E11 Final Report

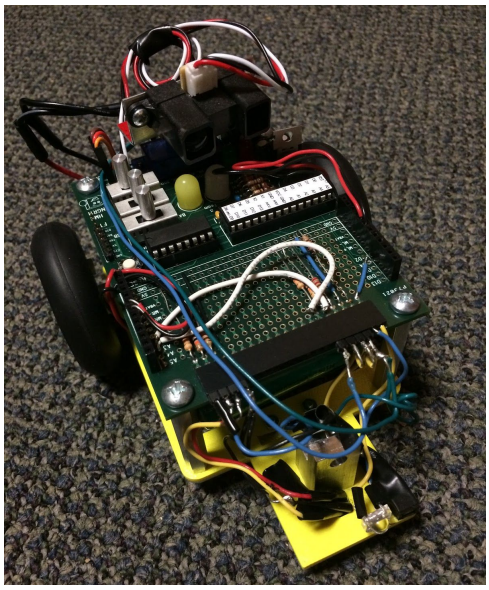
Team WIBSTR
Section 1, 3
Elizabeth Poss and Evan Amason

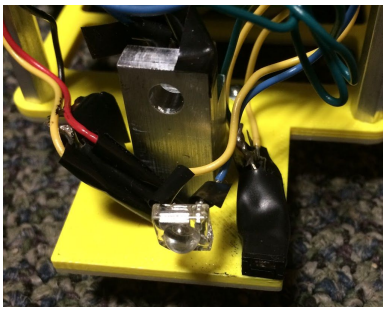
1. ABSTRACT

Our team built and designed a small autonomous robot capable of locating and interacting with beacons positioned on a mapped space. This report outlines the hardware modifications made to the robot, and the algorithm used to program its movement. Our hardware modifications centered on the the addition of a second sensor array designed to aid in navigation, while the robot's navigation programing used line following technique to trace an optimal route through the beacons. These additions aided our robot in the competition, though in practice our robot failed to achieve the goals set out in the code.

2. INTRODUCTION

Our goal was to design a robot capable of finding and interacting with specific points on a mapped terrain. In practice this took the form of building and coding a small robot that would tag beacons on a game board. These beacons could be activated by force- "bump beacons," or by broadcasting binary sequences called gold codes at the beacon with a diode- "flash beacons." Beacons were assigned point values, and the robots were allowed 70 seconds to interact with the game board, competing against the other robot to gain the highest number of points. The completed robot followed the class guidelines, mounting a battery pack and Mudduino controller atop a gearbox and wheels. It interacted with the game board using a navigation code we designed.





Our hardware modifications included additional sensors and changes to the front of the robot, designed to allow it to better interact with the beacons. As seen in the detail, a second reflectance sensor is attached to the front of the robot in place of a bumper (black, right side), and there is a second backfacing photosensor (not shown), as well as a diode for flashing gold codes (clear, center). These hardware modifications improved the robot's ability to avoid collisions and determine its position.

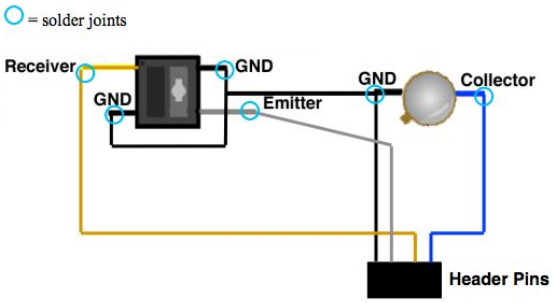
Our movement algorithm took advantage of the fact that we knew the placement and type of each beacon to direct the robot on an optimal route through the beacons. Given the starting position and the location of each beacon, our robot relied primarily on the basic line-following code to make its way from the starting square to the first flash beacon. From there it would follow the blue line to the bump beacon in the inner circle, then move across the board to the other blue line, which it would follow to get the other two beacons on that side of the board. The robot would then follow a similar pattern on the other team's side of the board, and repeat these processes until the match was over. While it never addressed the bonus beacon, this algorithm succeeded in getting the robot to all of the remaining beacons.

The rest of this report will describe in detail the hardware modifications made to the robot, and fully describe the algorithm used to control the robot. We will also discuss the robot's performance, both in testing and the competition, and outline improvements for future developments.

3. PHYSICAL MODIFICATION

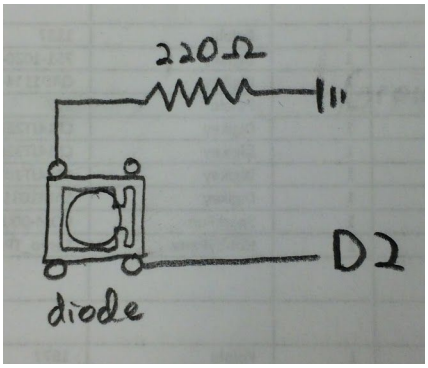
We made three main physical modifications to our robot. The first was to duplicate the reflectance/photosensor circuit already on the robot (shown in the diagram below). This allowed us to position a reflectance sensor on the front of the robot. This was useful because of the

inaccuracy of the distance sensor at short distances. Because we relied on the line following code, instead of distance from the walls, to navigate the robot, the only use we had for a distance sensor was to check if the robot was physically touching anything. Ultimately we decided to add a second reflectance sensor rather than relying on the distance sensor. Because any object the robot could run into registered a very different reflectance than open space, we were able to use the reflectance sensor to alert the robot if it had crashed. (This was executed with the wallCheck() function, which can be seen throughout the navigation code.)



