Robot

Ramp: The ramp was made by bending a sheet of metal to a 30 degree angle and punching ¾" holes to allow for screws. The ramp was sloped backwards towards the wheels. It was mounted on a C shaped piece of metal made by bending sheet metal at 90 degrees. The C shaped metal bracket was fastened to the board with screws. The ramp was fastened to the metal bracket using screws and it was reinforced using hot glue. To increase friction on the ramp we made patterns out of hot glue.

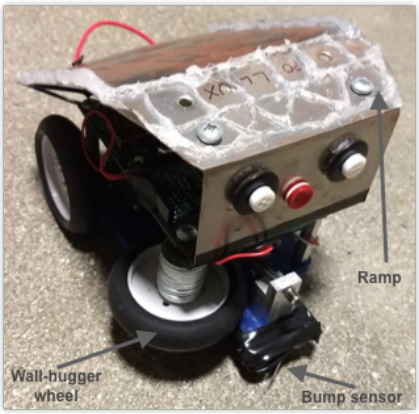

Figure 3 The side robot and its features.

There are two pairs of holes on the top of the ramp (Figure 3); the original pair caused the ramp to jut out too much and hit the corners of the game board, meaning the beacons there could not be bumped. We then punched another pair of holes closer to the front of the ramp, and cut off the front corners of the (originally rectangular) piece of sheet metal to minimize the chances of ramp interference.

Sideway wall-hugger wheel: We attached a wheel to the threaded rod on the front right corner. We used washers as spacers to restrict the wheels motion in the vertical direction. By coding the left wheel to have a slightly higher power than the right wheel, we allowed the robot to follow the wall at all times.

Bump sensor (limit switch) : We used a limit switch in the front of the robot to sense if the robot had bumped head on into the wall or another obstacle. This enabled the robot to turn away from the wall or obstacle after a collision.

We removed the following from the side robot because it was not needed or interfered with our ramp:

- Servo motor
- Distance Sensor
- Reflectance sensor

3.2 Center Robot

LEDs: We installed two LEDs on each side of the robot (Figure 4). We soldered the positive and negative pins of two LEDs to two wires, and that was then soldered onto male header pins. Each pair of header pins were then plugged into the Mudduino board. One pair of LEDs were connected to pin 5, and the other pair to 13 (Figure 5). This allowed us to program the LEDs separately without having to needing to free up one pin for each of the four LEDs.

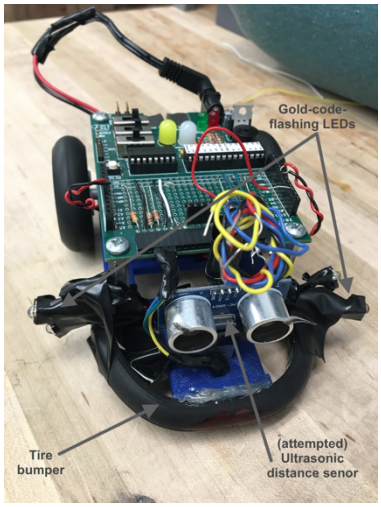

Figure 4 The center robot and its features.

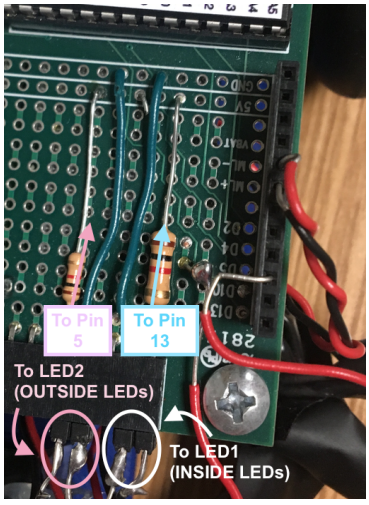

Figure 5 The LEDs on the board.

Tire bumper: We initially had issues with the LEDs accidentally grazing other robots or the beacons in the center of the board, which would cause them to start flashing gold codes at an inefficient angle. We decided to add a bumper to the front of the robot to protect the LEDs, and materials such as popsicle sticks and cut-open PVC pipes were considered. However, hot gluing a tire cut into half onto the chassis was the ideal solution, as it was easy to do, and the rubber of the tire would absorb shocks, protecting not only the LEDs, but the other components on board.

