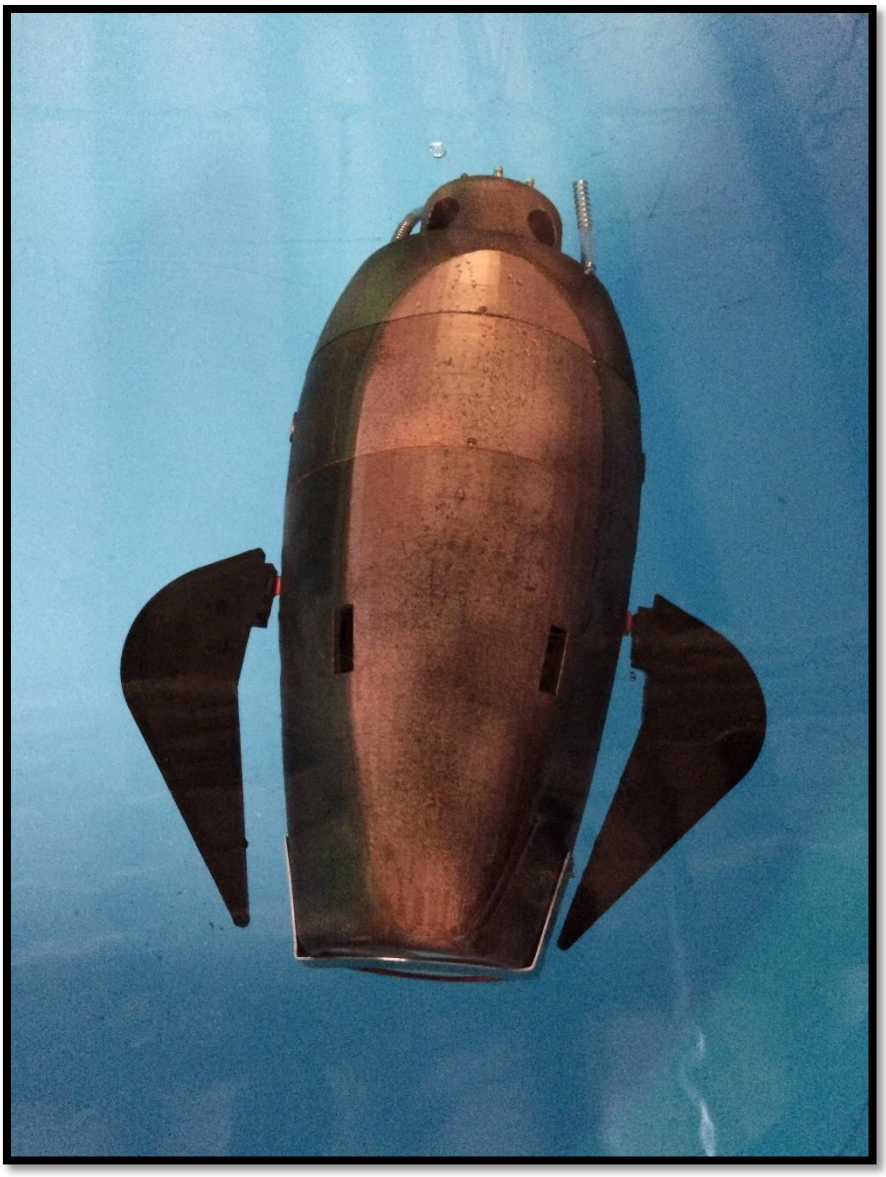

Fundamentals of Robotics; Spiral 2

December 15, 2017

1 Execution

When we set out on this adventure, we decided to create an approximately 1.5' long squid able to make a triangle in the pool by tracking LED buoys. It was to have 3 steering systems and long tentacles to add to the aesthetic. The squid was to be bio-inspired, the very reason we chose 3 steering systems. Because squids use jet propulsion with turning tubes to change direction, we wanted to have controllable tubes that could not only be pulled one direction or the other to turn but could also have a valve turned on or off so as to make turning easier by turning off one flow of water. We also had fins on the front of the boat to turn, which were inspired off of fins that squids also use to assist their turning. I will go more in depth on the different steering (tubes, fins, and valves) systems later in the report. Our goal in this project was to create a steering system that was unique and challenging for us.

We ended up not succeeding to make all of the steering systems work. In fact, we only had about 1.5 steering systems working on demo day. On top of that, our boats propulsion was not strong enough to move the boat and because of the speed, the steering units that could move, couldn't turn our boat. We ended up coming in last in the race with 19 points. For the most part, our boat tried to go in a straight line but ended up veering off because of the half working steering system. We set out with a crazy idea of making this multi complex system robot and unfortunately did not succeed. However, we were very proud with what we were able to do, even if it wasn't much.

2 Mechanical

Figure 1 Full CAD of the squid robot

Since we were making fully enclosed submarines, we had to have access points to the interior of the robot so we could change batteries and access the pressure hull full of electronics. In order to change batteries we created a small hatch on the bottom of the boat that could come off with two screws. Inside of that hatch where two battery mounts which help secure the batteries very tightly. In addition to power, the batteries provided some ballast at the bottom of the ship.

Figure 2 Battery hatch on the robot

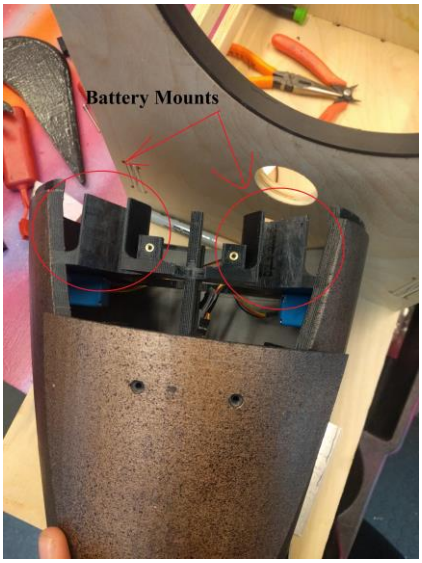

Figure 3 Battery mounts in the robot

In order to access the electronics, the designed our squid so that the pressure hull could pull out of the front of the boat. We made the hole in the front big enough to simply pull the pressure hull out. Unfortunately, we didn't plan our electronics set up accordingly. The Arduino we had to access the most in order to change code and read output, was on the opposite end of the tube. To get to that Arduino we had to open the squid at its halfway point in order to pull the entire pressure hull out. Since we made the front hole big enough to pull the pressure hull out of, we had to secure it in or else it would shoot out the front when we put the boat in the water. We ended up attaching a metal bar that folded over each edge and screwed to the sides.

Figure 4 Hole for the pressure tube. Bar over to keep the tube in the squid.

Since squids move through propulsion we decided to also use jet propulsion in order to move our boat. To do this we installed a water pump in the back of the boat. Originally this was going to suck water out of the back of the boat, then push water out the same direction. After printing the parts and thinking about this a little more, we realized that if we sucked water in from the back,

because of physics, we would most likely end up pulling the entire boat backwards. When we realized this, we created a cap for the hole in the back. We left smaller side holes that water could be sucked in from, but the problem of the big back hole had been solved.

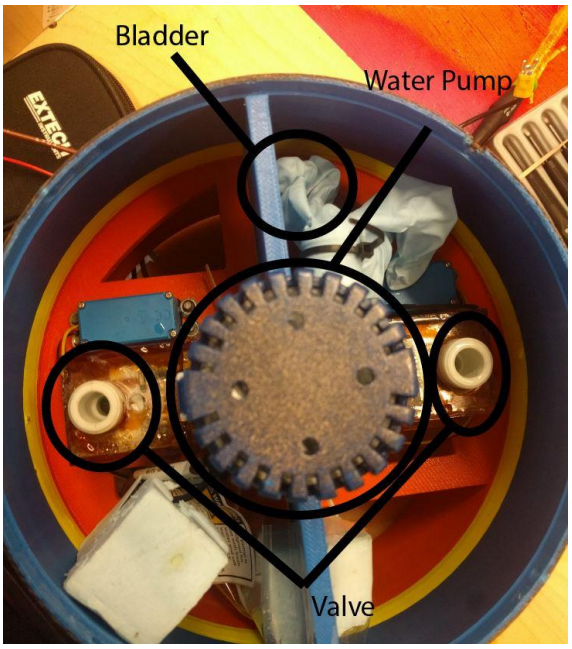

Figure 5 Where all the tubes go for water to move in order to propel and turn

As mentioned earlier, we planned to have three steering systems on our boat. We were going to have the pump send water into a bladder, in this case a rubber glove, and then split the bladder two directions into valves. These valves were connected to steering tubes that came out of the back of the boat. These tubes were designed to be connected by fishing line to a servo further up on either side of the boat. The plan was to have the ability to open and close the valves, so if we turned left, only the right valve would be open, forcing the boat to turn left, and if we were going straight we would have both valves open. We put the bladder between the pump and tubes with the hope that this would build up more pressure and help propel the boat forward. The tubes were to connect to the servos so we could pull the tubes to help turn the boat. We called this *direction tube drive*. The thought was that if we were turning left, for example, we could pull the right tube to the right and it would help to push the squid in the proper direction. We decided that these were complex steering systems so we wanted a backup system, just in case. For a back us system, we put two fins at the front of the boat. These fins were mounts to servos so they could spin down and create drag, helping to turn the boat. Again, if we wanted to turn left we could put the left fin down and it would help create drag to turn us that direction.

