E11 Final Report

EDC

Section #3

Lorelei Bivins, Howard Deshong, and Caleb Norfleet

1. ABSTRACT

This report discusses the robot that team EDC designed for a capture-the-flag game in Harvey Mudd College's Autonomous Vehicles elective. It outlines changes the team made from the stock vehicle and details results from testing, a scrimmage, and the final competition. Modifications include an array of reflectance sensors that sense the ground and two series of LEDs that flash to broadcast codes. The robot switches between 4 control states and it showed itself able to capture all beacons and finish on the starting square. The robot won its scrimmage and placed eighth in the final competition, losing the quarterfinals to the first-place winner. The team also discusses the lessons learned from this project.

2. INTRODUCTION

In the Autonomous Vehicles elective at Harvey Mudd College ("E11"), students build small robots to compete in a capture-the-flag style competition.

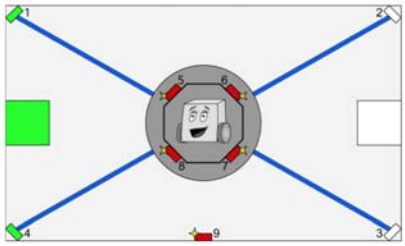

Figure 1. The competition arena. Two robots face off in each match, one team white, the other green. The robots start the match on opposite sides of an arena marked with blue and black paths. Nine numbered beacons mounted with LEDs and phototransistors are placed around the arena. Each beacon continuously flashes an LED on and off in a unique pattern ("Gold Code"). The robots have 70 seconds to claim beacons by flashing Gold Codes and earn a point per beacon. At the end of the match, robots earn an extra point for returning to the start zone.

EDC's robot differs from the "stock" robot in that it features extra reflectance sensors, giving the robot a better sense of the ground in front of it. This lets it make better informed decisions on where to move. The robot also has three LEDs on each side that continuously broadcast Gold Codes.

The robot's general game strategy involves driving forward and then circling the center black zone while flashing Gold Codes. Upon encountering the start of each blue line, the robot turns toward it. If the robot detects that the corner beacon at the end of the path is already captured, it turns around; otherwise it follows the line and captures the beacon. Finally, if the match has less than 10 seconds remaining and the robot detects it is near its starting square, it returns home.

The robot performed well in testing, consistently following arena lines and capturing beacons. It encountered difficulties in its scrimmage match, misjudging where it was and spinning in circles. Nonetheless, the robot won 2 of its 3 matches and placed 1st in the scrimmage. During the competition the robot captured several beacons, but it had trouble interpreting the newly-repainted arena lines and moving as quickly as its opponents. The EDC robot placed 8th, losing to the overall winner of the competition.

3. PHYSICAL MODIFICATION

The EDC build differs from the standard design in several ways. Its chassis is unchanged except for the gearbox's gear ratio, which was changed from 12.7:1 to 114.7:1. This made it faster. The robot features additional hardware and electronics described below (Figure 2-4).

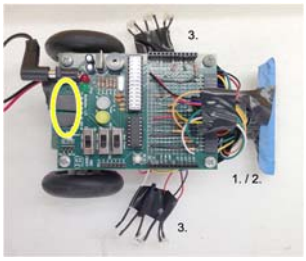

Figure 2. Top-down view of EDC robot design. The stock design features a distance sensor in the yellow circled region and a phototransistor and infrared reflectance sensor at the robot's front. EDC removed the distance sensor and reflectance sensor. Instead, the robot features a prefabricated array of 5 infrared reflectance sensors (1). This gives the robot a wider view of the ground than the stock reflectance sensor could provide. Also added was a cardboard bumper (2) to protect this sensor array. Finally, the robot features two arrays of 3 LEDs each (3). These continuously broadcast Gold Codes so the robot can capture beacons. Multiple LEDs are present on each side to strengthen the signal so that the beacons can easily read the robot's Gold Codes.

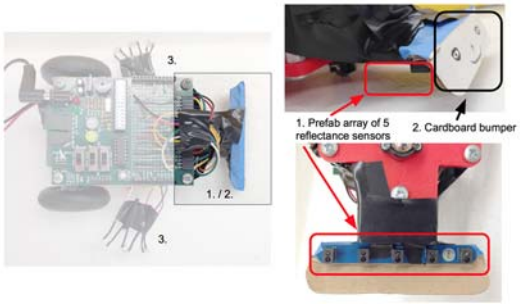

Figure 3. Front additions to robot. The prefabricated array of 5 reflectance sensors (1) aims at the ground and aids navigation. The cardboard bumper (2) protects the sensor array when the robot bumps into obstacles or beacons.