Ultrasonic distance sensor (attempted): Since the distance sensors provided to us was not accurate enough at close distances, we tried using an ultrasonic distance sensor (model HC-SR04) to detect whether there were other robots in our way, to help design an algorithm that could detect approaching vehicles and tell the center robot to move out of the way. However, we discovered a day before the final competition that the distance sensor was broken, possibly after being dropped on the floor. This rendered it unusable.

We removed the following from the center robot because it was not needed:

- Servo motor
- Distance Sensor
- Photosensor

Table 1: Bill of Materials

Component	Description	Supplier	Supplier Part #	Unit Price	Quantity	Total
R1-R7	5mm Red High Flux LED - 630 nm	DigiKey	HF5-R5590	$0.50	4	$2.00
C1	Wheel	DigiKey	P4525-ND	$0.18	2	$0.36
U1	Ultrasonic Sensor	Sparkfun	HC-SR04	$3.95	1	$3.95
					Grand Total	**$6.31**

4. Algorithms

4.1 Wall-hugger / Side Robot

1. Wait 1 second, drive forward and then turn right into the wall.
2. Drive forward until photosensor detects a gold code between 1 and 4 that is claimed by our team, or until it bumps into something
3. If it bumps into something, it stops and does a turn about a point to get itself unstuck
4. If it senses a gold code that we have claimed, it turns left so as to avoid bumping into a claimed Gold Code
5. Once 50 seconds are up, it attempts to return to base. For this, it tries to sense look for base 2 on team white, or base 4 on team green and halts forever once it detects them.

The drive forward was biased towards the right, which ensured that the side robot was always in contact with the wall. For detecting whether we had claimed the gold code or not, we modified the Homework code to return true if the team colors matched the color of the gold code it detected. Else it returned false. Due to the lack of a distance sensor, we used gold code correlation as a pseudo-distance measuring tool. At a correlation of 31, we were about 17 cm away from the source of the gold code.

4.2 Center Robot

1. Drive backward for 3 seconds, to ensure a successful drop and alignment.
2. Drive forward until the reflectance sensor reading exceeds 315 (black line), then turn right and go forward until the reflectance falls back under 315, using a while loop.
3. When the reflectance is under a threshold value, meaning the robot is over the white board, and drive left and forward until the reflectance is too high again, using a while loop.

The code for flashing gold codes is integrated into the while loops, so that every time the robot checks the reflectance sensor, it also flashes each gold code once. This was the only way we could integrate the code for moving the robot and the code for flashing gold codes. Though the line-following code is not perfect, the slight swerving actually helped us improve gold code flashing, because it allowed the LEDs to get close enough to the beacon.

The code for flashing used several nested loops.

1. The gold codes were hard-coded, and after checking pin 3 for whether the team was set to green or white, the code may enter a loop with inverted bits.
2. The gold codes were flashed twice in a row instead of one to increase reliability of flipping gold codes.

5. Results

In the scrimmage, we came second, winning one match, tying one, and losing one. This was due to issues with motor control; we only used the center robot, and it turned too much in the beginning and missed the black circle. It either turned in circles, as a result of failed line following code, or it hit the wall. At the time, the side robot wasn't ready and gold code flashing took an average of 5 rounds around the center to successfully flash the 4 center bases, while beacon 9, on the side, was a matter of chance.

In the final competition, we managed to come second overall, advancing to the finals. Several issues arose during the competition. Firstly, the side robot's ramp got caught on beacon 9 in the semi-final round. In the final round, the drop failed and the center robot was stuck on the ramp. We believe it was an issue of placing the wheel so that it caught on the edge of the ramp, and the hot glue caused it to stay on top. In addition, the side robot's return algorithm did not function in any of the rounds. Prior to this, during testing, we had not encountered these issues and our robot had a 100% success rate on these tasks.

Our robots were capable of winning all the 10 points, had there been no interference on the board. There were several instances when one robot would crash into an opponent's robot, causing both to be stuck. Luckily, this would often allow our other robot to claim their beacons without disruption. Unfortunately, the robots were limited as they were not able to react to external robots, and many of our functions were coded with ideal conditions in mind, which is never the case in real life.