While not used in the match, the second photosensor was added at the rear of the robot. This was intended to allow the robot to scan behind it for gold codes. This would have aided our alternate method of navigation, which relied on the robot turning around to scan for gold codes, then following those detected beacons. Fortunately, our robot had a correct gold code detection range of about half the board length, though this method of navigation was rejected because too great a precision was necessary in angling the photosensor toward the beacon.

Our second modification was originally accidental, but proved useful in the mounting of the second reflectance sensor. When the chassis was originally printed, it was of a saved file that didn't include the final step- adding the bumper to the front of the chassis. Instead, this robot just maintains original thickness to the front of the robot. This change allowed us to mount the reflectance sensor flat on the robot chassis. However, in future iterations this modification could simply be replaced by a chassis designed with a slot for the reflectance sensor.



Our third and final modification was the addition of a diode and accompanying resistor so that our robot could broadcast gold codes at the flash beacons (diagram of the circuit to the right). The resistor was placed on the Mudduino, and reduced the current flowing through the diode so that it wouldn't overload. The diode was at the end of wires positioned at the front of the robot. Unlike the reflectance sensor, which was superglued to the chassis, this was left so that its position and angle could be changed by fiddling with the wires. This proved useful when testing our navigation code, as the optimal angle between the diode and the beacon's photosensor changed depending on the distance range at which we were attempting to flash the gold code.

Bill of Materials

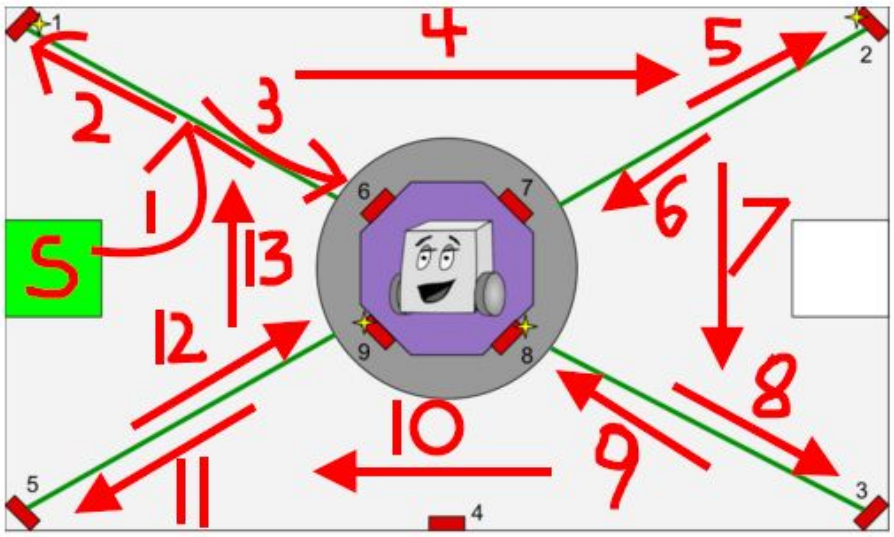
Component	Description	Supplier	Supplier Part #	Unit Price	Quantity	Total

BPW77NA	Phototransistor	Digikey	751-1020-ND	$1.91	1	$1.91
QRD1114	Reflectance Sensor	DigiKey	QRD1114-ND	$1.31	1	$1.31
MH4	4-pin male header	DigiKey	SAM1031-50-ND	$0.19	2	$0.38
R220, R320k	220 Ohm and 320 kOhm Resistors	DigiKey	CF14JT220RCT-ND; CF14JT330KCT-ND	$0.08	4	$0.32
D1	Diode	DigiKey	OVFSRAC8	$0.25	1	$0.25
					Total Cost:	$4.17

4. ALGORITHM

The algorithm that we used for the competition is based around using the blue lines on the board to navigate. In our setup function, our robot begins by initializing a number of inputs and variables before beginning to move. Everything is initialized at the beginning of the match so that we do not have to worry about something not initializing during the match. After that, the code sent the robot forward before telling it to turn left or right depending on which team the robot belonged to at the moment. The code sends the robot to the line that begins with one flash beacon unclaimed and one bump beacon belonging to the other team. Once the robot reaches the line, the code has it move forward a bit farther, then turn until it finds the line again. Afterwards, the robot moves forward slightly, then turns in the opposite direction until it finds the line for the third time. All of this turning, stopping, and starting is designed to allow the robot to center itself along the blue line. Then, the code has the robot detect for the gold codes being emitted by the beacons. Since there is a beacon at each end of every line, ideally the robot would be lined up well enough to read the gold code immediately, but the code accounts for the fact that this may not always be the case. If the robot does not detect a gold code with a high enough correlation immediately, the robot swivels very slightly back and forth with right and left turns, detecting gold codes at regular intervals. This swiveling causes the robot to move forward slightly, but this just helps it detect the gold code. Once a gold code is finally detected, the code's instructions vary depending on which beacon was detected. The code varies depending on whether the beacon is a flash or bump beacon and whether the beacon is in the center of the course or the outer edge of it.