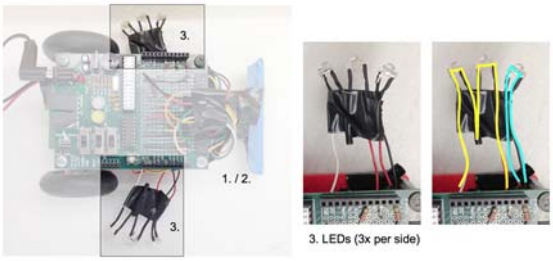

Figure 4. Side additions to robot. Each side features 3 LEDs that continually flash Gold Codes (3). Each array consists of 1 standalone LED (cyan wire) and 2 LEDs in series (yellow wire). Pins 13 and 10 are used to control the right side LEDs, and pins 5 and 2 are used to control the left side LEDs.

Component	Description	Supplier	Supplier Part #	Unit Price	Quantity	Total
S1	Infrared Line Tracking Sensor	RobotShop	RB-Wav-97	$5.49	1	$5.49
D1-D6	2.2 Volt Red LED	HMC	N/A	$0.25	6	$1.50
R1-R6	10k Ohm Resistor	HMC	N/A	$0.25	6	$1.50
R7-R10	220 Ohm Resistor	HMC	N/A	$0.25	4	$1.00

Table 1. Bill of materials for the robot's hardware modifications.

4. ALGORITHM

The robot has 4 control states. It begins in the "start" state, where it moves forward until encountering the black center zone. It then moves back and forth between the "on blue line" and "not on blue line" states for most of the match. At the end it can switch to the "go home" state.

In "not on blue line," the robot checks if it has found a blue line (in which case it switches to "on blue line"), has arrived at the edge of a blue line (in which case it turns right to face the line and switches to "on blue line"), is along the edge of the black circle (in which case it follows the edge counterclockwise), or is lost (in which case it turns in a counterclockwise circle until it recognizes a landmark).

In "on blue line," the robot tries to follow the blue line until it detects a claimed corner bump beacon (either because it was claimed previously or because the robot just claimed it). Then the robot backs up, turns around 180°, and tries to follow the line back to the center. If the robot detects that it is no longer on the blue line (either because it is lost or has returned to black circle), it returns to "not on blue line" and resumes navigating. Additionally, if the robot is in "on blue line" and detects that a) less than 10 seconds remain in the match and b) it is near one of the

corner bumper beacons by its starting square, the robot enters a special "go home" state. It drives back home, stops, and then plays a fun song using a piezo buzzer.

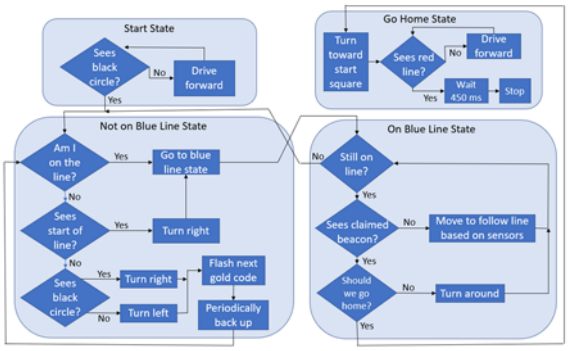

Figure 5. Control flow diagram for the robot algorithm. The robot switches between four states: a starting state, an ending state for returning to the home square, a state for being on a blue line, and a state for not being on a blue line. Each state has several behaviors, and the algorithm decides which behavior to implement based on sensor readings. These behaviors were sorted into different states to allow the robot to perform more actions and to reduce the effect of sensor noise.

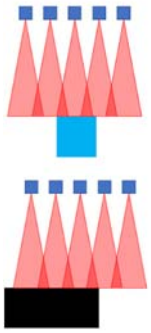

Figure 6. An illustration of how the robot's IR sensor array detects elements painted on the field. Note that each sensor detects more than just the ground directly beneath it.

The algorithm features several unique elements: a method of flashing Gold Codes to increase the rate at which the robot can read sensors and update its behavior, a way of following line edges intelligently using an array of IR-reflectance sensors, control states to augment robustness against sensor noise, and a method for getting unstuck when lost or pinned.

To minimize jerky or sinusoidal motion when line-following, the robot should collect new data and adapt its movement as often as possible. In initial tests, the robot repeatedly flashed all Gold Codes in order. However, flashing even a single 31-bit Gold Code at 250 µs/bit takes almost 8 milliseconds. Thus, the robot could not collect data often and responded slowly to its environment. In later iterations the robot flashed just one Gold Code each time it went through its code loop. This let it collect data approximately 4x as often and thus drive more smoothly.

Another benefit comes from how the IR-reflectance sensor array is used to follow blue lines and the black circle. The IR-reflectance sensors do not only sense the reflectance of the point directly below them. Since the surface does not exhibit perfect specular reflection, the IR-reflectance of nearby areas also affects the sensed value (Figure 6). Taking advantage of this fact allows more precise measurements of line edges than just using reflectance readings as binary indicators of whether lines are directly below the sensor. This method allows programmers to choose a precise radius for circling the black zone. The radius can be optimized so the center beacons can precisely read the robot's Gold Codes while the robot can reliably spot the start of blue lines. This is also used to enable the robot to determine the relative location of the blue line. Then the robot uses proportional control to better follow the line.