6. Lessons Learned and Improvements

6.1 Lessons Learned

Leading up to the competition, we learned that duct tape and hot glue are wonderful materials, and surprisingly reliable when screws are just too much work. We also realized that it is better to be redundant, because what can go wrong will go wrong, and conditions are highly unpredictable. External factors, such as other robots, could affect your vehicles, and it is important to keep that in mind when designing and coding. If a code that is supposedly simple has not worked after four hours, try rewriting it again with a different approach, instead of trying to debug the same thing repeatedly.

6.2 Improvements

Given more time, we would have made the ultrasonic work on both robots. We could also have the wall-hugger to run both clockwise and counterclockwise around the board. However, the most important thing would be to perfect the initial drop, as well as the final return of the side robot. We would also like to use stronger LEDs on the center robot, and reduce the time needed to claim all of the center beacons.

APPENDIX A:

SIDE ROBOT

Side.ino :

```
#define boopPin 13 //Pin to read the bump sensor value
#define boopOut 5 //Pin to send current to bump sensor
#define returnTime 50000 // Time in ms to return back to base

// Initialization of global variables
int x = 0;
int m = 0;
bool isWhite = true;
long whendidwestart = millis();

void setup() {
  long startTime = millis(); // Set starttime for gold code detection
  bool letsReturn = false; //To return or not to return
  Serial.begin(9600); // initialize Serial
  //Initializes pins
  pinMode(3, INPUT);
  pinMode(boopPin, INPUT);
  pinMode(boopOut, OUTPUT);
  digitalWrite(boopOut, HIGH);
  whendidwestart = millis();
  isWhite = digitalRead(3); //Checks if team is green actually because the LEDs were wired
reversely
  initMotors(); //Initializes motor pins
  //initUS();
  setupMove(); //Moves forward and then turns right


}

void loop() {
  workingWithReturn(); // Main function that worked with return

}

void workingWithoutReturn(){// Main function without returning

  while (!(detectGoldCode()) and !(digitalRead(boopPin))) { //While it didnt detect a gold code or
didnt bump into anything
    motorControl(255, 240); // Go almost straight slightly towards the right
  }
  if (detectGoldCode() ) { // If it detects a gold code claimed by us
    runAway(); // Turn away
  }
  if (digitalRead(boopPin)) { //If it bumps into something
    halt(); //stop
    while (digitalRead(boopPin)) { //while its bumped
      motorControl(255, -255); //turn about a point with full power.
      delay(400);
    }
  }

}
```

```
void workingWithReturn(){

  while (!(detectGoldCode()) and !(digitalRead(boopPin))) {//While it didnt detect a gold code or
didnt bump into anything

    motorControl(255, 240); // Go almost straight slightly towards the right
    returnStrat(); // try returning back to base

  }
  if (detectGoldCode() ) {// If it detects a gold code claimed by us

    returnStrat(); // try returning back to base
    runAway(); // Turn away
  }
  if (digitalRead(boopPin)) { //If it bumps into something
    halt(); // stop
    while (digitalRead(boopPin)) { //while its bumped
      motorControl(255, -255); //turn about a point with full power.
      delay(400);
    }
  }

}
```