The code is simpler, although not much so, for the beacons on the outer edge of the course. These beacons are aligned directly with the line, which means that ideally the robot need only move straight to reach them. However, we recognized that this was not always the case, as the robot could be off, so we accounted for that. If the beacon is an outer bump beacon (beacons 3 & 5), the code tells the robot to move for a set amount of time before beginning to swivel back and forth rapidly. This swivel allows the robot to cover more ground, helping to prevent it from missing the beacon entirely. Once the robot senses using our second reflectance sensor that is has hit a wall, it assumes that it has hit the beacon and then backs up. Flash beacons along the outer edge (beacons 1 & 2) are tackled in a similar way. The robot moves forward and swivels much like it did for the bump beacons, but it also broadcasts the proper gold code to change the beacon to the robot's team. Once the robot senses that the beacon's team matches the robot's, the code instructs it to back up. The flash beacons in the center (beacons 8 & 9) are handled identically except for a small turn to account for the fact that they are not perfectly aligned with the blue lines. The center bump beacons (beacons 6 & 7) are handled differently than the outer ones, however. Since we can now take advantage of the black circle, the robot is instructed to move forward until it detects the change from blue and white to black. Once it does, after a small turn to account for the beacon's orientation, the robot moves forward for a set amount of time, just long enough to hit the beacon, before the code instructs it to back up. After backing up, no matter which beacon was just hit, the robot then turns around and centers itself along the line in a manner very similar to the way it did so after finding its first blue line. Then, if this was the first beacon that the robot handled this line, it handles the next beacon on the line. If either of the beacons already belong to the robot's team, it is instructed to turn around and simply ignore them. Finally, after handling two beacons for a line, the robot turns right and moves forward until it finds the next blue line. It then centers itself along that line and handles the beacons in the manners listed above. This code then loops for the rest of the match.



5. Results

While our algorithm seemed sound, our robot never performed as smoothly as we would have liked it to. In testing, the robot was usually able to get at least the first beacon, often the second, but then had trouble finding the next line and acquiring any beacons there. During the scrimmage, we were only able to capture a single beacon before we were hit by the other team's robot. Since our code had nothing written to handle collisions, the robot remained still for the rest of the match. Unfortunately, we could not think of a way around this, so our robot went into the final competition with this flaw. However, a collision is not what caused our biggest problem in the final competition. There, our robot was able to turn the first flash beacon to our team, but it failed to detect the change in the beacon's team and slammed into the wall. We had forgotten to account for this possibility, so the robot remained stuck in a corner until we spent a point to get a reset. Then, the robot correctly identified that the beacon now belonged to us, but when it attempted to get the center bump beacon on the other end of the line, it collided with the other team's robot and was essentially out for the rest of the match. Because of these flaws in our code, we did not advance beyond the first round of the tournament.

6. Lessons Learned

We learned quite a lot about engineering and programming in this course. However, there are a number of specific things that we could improve upon if we were to do everything again. First, we would make sure to find the specification for our gearbox that would give our robot the optimal speed for the tasks we assign it. In practice, this would mean making sure that the gearbox was tested and modified before other modifications made fully disassembling the robot too risky. The gearbox configuration that we used for the competition was the slowest of all of the possible make-ups, and that put us at a strong disadvantage compared to the other teams.

Second, we would attempt to find more accurate sensors, even if they cost a little bit more. Our distance sensor was so unreliable that we choose not to use it during the competition, and it could have been a real boon to us if it had been working properly. The next thing we would change would be our navigation algorithm. The way we designed our code was far too rigid for a competition with an uncontrolled variable like the other team's robot. Instead, we would utilize a more fluid code that would allow the robot to move around without relying so heavily on a specific order of beacons. Finally, we would have more back up functions in our code that could kick in if the robot gets hit by the other team's robot or gets stuck in a corner. This would solve the biggest two problems that we had during the competition.