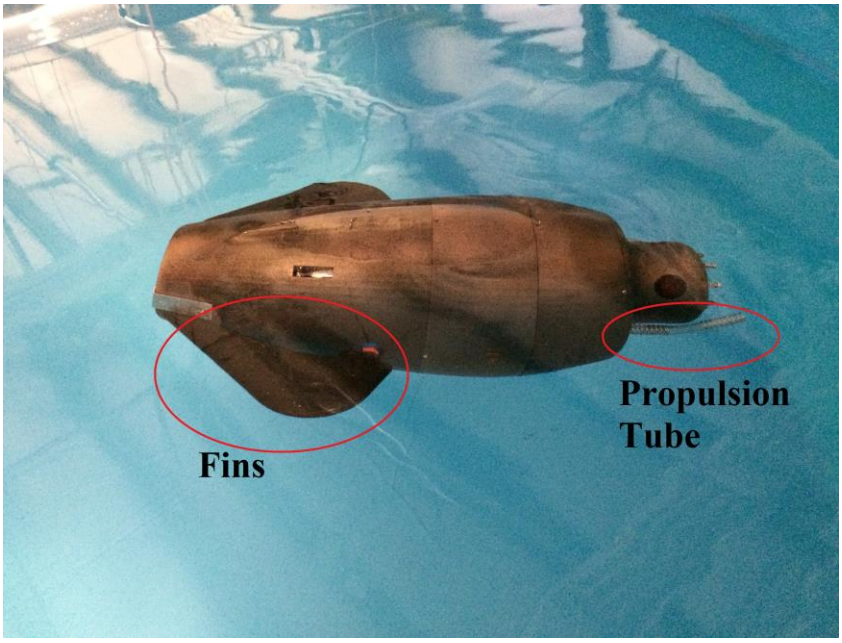

Figure 6 Full boat with "working" systems

Unfortunately, not all of our plans panned out. Because a lot of electrical pin losses through Arduino communication and things connected, like shields and cameras, we did not have enough pins for all of the systems. We ended up knocking out the directional tube driving because we didn't have space to power the servos, or time to try to make it work. The valves were tricky. Because of a lack of pins, we had one valve wired to the motor shield as a motor and the other connected to a relay. We luckily had space for this system; however, the two valves never both worked, which will be discussed later. The only system that truly worked was the fins. However, though the servos could move to a proper angle, the fins would never stay connected to the servo shaft, no matter how many times we tried to glue them on. Because of all of these failures we had about 1 – 1.5 working steering system, including the half working valves and the broken, but working fins.

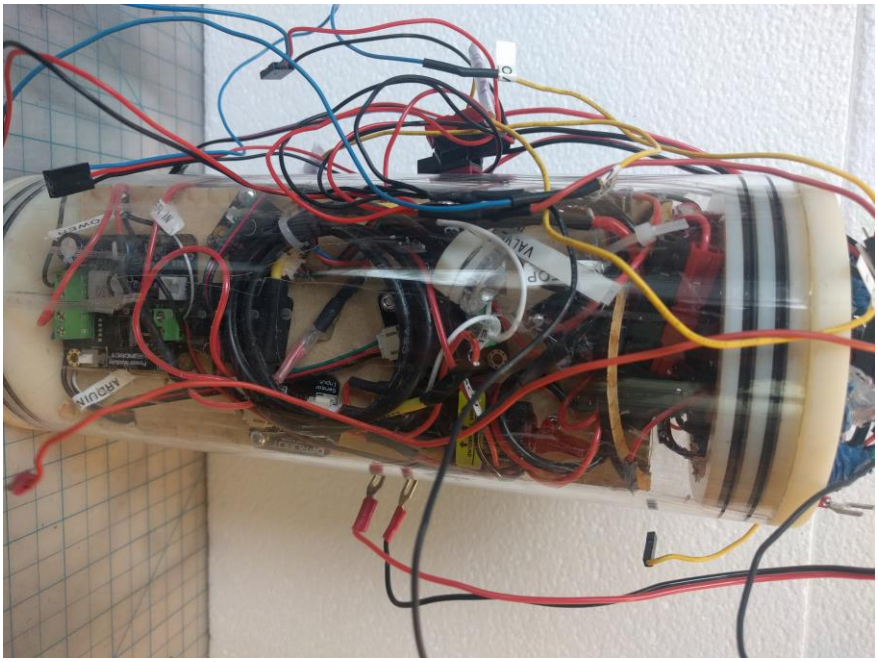

Figure 7 Full electronics tube

Since these boats were to be fully submersible, we had to fully waterproof the electronics. The electronics were put into a fully sealed pressure hull. The was a long clear tube with very tight end caps that we waterproofed along the edges because of the o rings used to snug the cap into the tube. However, since we had to power electronics outside of the tube, such as servos, and receiver power from the batteries that were not in the hull, we had to make wires to connect outside the hull. To do this, we screwed holes in the end cap to put bored-out bolts in to trail the wires through. After making the holes in the end cap we had to work very hard to waterproof it. We tried sealant and gasket sealant and a few other thigs but nothing worked. We began to give up hope. However, we then created an epoxy lake in the cap that, luckily, worked amazingly. The other endcap was a lot easier to seal. We had to drill a hole in the endcap and put an acrylic window in so that our camera could see out of the boat. Happily for us, we used sealant around the edges of that and it worked on the first try.

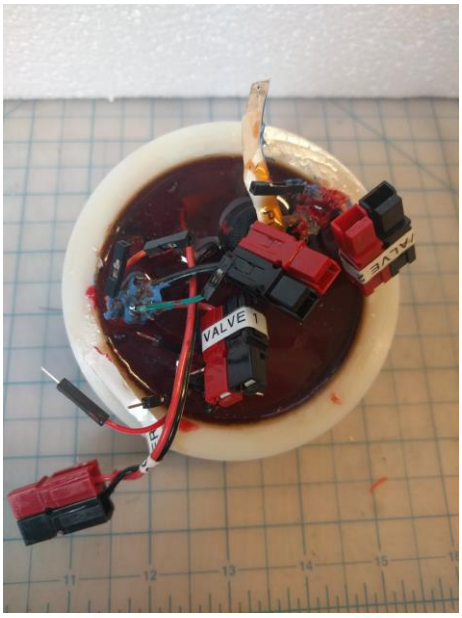

Figure 8 Endcap filled with epoxy

Since we were making such a cool aquatic creature, we wanted it to look really cool. We decided early on that we would attach tentacles and they would light up. Later we also decided that we wanted to create a steampunk squid. Painting was able to happen so the squid looked gold and steam punky, additionally some decorative parts, like portholes, were put on. We were also able to find tentacles and thread them with LEDS and put steam punk end caps on them. Our boat was looking great! Unfortunately, since our boat didn't work great, we never got a chance to water test with the light up tentacles.

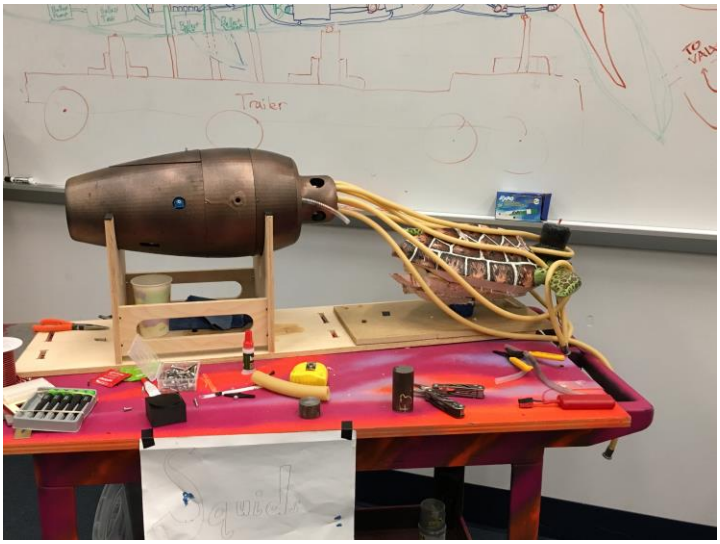

Figure 9 The full steampunk squid with tenticals!