GoldCodes.ino :

```
// Name: Aditya Khant . Worked with Daphne Poon

//Initial constants
#define PHOTSENSOR 19
#define sampLength 31
#define gcLen 31
#define lengthGoldCodes 31
#define numGoldCodes 9
#define returnCor 30

//initialize all variables
unsigned long sampleStart = 0;
int lastdigit = 0;
int rawSamples[sampLength] = {0, 0 , 0 , 0 , 0, 0, 0 , 0 , 0 , 0, 0, 0 , 0 , 0 , 0, 0, 0 , 0 , 0 ,
0, 0, 0 , 0 , 0 , 0, 0, 0 , 0 , 0 , 0, 0};
int mainSamples[sampLength] = {0, 0 , 0 , 0 , 0, 0, 0 , 0 , 0 , 0, 0, 0 , 0 , 0 , 0, 0, 0 , 0 , 0 ,
0, 0, 0 , 0 , 0 , 0, 0, 0 , 0 , 0 , 0, 0};
int WGC[numGoldCodes][sampLength] =
{ {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1},
  {1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0},
  {0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0},
  {1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1},
  {0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1},
  {1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0},
  {0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0},
  {1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1},
  {0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1}
};
int returnVal[2] = {0, 0};

int d[sampLength];
int e[sampLength];
int b[sampLength];
```

```
int c[sampLength];
int arrayw[numGoldCodes];
int D1, D2;
int lastdigit1, lastdigit2;
int tempWGC[sampLength];
char color = 'w';

int goldCodes[numGoldCodes][gcLen];
int goldcTemp[gcLen];



boolean detectGoldCode() {
  //run all functions
  sampleCollect();
  toBinary();
  correlateR();
  //pnt();
  if (abs(D1) > 30 and D2 < 5) {//If correlation is above 30, and Gold Code is below 5
    if (color == 'w') { //If the color of gold code is white
      if (isWhite) { //If we are team green
        return false; //return false
      } else {
        return true; // return true
      }
    } else if (color == 'g') {//if the color of the gold code is green
      if (isWhite) { // If we are team green
        return true; // return true
      } else {
        return false; // else return false
      }
    } else {
      return false; // else return false
    }
  } else {
    return false; //else return false
  }
}

void sampleCollect() {
  int i = 0; //initialize counter
  sampleStart = micros(); // initialize start time
  while ( i < sampLength) { //loop until i < 31
    if (micros() - sampleStart > 250 * i ) { // tries to find intervals of time that are multiples
of 250
      rawSamples[i] = analogRead(PHOTSENSOR); //stores the value of the photosensor in raw Samples
      i++; //increase the counter

    }
  }


}

void toBinary() { //convert the raw values to binary
  //initialize variables
  int average = 0;
  int sum = 0;
  //find the sum
```

```
  for (int i = 0; i < sampLength ; i++) {
    sum += rawSamples[i];
  }
  //find the average
  average = sum / (sampLength );
  //if value is below average, set it to 1, else, set it to 0
  for (int i = 0; i < sampLength ; i++) {
    if (rawSamples[i] < average) {
      mainSamples[i] = 1;
    } else {
      mainSamples[i] = 0;
    }

  }
}

void correlateR()
{
  // this takes g and finds the gold code of index g and correlates it with m
  for (int g = 0; g < numGoldCodes; g++)
  {
    for (int i = 0; i < sampLength; i++) // copies gold codes into arrays tempGC and b.