Another key element of the algorithm is the use of multiple control states to govern robot behavior. This technique allows for more accuracy in determining whether the robot is on a blue line: stricter thresholds are used when switching to a different state than when remaining in the same state. This reduces the impact of sensor noise on the robot's behaviors. It also allowed easier code development and allowed the robot to respond to scenarios with a myriad of behaviors.

Finally, the robot attempts to get unstuck when it is lost, in a corner, or pinned by another robot. The robot backs up once every eight seconds if it is in the "on blue line" or "not on blue line" states. The robot backs up far enough that it should end up in a new place. This aims to prevent the robot from getting lost and spinning in circles continuously. In the final competition, it also helped the robot get unpinned.

5. RESULTS

INITIAL TESTING

The EDC robot generally followed the arena lines and captured beacons. However, at several points the robot stopped following the blue lines and spun in circles because it misinterpreted the data from its reflectance sensors. Some dirty gray regions were adjacent to the blue lines the robot used to navigate. The robot did not see a clear blue/white divide, so it kept spinning, looking for a nonexistent boundary. We addressed this issue in two ways: modifying the range of values considered "blue" and adjusting the robot's movement code. The movement code adjustment makes the robot drive backwards at regular intervals. Thus, when it is lost, it can return to an area with landmarks instead of spinning indefinitely. We also timed the period of reversing such the robot moves in an overall linear direction when driving in circles and periodically reversing. This increases the chance that it can make its way to a blue line or the center circle.

SCRIMMAGE

The robot narrowly lost the first round; it claimed a beacon, but its opponent claimed multiple. It also began spinning in circles partway through the match despite our earlier algorithm changes. Our robot won the second round without issue; it claimed a beacon and its opponent did not. In the final round, the robot claimed a beacon but then collided with its opponent and reached a deadlock. The robot was backed up against the arena's center pillar, so it could not escape even with its frequent reversing. This deadlock lasted the rest of the match, but since our opponent did not score, our robot narrowly won the match and the scrimmage.

Two main issues occurred during the scrimmage: the robot spun as before, and it had difficulty claiming beacons. To resolve the spinning issue, we adjusted the blue threshold several times after the scrimmage. However, we concluded no one blue threshold would resolve spinning for all 4 blue lines, because some parts of the arena were dirtier (grayer) than others. However, we concluded that this issue would likely resolve itself when the board was re-painted. Our robot also had difficulties claiming beacons; it flashed Gold Codes, but the beacons did not always register them. We resolved this issue by installing more LEDs on the robot's sides. Each side now features 3 LEDs instead of 1, so the robot's Gold Code broadcasts are brighter.

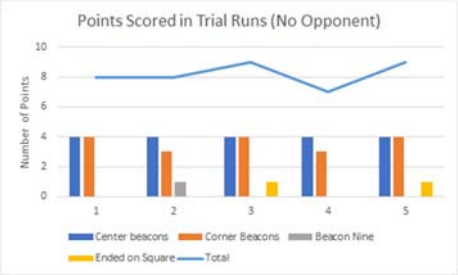

Figure 7. Points scored in five trial runs during the weekend before the final competition. The field was set up in the competition configuration and the robot was given 70 seconds to score as many points as possible without an opponent. Points are organized into categories based on how they were scored. The robot demonstrated the capability to score all three categories of beacons and the ability to navigate to the starting square at the end.

FINAL COMPETITION

Our robot participated in 3 rounds: two preliminary rounds and the quarterfinal. It won its first preliminary round, lost its second, and lost the quarterfinal to the overall first place winner.

During the competition, our robot captured beacons around the black central circle. It could not detect and follow the blue lines as they were covered with reflective tape, not matte paint as expected. Since many teams did not follow the blue lines, the center of the arena was often crowded; indeed, all three rounds the robot collided with its opponent. Luckily the robot often untangled itself by reversing.

When the robot collided with its opponents, it was pushed around and often got stuck. In the first match, our robot got stuck on a wall, freed itself, but it hit the other robot and got stuck again. Our robot barely won the round. In the second match, our robot was trapped again because its opponent hit it from behind. Our robot lost this round, but it advanced to the quarterfinals as the 8th place seed. In the third match, the opponent had two fast robots. Our robot captured the center beacons and was briefly in the lead, but it got stuck on an opponent robot and flipped over. This left our robot immobile. The opponents quickly captured EDC's beacons, eliminating our robot from the round and the competition. The team that eliminated our robot eventually won the competition.

6. Lessons Learned

Designing and building our robot taught us several skills and lessons. With the competition finished, we also have ideas on how we could improve our robot in the future.

First, the course taught us how to work with new tools. Not all of our team had experience soldering or machining parts prior to the course. The early labs, in which we soldered boards and constructed metal components, taught us these technical skills that will no doubt prove useful in our future endeavors at Harvey Mudd and beyond.

