Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

The Autonomous Vehicle

Final Report, Hilary Term 2018

Josefine Klintberg
Calvin Hubbard
Jack Hay

Introduction

With sensors and control systems to analyse data, this project aimed to construct a self-driven Buggy that can operate in a simplified environment consisting of a white line on a black mat.

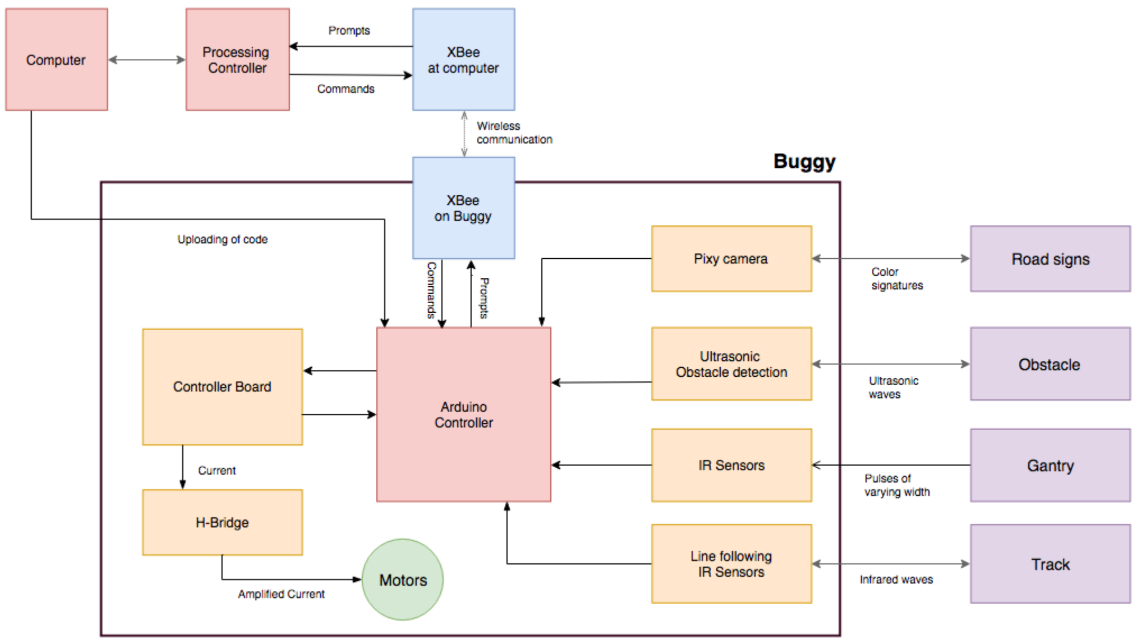
The project was divided in two parts that led to the Silver and the Gold challenge. To complete the silver challenge, the buggy needed to be able to drive two laps around a known track and report its progress to the computer telemetry, sense obstacles along the way and stop if an obstacle was detected. It also needed to use a pixy cam for taking shortcuts at forkings of the road, speed up and slow down according to specific color signatures. Lastly it also needed to sense gantries that were placed along the track and sended out pulses of varying width and report back to the telemetry which gantry it passed.

The gold challenge consisted of a previously unknown track that the buggy should perform as quickly as possible. This challenge opened up for a more creative approach to the programming and we could choose what hardware parts we wanted to use and not to use.

Table of contents

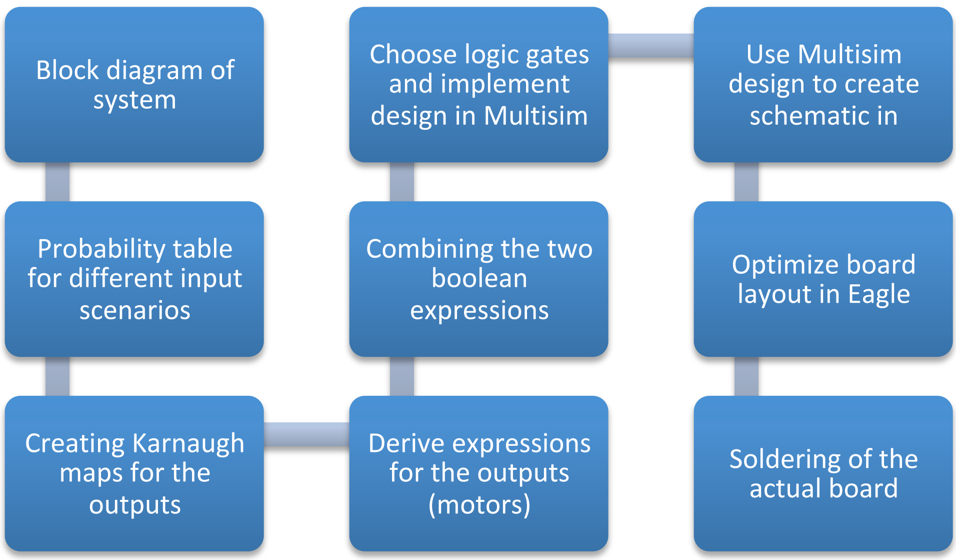
Block diagram

For an overview of the design for this project, we created a block diagram. This shows a description of how our computer's interface and the buggy operate.



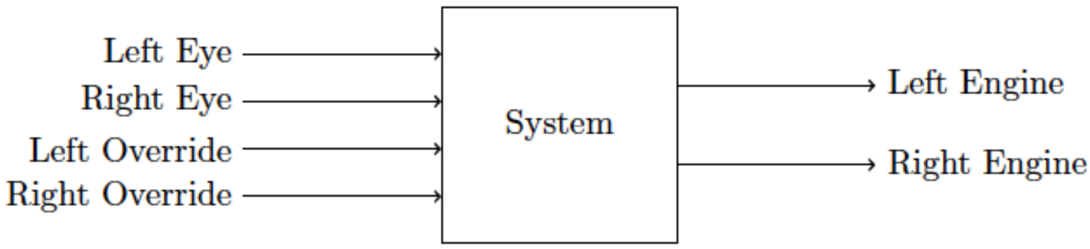
Design flow

The hardware development of our buggy can be broken down into multiple steps. An overview of the workflow that we've gone through can be seen in the following figure:



Model of the system

The first thing we had to do was to simplify our system. We used basic control theory to create a simple block diagram where we define our different inputs and outputs. We represent our buggy as a system that has 4 different inputs and 2 outputs. The inputs are the left eye and the right eye that is our line following sensors that can see the line on the floor and the overrides for respective eye. The outputs are the left and the right engine.



Derivation of the logic equations

Truth table

When we had clearly defined our inputs and outputs for the system, we could continue on with the derivation of the logic equations that will allow our buggy to follow the line on the floor. The line following sensors use infrared waves to read input. They can take one out of two values, 0 and 1. 0 is equal to a low threshold and is representing the eye value white and 1 is high threshold and representing eye value black. For the motors, 0 is representing engine turned off and 1 is engine turned on.

The purpose of deriving these equations is so that we can run the correct motors based on the inputs. With the left and the right eye we can make sure that the buggy stays on the line. The left and right override then allows us to override a low treshold value that the eyes are seeing with a high treshold value instead. The purpose of this is so that we can bypass the main track path and make the buggy turn and choose the way it should go when faced with a crossroad.

Since we have 4 different inputs, we can arrange them in 16 different combinations that are scenarios that can occur. We look at the different scenarios one by one to decide what should happen with the engines in this particular case for making the buggy move in the desired direction. We want it to stay on the line at all times and turn when the line turns, so we need it to recognize when it is moving out from the line. We put all the different cases into a truth table and this can be seen below:

L_{eye}	R_{eye}	L_o	R_o	L_{engine}	R_{engine}
0	0	0	0	1	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	1	1
0	1	1	1	1	1
1	0	0	0	1	0
1	0	0	1	1	1
1	0	1	0	1	0
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1

If we look at the first case, both the left and the right eye is seeing white and none of the overrides are giving a high value so the buggy is on the line and both off the engines should be turned on for the buggy to keep moving forward on the line.