3 Electrical

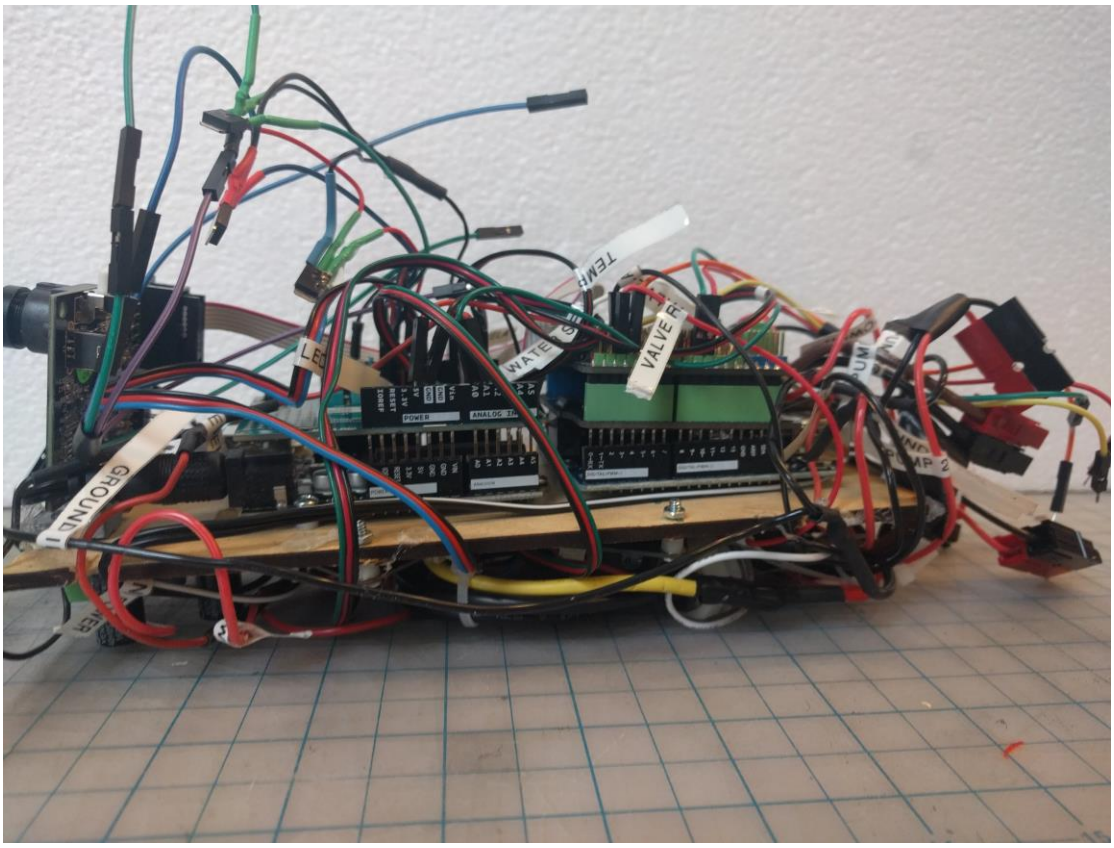

Figure 10 Full electronics system

The main components of the electrical system include: 2 Arduinos, a Pixy Cam, an X-Bee, the motors/servos, and the sensors.

The Pixy Cam was the sensing system for this boat. A Pixy Cam can be trained on certain colours. Once it is trained it is able to look at something of that colour and report back the colour, the size, which can tell the distance it is away from it, and the angle it is away from it. This was extremely useful as we were trying to sense buoys to drive to in order to traverse a triangle across the pool. Originally we had the Pixy Cam hooked up to what we called the “Sense” Arduino. The information collected - the arrays of colours, angles, and distances - were then sent to the “Think/Act” Arduino. After a lot of testing with a lot of Serial communication problem (I will talk about this more in depth later) we realized it would make more sense to hook the Pixy Cam directly to the Think/Act Arduino. This mean the information was directly on that Arduino with no need to transfer data.

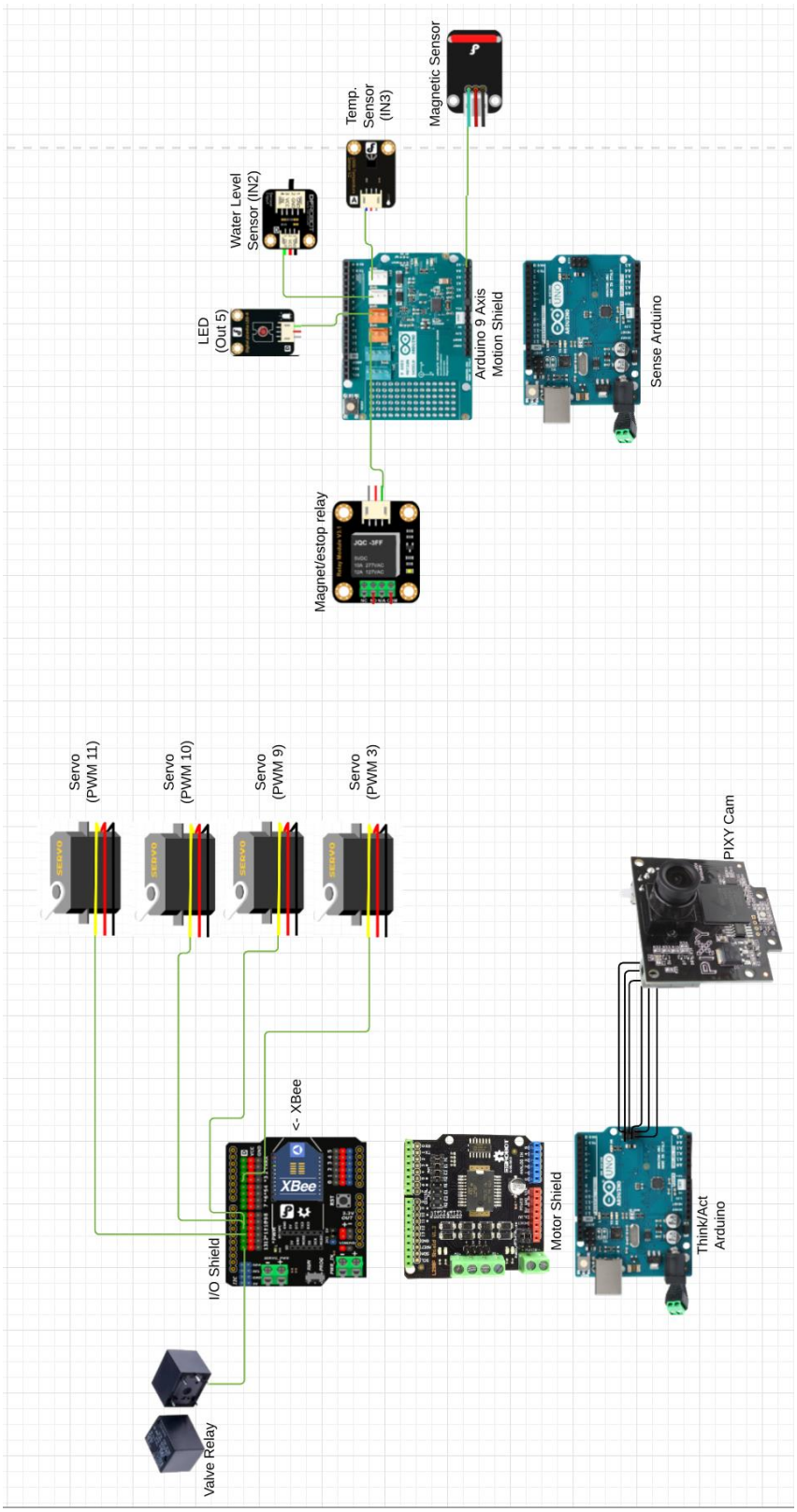

Figure 11 Data diagram

In order to get the power we needed, we hooked two batteries up in parallel so that even if one ran low, the robot would continue to get the power it needed. Even still, the robot seemed to run a little low on power, just because we had so much going on. When testing, we ran into a lot of problems with the power not being hooked up properly so the Arduinos were not behaving in the expected manner. However, by the end, we had all of the electronics, including power, wired up properly.