Programming on the Arduino taught us how to program within computational constraints. For example, we encountered trouble reading Gold Codes during testing; the robot misread codes if we asked it for the current time, a slow operation, too frequently. We thus learned the importance of efficient program design.

If we had more time, we would have implemented our original robot design. We planned to use two robots, one atop the other. We would also work on obstacle avoidance, such as maneuvering around an opponent robot by detecting its presence. We would also try to make the robot stop spinning in circles. Finally, we could have included two Mudduino boards on one robot. The boards could have used parallel computing, communicating with each other using I2C or similar methods. This would increase the computational power of the robot and enable us to use more complicated computational techniques to analyze our robot's surroundings and plan paths and behaviors.

7. References

Maria-Klawe. www.themarysue.com/wp-content/uploads/2011/09/maria-klawe.jpg.

8. CODE APPENDIX

APPENDIX A: MUDDUINO CODE

robot_code

```
[1] /**
[2]  * Behavior list: (divided among three (and a half) states)
[3]  * special: at start, go forward until find black circle (see start)**
[4]  * special: if at end of game and at one of the claimed beacons next to start,
     try to return to start **
[5]  * if see unclaimed bump beacon, slow down and go forward along blue line**
[6]  * if see claimed bump beacon, turn around**
[7]  * if found blue line, follow it**
[8]  * if on black circle, turn right to stay on circle
[9]  * if lost, bank turn left
[10]   */
[11]
[12]   #define LEDpin 13
[13]   #define LEDpin2 10
[14]   #define ninePin2 5
[15]   #define teamPin 3
[16]   #define ninePin 2
[17]   #define buzzerPin 4
[18]   boolean whiteTeam = false;
[19]   int gcMult = -1;
[20]
[21]   //IR reflectance thresholds
[22]   #define lineThreshold 790
[23]   #define circleThreshold 640
[24]
[25]   //white = 970 +- 10
[26]   //black circle = 590 +- 25
[27]   //blue line = 700 +- 70
[28]
[29]   //line following speeds
[30]   #define outSpeedR 0.65
[31]   #define outSpeedL 0.2
[32]   #define inSpeedL 0.7
[33]   #define inSpeedR -0.3
[34]   #define blueSpeedL 0.6
[35]   #define blueSpeedR -0.4
[36]
[37]   unsigned long startTime;
[38]   unsigned long stuckTime;
[39]
[40]   void setup()
```

```
[41]   {
[42]     tone(buzzerPin, 500);
[43]     Serial.begin(9600);
[44]     initMotors();
[45]     initSensors();
[46]     pinMode(LEDpin, OUTPUT);
[47]     pinMode(LEDpin2, OUTPUT);
[48]     pinMode(ninePin, OUTPUT);
[49]     pinMode(ninePin2, OUTPUT);
[50]     pinMode(teamPin, INPUT);
[51]     digitalWrite(LEDpin, LOW);
[52]     digitalWrite(LEDpin2, LOW);
[53]     startTime = millis();
[54]     delay(200);
[55]     whiteTeam = (digitalRead(teamPin) == LOW);
[56]     if(whiteTeam) { gcMult = 1; }
[57]     noTone(buzzerPin);
[58]
[59]     //start "state":
[60]     //behavior (special): at start, go forward until it finds black circle
[61]     setR((int) (255 * 0.6));
[62]     setL((int) (255 * 0.8));
[63]     while(readIrAvg() > 870)
[64]     { delay(1); }
[65]     setR((int) (255 * inSpeedR));
[66]     setL((int) (255 * inSpeedL));
[67]     delay(50);//*/
[68]     halt();
[69]     stuckTime = millis();
[70]   }
[71]
[72]   /**
[73]    * Main state ("loop")
[74]    * Behaviors:
[75]    * if found blue line, turn to follow it and switch to blue line state
[76]    * if on black circle, turn right to stay on circle
[77]    * if lost, bank turn left
[78]    */
[79]    int ttt = 0;
[80]   void loop()
[81]   {
[82]     double moveSpeed = 200.0; ttt++;
[83]     if(ttt > 200) //try to get unstuck
[84]     { backUp(); }
[85]     else if(getBlueLineLoc() != -1) //if currently on blue line, jump to blue
   line state
```

```
[86]     { halt(); followBlueLine(); }
[87]     else if((readIr(1) + readIr(2) + readIr(3))/3 < 720 + random(50)
[88]       and ((readIr(4) > 720 + random(20)
[89]       and readIr(4) < 900 + random(50))
[90]       and (readIr(5) > 720 + random(20)
[91]       and readIr(5) < 900 + random(50)))) //if it sees the blue line and is on
   black circle, turn right and go to blue line state
[92]     {
[93]       setR((int) (moveSpeed * blueSpeedR));
[94]       setL((int) (moveSpeed * blueSpeedL));
[95]       delay(600);
[96]       followBlueLine(); //go to following blue line state
[97]     }
[98]     else if(readIrAvg() < 720 + random(100)) //if in circle, turn right
[99]     {
[100]      setR((int) (moveSpeed * inSpeedR));
[101]      setL((int) (moveSpeed * inSpeedL));
[102]      flashNextCode();
[103]    }
[104]    else //if lost, bank turn left
[105]    {
[106]      setR((int) (moveSpeed * outSpeedR));
[107]      setL((int) (moveSpeed * outSpeedL));
[108]      flashNextCode();
[109]    }
[110]  }
[111]
[112]  void backUp()
[113]  {
[114]    if(millis() > stuckTime + 8000)
[115]    {
[116]      tone(buzzerPin, 400);
[117]      setR(-128); setL(-128);
[118]      delay(500);
[119]      halt();
[120]      stuckTime = millis();
[121]      noTone(buzzerPin);
[122]    }
[123]    ttt = 0;
[124]  }
```