This is similar to the last case when both eyes are seeing black and both overrides are on. Then the buggy is not on the line and both of the engines should be turned on to keep moving forward until the buggy detects a line.
Every case when both of the eyes is giving the same values, either both high or both low, (this can occur both by what the eyes are seeing or by overrides turning low to high) will give scenarios where both the engines should be turned on and the buggy will be moving straight forward.

Another scenario is when the buggy is supposed to turn. This can be seen in case number 5. Then the left eye is seeing white but the right eye is seeing black, still no high value from override, so we are a bit off track on the right side. To get back to the line, the left engine should be turned off and the right engine on so the buggy turns to the left. The exact same case but in the opposite direction is in case number 9.

Other than keeping on the line, as previous said, we need to be able to use the overrides to force the buggy to take a decided turn when facing a fork in the road. We can see an example of this in case number 3 in the truth table. Both of the eyes are seeing white so we are on the line but the left override forces the left eye to give a high treshold value and therefore signals that the left eye is seeing black (even though it's not) so the left engine is turned on and the right engine is turned off to enable the buggy to turn right. This is the case we want to use if we want the buggy to take a right turn in a crossroad.

By completing the truth table, we are able to move forward and continue on in the work of deriving the logic equations for the line following logic.

Karnaugh maps

In the next step, we transferred the values from our truth table into Karnaugh maps. This is a method where we use a two-dimensional grid and the cells in it are ordered in Gray code. Each cell is representing one combination of input and by identifying optimal groups we can use it to simplify our Boolean algebra expressions and from that implement physical logic gates.

R_eL_e \ R_oL_o	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	1	1	1	1
10	0	1	1	0

R_eL_e \ R_oL_o	00	01	11	10
00	1	0	1	1
01	0	0	1	1
11	1	1	1	1
10	1	1	1	1

From the two Karnaugh maps we can find the equations for our two outputs, the motors, which are as following:

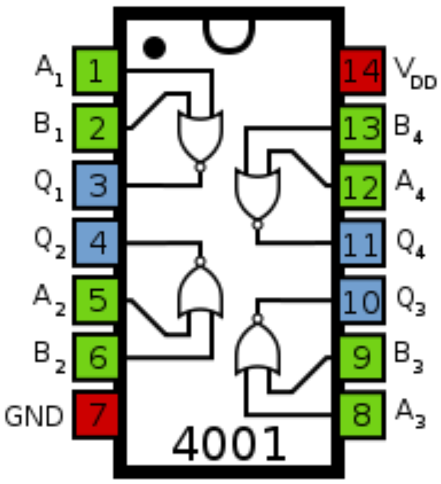
$$Right_{motor} = (L_e + L_o)' + R_e + R_o$$
$$Left_{motor} = (R_e + R_o)' + L_e + L_o$$

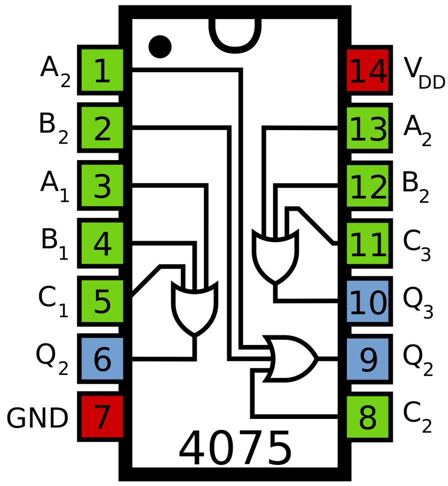
These equations are our line following logic and the next step was to actually design this by using logic gates. A logic gate is a physical device implementing a Boolean function. It performs a logical operation on binary inputs and produces a single binary output.

Implementation of the design

Logic gates

To decide which logic gates we should have in our design we used de Morgan's laws and we find out that we should use one NOR- and one OR-gate in the equation for respective engine. By putting the equations together we are able to use just two gates instead of four for optimization the circuit. We used 4000 series logic to implement our equations in Multisim. We used a two input NOR-chip (4001) and a three input OR-chip (4075). The NOR-gate is producing an output that is low only if all its inputs are high. The OR-gate gives a high output if one or both of the inputs are high. By looking up images of the chips that we decided to use, we got images of which pins that are inputs and which are outputs on respective chip.

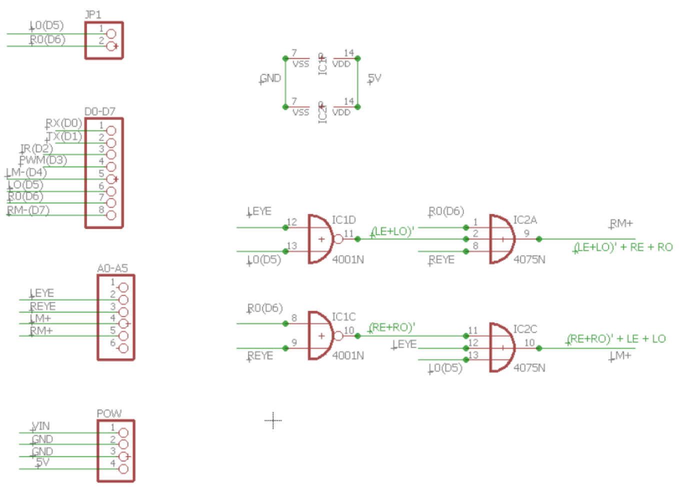




Multisim & Eagle

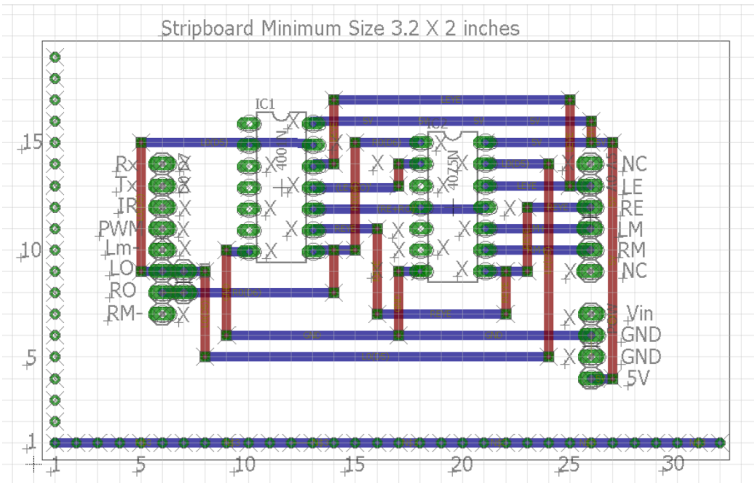
We used the software Multisim as a way to test that our logic equations are the correct ones by a test program. Once we cleared that everything worked properly, we moved on to using the software Eagle. The EAGLE (Easily Applicable Graphical Layout Editor) software is a scriptable electronic design automation and has two different views that are connected to one another. We used the schematic editor for designing our circuit diagram from the model we created in Multisim.

In the figure below, we can see our schematic from Eagle. In green, the gate outputs are shown for our implemented design. The total outputs for the engines are to be seen to the furthermost right in the figure and corresponds to our logic equations that we received before from the Karnaugh maps. We also connected the pins to the corresponding gate.