    {
      tempWGC[i] = WGC[g][i];
      b[i] = mainSamples[i];
    }

    for (int i = 0; i < sampLength; i++)
      // run shift() through 31 times.
      // also prints the correlated value each time.
    {
      // sets up an array that is the correlated version of b and tempGC, with tempGC being shifted
31 times.
      d[i] = correlate(b, tempWGC);
      shift();
    }

    // sets the maximum correlated value forcolors with the maximum aboslute value
    arrayw[g] = maximum2(d, sampLength);

  }
  maximum(arrayw, numGoldCodes);
  D1 = returnVal[0] ; // Get the correlation value
  D2 = returnVal[1] + 1; // Get the ID

  if (D1 < 0) { // set the color to green if the value is negative, else set the color to positive
    color = 'g';
  } else {
    color = 'w';
  }

}

void pnt() {
  //Print out all the relevant values
  Serial.print("ID Number: ");
  Serial.println(D2);
  Serial.print("The corresponding correlation: ");
  Serial.println(D1);
```

```
  Serial.print("Team color: ");
  //If the color is w print "White" else print "Green"
  if (color == 'w') {
    Serial.println("WHITE");
  }
  else if (color == 'g') {
    Serial.println("GREEN");
  }
  else {
    Serial.println("something went wrong");
  }

}

void shift()
{
  //takes the last digit of the array and saves it
  lastdigit1 = tempWGC[sampLength - 1];

  // shifts everything in the sequence one right
  for (int q = (sampLength - 1); q >= 0; q--)
  {
    tempWGC[q] = tempWGC[q - 1];

  }
  // takes the saved digit and moves it to the first index
  tempWGC[0] = lastdigit1;

}

// takes 2 arrays and correlates them, and then outputs the dot product
int correlate(int j[sampLength], int k[sampLength])
{
  //defines two variables
  // i is iterated
  // val is the value (dot product) which is the sum of all the ...
  // ... dot products of each index under j[i] and k[i]
  int i = 0;
  int val = 0;
  while (i < sampLength) // iterating i 31 times to correlate
  {
    if (j[i] != k[i]) // if the one value DOES NOT equal the other value
    {
      val = val - 1 ; // take away one point
    }
    else // if it does
    {
      val = val + 1 ; // add a point
    }
    i++;
  }

  return val;
}

void maximum(int x[], int y) //finds the value of d with maximum absolute value
{
  // defines variable maxx
  int maxx = 0;
```

```
  // loops a

  for (int a = 0; a < y; a++)
  {
    if (abs(x[a]) > abs(maxx)) {
      maxx = x[a];
      returnVal[0] = x[a]; // if it's bigger than the stored maximum value, replace maxx with x[a]
      returnVal[1] = a;
    }

  }

}

int maximum2(int x[], int y) //finds the value of d with maximum absolute value
{
  // defines variable maxx
  int maxx = 0;
  int returnthis = 0;
  // loops a

  for (int a = 0; a < y; a++)
  {
    if (abs(x[a]) > abs(maxx)) {
      maxx = x[a]; // if it's bigger than the stored maximum value, replace maxx with x[a]
      returnthis = a;
    }

  }
  return maxx;
}

//Return Strategy
void returnStrat() {
  if (millis() - whendidwestart > returnTime) {

    if (!isWhite) {
      if (D2 == 2 and abs(D1) > returnCor) {

        while (true) {
          halt();
        }
      }

    } else {
      if (D2 == 4 and abs(D1) > returnCor) {
        while (true) {
          halt();
        }
      }
    }
  }

}
```