blue_line_code

```
[1] /**
[2]  * State for following blue line
[3]  * Behaviors:
[4]  * if see unclaimed bump beacon, slow down and go forward along blue line**
[5]  * if see claimed bump beacon and time is almost up, go to go home state **
```

```
[6]  * else if see claimed bump beacon, turn around**
[7]  * if can't find blue line anymore, return to main state
[8]  */
[9]  double blueMoveSpeed = 120.0;
[10]    unsigned long lastTime;
[11]    int t;
[12]   #define timeToGoHome 60000
[13]   void followBlueLine()
[14]   {
[15]     t = 0;
[16]     lastTime = millis();
[17]     double blueLoc = getBlueLineLoc();
[18]     while(blueLoc != -1)
[19]     { t++; ttt++;
[20]       int bNum = readBeaconWell();
[21]       if(ttt > 200) //try to get unstuck
[22]       { backUp(); }
[23]       else if(isCornerBeacon(bNum) and not (XOR(bNum < 0, whiteTeam)))
[24]       { //if see claimed bump beacon:
[25]         if(millis() > startTime + timeToGoHome) //if time is almost up, go
   home
[26]         { headHome(bNum % 2 == 0); }
[27]         else //otherwise, turn around
[28]         {
[29]           tone(buzzerPin, 800);
[30]           setR(-255); setL(-255);
[31]           delay(250);
[32]           setR(255); setL(-255);
[33]           delay(200);
[34]           noTone(buzzerPin);
[35]         }
[36]       }
[37]       else if (t > 50)
[38]       {
[39]         unsigned long m = millis();
[40]         if(m > lastTime + 6000)
[41]         {
[42]           setR(-255); setL(-255);
[43]           delay(100);
[44]           halt();
[45]           lastTime = m;
[46]         }
[47]         t = 0;
[48]       }
[49]       else //if it's on the blue line and it doesn't see any claimed corner
   beacons, follow the line
[50]       {
```

```
[51]         int turnRightAmt = blueMoveSpeed * 0.2 * (blueLoc - 3);
[52]         setR(blueMoveSpeed * 0.8 - turnRightAmt);
[53]         setL(blueMoveSpeed * 0.7 + turnRightAmt);
[54]       }
[55]       blueLoc = getBlueLineLoc();
[56]     }
[57]   }
[58]
[59]   double getBlueLineLoc()
[60]   {
[61]     //classify what color the IR readers are on
[62]     int IR[] = { readIr(1), readIr(2), readIr(3), readIr(4), readIr(5) };
[63]     int irClassified[5];
[64]     for(int i = 0; i < 5; i++)
[65]     { irClassified[i] = classifyIR(IR[i]); }
[66]
[67]     //count black and blue readings, and also sum up the total of the reading
   classification values
[68]     int countBlack = 0; int countBlue = 0; double sum = 0.0;
[69]     for(int i = 0; i < 5; i++)
[70]     {
[71]       if(irClassified[i] == 2)
[72]       { countBlack++; sum+= i + 1; }
[73]       if(irClassified[i] == 1)
[74]       { countBlue++; sum+= i + 1; }
[75]     }
[76]
[77]     //if no clear blue line or too many black circle readings, return -1
   (invalid)
[78]     if(countBlack > 1 or (countBlue + countBlack) == 0) { return -1; }
[79]
[80]     //turn towards area with highest number (darkest spot)
[81]     return (sum/(countBlack + countBlue));
[82]   }
[83]
[84]   //0 = white, 1 = blue, 2 = black
[85]   int classifyIR(int ir)
[86]   {
[87]     if(ir > 720 + random(20) and ir < 900 + random(50)) //blue
[88]     { return 1; }
[89]     else if (ir > 880) //white
[90]     { return 0; }
[91]     else //black
[92]     { return 2; }
[93]   }
[94]
[95]   bool amOnBlueLineNow()
```

```
[96]  {
[97]    int bNum = readBeaconWell();
[98]    if(isCornerBeacon(bNum) and (XOR(bNum > 0, whiteTeam)))
[99]    { return false; }
[100]
[101]   //classify what color the IR readers are on
[102]   int IR[] = { readIr(1), readIr(2), readIr(3), readIr(4), readIr(5) };
[103]   int irClassified[5];
[104]   for(int i = 0; i < 5; i++)
[105]   { irClassified[i] = classifyIR(IR[i]); }
[106]
[107]   //count black and blue readings in center
[108]   int countBlack = 0; int countBlue = 0;
[109]   for(int i = 1; i < 4; i++)
[110]   {
[111]     if(irClassified[i] == 2)
[112]     { countBlack++; }
[113]     if(irClassified[i] == 1)
[114]     { countBlue++; }
[115]   }
[116]
[117]   if(countBlack < 2 and countBlue + countBlack > 1
[118]     and IR[0] > 720 == 0 and IR[0] > 720)
[119]   { return true; }
[120]   else
[121]   { return false; }
[122] }
[123]
[124] int cornerBeacons[] = {1, 2, 3, 4};
[125] bool isCornerBeacon(int bNum)
[126] {
[127]   bNum = abs(bNum);
[128]   for(int i = 0; i < 4; i++)
[129]   {
[130]     if(bNum == cornerBeacons[i])
[131]     { return true; }
[132]   }
[133]   return false;
[134] }
[135]
[136] bool isHomeBeacon(int bNum)
[137] {
[138]   bNum = abs(bNum);
[139]   return (((bNum == 2 or bNum == 3) and whiteTeam)
[140]        or ((bNum == 1 or bNum == 4) and (not whiteTeam)));
[141] }
```