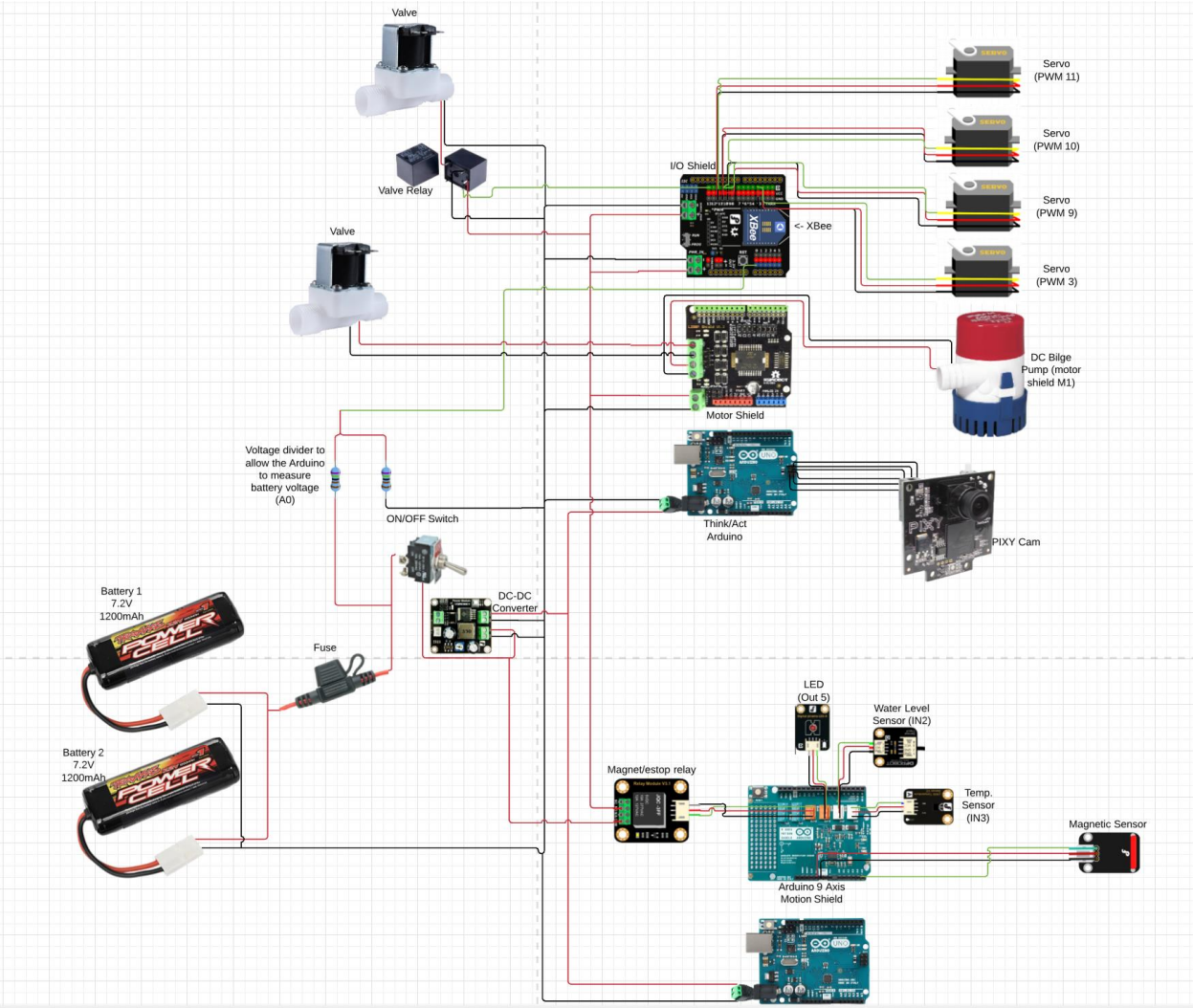

Figure 12 Power diagram

Communication was a big thing with this robot. We had to communicate between two Arduinos as well as wirelessly between the Think/Act Arduino and the computer using an X-Bee. We had a lot of trouble connecting the Arduinos through Serial (which will be talked about later) because of all of the pins that the different motors, servos and the Pixy Cam took up. Also, some things, like servos, didn't work with some of the Serial connections we tried to use. We ended up not communicating between the Arduinos. In the end we had the Pixy Cam connected to the Think/Act Arduino. We used the sense Arduino to hook up our e-stop magnetic sensor, our water flood sensor, and our temperature sensor. The magnetic sensor to determine e-stop was hooked to a relay that when triggered, would turn the power to the motors off. The hope was that if the pressure hull was taking on too much water, or if the temperature

got too high, or the e-stop magnet was pulled, the boat could e-stop. However, since we couldn't even get our boat running, we never fully integrated the code for this into the robot.

Figure 13 Magnet holder for the estop

In order to communicate with the robot we had to connect an X-Bee to the computer and one to an Arduino. The hope was that we could upload code and talk to the boat over X-Bee. Since we wanted to be able to upload code, we decided to hook the X-Bee up to the Think/Act Arduino as that had more variables- likely to change in the code. The X-Bee was hooked to the I/O shield, which has a spot made for it. The X-Bee is a wireless communication device that is commonly used to communicate between Arduinos and computers. The hope was that we would be able to upload code wirelessly; unfortunately, we could not figure out how to make that work. Luckily, we were able communicate through the X-Bee so we could have the Arduino send us messages and we could send messages back. This allowed us to send missions to the Arduino and allowed us to see the data the Pixy Cam was sending so we could debug and see if the robot actually saw the targets.

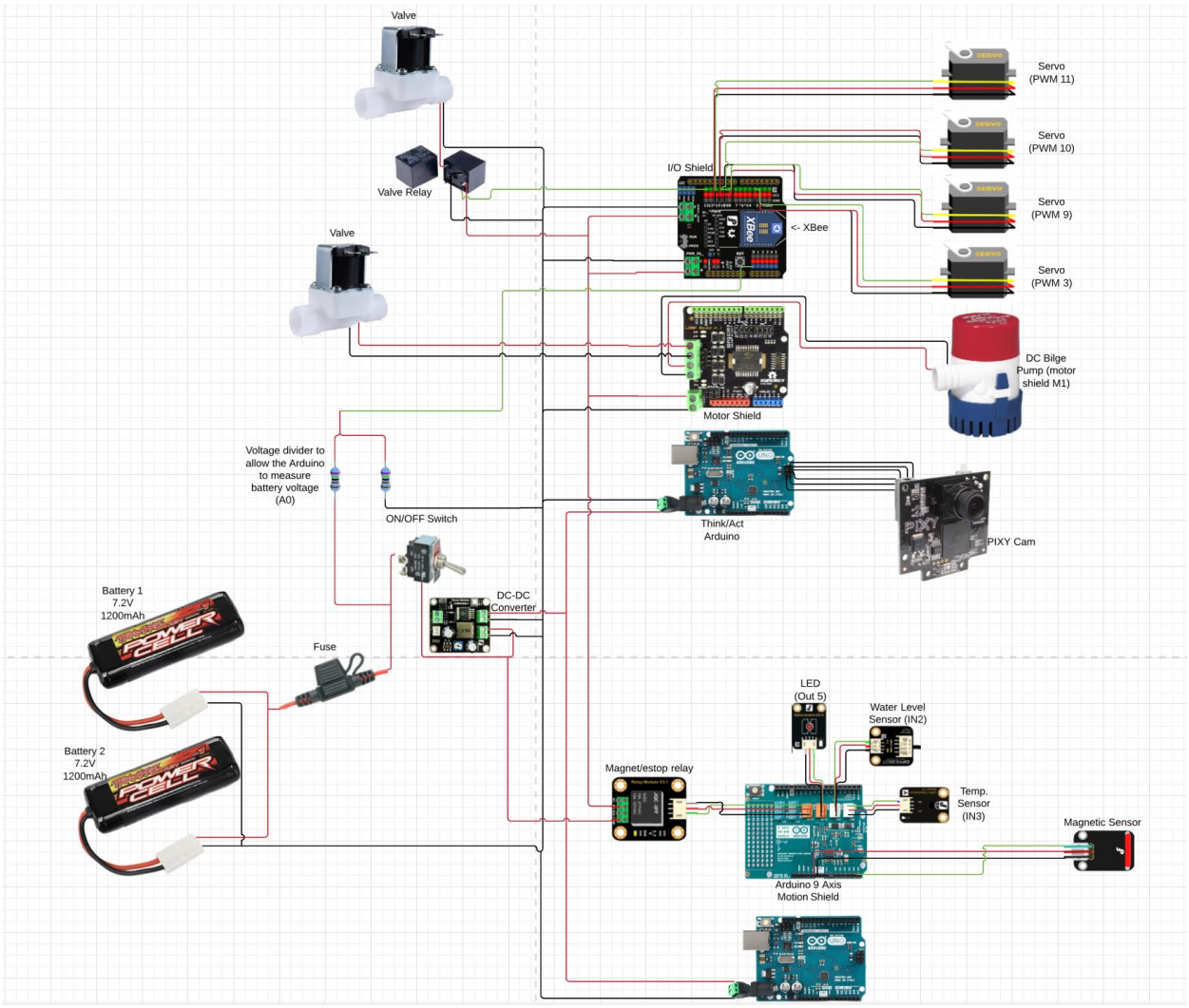

Figure 14 Full data and power diagram

4 Software

[See appendix for full code]