APPENDIX A: Mudduino Code

Navigation Tab

```
// E11 Competition Code: Final Copy
// Team: WIBSTIR
// Team Members: Evan Amason and Elizabeth Poss

// Reflectance Reading of White Ground: 475 - 550
// Reflectance Reading of Blue Ground: 600 - 700
// Reflectance Reading of Black Ground: 750 - 850

// We start off the code by defining a number of variables that will be used
throughout the code.
int reflectance;
int distance;
int photosensor;
int refl2;
int beaconnumber = 0;
int team = 0;
// Team White is 1, Team Green is 0.
int OurTeam;
#define DISTSENSOR 14
#define REFLECTSENSOR 18
#define REFLECTSENSOR2 16
#define PHOTO 19
#define PHOTO2 17
// These next few variables can and need to be changed depending on how long it has
been
// since the course has been painted and what the lighting is like.
#define WALLTHRESH 50
#define BUMPER 500
#define Blue 600
#define Black 750
boolean crash = false;
// The following variables are those for the gold code broadcasting functions.
const int LEDPIN = 2;  // The right LED pin.
const int ASCII_OFFSET = 48;  // To convert Serial "char" input to "byte".
unsigned long delayTime = 250;
unsigned long nextTrigger = 0;
unsigned long currTime = 0;
byte whichGoldCode = 1;  // Which gold code is being detected.

boolean invertBits;
byte j = 0;  // which element of the gold code sequence to light up.

// The following is a list of all gold codes that will be used during the competition.
boolean allcodes[19][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},
  {0,1,0,0,0,0,1,1,0,1,0,0,0,0,1,0,1,1,1,1,1,1,0,1,0,1,0,1,1,1,0},
```

```
  {1,0,1,0,0,0,0,0,0,0,1,1,0,1,1,1,1,1,1,1,0,1,0,0,0,0,1,1,1,0,1},
  {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},
  {0,1,1,0,0,1,1,0,1,1,0,1,1,1,0,1,1,1,1,1,0,0,1,1,0,1,1,1,1,0,1,0},
  {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},
  {1,1,0,1,0,1,0,1,0,0,1,1,1,1,1,1,0,1,0,0,1,0,0,1,1,1,1,1,1,1,1,0},
  {0,1,0,1,0,0,0,1,1,0,0,0,1,1,0,1,0,1,1,1,0,0,0,0,1,0,0,0,1,0,0},
  {1,0,1,1,0,0,1,0,1,1,1,1,1,1,0,0,0,0,1,1,1,1,0,0,1,1,1,1,1,0,1,1,1},
  {0,0,1,1,0,1,1,0,0,1,0,0,1,0,1,1,1,1,0,0,1,1,0,1,0,1,0,1,0,0,1,1,0,1},
  {1,1,1,1,0,0,0,0,1,0,1,0,0,0,0,1,1,0,0,0,1,0,0,0,0,1,0,1,0,1,0},
  {0,1,1,1,0,1,0,0,0,0,0,1,0,0,1,0,0,1,1,0,1,0,1,1,0,0,1,0,0,0,0},
  {1,0,0,0,1,1,0,0,1,1,0,0,1,1,1,1,1,0,0,1,0,1,0,0,1,0,1,1,1,1,0,0},
  {0,0,0,0,1,0,0,0,0,0,1,1,1,1,1,1,0,0,1,1,0,0,1,0,1,0,0,0,0,0,1,1,0},
  {1,1,0,0,1,1,1,0,1,0,0,1,0,1,1,0,1,1,0,1,1,0,0,0,1,1,0,0,0,0,1},
  {0,1,0,0,1,0,1,0,0,0,1,0,0,1,0,1,0,0,1,1,1,0,1,1,1,0,1,1,0,1,1} };
  // The 10 - 19 gold codes were necessary to handle unclaimed beacons.

// Quite a bit of helpful information will be shown if this is changed to true.
// However, it will clog the serial port with text.
#define debugmain false

void setup() {
  Serial.begin(9600);
  if (debugmain) {
    Serial.println("Debug code initiated.");
  }
  // The first several commands are just turning on parts of the robot and setting
some variables.
  testServo();
   primeSensors();
  motorEn();
  initMotors();
  OurTeam = digitalRead(3);
  // The team of the robot will be determined by which LED is on, and that information
  // will be used to determine behaivors throughout the code.
  if (OurTeam == 1) {
    invertBits = false;
  }
  if (OurTeam == 0) {
    invertBits = true;
  }
  setupBroadcast();

  if (debugmain) {
    Serial.print("Our Team Number is ");
    Serial.println(OurTeam);
  }
  // The following several lines of code allow the robot to reach the line closest
  // to its starting point that has one unclaimed beacon and one beacon belonging to
```

```
// the other team.
forward();
delay(2000);
halt();
if (OurTeam == 1) {
  turnR();
}
if (OurTeam == 0) {
  turnL();
}
delay(1400);
halt();
forward();
// The robot uses the blue of the line to find the line and then center
// itself on it.
reflectance = analogRead(REFLECTSENSOR-14);
while(reflectance < Blue) {
  reflectance = analogRead(REFLECTSENSOR-14);
}
halt();
forward();
delay(500);
 halt();
reflectance = analogRead(REFLECTSENSOR-14);
if (OurTeam == 1) {
  turnR();
}
if (OurTeam == 0) {
  turnL();
}
while(reflectance < Blue) {
  reflectance = analogRead(REFLECTSENSOR-14);
}
halt();
if (OurTeam == 1) {
  turnR();
}
if (OurTeam == 0) {
  turnL();
}
delay(50);
halt();
if (debugmain) {
  Serial.println("Moving to HandleLine function.");
}
// The setup code is essentially over here. Now, we use a select number of
// functions repeatedly to handle the rest of the round.
HandleLine();
if (debugmain) {
  Serial.println("End of setup code.");
```

```
  }
}

void loop() {
  if (debugmain) {
    Serial.println("Starting loop code.");
  }
  // This function is intended to have the robot go from line to line,
  // pausing to handle unclaimed beacons or beacons belonging to the other
  // team at the ends of the lines.
  FindLine();
  // The wall check function checks to see if the robot has somehow crashed
  // into a wall. If it has, it tries to correct the robot's position.
  wallCheck();
  HandleLine();
  wallCheck();
}

// This loop operates under the assumption that the robot is facing
// towards the outer wall of the course while at the center of a line.
void FindLine() {
  turnR();
  // Certain turns take less time than others.
  if (beaconnumber == 7 or beaconnumber == 9) {
    delay(1850);