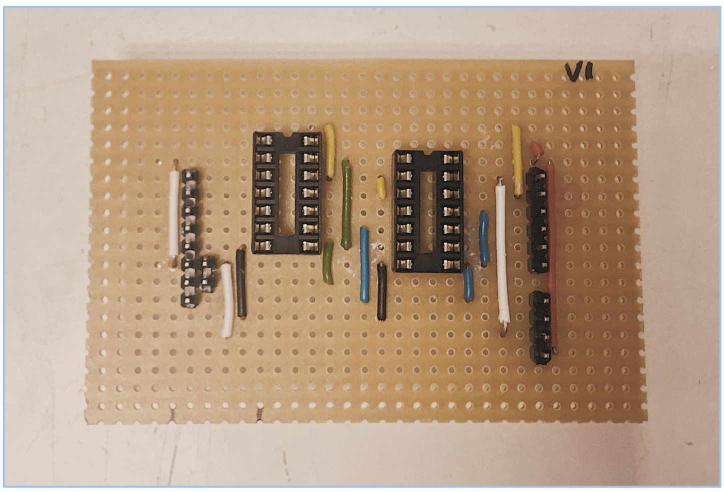


After finalizing our design in the schematic view, we switched into board view in Eagle and started to design our actual board. Our two components appeared on the layer and from that we made a sketch of how our wires should be connected by soldering. The printed circuit board layout allows back-annotation to the schematic and connects traces based on our defined connections. We started with locating the headers and IC sockets and then continued on with the wires.

The challenge in this part of the design process was to optimize as good as possible. We had minimalized the number of chips but we also wanted to minimize the number and length of wires that we used. This we tried to do both by thinking of the placement of the chips and also by using unused input gates in the chips to connect wires in a better way. We ended up with 8 unused gates out of 28 possible and we feel that we were successful with optimizing the lengths of the wires and got a simple and clean design that is easy to follow.



The image above shows our finalized board that we created in Eagle. The blue colour is representing the bottom layer and is the actual copper tracks on the stripboard. The red colour is the top layer and is the wires that we create by soldering. The breaks are displayed as crosses. The board after we had done all the soldering is to be seen in the image below and the colour coding is displayed in the table next to it.



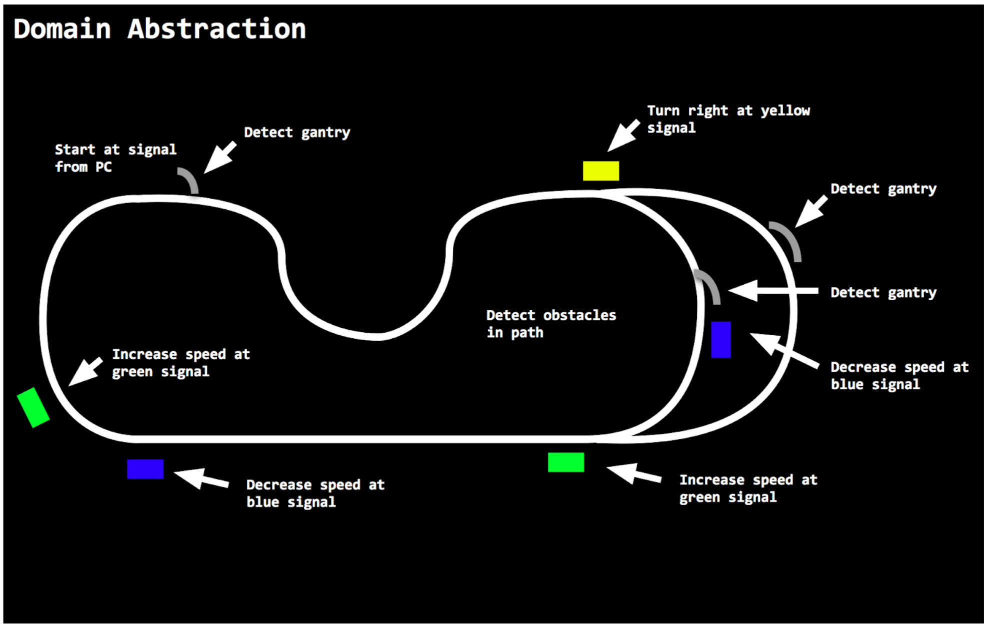
Color	Representing
Red	5V
Black	Ground
Yellow	Left eye
Blue	Right eye
White	Left override
Green	Right override

Our group achieved both the Silver and the Gold Challenge and in this section we will describe the software engineering design that we made during this project.

Silver Challenge: Software Engineering Design

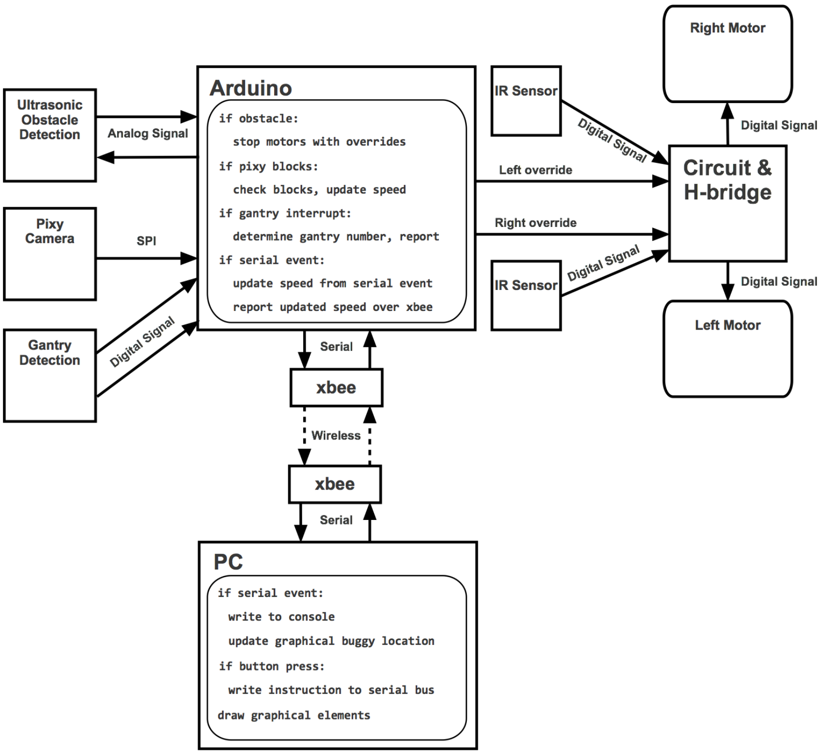
Domain Model

For the Silver Challenge, the track consisted of the following problems.



System Model

The following is a system model of our silver challenge buggy:



Telemetry Code Specifications

General:

All global variables and global constants have variable name with uppercase letters by convention. These are all declared at the top of the processing file before the declaration of functions. Included in the global variables are various list-based data structures used to group relevant information. These include integers that define the positions of buttons and floats that define graphical positions that the position-reporting buggy can take on the screen.

Functional decomposition:

The `setup()` function loads graphical elements from the data directory and attempts to setup a serial interface with the XBee. The `draw()` component invokes `draw_components()` which is used to simplify the `draw()` function. The `draw_components()` function draws graphical elements layer by layer and also handles any new serial communications that have been handled in the event handler. The flag `SERIAL_AVAILABLE` notifies the draw function that a new message is ready to be parsed. The `draw_components()` function distributed various graphical tasks to the supporting functions `draw_map()`, `draw_buggy_on_map()`, `draw_button()`, `add_console_message()`, `draw_graphical_console()`, `draw_window()`, `draw_text()`, `draw_component_at_angle()`.

Event driven functions:

The functions `send_instruction()`, `serialEvent()`, `button_clicked()`, and `mouseClicked()` are event-driven or support event-driven functions.

Telemetry to Arduino Signal Specifications

The main telemetry program communicates various speed control signals over the serial interface. These include four button driven speed controls:

Set motor to 0% speed	`send_instruction("x\n");`
Set motor to 50% speed	`send_instruction("s\n");`
Set motor to 78% speed	`send_instruction("m\n");`
Set motor to 100% speed	`send_instruction("f\n");`

The buggy will respond if its current speed is modified by the instruction and this message will be displayed on the telemetry console.

Arduino to Telemetry Signal Specifications

The buggy notifies the main telemetry system of any obstacles that it encounters and any gantries it passes. These messages take the following form:

Gantry/slow sign passed	location_event
Obstacle Detected	<dist> cm

These messages are displayed in the telemetry console and update the position of the graphical buggy in the Reported Location window.