Ultrasonic.ino :

```
//ULTRASONIC SENSOR CODE - WE DIDNT USE THIS BECAUSE IT DIDNT WORK
```

```
//Initialize pins
#define trigpin 5
#define echopin 10

void initUS(){
  pinMode(trigpin, OUTPUT);
  pinMode(echopin, INPUT);
}


float calcDist(){
  //Initial Variables
  float cmVal = 1.0;
  float cmCon = (340.0/10000.0)/2.0;
  float inCon = (1.0/2.54);
  int durationIn = 1;
  //First write low and then write High after 2 microseconds
  digitalWrite(trigpin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigpin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigpin, LOW);
  //use pulsein to get a reading
  durationIn = pulseIn(echopin, HIGH, 50000);
  //convert reading to cm
  cmVal = cmCon * durationIn;
  //inchVal = cmVal * inCon;
  // Serial.print("Distance in cm: ");
  Serial.println(durationIn);
  return cmVal;
  //Serial.print("Distance in inch: ");
  //Serial.println(inchVal);
}
```

motors.ino :

```
//Initialize Pins
#define RPLUS 7
#define RMINUS 12
#define LPLUS 9
#define LMINUS 8
#define LEN 6
#define REN 11

// int powerLevel = 255;

void initMotors(){ // Initialize Motors
  pinMode(RPLUS, OUTPUT);
  pinMode(RMINUS, OUTPUT);
  pinMode(LPLUS, OUTPUT);
  pinMode(LMINUS, OUTPUT);
  pinMode(LEN, OUTPUT);
  pinMode(REN, OUTPUT);
}
void halt(){ //Halt all motors
  analogWrite(LEN, x);
  analogWrite(REN, m);
  digitalWrite(LPLUS, LOW);
```

```
  digitalWrite(LMINUS, LOW);
  digitalWrite(RPLUS, LOW);
  digitalWrite(RMINUS, LOW);
}

void forward(){ // Go Forward
  analogWrite(LEN, x);
  analogWrite(REN, m);
  digitalWrite(LPLUS, HIGH);
  digitalWrite(LMINUS, LOW);
  digitalWrite(RPLUS, HIGH);
  digitalWrite(RMINUS, LOW);
}

void backward(){ // Go Backwards
  analogWrite(LEN, 225);
  analogWrite(REN, 225);
  digitalWrite(LPLUS, LOW);
  digitalWrite(LMINUS, HIGH);
  digitalWrite(RPLUS, LOW);
  digitalWrite(RMINUS, HIGH);
}

void turnL(){ //Turn Left
  analogWrite(LEN, x);
  analogWrite(REN, m);
  digitalWrite(LPLUS, LOW);
  digitalWrite(LMINUS, HIGH);
  digitalWrite(RPLUS, HIGH);
  digitalWrite(RMINUS, LOW);
}

void turnR(){ //turn Right
  analogWrite(LEN, 255);
  analogWrite(REN, 255);
  digitalWrite(LPLUS, HIGH);
  digitalWrite(LMINUS, LOW);
  digitalWrite(RPLUS, LOW);
  digitalWrite(RMINUS, HIGH);
}

void motorControl(int y, int z){ //Custom motor control to control each individual motor
  if (y > 0){ // if y is positive
  analogWrite(LEN, y); // Write power to Left
  //Go forward
  digitalWrite(LPLUS, HIGH);
  digitalWrite(LMINUS, LOW);

  } else {
  analogWrite(LEN, abs(y));//Write abs power to Left
  //Go back
  digitalWrite(LPLUS, LOW);
  digitalWrite(LMINUS, HIGH);

  }

  if (z > 0){ //if z is positive
  analogWrite(REN, z);// Write power to Z
  //Right go forward
```

```
  digitalWrite(RPLUS, HIGH);
  digitalWrite(RMINUS, LOW);

  } else {
    analogWrite(REN, abs(y)); //write abs power to Right
  //Left go forward. While this was a coding error, fixing it worsened our robot.
  digitalWrite(LPLUS, LOW);
  digitalWrite(LMINUS, HIGH);

  }
}

void setPowerLevel(int pwr){ // Motor Power Library
  x = pwr;
  m = x * 0.75;
}

void runAway(){ //Stop, turn left, run into the wall
  halt();
  motorControl(0, 255);
  delay(450);
  motorControl(255, 200);
  delay(1500);
  halt();

}

void setupMove(){// Wait for 1s go straight, and then turn Right into the wall
  delay(1000);
  motorControl(255, 255);
  delay(1000);
  turnR();
  delay(300);

}
```

sensors.ino :

```
// NOTE: NOT USED FOR SIDE ROBOT

#define DISTSENSOR 14
#define REFLSENSOR 18
#define CREFLSENSOR 16


// values of reflectance for arena
int rblack = 323;
int rwhite = 52;
int rred = 65;
int rblue = 230;

int dist;
int phot;
int refl;

void testSensors() {//Outputs value for refl sensor

  pinMode(REFLSENSOR, INPUT);
```

```
  refl = analogRead(REFLSENSOR-14);

  Serial.print("Reflectance: ");Serial.println(refl);

  delay(1500);
}

void fwd(){ //go forward
  pinMode(REFLSENSOR, INPUT);
  setPowerLevel(90);
  while(analogRead(REFLSENSOR-14) < 300){
     forward();
  }
  halt();
}

void initialturn(){//First turn while line following
  pinMode(REFLSENSOR, INPUT);
  while(analogRead(REFLSENSOR-14) > 300){
    turnR();
  }
  halt();

}
void turnonblack(){ //turn right on black
  pinMode(REFLSENSOR, INPUT);
  while(analogRead(REFLSENSOR-14) > 300){
    motorControl(140,70);
  }
  halt();

}

void turnonwhite(){ // turn left on white
  pinMode(REFLSENSOR, INPUT);
  while(analogRead(REFLSENSOR-14) < 300){
     motorControl(70,140);
  }
  halt();

}
```