  }
  else {
    delay(2250);
  }
  halt();
  // The beacon number is then reset.
  beaconnumber = 0;
  // The robot then finds the next line in much the same way as it
  // found the first line.
  forward();
  reflectance = analogRead(REFLECTSENSOR-14);
  while(reflectance < Blue or reflectance > Black) {
    reflectance = analogRead(REFLECTSENSOR-14);
    wallCheck();
    forward();
  }
  halt();
  forward();
  delay(500);
  halt();
  reflectance = analogRead(REFLECTSENSOR-14);
  turnL();
  while(reflectance < Blue) {
    reflectance = analogRead(REFLECTSENSOR-14);
  }
```

```
  halt();
  turnR();
  delay(50);
  halt();
}

void HandleLine() {
  // This loop will continue to run until a beaconnumber is determined.
  while(beaconnumber == 0) {
   Gold_Code_Detection();
   if (beaconnumber != 0) {
     break; // The loop will break as soon as a beacon is properly read.
   }
   // Until a beacon is properly read, the robot will swivel back and forth.
   turnL();
   delay(50);
   halt();
   Gold_Code_Detection();
   if (beaconnumber != 0) {
    break;
   }
   turnR();
   delay(100);
   halt();
   Gold_Code_Detection();
   if (beaconnumber != 0) {
    break;
   }
   turnL();
   delay(50);
   halt();
   Gold_Code_Detection();
   if (beaconnumber != 0) {
    break;
   }
   forward();
   delay(100);
   halt();
  }
  if (debugmain) {
    Serial.println();
    Serial.print("Beacon Number is ");
    Serial.println(beaconnumber);
  }
  // Here, we move to another flexible function.
  BeaconInstructions();
  // The beacon number is reset to zero so the next beacon number can be read.
  beaconnumber = 0;
  // This next loop is identical to the first.
  while(beaconnumber == 0) {
```

```
   Gold_Code_Detection();
   if (beaconnumber != 0) {
    break;
   }
   turnL();
   delay(50);
   halt();
   Gold_Code_Detection();
   if (beaconnumber != 0) {
    break;
   }
   turnR();
   delay(100);
   halt();
   Gold_Code_Detection();
   if (beaconnumber != 0) {
    break;
   }
   turnL();
   delay(50);
   halt();
   Gold_Code_Detection();
   if (beaconnumber != 0) {
    break;
   }
   forward();
   delay(100);
   halt();
  };
  if (debugmain) {
    Serial.println();
    Serial.print("Beacon Number is ");
    Serial.println(beaconnumber);
  }
  BeaconInstructions();
  // The beacon number is not reset here, as it will be used in the first
  // part of the FindLine() function.
}

void BeaconInstructions() {
  if (debugmain) {
    Serial.println("Starting BeaconInstructions.");
  }
  // The robot will turn around and ignore a beacon if it already belongs to
  // the robot's team.
  if (OurTeam == team) {
    TurnAround();
    return;
  }
  // This code handles bump beacons on the outer edge of the course.
```

```
  if (beaconnumber == 3 or beaconnumber == 4 or beaconnumber == 5 or beaconnumber ==
13 or beaconnumber == 14 or beaconnumber == 15) {
    if (debugmain) {
      Serial.println("Launching Outer Edge Bump Beacon Instructions.");
    }
    forward();
    delay(1000);
    halt();
    Gold_Code_Detection();
    int refl2 = analogRead(REFLECTSENSOR2-14);
    // The robot swivels here in order to allow it to hit the beacon if it is not
    // approaching the beacon spot on.
    while( OurTeam != team) {
      turnR();
      delay(50);
      halt();
      turnL();
      delay(100);
      halt();
      turnR();
      delay(50);
      halt();
      delay(25);
      // This allows the beacon to know if it has hit a wall.
      // If it has, it assumes that this beacon has been handled and turns around.
      refl2 = analogRead(REFLECTSENSOR2-14);
      if (refl2 < BUMPER) {
        break;
      }
      Gold_Code_Detection();
      if (OurTeam != team) {
        forward();
        delay(100);
        halt();
      }
    }
    team = 2;
    BackUpLine();
    wallCheck();
  }
  // The bump beacons in the center of the course need to be handled individually, as
they
  // are not perfectly straight with the line, and one lies to the right of the line
  // while the other lies to the left.
  if (beaconnumber == 6 or beaconnumber == 16) {
    if (debugmain) {
      Serial.println("Launching Center Bump Beacon Instructions.");
    }
    forward();
    delay(500);
```

```
    halt();
    turnL();
    delay(200);
    halt();
    forward();
    Gold_Code_Detection();
    // The robot goes forward indefinitely until it reaches the black inner
    // circle, and then it only goes forward for another 1.3 seconds, which
    // should be long enough for it to hit the beacon.
    reflectance = analogRead(REFLECTSENSOR-14);
    while(reflectance < Black) {
      reflectance = analogRead(REFLECTSENSOR-14);
    }
    delay(1300);
    halt();
    team = 2;
    BackUpLine();
    wallCheck();
  }
  // Other than the small turn right, this is identical to the code for beacon 6.
  if (beaconnumber == 7 or beaconnumber == 17) {
    if (debugmain) {
      Serial.println("Launching Center Bump Beacon Instructions.");
    }
    forward();
    delay(500);
    halt();
    turnR();
    delay(200);
    halt();
    forward();
    reflectance = analogRead(REFLECTSENSOR-14);
    while(reflectance < Black) {
    reflectance = analogRead(REFLECTSENSOR-14);
    }
    delay(1300);
    halt();
    team = 2;
    BackUpLine();
    wallCheck();
  }