Pseudo Code (supporting graphical functions omitted):

```
setup():
	init all relevant graphical components (windows, buttons)
	try:
		connect to serial interface
		set SERIAL_CONNECTED flag to true
		write relevant info to console
	catch:
		set SERIAL_CONNECTED flag to false
		write relevant info to console

loop: draw():
	<via draw_components()>
	draw graphical components using supporting functions
	if SERIAL_AVAILABLE:
		parse serial message from SERIAL_QUEUE
		write to console window
		update graphical with CURRENT_REPORTED_GANTRY

draw_buggy_on_map():
	use CURRENT_REPORTED_GANTRY data to display object

serialEvent():
	respond to new event from connected serial interface
	set SERIAL_AVAILABLE flag
	put message in SERIAL_QUEUE

mouseClicked():
	loop for all buttons:
		if mouseX && mouseY in bounds of button[i]:
			send corresponding instruction over serial interface

add_console_message():
	if new message available:
		prune current list
		if SERIAL_CONNECTED:
			add message
```

Arduino Code Specifications

Pseudo Code:

```
setup():
	establish serial interface
```

```
      establish all variables
      set pin modes
      set pins to digital write
      set interrupt pins
      turn motor off

loop() for detecting gantry and obstacles with IR-sensors:
      retrieve ultrasonic information
      Check if it’s an obstacle within 15cm ahead of the buggy and
      if it is, stop and report the speed to the computer
      if SERIAL_AVAILABLE:
           send to interpreting function
           set SERIAL_AVAILABLE to false
      if IrInterrupt:
           detect the pulse
           combine the pulse with the corresponding gantry
           report that the buggy passed a gantry and print out the
           name on which gantry it was
           set IrInterrupt variable to false

report_speed_percentage():
      calculate percentage with the help of the current speed and
      send over serial interface

interpret_master_instr():
      if the serial message matches predefined control signals we
      set the updated speed to the corresponding value

motors_granular_speed():
      writes a new value to the motor control pin

serialEvent():
      reads in new serial strings
      sets the SERIAL_AVAILABLE to true

IR_Interrupt():
      sets IrInterrupt variable to true
```

The following outlines the high-level control structures that we implemented in our code for the Silver challenge on the arduino:

```
setup()
      set input and output pins
      set interrupt pin for IR detection
      set individual motor speeds to 0
      set SERIAL_AVAILABLE to false
      set SERIAL_STRING to “”
```

```
        set irInterrupt, report, gantry1, gantry2, gantry3 to false
        set currTime and prevTime to 0
        set CURRENT_SPEED and UPDATE_SPEED to 0
        set YELLOW_DETECTED, BLUE_DETECTED, GREEN_DETECTED to false
        initiate the pixy cam
        set up the serial for xbee communication
loop()
   // ultrasonic sensor code
        send out an ultrasonic pulse for obstacle detection and..
   ..calculate distance
   if obstacle is < 15 cm away:
        set motor speed to 0
        if CURRENT_SPEED != 0:
        report speed is 0 to terminal
        report how far obstacle is to terminal
        set CURRENT_SPEED to 0
   else if CURRENT_SPEED == 0:
       //buggy is stopped but no obstacle is visible
       CURRENT_SPEED = UPDATE_SPEED
       set motors to CURRENT_SPEED
       if UPDATE_SPEED != 0:
        report CURRENT_SPEED to terminal
   else if CURRENT_SPEED != UPDATE_SPEED:
        // CURRENT_SPEED is out of date
        set motors to UPDATE_SPEED
        report UPDATE_SPEED to terminal
       CURRENT_SPEED = UPDATE_SPEED

   // pixy code
   check if the pixy cam detects any blocks
   if blocks detected is > 0:
       set max_index to 0
       set max_size to 0
       //loop through all the blocks and select the biggest
       for (int b=0;b<blocks;b++)
           if width*height of block is > max_size:
               set max_size to width*height
               set max_index to b //b is the index of..
               .. the current block
       set area_threshold to 2400
       set sig = the index of the biggest block
       if (max_size > area_threshold) && (sig == 1) && ..
       .. (YELLOW_DETECTED == false):
           //fork in the road
           //take the path to the right
           set the left override to HIGH
           delay(2000)
           //vehicle turns right
           set the left override to LOW
```

```
            report “yellow signal” to terminal
            report “location_event” to terminal
            YELLOW_DETECTED = true
            BLUE_DETECTED = false
            GREEN_DETECTED = false
        else if (max_size > area_threshold) && (sig == 2) &&..
        .. (BLUE_DETECTED == false):
            set the motors to a slower speed
            report “blue signal” to terminal
            report “location_event” to terminal
            YELLOW_DETECTED = false
            BLUE_DETECTED = true
            GREEN_DETECTED = false
        else if (max_size > area_threshold) && (sig == 3) &&..
        .. (GREEN_DETECTED == false):
            set the motors back to CURRENT_SPEED
            report “green signal” to terminal
            report “location_event” to terminal
            YELLOW_DETECTED = false
            BLUE_DETECTED = false
            GREEN_DETECTED = true

    // communication sent from main computer code
    if SERIAL_AVAILABLE == true:
        set SERIAL_AVAILABLE to false
        SERIAL.STRING.trim()
        interpret the SERIAL.STRING

    // infrared sensor code
    if irInterrupt == true
        //detect the pulse being sent from the gantry
        int pulse = pulseIn(IR_PIN,LOW)
        if (pulse >= 1000) && (pulse <= 1100)
            if gantry1 == false
                currTime = millis()
                if currTime - prevTime >= 2000
                    report = true
                if report == true
                    prevTime = millis()
                    report = false
                    gantry1 = true
                    gantry2 = false
                    gantry3 = false
                    report “Gantry 1” to terminal
                    report “location_event” to terminal
        if (pulse >= 2000) && (pulse <= 2100)
            if gantry2 == false
                currTime = millis()
                if currTime - prevTime >= 2000
```

```
                    report = true
                if report == true
                    prevTime = millis()
                    report = false
                    gantry1 = false
                    gantry2 = true
                    gantry3 = false
                    report “Gantry 2” to terminal
                    report “location_event” to terminal
        if (pulse >= 2900) && (pulse <= 3000)
            if gantry3 == false
                currTime = millis()
                if currTime - prevTime >= 2000
                    report = true
                if report == true
                    prevTime = millis()
                    report = false
                    gantry1 = false
                    gantry2 = false
                    gantry3 = true
                    report “Gantry 3” to terminal
                    report “location_event” to terminal
    set irInterrupt to false

// the following code are seperate functions that we call upon.. ..during the loop to help us
perform some of the functions

void send_data(String data)
    Serial.println(data)
    // whenever we want to report a string to the terminal we..
    ..use this function

void report_speed_percentage(int val)
    int percent = (val/255)*100
    //change the percent integer to a string
    String out = String(percent)
    //call on send_data function
    send_data(“Motor power: “ + out + “%”)
    // we use this function whenever we want to report that..
    ..there has been a speed change in our motors

void interpret_master_instr(String serialInput)
    if serialInput == “x”
        UPDATE_SPEED = 0;
    else if serialInput = “s”
        UPDATE_SPEED = 127
    else if serialInput = “m”
        UPDATE_SPEED = 150
    else if serialInput = “f”
```

```
        UPDATE_SPEED = 255
    //we use this function to control the speed from the main..
    ..computer

void motors_granular_speed(int val)
    analogWrite(motors pin, val)
    //we use this function to change the speed of the motors..
    ..within the loop

void serialEvent()
    read SERIAL_STRING until it ends (until it reads “\n”)
    set SERIAL_AVAILABLE to true
    // this function is responsible for receiving and reading..
    ..data that comes through the xbee from the main computer

void IR_ISR()
    irInterrupt = true
    // this is our interrupt function for the infrared detection
```

The following is a high-level overview of the processing system telemetry code running on a PC in tandem with the buggy:

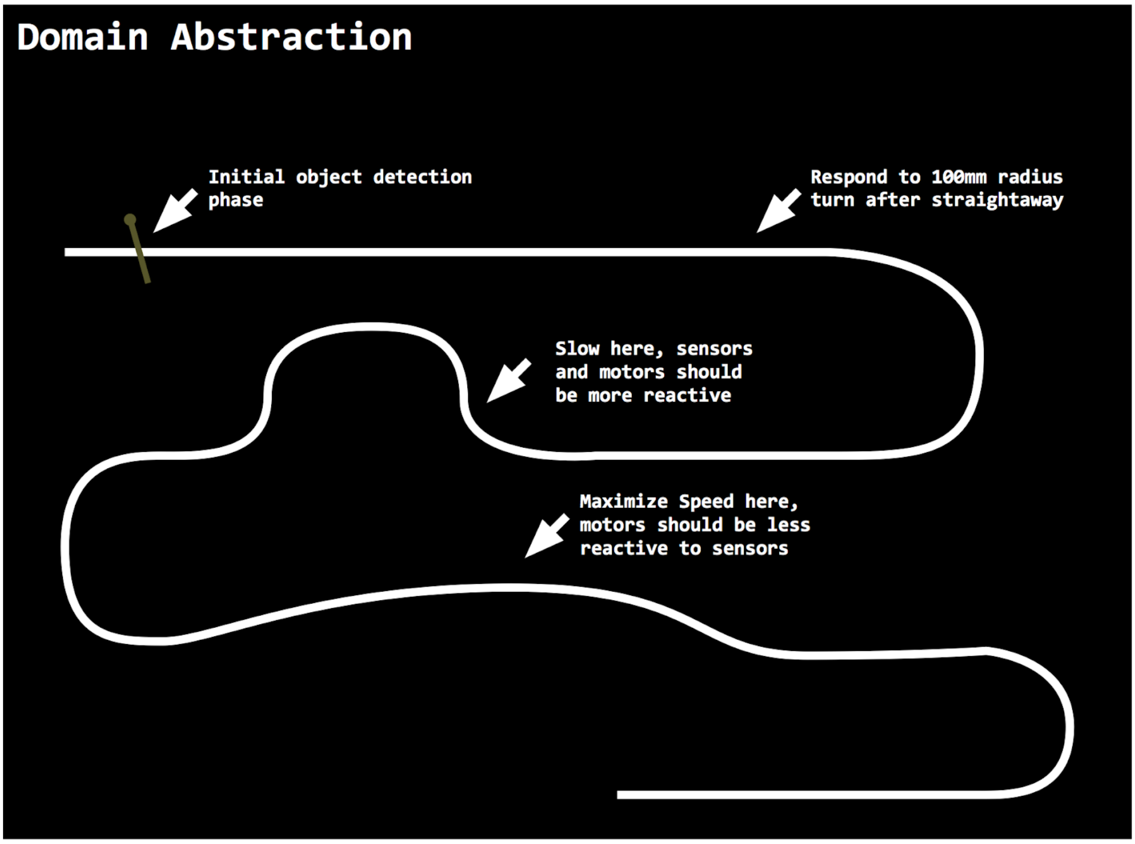
```
setup()
        connect to xbee serial interface
        initialize graphical elements
draw()
        call draw_components to draw graphical elements
        draw buggy given current location on track
        if SERIAL_AVAILABLE:
        parse message and display
draw_components()
        draw graphical components including track, console, and buttons
draw_button()
        draw a button object and text
add_console_message()
        add a new message to the console list and remove the oldest
draw_graphical_console()
        draw all messages in console list to console window
draw_window()
        draw a window object
draw_text()
        draw a text string to the window
send_instruction()
        if serial connected:
        send the given instruction over serial interface
serialEvent()
        event-handler for serial port
        set SERIAL_AVAILABLE to true and store input
```

```
draw_component_at_angle()
        draw an image to window at an angle
mouseClicked()
        mouseclick-handler
        if within a button in list:
        send instruction associated with that button
```

Gold Challenge: Software Engineering Design

Domain Model

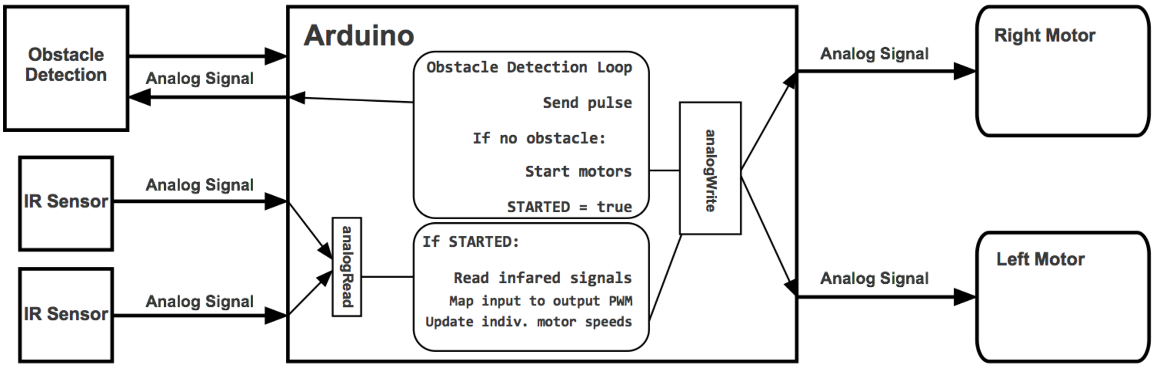
The following is a general abstraction of the challenge environment and outlines some of the key design considerations. We designed an infrared-to-motor system that was reactive enough for curved sections with a minimum radius of 100mm but also maximized speed and smooth travel on the straight sections. Our motor-control code is thus designed according to these two characteristics of the track. The starting gate required us to update our obstacle detection code as well so that it would be the arduino's priority at first but would be disabled once the buggy had started along the track.



We choose not to extend our code for the pixy to meet this new challenge in order to focus our efforts on the infrared sensors in analog mode. We decided that if the analog input signals were correctly mapped to speed updates for each motor, we would not need to detect areas where the buggy's speed should be faster or slower and instead we could simplify the overall logic of the system and focus on the reliability of the central line-detection functionality.

System Model

The following diagram demonstrates the general software/hardware interface of our Gold Challenge code and hardware configuration:



The outline of our code running on the Arduino is as follows (unused functionality from previous challenges omitted):

```
setup()
        set input and output pins
        set individual motor speeds to 0
        set STARTED to false
        set "previous" infrared readings to 32
        set left and right speed reductions to 0
        set FULL to 200
loop()
        if !STARTED:
        send obstacle detection pulse
        calculate distance
        if distance > 10:
        set STARTED to true
        set motor speed to FULL
        else:
        get left and right infrared readings
        // map readings to speed reductions:
        right_reduce = right_reading*250/100
        left_reduce = left_reading*250/100
        if left_reading > 38:
        if left_reading > left_previous:
// in this case, the distance from center is
// getting worse
                left_reduce += 50
        left_power = min(FULL - left_reduce, 0);
```

```
else:
left_power = FULL
if right_reading > 38:
if right_reading > right_previous:
        right_reduce += 50
right_power = min(FULL - right_reduce, 0);
else:
right_power = FULL
set previous reading variables to current readings
write left_power to left analog control pin
write right_power to right analog control pin
```

For the Gold challenge we did not continue to use the processing program that we had written to run on a supervising computer. Therefore, to focus our efforts on the Gold challenge, we did not continue to update the supporting communications and reporting code that we implemented for previous challenges. While some of this code still exists in our Gold software package, it was not updated beyond the Silver challenge.