Since we had two Arduinos we had to have two scripts of Arduino code, one for Sensing and one from Thinking and Acting. These Arduinos had to communicate through serial in order to transfer data from one to the other. Software Serial can only transfer integers so we decided to use an Arduino library called "Easy Transfer." This library allows you to end almost any type of data between two Arduinos, even arrays. This transfer is set up by creating a send Structure and a receive Structure. Inside of the structures you create the variables you want to send between the Arduinos, making sure it is the same in the send and the receive on the two Arduinos. In the code you can call the structure. Variable and set it equal to something. When you are done assigning values you can then send the variable to the other Arduino. On the other Arduino you can receive those values and use structure.variable in order to use the variable.

```
[Some simple easy transfer code] struct SEND_DATA_STRUCTURE { //put your variable definitions here for the data you want to receive //THIS MUST BE EXACTLY THE SAME ON THE OTHER ARDUINO
    int16_t test;
    boolean mag;
};

void loop() { // put your main code here, to run repeatedly:
        for (int i = 0; i < 100; i++) {
            txdata.test = i;
            txdata.mag = digitalRead(A0);
            ETout.sendData(); //Serial.println(txdata.test);
            delay(1000);
        }
```

Unfortunately, we were unable to use Software Serial on the Sense Arduino because the Pixy Cam interfered with it. Instead, we sent data using Serial from the Sense Arduino, this was still able to work with Easy Transfer. The only reason this was a pain is because we could no longer see the data we were sending on the actual Serial Monitor. We then realized Software Serial also didn't work on the Think/Act Arduino. The Think/Act Arduino had servos plugged in as well as the X-Bee. Because of the X-Bee we were not able to use the regular servo library, this had something to do with the timers in the Servo and X-Bee library interfering in some way. Since the regular Servo library did not work we had to use a library called ServoTimer2. This allowed the servos to move properly again! However, this new servo library did not work with the Software Serial. Because of this we tried a new serial library, AltSoftSerial. Unfortunately, this type of serial did not work with the Easy Transfer library. At this point, we decided it was not worth trying to make the Arduinos communicate through Serial with all the things we had going on. We realized we had two choices, either move the send information through analog pins (ie. Send different voltage values to reference straight, right, left), try to use I2C transfer instead of Easy Transfer, or not have the Arduinos connect and instead have the Sense Arduino be in control of the sensors, including e-stop, and have the Pixy Cam connect to the Think/Act Arduino. We decided the easiest process to switch to would be to move the Pixy Cam and disconnect the Arduinos. Luckily, the code could remain almost the same, just located in a different place. The following is the code used to extract data from the Pixy Cam, we were able to use the same code in both the Sense Arduino and when we moved to the Think/Act Arduino.

```
1.  void readSenseArduino() {
2.          int n = pixy.getBlocks();
3.          for (int i = 0; i < MAX_BLOCKS; i++) {
4.              signatures[i] = 0;
5.          }
6.          for (int i = 0; i < min(n, MAX_BLOCKS); i++) {
7.              widths[i] = pixy.blocks[i].width;
8.              positions[i] = pixy.blocks[i].x;
9.              signatures[i] = pixy.blocks[i].signature;
10.         }
11.         distance = -1;
12.         angle = 0;
13.         for (int i = 0; i < MAX_BLOCKS; i++) {
14.             if (signatures[i] == mission[target] - 2) { // G,Y,R,H = 1,2,3,4
15.                 distance = CAMERA_RATIO * widths[i];
16.                 angle = positions[i] - 159; // 159 = center of screen
```

The Pixy Cam returns something called “blocks” which holds all of the data. In order to make this data easier to process, we split the blocks up into 3 parts, colour, size, and angle. By putting this into arrays we were able to easily manipulate and use them in the code. To see this break down, see the code above. In order to have the Pixy Cam recognize a colour you press the white button on the top right, seen in the picture below.

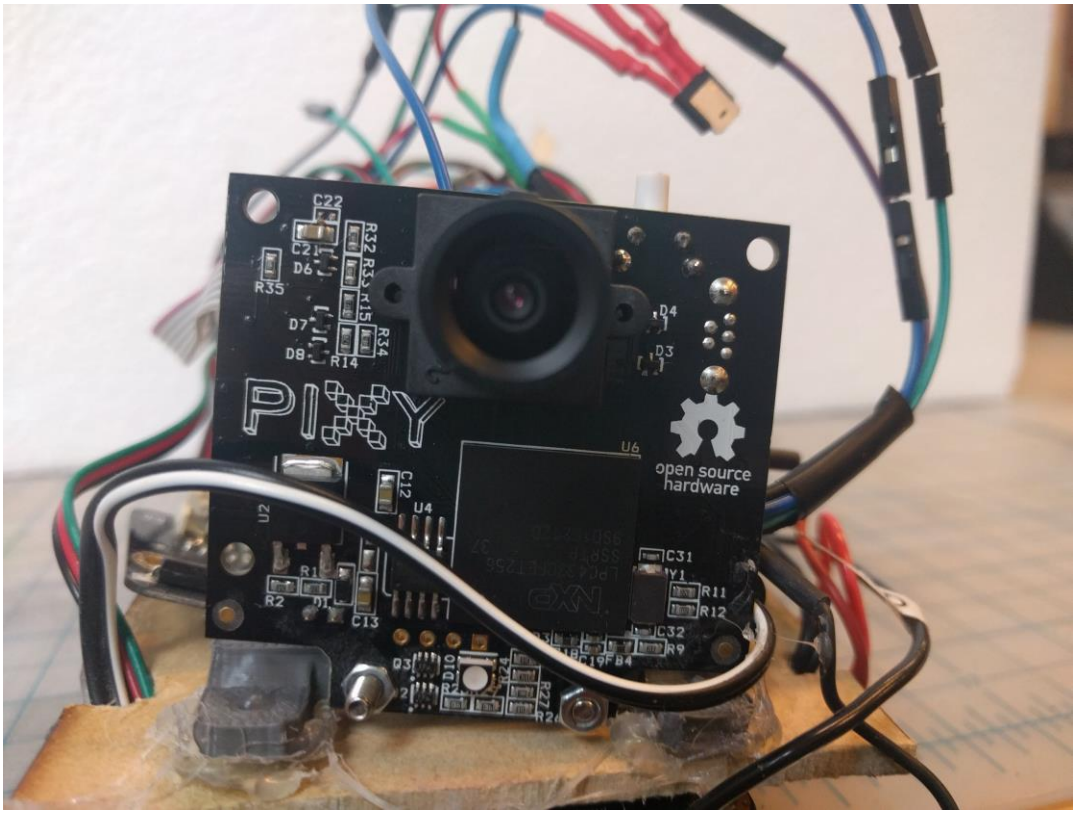

Figure 15 Pixy Cam with white calibration button on top right corner

Since we were to have so many steering capabilities, we had some complex, but nicely designed, code to determine what part needed to move and how much it needed to move. We had a Think function that looked at which mission was chosen and determined which direction the boat should drive, straight, right, or left. It also restarted the mission if we input the mission as a loop. This function set the variable "direction" to straight, right, left, or none.

```
void think() {
        if (mission[target] <= 2) { // Manual override
            direction = mission[target];
        } else if (mission[target] == DANCE) { // Dance code
            direction = DANCE;
        } else if (distance < 0) { // Target not visible
            direction = LEFT;
        } else if (distance > APPROACH_DIST) { // Reached target
            target++;
            if (mission[target] == LOOP) { // Restart mission
                target = 0;
            }
            direction = NONE;
        } else { // Target visible
            direction = STRAIGHT;
        }
```

} The direction variable was used in the act loop. In the act loop we had a switch statement that went through all the possibilities for the direction. In each case, the move function would be called. The move function required 2 arguments: velocity and angle. The move function took these arguments and determined what to do with each of the steering systems.

```
            switch (Serial.read()) {
                case '>':
                    return; // Debug message
                case '2':
                    mission[i] = STRAIGHT;
                    break;
                case '1':
                    mission[i] = LEFT;
                    break;
                case '3':
                    mission[i] = RIGHT;
                    break;
                case 'r':
                    mission[i] = RED;
                    break;
                case 'y':
                    mission[i] = YELLOW;
                    break;
                case 'g':
                    mission[i] = GREEN;
                    break;
                case 'h':
                    mission[i] = HOME;
                    break;
                case 'd':
                    mission[i] = DANCE;
                    break;
                case 'l':
                    mission[i] = LOOP;
```

```
                    break;
                default:
                    mission[i] = NONE;
            }
```