  // This code handles flash beacons on the outer edge of the course.
  if (beaconnumber == 1 or beaconnumber == 2 or beaconnumber == 11 or beaconnumber ==
12) {
    if (debugmain) {
      Serial.println("Launching Flash Beacon Instructions.");
    }
    // The first thing that happens is the robot determines which code to flash
    // in order to change the beacon to the appropriate team's color.
    if (beaconnumber == 11 or beaconnumber == 12) {
```

```
    whichGoldCode = (beaconnumber - 10);
  }
  if (beaconnumber == 1 or beaconnumber == 2) {
    whichGoldCode = beaconnumber;
  }
  forward();
  delay(750);
  halt();
  Gold_Code_Detection();
  // The robot will continuously flash the correct gold code, swivel, and
  // attempt to read the beacon until it reads that the beacon's team matches
  // the robot's.
  while(OurTeam != team) {
    turnR();
    delay(50);
    halt();
    FlashGoldCode();
    turnL();
    delay(100);
    halt();
    FlashGoldCode();
    turnR();
    delay(50);
    halt();
    FlashGoldCode;
    Gold_Code_Detection();
    if (OurTeam != team) {
      forward();
      delay(50);
      halt();
    }
  }
  team = 2; // The team variable is then reset.
  BackUpLine();
  wallCheck();
  }
// This next part o the code handles the flash beacons on the inner part
// of the course. Like the bump beacons, this was done since the center flash
// beacons are not perfectly in line with the blue lines.
if (beaconnumber == 8 or beaconnumber == 18) {
  if (debugmain) {
    Serial.println("Launching Center Flash Beacon Instructions.");
  }
  if (beaconnumber == 18) {
    whichGoldCode = (beaconnumber - 10);
  }
  if (beaconnumber == 8) {
    whichGoldCode = beaconnumber;
  }
  forward();
```

```
  delay(1250);
  halt();
  turnL();
  delay(100);
  halt();
  Gold_Code_Detection();
  // This part of the code is identical to the one that handles the beacons
  // on the outer edge of the course.
  while(OurTeam != team) {
    turnR();
    delay(50);
    halt();
    FlashGoldCode();
    turnL();
    delay(100);
    halt();
    FlashGoldCode();
    turnR();
    delay(50);
    halt();
    FlashGoldCode;
    Gold_Code_Detection();
    if (OurTeam != team) {
      forward();
      delay(50);
      halt();
    }
  }
  team = 2;
  BackUpLine();
  wallCheck();
  }
// Other than the small turn right, this code is identical to the code used
// for beacon 8.
if (beaconnumber == 9 or beaconnumber == 19) {
  if (debugmain) {
    Serial.println("Launching Center Flash Beacon Instructions.");
  }
  if (beaconnumber == 19) {
    whichGoldCode = (beaconnumber - 10);
  }
  if (beaconnumber == 9) {
    whichGoldCode = beaconnumber;
  }
  forward();
  delay(1250);
  halt();
  turnR();
  delay(100);
  halt();
```

```
    Gold_Code_Detection();
    while(OurTeam != team) {
      turnR();
      delay(50);
      halt();
      FlashGoldCode();
      turnL();
      delay(100);
      halt();
      FlashGoldCode();
      turnR();
      delay(50);
      halt();
      FlashGoldCode;
      Gold_Code_Detection();
      if (OurTeam != team) {
        forward();
        delay(50);
        halt();
      }
    }
    team = 2;
    BackUpLine();
    wallCheck();
    }
}

// This function simply allows the robot to turn around after reading
// that a beacon already belongs to it.
// This function assumes that the robot is at about the midpoint of the
// line.
void TurnAround() {
  forward();
  delay(500);
  halt();
  turnL();
  delay(500);
  // Much like the setup code, the turning around is based all around
  // the blue lines.
  reflectance = analogRead(REFLECTSENSOR-14);
  while(reflectance < Blue) {
    reflectance = analogRead(REFLECTSENSOR-14);
  }
  halt();
  forward();
  delay(500);
  halt();
  turnR();
  reflectance = analogRead(REFLECTSENSOR-14);
  while(reflectance < Blue) {
```

```
    reflectance = analogRead(REFLECTSENSOR-14);
  }
  halt();
}

// This function allows the robot to back up and turn around after handling
// a beacon.
// This function assumes that the robot is at a beacon at the end of a line.
void BackUpLine() {
  backward();
  delay(1200);
  halt();
  turnR();
  delay(500);
  // Again, this code is based around the blue lines.
  reflectance = analogRead(REFLECTSENSOR-14);
  while(reflectance < Blue) {
    reflectance = analogRead(REFLECTSENSOR-14);
  }
  halt();
  forward();
  delay(500);
  turnL();
  reflectance = analogRead(REFLECTSENSOR-14);
  while(reflectance < Blue) {
    reflectance = analogRead(REFLECTSENSOR-14);
  }
  halt();
}

// This code projects a gold code from an LED on the robot.
void FlashGoldCode() {
  Gold_Code_Detection();
  // The timing here is very important, otherwise this function will
  // not work.
  nextTrigger = micros();
  unsigned long startTime = millis();
  while ((millis() - startTime) < 250) {
    BroadcastGC();
  }
}

// This is a back-up function in case the robot hits a wall.
// It checks using the second reflectance sensor to see if it is hitting
// anything. If it is, it backs up and turns right.
void wallCheck() {
  int refl2 = analogRead(REFLECTSENSOR2-14);
  if (refl2 < WALLTHRESH) {
    backward();
    delay(500);
```

```
    halt();
    // This turn assumes that the robot needs to make a 90 degree turn
    // to be facing the next blue line.
    turnR();
    delay(1350);
    halt();
  }
}
```