Appendix

Processing code

```
import processing.serial.*;

Serial PORT;
//index of expected serial interface
int EXPECTED_PORT = 5;
int BAUD_RATE = 9600;
String SERIAL_CONFIG = "ATID 3200, CH C, CN";
String[] SERIAL_QUEUE;
boolean SERIAL_AVAILABLE;
boolean SERIAL_CONNECTED;

//GUI color presets
color BG_COLOR = color(18, 18, 18);
color SUB_WINDOW_COLOR = color(245, 220,80);
color BUTTON_COLOR = color(51, 105, 64);
color BUTTON_CLICKED_COLOR = color(239, 42, 44);
color TEXT_COLOR = color(0);

int TITLE_SIZE = 20;

//GUI arrangement params
int WINDOW_WIDTH = 800;
int WINDOW_HEIGHT = 600;
int SUB_WINDOW_SPACING = 20;

int MAP_WIDTH = 549;
int MAP_HEIGHT = 400;

//GUI images
PImage TRACK_IMAGE;
PImage GARY_IMAGE;
PImage BUGGY_LARGE;

//standard button sizes
int BUTTON_WIDTH = 122;
int BUTTON_HEIGHT = 60;

//toggled buttons
boolean START_BUTTON_CLICKED = false;
boolean STOP_BUTTON_CLICKED = true;
boolean SPEED_PRESET_ONE_CLICKED = false;
boolean SPEED_PRESET_TWO_CLICKED = false;

//button sizes and spacing
int[] STOP_BUTTON = {SUB_WINDOW_SPACING,
```

```
                 (2 * SUB_WINDOW_SPACING) + MAP_HEIGHT,
                 BUTTON_WIDTH,
                 BUTTON_HEIGHT};
int[] START_BUTTON =  {(2 * SUB_WINDOW_SPACING) + BUTTON_WIDTH,
                 (2 * SUB_WINDOW_SPACING) + MAP_HEIGHT,
                 BUTTON_WIDTH,
                 BUTTON_HEIGHT};
int[] SPEED_PRESET_ONE =  {(3 * SUB_WINDOW_SPACING) + (2 *
BUTTON_WIDTH),
                    (2 * SUB_WINDOW_SPACING) + MAP_HEIGHT,
                    BUTTON_WIDTH,
                    BUTTON_HEIGHT};
int[] SPEED_PRESET_TWO =  {(4 * SUB_WINDOW_SPACING) + (3 *
BUTTON_WIDTH),
                    (2 * SUB_WINDOW_SPACING) + MAP_HEIGHT,
                    BUTTON_WIDTH,
                    BUTTON_HEIGHT};

//x, y, angle at which to draw buggy
//locations at which to draw the graphical progress tracker
float[] GANTRY_ONE = {141, 55, radians(90)};
float[] GANTRY_TWO = {360, 48, radians(100)};
float[] GANTRY_THREE = {395, 103, radians(175)};
float[] GANTRY_FOUR = {400, 150, radians(180)};
float[] GANTRY_FIVE = {356, 230, radians(260)};
float[] GANTRY_SIX = {151, 233, radians(270)};
float[] GANTRY_SEVEN = {47, 203, radians(350)};

int CURRENT_REPORTED_GANTRY = 0;

ArrayList<float[]> GANTRY_PATH = new ArrayList<float[]>();

//console messages to draw to the screen
ArrayList<String> CONSOLE_MESSAGES = new ArrayList<String>();
int CONSOLE_MAX_MESSAGES = 17;

void setup() {
  //setup screen
  size(1160, 600);
  background(0);
  //write to console
  add_console_message("> Starting...");

  //load images
  BUGGY_LARGE = loadImage("Buggy.png");
  TRACK_IMAGE = loadImage("TCD_Track_rev4.png");
  GARY_IMAGE = loadImage("gary_shadows_small.png");

  //attempt to connect to expected serial port
  try {
```

```
    PORT  = new Serial(this, Serial.list()[EXPECTED_PORT], BAUD_RATE);
    PORT.write("+++");
    delay(1100);
    //xBee setup

    PORT.write(SERIAL_CONFIG);
    delay(1100);
    PORT.bufferUntil( 10 );
    SERIAL_CONNECTED = true;
    add_console_message("> Using Serial config: " + SERIAL_CONFIG);
    add_console_message("> Serial connected on " + Serial.list()[EXPECTED_PORT]);

  } catch (Exception e) {

    //problem connecting, display options
    e.printStackTrace();
    String errorMsg = "> ERROR: problem connecting serial port with index " +
EXPECTED_PORT;
    SERIAL_CONNECTED = false;
    add_console_message(errorMsg);
    add_console_message("> Serial config: " + SERIAL_CONFIG + ".");
    add_console_message("> Possible interfaces printed to std out.");
    println(Serial.list());
  }
  //add buggy locations to list
  GANTRY_PATH.add(GANTRY_ONE);
  GANTRY_PATH.add(GANTRY_TWO);
  GANTRY_PATH.add(GANTRY_THREE);
  GANTRY_PATH.add(GANTRY_FOUR);
  GANTRY_PATH.add(GANTRY_FIVE);
  GANTRY_PATH.add(GANTRY_SIX);
  GANTRY_PATH.add(GANTRY_SEVEN);

}

void draw() {

  //draw graphical components
   draw_components();

   //draw buggy image
   image(BUGGY_LARGE, 890, 280, BUGGY_LARGE.width/2,
BUGGY_LARGE.height/2);
   fill(255);
   rect(0,520, 950, 20);
   fill(255);
   text("GARY'S AMAZING ADVENTURES ", 20, 580);
}

void draw_components() {
```

```
  //draw window components
  //draw maps
  draw_map();
  //draw track image
  image(TRACK_IMAGE, SUB_WINDOW_SPACING + 20, SUB_WINDOW_SPACING +
30, width/2.2, height/2.2);
  draw_buggy_on_map(GARY_IMAGE,
GANTRY_PATH.get(CURRENT_REPORTED_GANTRY));

  //draw console and buttons
  draw_graphical_console();
  draw_button(START_BUTTON, "50%", START_BUTTON_CLICKED);
  draw_button(STOP_BUTTON, "0%", STOP_BUTTON_CLICKED);
  draw_button(SPEED_PRESET_ONE, "78%", SPEED_PRESET_ONE_CLICKED);
  draw_button(SPEED_PRESET_TWO, "100%", SPEED_PRESET_TWO_CLICKED);

  //if a new serial message is available, check if graphical progress needs to be updated, write
to console
  if (SERIAL_AVAILABLE) {
      if (SERIAL_QUEUE[0].equals("location_event")){
        CURRENT_REPORTED_GANTRY = (CURRENT_REPORTED_GANTRY + 1) %
GANTRY_PATH.size();
      }
      SERIAL_AVAILABLE = false;