Originally we decided whether to turn the pump on or off in the move function. However, for demo we commented this out and just had the pump constantly on. We also had logic to determine how to turn the tubes for the differential tube steering. This was obviously unused since we didn't set this system up. The fins had another logic statement set to determine how to move each of the fins. The fin code worked great! Unfortunately, the fins kept falling off and they were not impactful enough with such slow movement to turn the boat.

```
        if (TURN_FINS && ang >= TURNING_ANGLE) { // Left //Serial.println("turn fins left");
            leftFin.write(SERVO_MIN_POSITION + FIN_TURN_ANGLE - 200);
            rightFin.write(SERVO_MAX_POSITION - FIN_FORWARD_ANGLE - 100);
        } else if (TURN_FINS && ang <= -TURNING_ANGLE) { // Right //Serial.println("turn fins right");
            leftFin.write(SERVO_MIN_POSITION + FIN_FORWARD_ANGLE - 200);
            rightFin.write(SERVO_MAX_POSITION - FIN_TURN_ANGLE - 100);
        } else { // Straight //Serial.println("turn fins straight");
            leftFin.write(SERVO_MIN_POSITION + FIN_FORWARD_ANGLE - 200);
            rightFin.write(SERVO_MAX_POSITION - FIN_FORWARD_ANGLE - 100);
        } // Set valve states
        if (TURN_VALVES && ang >= TURNING_ANGLE) { // Left //Serial.println("turn VALVE left");
```

We also had logic statements to determine which valves should be on or off. Since one valve was wired to the motor shield and the other to a relay, the control of the two pumps was different.

```
        if (TURN_VALVES && ang >= TURNING_ANGLE) { // Left //Serial.println("turn VALVE left");
            digitalWrite(VALVE1M, LOW);
            analogWrite(VALVE1E, 0);
            digitalWrite(VALVE2, HIGH);
        } else if (TURN_VALVES && ang <= -TURNING_ANGLE) { // Right //Serial.println("turn VALVE right");
            digitalWrite(VALVE1M, HIGH);
            analogWrite(VALVE1E, 255);
            digitalWrite(VALVE2, LOW);
        } else { // Straight //Serial.println("turn VALVE straight");
            digitalWrite(VALVE1M, HIGH);
            analogWrite(VALVE1E, 255);
            digitalWrite(VALVE2, HIGH);
        }
```

Unfortunately on demo day both of the valves wouldn't work. One valve seemed to always be on and the other would never turn on, causing us to be able to move in a very slow arc "across" the pool.

5 Demo Day

The night leading up to demo day and the morning of, you could find our team in the LPB working feverishly to get our squid to work. The night before is when we discovered most of our Arduino connection problems. Unfortunately, in the testing leading up to the night before, we did not uncover all of these problems. Before that night, we discovered that the Pixy Cam couldn't work with Software Serial so we would have to work with Serial to communicate instead. Unfortunately, all of the other Serial and Servo problems we find the night before.

Early on that night we decided to move away from Arduino communication. Unfortunately, the weeks leading up to demo day we worked to get the Pixy Cam working properly, the X-Bee communicating, and the Serial communication to work properly so we didn't get to test much with the actual movement systems. This cause the problems we had for demo.

That night, we realized that not only did our servos not work, something fixed as soon as we could look up what the problem was. The night before we realized we had another problem, the pump did not always turn on or off when it was supposed to. In order to fix this problem we turned the pump to constantly on

```
90. digitalWrite(PUMPM, HIGH);
91.         analogWrite(PUMPE, 255);
```

We then had many valve problems. Originally it appeared both of our valves worked, when tested with a simple code to test the valves turning on and off. For some reason, once we put the actual code on the Arduino the valves stopped working properly.

We tried many things to fix this. We tried to hook both valves up to the motor controller together so they would just be open, so we could propel ourselves in a straight line forward. For some reason, this did not work, only one valve would stay open. We tried the original orientation, with both set to open, no luck. We tried to original orientation with the original code again, still no luck. When we plugged them both into the motor controller the second time, the valve that hadn't been working started to work, but the other one remained closed. We are very unsure why this was happening. I would guess it has something to do with power. It is possible the valve was broken, except they did work in the test code. If we had more time I would try to figure out what was wrong with the valves.

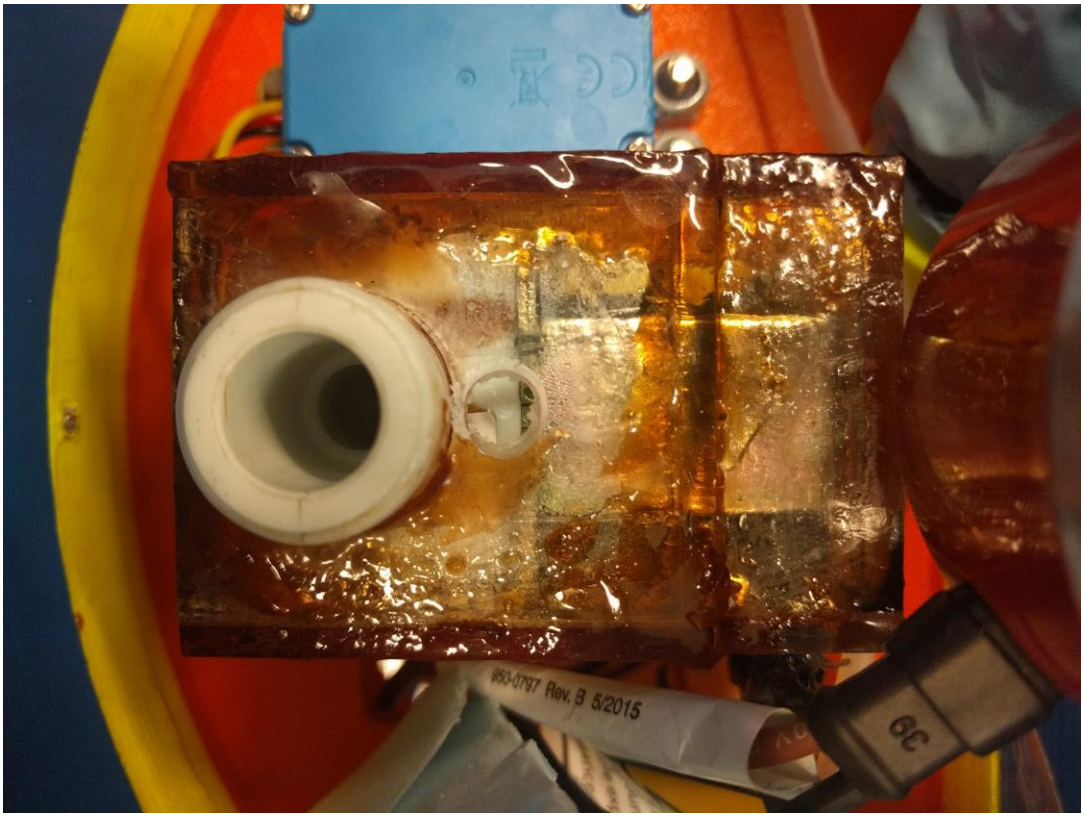

Figure 16 A valve, attempted to be waterproofed, ended up not needing it.

It was very frustrating to be working the night before and the morning of and still be taking systems out and moving things like the camera. It was frustrating to realize more pins than we expected were taken up by the shields or other pieces (like servo, Pixy Cam, and X-Bee). It was very sad to not see the part we were all most excited about, the direction steering, work and to see even the simplest systems, like the valves and fins, not behave the way we expected them to, wither mechanically or electrically/code. I was definitely sad that we never got to see our squid swim around with its light up tentacles. I think that there were a lot of draw-backs this spiral, including mechanical parts coming slowly so it was hard to test, but I also think we should have done more testing in the beginning, even without the mechanical parts all there. It was a difficult process, trying to figure out the X-Bee and Arduino serial connection, finally thinking you had it working, and then having it not actually work.

Over all, I was super excited to see the squid float in the pool. When we first tested, the squid was butt heavy and the tube wanted to pop out the top, so it was great to see it actually balanced.

Figure 17 The pressure hull trying to burst out of the water

The morning of, we were able to miraculously get the squid to move. The valves decided to both stay open, the pump worked and the fins were working. The propulsion was not strong enough to really direct the squid so it kind of just drifted around, but none the less it was exciting to see it working. It was also very exciting to see the information sent from the X-Bee to the computer because it meant we got wireless communication working. One of the coolest things, was seeing the data from the Pixy Cam behave exactly how we wanted, and seeing the robot try to turn the fins the correct direction according to the information from the Pixy Cam. I honestly believe that if we had had just a little more time we could have figured everything out. It felt like we had all the pieces there, they just weren't all working quite perfectly together. Overall, I am very happy with how our squid turned out, I just wish we could have had a little more time to actually make it work.