Gold Code Broadcasting Tab

```
// This tab handles all gold code broadcasting.
// This code was taken from the E11 website. Small alterations
// were made so that it would fit our code.

// This function intializes the LED used for broadcasting
// gold codes.
void setupBroadcast()
{
  pinMode(LEDPIN,OUTPUT);
}

void BroadcastGC()
{
  currTime = micros();  // Total elapsed microseconds.

  // Perform a time-sensitive task when both times match:
  if ( currTime >= nextTrigger)
  {
    nextTrigger = nextTrigger + delayTime;

    // Display the next Gold Code (or inverse Gold Code) element.
    if (invertBits == false) {
      digitalWrite(LEDPIN, allcodes[(whichGoldCode - 1)][j] );  // 1 is HIGH. 0 is LOW
    }
    else {
      digitalWrite(LEDPIN, !(allcodes[(whichGoldCode - 1)][j]) );
    }
    j++;

    if (j > 30)
    { j = 0;}  // Reset index.
  }
}
```

Gold Code Detection Tab

```
// This tab handles detecting gold codes emitted by the beacons.
```

```
// A number of variables need to be initialized before moving
// on to the function itself.
#define debug false // Debugging proved very helpful in this function.
#define PHOTOSENSOR 19 // Defining this for later.
int FlashValues[31] = {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};
// This array will hold the readings from the photosensor.
boolean DetectedGoldCode[31] =
{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}; // This array will
hold the detected gold code.
int average = 0; // Just storing this for now.
int maxcounter = 0;
int mincounter = 0;
int HighestCorrelation = 0;

// This function will work multiple times within a program.
void Gold_Code_Detection() {
  detectGC();
  delay(100);
  CorrelateGC();
  delay(100);
  // These variables need to be reset every time the function
  // is run.
  HighestCorrelation = 0;
  maxcounter = 0;
  mincounter = 0;
  delay(100);
}

void detectGC() {

  pinMode(PHOTOSENSOR, INPUT); // Initialize the photosensor.

  if (debug) {
    Serial.println();
    Serial.println();
    Serial.print("Flash Value is: ");
  }
  // Kate Woolverton and Maggie Gelber showed us how to implement this method of
keeping time.
  int i = 0;
  unsigned long time1 = micros();
  unsigned long time2 = micros() + 250;

// I avoided debugging statements in the loop just in case they messed with the
timing.

  while ( i < 31 ) {
    while ( time1 <time2) {
      time1 = micros();
```

```
    }
    time2 = time2 + 250;
    FlashValues[i] = analogRead(PHOTOSENSOR - 14);
    i++;
    }
  // This seemed like the best way to keep time with minimal delays.
  if (debug) {
    for (int i = 0; i<31; i++) {
      Serial.print(FlashValues[i]);
      Serial.print(" ");
    }
  }

  if (debug) {
    Serial.println();
  }

  for (int i = 0; i<31; i++) { // This sums the values detected by the photosensor.
    average = average + FlashValues[i];
  }

  average = average / 31; // This is the final average.

  if (debug) {
    Serial.println();
    Serial.print("Average: ");
    Serial.println(average);
  }

  for (int i = 0; i<31; i++) { // I wish there was an easier way to print an array.
    if (FlashValues[i] <= average) { // Corresponds to the LED being on.
      DetectedGoldCode[i] = 1;
    }
    else { // Corresponds to the LED being off.
      DetectedGoldCode[i] = 0;
    }
  }
}

void CorrelateGC() { // Second part of the overall function.
// I needed to initialize two variables and an array up here.
  boolean ClosestCode[31] =
{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 m; //a marker for red beacons

  if (debug) {
    Serial.println();
    Serial.println();
  }
```

```
  for (int i = 0; i<19; i++) {
    correlateshift( DetectedGoldCode, allcodes[i]); // Using parts from the Gold Code
Correlation homework.

    if (debug) {
      Serial.println();
      Serial.print("Correlating Gold Code ");
      Serial.println(i+1);
    }

    if (abs(maxcounter) > abs(HighestCorrelation)) {
      HighestCorrelation = maxcounter;
      for (int j = 0; j<31; j++) {
        ClosestCode[j] = allcodes[i][j];
      }
      // If this gold code correlates better than any of the previous ones, set the
Closest Code to it and change the beacon number.
      if (i<10) { // This handles claimed beacons.
        beaconnumber = i + 1;
        m = 0;
      }

      else { // This handles unclaimed beacons.
        beaconnumber = i - 9;
        m = 1;
      }

    }

    if (abs(mincounter) > abs(HighestCorrelation)) {
      HighestCorrelation = mincounter;
       for (int j = 0; j<31; j++) {
        ClosestCode[j] = allcodes[i][j];
      }

      if (i<10) {
        beaconnumber = i + 1;
        m = 0;
      }

      else {
        beaconnumber = i - 9;
        m = 1; // A marker for a red beacon.
      }

    }

  if (debug) {
    Serial.print("Highest Correlation: ");
    Serial.println(HighestCorrelation);
```

```
  }

  }

  // This next if statment makes sure that the detected gold code falls within an
acceptable margin of error.
  if (abs(HighestCorrelation) < 28) {
    Serial.println();
    Serial.println();
    Serial.println("Error made.");
    beaconnumber = 0;
    team = 2;
    return; // Ends the function.
  }

  for (int i = 0; i<31; i++) { // Again, I wish there was an easier way to print
arrays.
    boolean a;
    a = ClosestCode[i];
  }

  if (m == 1) {
    //Serial.println("This beacon belongs to no team.");
    team = 2;
  }
  if (HighestCorrelation > 0 and m != 1) { // This means that the code belongs to the
white team.
    //Serial.println("This beacon belongs to the white team.");
    team = 1;
  }

  if (HighestCorrelation < 0 and m != 1) { // This means that the code is inverted and
belongs to the green team.
    //Serial.println("This beacon belongs to the green team.");
    team = 0;
  }
}

// Everything below this is strictly Elizabeth Poss' code with some
// minor alterations to fit the competition code.

int correlateshift(boolean goldcode1[31], boolean goldcode2[31]) {
  if (debug) {
    Serial.println("Running Correlate Shift.");
  }

  int sum = 0;
  // The shifted gold code.
  boolean tempcode[31];
  // counter for max and min
```

```
  for (int i=0; i<31; i++)
  {
     // Uses two for loops to assemble temporary list of the shifted gold code.
     // One from i to end, next from 0 to i.
     int j = 0;
     for (int x=i; x<31; x++)
     {
      tempcode[j] = goldcode2[x];
      j += 1;
     }
     for (int x=0; x<i; x++)
     {
      tempcode[j] = goldcode2[x];
      j += 1;
     }

     // Tests each correlation value for min or max.
     if (correlate(goldcode1, tempcode) > maxcounter)
     {
        maxcounter = correlate(goldcode1, tempcode);
     }

    if (correlate(goldcode1, tempcode) < mincounter)
    {
      mincounter = correlate(goldcode1, tempcode);
    }


  }

}

// Correlates two gold codes.
int correlate(boolean goldcode1[31], boolean goldcode2[31])
{
  int sum = 0;
  for (int i = 0; i<31; i++)
  {
    // +1 for identical, -1 for different
    if (goldcode1[i] == goldcode2[i])
    {
      sum += 1;
    }
    else
    {
      sum -= 1;
    }
  }
  return sum;
```

```
}
```