  }
}

void draw_map() {
  //draw the map window and text
  draw_window(SUB_WINDOW_SPACING,
            SUB_WINDOW_SPACING,
            MAP_WIDTH,
            MAP_HEIGHT,
            SUB_WINDOW_COLOR);
  draw_text("Reported Location",
          SUB_WINDOW_SPACING + 5,
          SUB_WINDOW_SPACING + 20,
          TITLE_SIZE,
          TEXT_COLOR);
}

void draw_buggy_on_map(PImage buggy, float[] loc_angle) {
  //draw the buggy image at the specified angle and location in progress tracker
  draw_component_at_angle(buggy, loc_angle[2], (int)loc_angle[0], (int)loc_angle[1]);
}

void draw_button(int[] button, String label, boolean toggle) {
  //draw a button based on preset
  if (toggle) {
    draw_window(button[0],button[1],button[2],button[3],BUTTON_CLICKED_COLOR);
```

```
  } else {
    draw_window(button[0],button[1],button[2],button[3],BUTTON_COLOR);
    noStroke();
  }
  draw_text(label, button[0] + 32, button[1] + 37, 25, TEXT_COLOR);
}

void add_console_message(String msg) {
  //add a message to the console and remove the oldest to maintain size (based on window
size)
  int console_length = CONSOLE_MESSAGES.size();
  if (console_length >= CONSOLE_MAX_MESSAGES) {
     int to_remove = console_length - CONSOLE_MAX_MESSAGES - 1;
     for (int i = 0 ; i < to_remove; i++) {
       CONSOLE_MESSAGES.remove(0);
     }
  }
  //add new message
  CONSOLE_MESSAGES.add(msg);
}

void draw_graphical_console() {
    //draw the graphical console from the message list
    String current;
    int top = SUB_WINDOW_SPACING;
    //draw console window
    draw_window((2*SUB_WINDOW_SPACING) + MAP_WIDTH,
                SUB_WINDOW_SPACING,
                MAP_WIDTH,
                MAP_HEIGHT,
                SUB_WINDOW_COLOR);

    //write messages
    for (int i = 0; i < CONSOLE_MESSAGES.size(); i++) {
        current = CONSOLE_MESSAGES.get(i);
        draw_text(current,
            (2*SUB_WINDOW_SPACING) + MAP_WIDTH + 5,
            (top * i)  + 40,
            12,
            TEXT_COLOR);
    }
}

void draw_window(int x, int y, int w, int h, color c) {
  //draw a window based on size and color
  fill(c);  // Use color variable 'c' as fill color
  rect(x, y, w, h);
}

void draw_text(String text, int x, int y, int size, color c) {
```

```
  //draw text at a specified location
  fill(c);
  textSize(size);
  text(text, x, y);
}

void send_instruction(String instr) {
  //send an instruction over the serial interface if connected
  if (SERIAL_CONNECTED) {
    PORT.write(instr);
  } else {
    add_console_message("> WARNING: serial interface disconnected.");
  }
}

void serialEvent(Serial portEvent) {
  //if serial event, set condition variable and parse message
   String newstring = trim(portEvent.readString());
   try {
     SERIAL_QUEUE = split(newstring, " ");
     SERIAL_AVAILABLE = true;
     if (!newstring.equals("")) {
       //if not empty, add to console
       add_console_message("> " + newstring);
     }
   } catch (Exception e) {
     e.printStackTrace();
     add_console_message("> ERROR: problem splitting serial string input");
   }
}

void draw_component_at_angle(PImage img, float angle, int x, int y) {
  //prevent translation/rotation from being permanent
  //draw an image at an angle
  pushMatrix();
      //push current window translation
      //update to draw at angle
      translate(x,y);
      translate(img.width/2, img.height/2);
      rotate(angle);
      image(img, -img.width/2, -img.height/2);
  //revert
  popMatrix();
}

boolean button_clicked(int[] button, int mx, int my) {
  //check if a mouse press is within a button location
  return (mx > button[0] && mx < button[0] + button[2]) && (my > button[1] && my <
button[1] + button[3]);
}
```

```
void mouseClicked() {
  //check if a mouse press is within one of the button presets
  //check if mouseX and mouseY in start button or end button
  if (button_clicked(START_BUTTON, mouseX, mouseY) &&
!START_BUTTON_CLICKED) {
    START_BUTTON_CLICKED = true;
    STOP_BUTTON_CLICKED = false;
    SPEED_PRESET_TWO_CLICKED = false;
    SPEED_PRESET_ONE_CLICKED = false;
    //send instruction
    send_instruction("s\n");
  }
  else if (button_clicked(STOP_BUTTON, mouseX, mouseY) &&
!STOP_BUTTON_CLICKED) {
    START_BUTTON_CLICKED = false;
    SPEED_PRESET_ONE_CLICKED = false;
    SPEED_PRESET_TWO_CLICKED = false;
    STOP_BUTTON_CLICKED = true;
    //send instruction
    send_instruction("x\n");
  }
  else if (button_clicked(SPEED_PRESET_ONE, mouseX, mouseY) &&
!SPEED_PRESET_ONE_CLICKED) {
    START_BUTTON_CLICKED = false;
    STOP_BUTTON_CLICKED = false;
    SPEED_PRESET_TWO_CLICKED = false;
    SPEED_PRESET_ONE_CLICKED = true;
    //send instruction
    send_instruction("m\n");
  }
  else if (button_clicked(SPEED_PRESET_TWO, mouseX, mouseY) &&
!SPEED_PRESET_TWO_CLICKED) {
    START_BUTTON_CLICKED = false;
    STOP_BUTTON_CLICKED = false;
    SPEED_PRESET_ONE_CLICKED = false;
    SPEED_PRESET_TWO_CLICKED = true;
    //send-instruction
    send_instruction("f\n");
  }
}
```