goldcode

```
[1] #define thresholdCor 23
[2] #define LEDpin 13
[3] #define LEDpin2 10
[4] #define ninePin2 5
[5] #define ninePin 2
[6]
[7] #define gcLength 31
[8] #define numRegisters 5
[9] #define numGoldCodes 8
[10]   #define intervalFlash 250 //micro sec
[11]   boolean GC[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},
[12]                        {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},
[13]                        {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},
[14]                        {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},
[15]                        {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},
[16]                        {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},
[17]                        {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},
[18]                        {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},
[19]                        {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}};
[20]                        //{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}};
[21]
[22]
[23]   ////////////// reading gc from beacon ////////////////
[24]
[25]   int readBeacon()
[26]   {
[27]     int thisIs = readCode();
[28]     int beaconNum = thisIs/100; //abs(thisIs)/100
[29]     return beaconNum;
[30]   }
[31]
[32]   int readBeaconWell()
[33]   {
[34]     int c = readCode();
[35]     int beaconNum = c/100;
[36]     if ((abs(c) % 100) < 20) { return 0; }
[37]     if(readBeacon() != beaconNum) { return 0; }
```

```
[38]     if(readBeacon() != beaconNum) { return 0; }
[39]     return beaconNum;
[40]   }
[41]
[42]   int readCode()
[43]   {
[44]     //read light values
[45]     unsigned long startReadTime = micros();
[46]     int lightVals[gcLength] = {};
[47]     int avgVal = 0;
[48]     for(int i = 0; i < gcLength;)
[49]     {
[50]       //Serial.println(micros() - startReadTime);
[51]       if(micros() > startReadTime + (intervalFlash * i))
[52]       {
[53]         int photo = readPhoto();
[54]         lightVals[i] = photo;
[55]         avgVal += photo;
[56]         i++;
[57]       }
[58]     }
[59]     //convert read values to binary gold code
[60]     boolean code[gcLength] = {};
[61]     avgVal = avgVal / gcLength;
[62]     for(int i = 0; i < 31; i++)
[63]     {
[64]       if(lightVals[i] > avgVal)
[65]       { code[i] = 0; }
[66]       else
[67]       { code[i] = 1; }
[68]     }
[69]
[70]     //correlation
[71]     int thisIs = whichCode(code);
[72]     /*Serial.print("ID # = "); Serial.println(abs(thisIs) / 100);
[73]     Serial.print("Corresponding correlation = "); Serial.println(thisIs %
   100);
[74]     Serial.print("Team color = ");
[75]     if(thisIs > 0) { Serial.println("White"); } else {
   Serial.println("Green"); } //*/
[76]     return thisIs;
[77]   }
[78]
[79]   //for weird reasons I decided to make this return 100 * the number of the
   code (indexed from 1) + the absolute value of the correlation
[80]   //also, the returned value is negative for "inverse" correlation (i.e. green
   team) and positive for white team
```

```
[81]   int whichCode(boolean code[gcLength])
[82]   {
[83]     int maxCorrelation = 0;
[84]     int bestFitCode = 0;
[85]     for(int i = 0; i < numGoldCodes; i++)
[86]     {
[87]       int corr = getCorrelation(code, GC[i]);
[88]       if(abs(corr) > abs(maxCorrelation))
[89]       { maxCorrelation = corr; bestFitCode = i; }
[90]     }
[91]     //return (bestFitCode + 1) * maxCorrelation / abs(maxCorrelation);
[92]     return (((bestFitCode + 1)*100) +  abs(maxCorrelation)) * (maxCorrelation
   / abs(maxCorrelation));
[93]   }
[94]
[95]   int getCorrelation(boolean code1[gcLength], boolean code2[gcLength])
[96]   {
[97]     int max = 0;
[98]     //find correlation at each offset
[99]     for(int k = 0; k < gcLength; k++)
[100]    {
[101]      int temp = getDotProduct(code1, code2, k);
[102]      //Serial.print(String(temp) + " ");
[103]      if(abs(temp) > abs(max)) { max = temp; }
[104]    }
[105]    return max;
[106]  }
[107]
[108]  int getDotProduct(boolean code1[gcLength], boolean code2[gcLength], int
   offset2)
[109]  {
[110]    int total = 0;
[111]    for(int i = 0; i < gcLength; i++)
[112]    {
[113]      int loc = i + offset2;
[114]      if(loc >= gcLength) { loc -= gcLength; }
[115]      total += dot(code1[i], code2[loc]);
[116]    }
[117]    return total;
[118]  }
[119]
[120]  int dot(int i, int j)
[121]  {
[122]    if(i == j) { return 1; }
[123]    else { return -1; }
[124]  }
[125]
```

```
[126] ////////// flashing gc to beacon /////////////////
[127]
[128] void flashCode(int beaconNum)
[129] {
[130]   unsigned long startReadTime = micros();
[131]   int interval = 250;
[132]   for(int i = 0; i < gcLength;)
[133]   {
[134]     if(micros() > startReadTime + (interval * i))
[135]     {
[136]       if(XOR(GC[abs(beaconNum - 1)][i], whiteTeam)) { digitalWrite(LEDpin,
   HIGH); digitalWrite(LEDpin2, HIGH); }
[137]       else { digitalWrite(LEDpin, LOW); digitalWrite(LEDpin2, LOW); }
[138]       i++;
[139]     }
[140]   }
[141] }
[142]
[143] boolean XOR(boolean a, boolean b)
[144] { return((a && !b) || (!a && b)); }
[145]
[146] void flashCodeAndNine(int beaconNum)
[147] {
[148]   unsigned long startReadTime = micros();
[149]   int interval = 250;
[150]   for(int i = 0; i < gcLength;)
[151]   {
[152]     if(micros() > startReadTime + (interval * i))
[153]     {
[154]       if(XOR(GC[abs(beaconNum - 1)][i], whiteTeam)) { digitalWrite(LEDpin,
   HIGH); digitalWrite(LEDpin2, HIGH); }
[155]       else { digitalWrite(LEDpin, LOW); digitalWrite(LEDpin2, LOW); }
[156]       if(XOR(GC[abs(9 - 1)][i], whiteTeam)) { digitalWrite(ninePin, HIGH);
   digitalWrite(ninePin2, HIGH); }
[157]       else { digitalWrite(ninePin, LOW); digitalWrite(ninePin2, LOW); }
[158]       i++;
[159]     }
[160]   }
[161]   digitalWrite(LEDpin, LOW);
[162]   digitalWrite(ninePin, LOW);
[163]   digitalWrite(LEDpin2, LOW);
[164]   digitalWrite(ninePin2, LOW);
[165] }
[166]
[167] int codesToFlash[] = { 5, 6, 7, 8 }; //who do we appreciate
[168] int curCodeIdx = 0;
[169] //this takes ~8 milliseconds
```

```
[170] void flashNextCode()
[171] {
[172]   flashCodeAndNine(codesToFlash[curCodeIdx]);
[173]   curCodeIdx++;
[174]   if(curCodeIdx > 4)
[175]   { curCodeIdx = 0; }
[176] }
```