Motors Tab

```
// This tab sets up several functions to control the motors.
// This code was primarily taken from an example given to us
// on the E11 website. The comments are our own.

#define LPLUS 9
#define LMINUS 8
#define RPLUS 7
#define RMINUS 12
#define LEN 6
#define REN 11

// Defining the various inputs I will be using.

void motorEn() { // This will enable the motors.
  digitalWrite(LEN, HIGH);
  digitalWrite(REN, HIGH);
}

void initMotors() {  // Assigns the motors as outputs.
  pinMode(LPLUS, OUTPUT);
  pinMode(LMINUS, OUTPUT);
  pinMode(RPLUS, OUTPUT);
  pinMode(RMINUS, OUTPUT);
  pinMode(LEN, OUTPUT);
  pinMode(REN, OUTPUT);
}

void halt() { // Stops any movement by the motors.
  digitalWrite(RPLUS, HIGH);
  digitalWrite(RMINUS, HIGH);
  digitalWrite(LPLUS, HIGH);
  digitalWrite(LMINUS, HIGH);
}

// The rest of the functions are fairly self explanatory.
void forward() {
  digitalWrite(RPLUS, HIGH);
  digitalWrite(RMINUS, LOW);
  digitalWrite(LPLUS, HIGH);
  digitalWrite(LMINUS, LOW);
}

void backward() {
  digitalWrite(RPLUS, LOW);
  digitalWrite(RMINUS, HIGH);
  digitalWrite(LPLUS, LOW);
```

```
  digitalWrite(LMINUS, HIGH);
}

void turnL() {
  digitalWrite(RPLUS, HIGH);
  digitalWrite(RMINUS, LOW);
  digitalWrite(LPLUS, HIGH);
  digitalWrite(LMINUS, HIGH);
}

void turnR() {
  digitalWrite(RPLUS, HIGH);
  digitalWrite(RMINUS, HIGH);
  digitalWrite(LPLUS, HIGH);
  digitalWrite(LMINUS, LOW);
}
```

Sensors Tab

```
// This tab simply handles initializing the various sensors.
// Originally, this tab had a few functions that helped us
// test to make sure that our sensors were working properly.

// Defining the pins for most of the sensors.
#define DISTSENSOR 14
#define REFLECTSENSOR 18
#define PHOTO 19

// This initializes the sensors when called.
void primeSensors() {
  pinMode(DISTSENSOR, INPUT);
  pinMode(REFLECTSENSOR, INPUT);
  pinMode(PHOTO, INPUT);
  pinMode(REFLECTSENSOR2, INPUT);
}
```

Servo Tab

```
// This tab simply adjusts the distance sensor to be
// facing forward at the beginning of every match.
// Again, most of this code was given to us from
// the E11 website.

#include <Servo.h>
// Pins
#define SERVOPIN 10
// Global variable for the servo information.
Servo servo;
```

```
void testServo() {
  initServo();
  servo.write(90);
  delay(2000); // In hindsight, this really does not
               // need to be here.
}

void initServo() {
  pinMode(SERVOPIN, OUTPUT);
  servo.attach(SERVOPIN);
}
```