6 Appendix

Think Act Code

```
/** * Sprint 2 Code * Think/Act * SquidBot * Mission: Drive straight to buoy, turn in circle, drive to next buoy, etc., *          then back home  * Team Squid: Aubrey, Diego, Gretchen, Jon, MJ, Paul   * 12/9/2017 * Version 1 */ // Library for Serial Transfer //#include <EasyTransfer.h> //#include <SoftwareSerial.h> //#include <AltSoftSerial.h> // Libraries included to use PixyCam
#
include < SPI.h > #include < Pixy.h > // Library included to use servos //#include<Servo.h>
    #include < ServoTimer2.h > // Libraries included to use motor and motion shield
    #include < Wire.h > Pixy pixy; //creates PixyCam object to use // CONSTANTS AND GLOBAL VARIABLES VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV // Constants
enum {
    RIGHT = -1, NONE = 0, LEFT = 1, STRAIGHT = 2
}; // Directions
       enum {
           GREEN = 3, YELLOW = 4, RED = 5, HOME = 6, DANCE = 7, LOOP = 8
       }; // Targets
       const int APPROACH_DIST = 100; // Distance from target to start turning (inches)
       const float K_P = 1.0; // Proportional constant for feedback control
       const int FORWARD_VELOCITY = 255; // Pump output for normal swimming
       const int TURNING_VELOCITY = 255; // Pump output for turning
       const int MAX_MISSION_LENGTH = 10; // Maximum number of targets in a mission
       const int CAMERA_RATIO = 1; // Distance from buoy divided by pixel width of buoy (inches/pixel)
       const int SERVO_MIN_POSITION = 1000; //0; // Minimum angle that servos can output
       const int SERVO_MAX_POSITION = 2000; //170; // Maximum angle that servos can output
       const int FIN_FORWARD_ANGLE = 500; //85; // Left fin servo value for going forward (right is reversed)
       const int FIN_TURN_ANGLE = 100; //120; // Left fin servo value for turning (right is reversed)
       const int TUBE_ZERO_ANGLE = 1500; // Left tube servo default position for going forward (right is reversed)
       const int TURNING_ANGLE = 2000; //170; // Angle output for initiating a turn, cutoff for applying valves and fins
       const int MAX_BLOCKS = 7; // Maximum number of blocks sent from pixycam
       const bool TURN_TUBES = true; // Whether to use tube servos for steering
       const bool TURN_VALVES = true; // Whether to use valves for steering
       const bool TURN_FINS = true; // Whether to use fins for steering // Pins
       const int FIN1 = 10; // Right fin
       const int FIN2 = 3; // Left fin
       const int TUBE1 = 9; // Right tube pull
       const int TUBE2 = 5; // Left tube pull
       const int VALVE2 = 2; // Left valve through relay
       const int PUMPE = 4; // Pump PLL speed control pin
       const int PUMPM = 5; // Pump motor plug
       const int VALVE1E = 7; // Valve PLL speed control pin
       const int VALVE1M = 6; // Valve motor plug // Objects
       ServoTimer2 rightFin, leftFin, leftTube, rightTube; //AltSoftSerial Arduino(12,13); //communicate with sense Arduino RX TX //EasyTransfer ETin, ETout; // State variables
       int direction = NONE; // Computed direction to travel
       int mission[MAX_MISSION_LENGTH]; // Ordered array of targets, e.g. {RED, YELLOW, WHITE, HOME, NONE}
       int target = 0; // Current target index
       int distance = 0; // Distance from target in inches
       int angle = 0; // Angle towards target in degrees CCW
       long previousMillis = 0; // Previous loop time in milliseconds
```

```
    boolean flood, temp = false; // E-Stop activated, hull flooding, electronics overheating
    int loops = 0;
    float widths[MAX_BLOCKS];
    int16_t signatures[MAX_BLOCKS];
    float positions[MAX_BLOCKS];
    boolean estop = false; // //// Serial send/recieve structures //struct RECEIVE_DATA_STRUCTURE{ //  //put your variable definitions here for the data you want to receive //  //THIS MUST BE EXACTLY THE SAME ON THE OTHER ARDUINO //  float widths[MAX_BLOCKS]; //  int16_t signatures[MAX_BLOCKS]; //  float positions[MAX_BLOCKS]; //  boolean estop; //}; // // //// Give a name to the group of data //RECEIVE_DATA_STRUCTURE rxdata; // SETUP ROBOT CODE (RUN ONCE) SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
    void setup() { // Serial transfer initialization
        Serial.begin(9600);
        pixy.init(); //Arduino.begin(4800); //ETin.begin(details(rxdata), &Arduino); // Serial.println("In setup"); // Pin initialization
        rightTube.attach(TUBE1);
        leftTube.attach(TUBE2);
        leftFin.attach(FIN2);
        rightFin.attach(FIN1);
        pinMode(PUMPM, OUTPUT);
        pinMode(VALVE1M, OUTPUT); // Right
        pinMode(VALVE2, OUTPUT); // Left
        Serial.println("About to system check");
        systemCheck();
        Serial.println("System check done");
    } // ROBOT CONTROL LOOP (RUNS UNTIL STOP) LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
    void loop() {
        delay(20);
        loops++; //  if(0&&ETin.receiveData()){ //    Serial.print("Magnet: "); //    Serial.println(rxdata.estop); //    Serial.print("Sig: "); //    Serial.println(rxdata.signatures[0]); //    Serial.println(rxdata.signatures[1]); //    Serial.println(rxdata.signatures[2]); //    Serial.println(rxdata.signatures[3]); //    Serial.println(rxdata.signatures[4]); //    Serial.println(rxdata.signatures[5]); //    Serial.println(rxdata.signatures[6]); //    Serial.print("Pos: "); //    Serial.println(rxdata.positions[0]); //    Serial.print("Width: "); //    Serial.println(rxdata.widths[0]); //  }
        digitalWrite(PUMPM, HIGH);
        analogWrite(PUMPE, 255);
        downloadMission();
        readSenseArduino();
        think();
        act();
        if (loops % 10 == 0) debug();
    } // CONTROL FUNCTIONS CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC // Delay loop
    void wait(int t) {
        previousMillis = millis();
        while (millis() - previousMillis <= t) {}
    } // Check for new mission over Serial in the format of a string of characters
    void downloadMission() {
        int n = Serial.available();
        if (n < 1) { // No message available
            return;
        }
        for (int i = 0; i < n; i++) { // Map input characters to desired targets
            switch (Serial.read()) {
                case '>':
                    return; // Debug message
                case '2':
```

```
                            mission[i] = STRAIGHT;
                            break;
                        case '1':
                            mission[i] = LEFT;
                            break;
                        case '3':
                            mission[i] = RIGHT;
                            break;
                        case 'r':
                            mission[i] = RED;
                            break;
                        case 'y':
                            mission[i] = YELLOW;
                            break;
                        case 'g':
                            mission[i] = GREEN;
                            break;
                        case 'h':
                            mission[i] = HOME;
                            break;
                        case 'd':
                            mission[i] = DANCE;
                            break;
                        case 'l':
                            mission[i] = LOOP;
                            break;
                        default:
                            mission[i] = NONE;
                    }
                }
                for (int i = n; i < MAX_MISSION_LENGTH; i++) {
                    mission[i] = NONE;
                }
                target = 0;
            } // Compute distance and direction from sense Arduino input
        void readSenseArduino() {
                int n = pixy.getBlocks();
                for (int i = 0; i < MAX_BLOCKS; i++) {
                    signatures[i] = 0;
                }
                for (int i = 0; i < min(n, MAX_BLOCKS); i++) {
                    widths[i] = pixy.blocks[i].width;
                    positions[i] = pixy.blocks[i].x;
                    signatures[i] = pixy.blocks[i].signature;
                }
                distance = -1;
                angle = 0;
                for (int i = 0; i < MAX_BLOCKS; i++) {
                    if (signatures[i] == mission[target] - 2) { // G,Y,R,H = 1,2,3,4
                        distance = CAMERA_RATIO * widths[i];
                        angle = positions[i] - 159; // 159 = center of screen
                    }
                } //  if(ETin.receiveData()){ //recieves data: n, blocks //    delay(20); //
    distance = -1; //    angle = 0; //    if(rxdata.estop){ //      eStop(); //  } //     //}
            } //Check all systems
        void systemCheck() {
                wait(1000);
                move(0, TURNING_ANGLE);
                wait(1000);
                move(0, -TURNING_ANGLE);
                wait(1000);
```

```
            move(0, 0);
        } //eStop function to shut off all motors
    void eStop() { //  rightFin.write(0); //  leftFin.write(0); //  leftTube.write(0); // rightTube.write(0);
            digitalWrite(VALVE2, LOW);
            digitalWrite(PUMPM, LOW);
            analogWrite(PUMPE, 0);
            digitalWrite(VALVE1M, LOW);
            analogWrite(VALVE1E, 0);
        } // Output current state over Xbee
    void debug() {
            Serial.print(">>> Mission: ");
            for (int i = 0; i < MAX_MISSION_LENGTH; i++) {
                Serial.print(mission[i]);
            }
            Serial.print(", Blocks: ");
            Serial.print(signatures[0]);
            Serial.print(signatures[1]);
            Serial.print(signatures[2]);
            Serial.print(signatures[3]);
            Serial.print(signatures[4]);
            Serial.print(signatures[5]);
            Serial.print(signatures[6]);
            Serial.print(", Target: ");
            Serial.print(target);
            Serial.print(", Direction: ");
            Serial.print(direction);
            Serial.print(", Distance: ");
            Serial.print(distance);
            Serial.print(", Angle: ");
            Serial.print(angle);
            Serial.print(", Flood: ");
            Serial.print(flood);
            Serial.print(", Temp: ");
            Serial.print(temp);
            Serial.print(", E-Stop: ");
            Serial.println(estop);
        } // THINK TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
    void think() {
            if (mission[target] <= 2) { // Manual override
                direction = mission[target];
            } else if (mission[target] == DANCE) { // Dance code
                direction = DANCE;
            } else if (distance < 0) { // Target not visible
                direction = LEFT;
            } else if (distance > APPROACH_DIST) { // Reached target
                target++;
                if (mission[target] == LOOP) { // Restart mission
                    target = 0;
                }
                direction = NONE;
            } else { // Target visible
                direction = STRAIGHT;
            }
        }
     // ACT AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    void act() {
            if (estop) {
                move(0, 0);
```

```
298.                     return;
299.                 }
300.                 switch (direction) {
301.                     case STRAIGHT: // Swim straight using proportional feedback control
302.                         move(FORWARD_VELOCITY, int(K_P * angle));
303.                         break;
304.                     case LEFT: // Turn left
305.                         move(TURNING_VELOCITY, TURNING_ANGLE);
306.                         break;
307.                     case RIGHT: // Turn right
308.                         move(TURNING_VELOCITY, -TURNING_ANGLE);
309.                         break;
310.                     case DANCE: // Show off your moves
311.                         break;
312.                     default: // Stop
313.                         move(0, 0);
314.                         break;
315.                 }
316.             } // Output motor values
317.         void move(int vel, int ang) { // Set pump output //  if(vel>0) { //    //Serial.print("
    vel when pump on: "); //    //Serial.println(vel); //    digitalWrite(PUMPM, HIGH); //    anal
    ogWrite(PUMPE, vel); //  } else { //    //Serial.print("vel when pump off: "); //    //Serial.
    println(vel); //    digitalWrite(PUMPM, LOW); //    analogWrite(PUMPE, 0); //  }
318.                 digitalWrite(PUMPM, HIGH);
319.                 analogWrite(PUMPE, 255); // Set tube angles
320.                 if (TURN_TUBES) {
321.                     int leftTubeAngle = min(max(TUBE_ZERO_ANGLE + ang, TUBE_ZERO_ANGLE), TUBE_Z
    ERO_ANGLE + TURNING_ANGLE);
322.                     int rightTubeAngle = min(max(TUBE_ZERO_ANGLE - ang, TUBE_ZERO_ANGLE), TUBE_
    ZERO_ANGLE + TURNING_ANGLE);
323.                     leftTube.write(SERVO_MIN_POSITION + leftTubeAngle);
324.                     rightTube.write(SERVO_MAX_POSITION - rightTubeAngle);
325.                 } else {
326.                     leftTube.write(SERVO_MIN_POSITION + TUBE_ZERO_ANGLE);
327.                     rightTube.write(SERVO_MAX_POSITION - TUBE_ZERO_ANGLE);
328.                 } // Set fin angles
329.                 if (TURN_FINS && ang >= TURNING_ANGLE) { // Left //Serial.println("turn fins le
    ft");
330.                     leftFin.write(SERVO_MIN_POSITION + FIN_TURN_ANGLE - 200);
331.                     rightFin.write(SERVO_MAX_POSITION - FIN_FORWARD_ANGLE - 100);
332.                 } else if (TURN_FINS && ang <= -
    TURNING_ANGLE) { // Right //Serial.println("turn fins right");
333.                     leftFin.write(SERVO_MIN_POSITION + FIN_FORWARD_ANGLE - 200);
334.                     rightFin.write(SERVO_MAX_POSITION - FIN_TURN_ANGLE - 100);
335.                 } else { // Straight //Serial.println("turn fins straight");
336.                     leftFin.write(SERVO_MIN_POSITION + FIN_FORWARD_ANGLE - 200);
337.                     rightFin.write(SERVO_MAX_POSITION - FIN_FORWARD_ANGLE - 100);
338.                 } // Set valve states
339.                 if (TURN_VALVES && ang >= TURNING_ANGLE) { // Left //Serial.println("turn VALVE
     left");
340.                     digitalWrite(VALVE1M, LOW);
341.                     analogWrite(VALVE1E, 0);
342.                     digitalWrite(VALVE2, HIGH);
343.                 } else if (TURN_VALVES && ang <= -
    TURNING_ANGLE) { // Right //Serial.println("turn VALVE right");
344.                     digitalWrite(VALVE1M, HIGH);
345.                     analogWrite(VALVE1E, 255);
346.                     digitalWrite(VALVE2, LOW);
347.                 } else { // Straight //Serial.println("turn VALVE straight");
348.                     digitalWrite(VALVE1M, HIGH);
349.                     analogWrite(VALVE1E, 255);
```

```
                    digitalWrite(VALVE2, HIGH);
                }
```