home_code

```
[1] /**
[2]  *   The song played upon completing the game is the death sound effect from
     Super Mario
[3]  *   Bros. The notes were transcribed into frequencies by hand by me from sheet
     music
[4]  *   posted to http://www.mariopiano.com/mario-sheet-music-death-sound.html.
[5]  */
[6]
[7] /**
[8]  * Go home (final/special) state
[9]  * if at end of game and at one of the claimed beacons next to start, try to
     return to start **
[10]    */
[11]   void headHome(bool rightSide) //state for heading home at end of game
[12]   {
[13]     if(rightSide)
[14]     {
[15]       //turn towards box
[16]       setR(0.1 * 255);
[17]       setL(0.9 * 255);
[18]       delay(450);
[19]       //go to the box
[20]       setR((int) (255 * 0.65));
[21]       setL((int) (255 * 0.80));
[22]     }
[23]     else
[24]     {
[25]       //turn towards box
[26]       setR(0.75 * 255);
[27]       setL(0.1 * 255);
[28]       delay(450);
[29]       //go to the box
[30]       setR((int) (255 * 0.55));
[31]       setL((int) (255 * 0.95));
[32]     }
[33]     while(readIrAvg() > 900)
[34]     { delay(1); }
[35]     delay(300);
```

```
    halt();

    //play a song b/c yay
    playEndSong();
    while(true)
    { delay(1); }
  }

  int melody[] = {
    1047, 1109, 1175, -1,
    784, 1175, -1, 1175,
    1175, 1047, 988,
    784, 659, -1, 659,
    523
  };
  #define s 1000
  int tempo[] = {
    s/16, s/16, s/8, 3*s/4,
    s/4, s/4, s/4, s/4,
    s/3, s/3, s/3,
    s/4, s/4, s/4, s/4,
    s/4, -1
  };

  void playEndSong()
  {
    int totalTime = 0; int i = 0;
    int currentTime = 0;
    while (true)
    {
      if (currentTime > totalTime)
      {
        if (tempo[i] < 0) { noTone(buzzerPin); return; }
        totalTime += tempo[i];
        tone(buzzerPin, 0);
        delay(7);
        tone(buzzerPin, melody[i]);
        i++;
      }
      currentTime++;
      delay(1);
    }
  }
```