Arduino code

Silver Challenge Code

```
#include <SPI.h>
#include <Pixy.h>
#define BAUD_RATE 9600
#define IR_PIN 2
#define MOTOR_CTRL 3
#define MOTOR_MINUS_LEFT 4
#define MOTOR_MINUS_RIGHT 7
#define OVERRIDE_LEFT 5
#define OVERRIDE_RIGHT 6
#define TRIGGER_PIN 9
#define ECHO_PIN 8

String SERIAL_STRING = "";
bool SERIAL_AVAILABLE = false;

//IR setup
volatile boolean irInterrupt = false;
volatile boolean gantry1 = false;
volatile boolean gantry2 = false;
volatile boolean gantry3 = false;
volatile boolean report = false;
unsigned long currTime = 0;
unsigned long prevTime = 0;

int CURRENT_SPEED;
int UPDATE_SPEED;
int width;

//variables for pixy cam code
boolean YELLOW_DETECTED = false;
boolean BLUE_DETECTED = false;
boolean GREEN_DETECTED = false;

Pixy pixy; //main pixy object

void setup() {
  //setup serial
  Serial.begin(BAUD_RATE);
  Serial.print("+++");
  delay(1100);
  Serial.println("ATID 3200, CH C, CN");  //PanID set on Buggy
  delay(1100);

  pixy.init(); //initiating the pixy cam
```

```
  SERIAL_STRING.reserve(200);
  CURRENT_SPEED = 0;
  UPDATE_SPEED = 0;

  //consume OK
  while(Serial.read() != -1) {};

  pinMode(MOTOR_CTRL,OUTPUT);
  pinMode(OVERRIDE_LEFT, OUTPUT);
  pinMode(OVERRIDE_RIGHT, OUTPUT);
  pinMode(MOTOR_MINUS_RIGHT, INPUT);
  pinMode(MOTOR_MINUS_LEFT, INPUT);
  pinMode(TRIGGER_PIN, OUTPUT); //Trigger pin will have pulses output
  pinMode(ECHO_PIN, INPUT); //Echo pin is input to get pulse width
  pinMode(IR_PIN, INPUT); //Interrupt pin for the infrared sensor

  //Sets control pins to low, to keep stationary until signal is recieved
  digitalWrite(OVERRIDE_LEFT, LOW);
  digitalWrite(OVERRIDE_RIGHT, LOW);
  digitalWrite(MOTOR_MINUS_RIGHT, LOW);
  digitalWrite(MOTOR_MINUS_LEFT, LOW);
  digitalWrite(TRIGGER_PIN, LOW);
  motors_granular_speed(CURRENT_SPEED); //sets motors to 0

  //IR detection setup
  attachInterrupt(digitalPinToInterrupt(IR_PIN), IR_ISR, RISING);

}

void loop() {

  int dur, dist;
  //send out ultrasonic pulse to detect any obstacles
  digitalWrite(TRIGGER_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIGGER_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN, LOW);
  dur = pulseIn(ECHO_PIN, HIGH);
  //calculate the distance to the obstacle
  dist = dur/58;

  if (dist < 15 && dist > 0) {
    //obstacle visible
    motors_granular_speed(0);
    if (CURRENT_SPEED != 0) {
      //report motor speed percentage to supervising PC
      report_speed_precentage(0);
      //report the distance to the supervising PC
      Serial.print(dist);
```

```
    Serial.println(" cm");
  }
  CURRENT_SPEED = 0;
}

else if (CURRENT_SPEED == 0) {
  //buggy stopped but no obstacle visible
  CURRENT_SPEED = UPDATE_SPEED;
  //set the motor speed back up again
  motors_granular_speed(CURRENT_SPEED);
  if (UPDATE_SPEED != 0) {
    Serial.println("Resuming...");
    //report motor speed percentage to supervising PC
    report_speed_precentage(CURRENT_SPEED);
  }
}

else if (CURRENT_SPEED != UPDATE_SPEED) {
  //current speed out of date
  //set motors to correct speed
  motors_granular_speed(UPDATE_SPEED);
  //report motor speed percentage to supervising PC
  report_speed_precentage(UPDATE_SPEED);
  CURRENT_SPEED = UPDATE_SPEED;
  }

//Pixy settings
static int i = 0;
uint16_t blocks;
char buf[32];
// grab blocks!
blocks = pixy.getBlocks();

//Pixy detecting signature blocks
if (blocks > 0) {
    int max_index = 0;
    int max_bounding_size = 0;
    //loop through all the blocks detected and select the biggest one
    for (int b = 0;b<blocks;b++) {
      if ((pixy.blocks[b].width * pixy.blocks[b].height) > max_bounding_size) {
        max_bounding_size = (pixy.blocks[b].width * pixy.blocks[b].height);
        max_index = b;
      }
    }
    //buggy will only respond to a block if it is atleast a minimum size
    int area_threshold = 2400;
    //when yellow is detected it performs better if it waits a little before turning right
    int respond_delay = 500;
    //when blue is detected set the motors to a slower speed
    int slow_speed = 120;
```

```
    int sig = pixy.blocks[max_index].signature;

    if (sig == 1 && (max_bounding_size >= area_threshold) && !YELLOW_DETECTED)
{
      //fork in the road: turn right by turning on the left override
      delay(respond_delay-425);
      digitalWrite(OVERRIDE_LEFT, HIGH);
      delay(2000);
      digitalWrite(OVERRIDE_LEFT, LOW);
      digitalWrite(OVERRIDE_RIGHT, LOW);
      //report the yellow signal to the supervising PC
      Serial.println("Yellow signal");
      //move the buggy image to the correct location on the PC map
      Serial.println("location_event");
      //boolean variables are used so that the buggy doesn't respond more than once to the
      //same signal
      YELLOW_DETECTED = true;
      BLUE_DETECTED = false;
      GREEN_DETECTED = false;
    }

    else if (sig == 2 && (max_bounding_size >= area_threshold) && !BLUE_DETECTED){
      delay(respond_delay);
      //blue signal means go slow
      motors_granular_speed(slow_speed);
      //report the blue signal to the supervising PC
      Serial.println("Blue signal");
      //move the buggy image to the correct location on the PC map
      Serial.println("location_event");
      //boolean variables are used so that the buggy doesn't respond more than once to the
      //same signal
      YELLOW_DETECTED = false;
      BLUE_DETECTED = true;
      GREEN_DETECTED = false;
    }

    else if (sig == 3 && (max_bounding_size >= area_threshold) &&
!GREEN_DETECTED){
      delay(respond_delay+1000);
      //green signal means return back to the faster speed
      motors_granular_speed(CURRENT_SPEED);
      //report the green signal to the supervising PC
      Serial.println("Green signal");
      //move the buggy image to the correct location on the PC map
      Serial.println("location_event");
      //boolean variables are used so that the buggy doesn't respond more than once to the
      //same signal
      YELLOW_DETECTED = false;
      BLUE_DETECTED = false;
      GREEN_DETECTED = true;
```

```
    }
  }

  if (SERIAL_AVAILABLE) {
    //if a signal is received from the supervising PC, determine what it is
    SERIAL_AVAILABLE = false;
    SERIAL_STRING.trim();
    //call on one of our functions to read the string
    interpret_master_instr(SERIAL_STRING);
  }

  //IR interrupt check
  if (irInterrupt == true) {
    //read the pulse
    int pulse = pulseIn(IR_PIN,LOW);

    if (pulse >= 1000 && pulse <= 1100) {
      if (gantry1 == false) {
        currTime = millis();
        //pulse is faulty if the last time a gantry was detected was less than 2 seconds ago
        if (currTime - prevTime >= 2000) {
          report = true;
        }
        if (report == true) {
          prevTime = millis();
          report = false;
          //boolean variables are used to stop the buggy from reporting the same gantry multiple
          //times
          gantry1 = true;
          gantry2 = false;
          gantry3 = false;
          //report gantry 1 to supervising PC
          Serial.println("Gantry 1");
          //move buggy image to the correct location on the PC map
          Serial.println("location_event");
        }
      }
    }

    else if (pulse >= 2000 && pulse <= 2100) {
      if (gantry2 == false) {
        currTime = millis();
        //pulse is faulty if the last time a gantry was detected was less than 2 seconds ago
        if (currTime - prevTime >= 2000) {
          report = true;
        }
        if (report == true) {
          prevTime = millis();
          report = false;
```

```
        //boolean variables are used to stop the buggy from reporting the same gantry
multiple
        //times
        gantry1 = false;
        gantry2 = true;
        gantry3 = false;
        //report gantry 2 to supervising PC
        Serial.println("Gantry 2");
        //move buggy image to the correct location on the PC map
        Serial.println("location_event");
      }
    }
  }

  else if (pulse >= 2900 && pulse <= 3000) {
    if (gantry3 == false) {
      currTime = millis();
      //pulse is faulty if the last time a gantry was detected was less than 2 seconds ago
      if (currTime - prevTime >= 2000) {
        report = true;
      }
      if (report == true) {
        // report location
        prevTime = millis();
        report = false;
        //boolean variables are used to stop the buggy from reporting the same gantry multiple
times
        gantry1 = false;
        gantry2 = false;
        gantry3 = true;
        //report gantry 3 to supervising PC
        Serial.println("Gantry 3");
        //move buggy image to the correct location on the PC map
        Serial.println("location_event");
      }
    }
  }
 //set the interrupt variable back to false
 irInterrupt = false;
 }

}

//use this function to report strings back to the supervising PC
void send_data(String data) {
 Serial.println(data);
}

//use this fucntion to report speed percentage back to supervising PC
void report_speed_precentage(int val) {
```

```
  int percent = (val / 255.0) * 100;
  //change the calculated integer to a string
  String out = String(percent);
  send_data("Motor power: " + out + "%");
}

//use this function to interpret string commands sent from the supervising PC
void interpret_master_instr(String serialInput) {
  //this is how we control the motor speed from the supervising PC
  if (serialInput.equals("x")) {
    UPDATE_SPEED = 0;
  }
  else if (serialInput.equals("s")) {
    UPDATE_SPEED = 127;
  }
  else if (serialInput.equals("m")) {
    UPDATE_SPEED = 150;
  }
  else if (serialInput.equals("f")) {
    UPDATE_SPEED = 255;
  }
}

//use this function to control the motor speed from within the loop function
void motors_granular_speed(int val) {
  analogWrite(MOTOR_CTRL,val);
}

//this function triggers an if statement in the loop and is the initial step in interpreting string
//commands sent from the supervising PC
void serialEvent() {
  SERIAL_STRING = Serial.readStringUntil("\n");
  SERIAL_AVAILABLE = true;
}

//this is our interrupt function for the infrared sensor
void IR_ISR() {
  irInterrupt = true;
}
```

Changelog of development

The following is a general changelog of our buggy development:

- Building logic truth tables and simplifying with DeMorgan's laws and Karnaugh maps
- Initial Board prototyping in Eagle and Multisim
- Board fabrication
- Board verification
- Initial processing code added to simplify UI workflow and standardize graphical elements
- Test xbee connectivity with chat program
- Modify xbee program to send vehicle control messages
- Processing console added to display messages
- Add code to arduino for obstacle detection
- Add code to arduino for speed changes
- Add code to arduino for pixy
- Add code to arduino for gantry detection
- Perform testing for ideal vehicle speed given the standard Silver/Bronze track
- Add speed overrides for each of the pixy color conditions
- Testing for silver/bronze
- Initial planning for Gold challenge
- Remove logic board and change h-bridge to respond to arduino
- Change infrared sensors to analog mode
- Update obstacle detection code to only run at beginning and stop once vehicle has started.
- Map analog infrared signals to individual motor speed reductions
- Testing to determine ideal speed and motor speed reduction values
- Add snail eyes and snail shell for speed and style
- Successfully compete in gold challenge