Sense Code

```
/** * Sprint 2 Code * Sense * SquidBot * Mission: Drive straight to buoy, turn in circle, drive to next buoy, etc., *          then back home * Team Squid: Aubrey, Diego, Gretchen, Jon, MJ, Paul   * 10/25/2017 * Version 1 */ //library for serial communication //#include <EasyTransfer.h> //#include <SoftwareSerial.h> //need this library to run Software Serial //libraries included to use PixyCam
#
include < SPI.h > #include < Wire.h > //#include <PixyI2C.h>
    #include < Pixy.h > //#include "PixyUART.h" //library included to use servos
    #include < Servo.h > //libraries included to use motor and motion shield //#include "NineAxesMotion.h"  //Constants and Global Variables VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
    Pixy pixy; //creates PixyCam object to use //EasyTransfer ETin, ETout; //creates serial structures to transfer data //flood True if hull flooding //temp true if electronics overheating

boolean flood, temp, estop = false;
const int COLUMNS = 16;
const int ROWS = 1;
const int FLOODPIN = A3;
const int MAX_BLOCKS = 7;
const int STOP = A0; // Magnetic sensor pin to determine eStop
const int TEMP = A2;
const int RELAY = 5;
int filter = 0; //struct SEND_DATA_STRUCTURE{ //  //put your variable definitions here for the data you want to receive //  //THIS MUST BE EXACTLY THE SAME ON THE OTHER ARDUINO //  float widths[MAX_BLOCKS]; //  int16_t signatures[MAX_BLOCKS]; //  float positions[MAX_BLOCKS]; //  boolean estop; //}; // ////give a name to the group of data //SEND_DATA_STRUCTURE txdata; //SETUP ROBOT CODE (RUN ONCE)SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
void setup() {
        Serial.begin(9600);
        pixy.init(); //Arduino.begin(4800); //ETout.begin(details(txdata), &Serial); //Serial.println("SETUP");
        pinMode(STOP, INPUT);
        pinMode(FLOODPIN, INPUT);
        pinMode(RELAY, OUTPUT);
        delay(100);
    } //ROBOT CONTROL LOOP (RUNS UNTIL STOP)LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
void loop() { //ETout.sendData(); //  Serial.print(txdata.signatures[0]); //  Serial.print(txdata.signatures[1]); //  Serial.print(txdata.signatures[2]); //  Serial.print(txdata.signatures[3]); //  Serial.print(txdata.signatures[4]); //  Serial.print(txdata.signatures[5]); //  Serial.println(txdata.signatures[6]);
        delay(20); //checkFlood(); //checkTemp();
    } //CONTROL FUNCTIONS CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
void checkFlood() {
    int liquidLevel = digitalRead(FLOODPIN);
    if (liquidLevel == HIGH) {
        flood = true; //Serial.println("FLOOD");
    } //else //Serial.println("Water Good");
}
void checkTemp() { //temp 150F
    int val = analogRead(TEMP); //Connect LM35 on Analog 0
    float dat = (double) val * (5 / 10.24);
    if (dat >= 65.5) {
```

```
37.         temp = true; //Serial.println("FIRE");
38.     } //else //Serial.println("Temp Good");
39. }
40. void eStop() {
41.     estop = digitalRead(STOP);
42.     if (eStop) {
43.         digitalWrite(RELAY, HIGH);
44.     }
45. }
```