motor_cn

```
#define LPlus 9
```

```
[2] #define LMinus 8
[3] #define RPlus 7
[4] #define RMinus 12
[5] #define LEN 6
[6] #define REN 11
[7]
[8] int powerLevel = 150;
[9]
[10]   void initMotors()
[11]   {
[12]     pinMode(LPlus, OUTPUT);
[13]     pinMode(LMinus, OUTPUT);
[14]     pinMode(RPlus, OUTPUT);
[15]     pinMode(RMinus, OUTPUT);
[16]     pinMode(REN, OUTPUT);
[17]     pinMode(LEN, OUTPUT);
[18]     digitalWrite(LEN, HIGH);
[19]     digitalWrite(REN, HIGH);
[20]     halt();
[21]   }
[22]
[23]   void halt()
[24]   {
[25]     analogWrite(RPlus, 0);
[26]     analogWrite(RMinus, 0);
[27]     analogWrite(LPlus, 0);
[28]     analogWrite(LMinus, 0);
[29]   }
[30]
[31]   void setR(int i)
[32]   {
[33]     if(i>=0)
[34]     {
[35]       analogWrite(RPlus, min(i,255));
[36]       analogWrite(RMinus, 0);
[37]     }
[38]     else
[39]     {
[40]       analogWrite(RPlus, 0);
[41]       analogWrite(RMinus, max(-i,-255));
[42]     }
[43]   }
[44]
[45]   void setL(int i)
[46]   {
[47]     if(i >= 0)
```

```
    {
      analogWrite(LPlus, min(i,255));
      analogWrite(LMinus, 0);
    }
    else
    {
      analogWrite(LPlus, 0);
      analogWrite(LMinus, max(-i,-255));
    }
  }

  //general function things:
  void setPowerLevel(int pwr)
  {
    if(powerLevel > 255) { powerLevel = 255; }
    else if (powerLevel < -255) { powerLevel = -255; }
    else { powerLevel = pwr; }
  }

  void forward()
  {
    setR(powerLevel);
    setL(powerLevel);
  }

  void backward()
  {
    setR(-powerLevel);
    setL(-powerLevel);
  }

  void turnR()
  {
    setR(-powerLevel);
    setL(powerLevel);
  }

  void turnL()
  {
    setR(powerLevel);
    setL(-powerLevel);
  }
```

sensors

```
#define photoSensor 14
#define irReflectSensor1 15
```

```
#define irReflectSensor2 16
#define irReflectSensor3 17
#define irReflectSensor4 18
#define irReflectSensor5 19

int irSensorPins[5] = { irReflectSensor1, irReflectSensor2, irReflectSensor3,
irReflectSensor4, irReflectSensor5 };

  void initSensors()
  {
    pinMode(photoSensor, INPUT);
    pinMode(irReflectSensor1, INPUT);
    pinMode(irReflectSensor2, INPUT);
    pinMode(irReflectSensor3, INPUT);
    pinMode(irReflectSensor4, INPUT);
    pinMode(irReflectSensor5, INPUT);
  }

  void printAllSensors()
  {
    Serial.print("Phototransistor Sensor: ");
    Serial.print(readPhoto());
    Serial.print(" and IR Reflectance Sensors: ");
    for(int i = 0; i < 5; i++)
    { Serial.print(readIr(i+1)); if(i != 4) { Serial.print(", "); } }
    Serial.println();
  }

  int readSensor(int pin)
  { return analogRead(pin - 14); }

  int readPhoto()
  { return readSensor(photoSensor); }

  int readIr(int num)
  { return readSensor(irSensorPins[num-1]); }

  int readIrAvg()
  {
    int total = 0;
    for (int i = 0; i < 5; i++)
    { total += readIr(i+1); }
    return total/5;
  }
```