CENTER ROBOT

CENTER.ino :

```
int x = 0;
int m = 0;
#define trigpin 17
#define echopin 16
// sets up pins and variables
// x and m were initially used in a function called setMotorPower
// to change the power output of the motors, but because we ended up using motorControl,
// variables x and m were only kept for the halt() function.

void setup() {
  Serial.begin(9600); // initialize Serial
  initMotors(); // initialize the motors
        gcsetup(); //sets up the gold code
```

```
  jumpoff(); //lets the center robot roll off the back of the side robot
  fwd(); //drives forward until a black line is hit
  halt(); //it stops instantly
  delay(5); //and waits for a brief moment
//
}

void loop() {
   turnonblack(); //if the reflectance sensor sees black, it drives right and forward
   turnonwhite(); //if it sees white, it drives left and forward
}
```

goldcodes.ino :

```
int LED1 = 13; //Inside
int LED2 = 5; //Outside
boolean invertBits = false; //we define invertBits, which is false by default
// hard-coding the gold codes
boolean goldCodes[9][31] = {{0,0,0,0,0,0,0,1,0,0,0,1,1,0,1,1,0,0,0,0,1,1,0,0,1,1,1,0,0,1,1},
                            {1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,1,0,0,0,1,1,1,1,0,0,0,1,0,1,0,0},
                            {0,1,0,0,0,0,1,1,0,1,0,0,0,0,1,0,1,1,1,1,1,1,1,0,1,0,1,0,1,1,1,0},
                            {1,0,1,0,0,0,0,0,0,0,1,1,0,1,1,1,1,1,1,1,1,0,1,0,0,0,0,1,1,1,0,1},
                            {0,0,1,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,0,1,0,1,1,1,0,1,0,0,1,1,1},
                            {1,1,1,0,0,0,1,0,0,1,1,0,1,1,1,0,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0},
                            {0,1,1,0,0,1,1,0,1,1,0,1,1,1,0,1,1,1,1,0,0,1,1,0,1,1,1,1,0,1,0},
                            {1,0,0,1,0,1,1,1,0,1,1,0,0,1,1,1,0,1,1,0,0,0,1,0,0,1,0,0,0,1,1},
                            {0,0,0,1,0,0,1,1,1,1,0,1,0,1,0,0,1,0,0,0,0,0,0,1,0,0,1,1,0,0,1}};
void gcsetup(){
  pinMode(3, INPUT);
  pinMode(LED1, OUTPUT); //initializes the pin for the inside LEDs
  pinMode(LED2, OUTPUT); //initializes the pin for the outside LEDs
  digitalWrite(LED1, LOW); //sets all 4 LEDs as low
  digitalWrite(LED2, LOW);
  invertBits = !(digitalRead(3));
  // If pin 3 is low (i.e. we are on team green), set invertBits as 1.
  // Otherwise, if we are on team white, there is no need to invert the bits.
}

void gcloop() {
  if (invertBits == false){ // if we are on team white
    for(int i=4; i<8; i++)
    //this loops through gold codes 5,6,7,8, which are on indexes 4 through 7.
    {
      for(int y=0; y<2; y++) //loops the gold code twice (flashing 5,5,6,6,7,7,8,8)
      {
        for(int z=0; z<31; z++) //runs through the 31 registers in one gold code
        {
            long startTime = micros(); //sets the variable startTime as the current time, in
microseconds
            while(micros() - startTime <= 250) //while 250 microseconds have NOT elapsed
            {
             if (goldCodes[i][z] == 1) // if the register on the iterating code we are currently
flashing is 1
                  {
                    digitalWrite(LED1, HIGH); // set the inner LED to be on high
                    if (goldCodes[8][z] == 1){ //we are constantly flashing gold code 9 (index 8)
on the outside LED
```

```
                    digitalWrite(LED2, HIGH); //if the register is equal to 1, set the outside
LED on high
                  }
                  else{
                    digitalWrite(LED2, LOW); //otherwise, have it be on low.
                  }
                }
             else //this does the same thing as the above loop, except this is run when the
register bit is 0.
                {
                  digitalWrite(LED1, LOW);
                  if (goldCodes[8][z] == 1){
                    digitalWrite(LED2, HIGH);
                  }
                  else{
                    digitalWrite(LED2, LOW);
                  }
                }
            }
          }
      }
    }
  }

  else{ // if we are on team green (this does everything the above code does, with minor
differences)
    for(int i=4; i<8; i++)
    {
      for(int y=0; y<2; y++)
      {
        for(int z=0; z<31; z++)
        {
            long startTime = micros();
            while(micros() - startTime <= 250)
            {
             if (goldCodes[i][z] == 1)
                  {
                    digitalWrite(LED1, LOW); //now, if the iterating register bit is 1, set the LED
to be low instead of high
                  if (goldCodes[8][z] == 1){
                    digitalWrite(LED2, LOW); //same thing, but for the outside LED
                  }
                  else{
                    digitalWrite(LED2, HIGH);
                  }
                }
             else
                  {
                    digitalWrite(LED1, HIGH);
                    if (goldCodes[8][z] == 1){
                      digitalWrite(LED2, LOW);
                    }
                    else{
                      digitalWrite(LED2, HIGH);
                    }
                  }
            }
          }
        }
```

```
      }
  }

}
```

motors.ino :

```
#define RPLUS 7
#define RMINUS 12
#define LPLUS 9
#define LMINUS 8
#define LEN 6
#define REN 11
//defines the above pins, which we later use to control the motors.


void initMotors(){ //sets all of the motor pins as output pins
  pinMode(RPLUS, OUTPUT);
  pinMode(RMINUS, OUTPUT);
  pinMode(LPLUS, OUTPUT);
  pinMode(LMINUS, OUTPUT);
  pinMode(LEN, OUTPUT);
  pinMode(REN, OUTPUT);
}

void halt(){ // turns off all motors to stop the robot.
  analogWrite(LEN, x);
  analogWrite(REN, m);
  digitalWrite(LPLUS, LOW);
  digitalWrite(LMINUS, LOW);
  digitalWrite(RPLUS, LOW);
  digitalWrite(RMINUS, LOW);
}

void motorControl(int y, int z){
  //drives forward, but because the motor power levels can be unequal, it could also turn one way
  analogWrite(LEN, y);
  analogWrite(REN, z);
  digitalWrite(LPLUS, HIGH);
  digitalWrite(LMINUS, LOW);
  digitalWrite(RPLUS, HIGH);
  digitalWrite(RMINUS, LOW);
}

void revControl(int y, int z){ // same as motorControl, but for driving backward
  analogWrite(LEN, y);
  analogWrite(REN, z);
  digitalWrite(LPLUS, LOW);
  digitalWrite(LMINUS, HIGH);
  digitalWrite(RPLUS, LOW);
  digitalWrite(RMINUS, HIGH);
}
```

sensorsandmovement.ino :

```
#define DISTSENSOR 14
#define REFLSENSOR 18
#define CREFLSENSOR 16
#define PHOTSENSOR 19
int dist;
int phot;
```

```
int refl;

void testSensors() { //this code was just used to read the reflectance values directly for
calibration
  pinMode(REFLSENSOR, INPUT);
  refl = analogRead(REFLSENSOR-14);
  Serial.print("Reflectance: ");
  Serial.println(refl);
  delay(1500);
}

void jumpoff(){ //the center robot would drive backward for 2.5 seconds, then stop.
  // this gave it ample time to fall off the back of the side robot and align itself with the wall.
  revControl(230,205);
  delay(2500);
  halt();
}

void fwd(){
  pinMode(REFLSENSOR, INPUT); //initializes the reflectance pin as input
  motorControl(100,70); //drives forward
  //(recall the left motor is weaker than the right one,
  // so we need a higher power level input on the left for the robot to drive forward)
  while(analogRead(REFLSENSOR-14) < 315){ //before hitting the black circle
    motorControl(100,70); //keep driving forward
  }
  halt(); //once the circle has been hit, and the loop is exited, stop
  revControl(50,100); //drive back briefly (this helps avoid colliding with the central blocks)
  delay(200);
}

void turnonblack(){
  while(analogRead(REFLSENSOR-14) > 315){ //while on the black circle
    motorControl(120,60); //drives right and forward
    gcloop(); // flashes one loop of the gold codes
  }
  halt(); //stops
}

void turnonwhite(){
  while(analogRead(REFLSENSOR-14) < 315){ //while on the white part of the board
     motorControl(60,120); //drives left and forward
     gcloop(); // flashes one loop of the gold codes
  }
  halt(); //stops
}
```