Phys 124 Final Report

Andrew Nguyen
Alex Shakouri

March 25, 2017

Project Title: *Julianne Stingray*

(subtitle: Bartending Bot)

Motivation and overall concept

It is obviously everyone's dream to become a bartender, preferably in a small, neatly kept place well known by the locals. You'd be so popular. You'd have so many friends. And from your customers, you'd hear stories---happy ones, sad ones, and of course, ones that would never be told if not for the drink of the brave.

So we designed and built an automatic bartender assembly expressly for wish fulfillment---vicarious drink mixing. For simplicity, we designed it to accept three ingredients in approximately bottle shaped containers, namely tequila, mixers, or syrups, which are arranged circularly on platforms. The platforms tilt, their hinges actuated by stepper motors and pulled by a rope, and weight (force) sensors judge the rate at which the drink is being poured to increase/decrease the tilt angle accordingly. The amount of each ingredient is implicitly specified by the user through serial input, where a drink can be selected through a menu, with the components of each drink having been pre-specified by us. We originally hoped to be able to mix the drink as well, but time constraints proved unwieldy.

Functional Definition

Our device is intended to correctly mix a user-specified drink with no mess.

Motors, run at a low speed (high torque), are attached to posts, which are themselves attached to platforms that hold the bottles. The posts are on a hinge, and the bottles poke out just above the posts, so that if it were at an oblique angle to pour, the liquid would land directly into a funnel below---see the picture.

To begin pouring, the motor turns, pulling a rope at the top of the post. To retain a reasonable pour rate, the ingredient's weight is read constantly and compared with the weight at the previous time step, constituting an approximate derivative of weight over time. (We assumed density of water throughout). So the the angle stops incrementing for fast rates of change, starts

again as the rate slows, and increments faster or slower depending on the magnitude of the “derivative”.

We calibrated the device so that the pour rates are not erratic, too slow, or too voluminous---but we did not need to limit the pour angle, which was more trouble than it was worth. As long as the motor ran slowly enough, it was perfectly fine.

Once enough liquid is observed to have been poured, the bottles return to an upright position.

Sensors

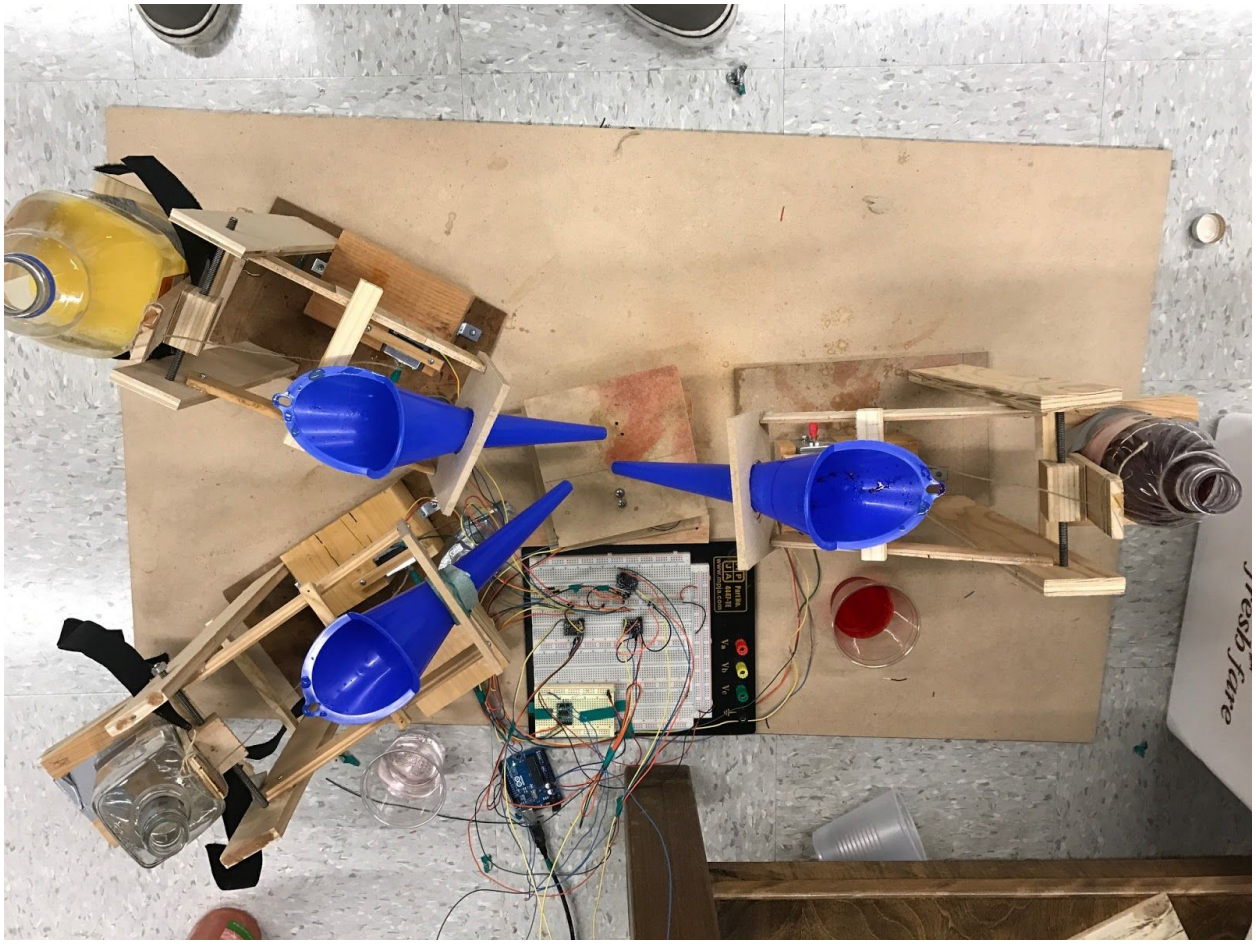
We used a single strain gauge load cells (TAL220, $7) for the force sensors, 1 kg capacity; We bolted the sensors with one bolt to the platform above and one to the platform below we can measure torque instead of compression force, which is more reliable, and convert that into the applied force of the ingredients resting on the platform (see picture above of how we mounted the load sensor). Their output is a (weak) voltage signal, which is more strongly positive for greater detected force.

It worked relatively well, all things considered. For those wishing to measure weight, we do recommend using load cells, and if desired, this brand. Precision was plus or minus five grams at worst.

They are powered by 5 V and ground; the output leads are connected to an amplifier, the HX711 IC, due to the weak signal of strain gauge load cells. Resistors and capacitors in the circuit are arranged in the configuration assumed by the HX711 library on Sparkfun; alternatively we can purchase a breakout board with resistors/so on soldered in for us ($10 at Sparkfun, but much cheaper elsewhere). The data (DAT) and clock (CLK) outputs from the IC go to any two GPIO pins on the Arduino.

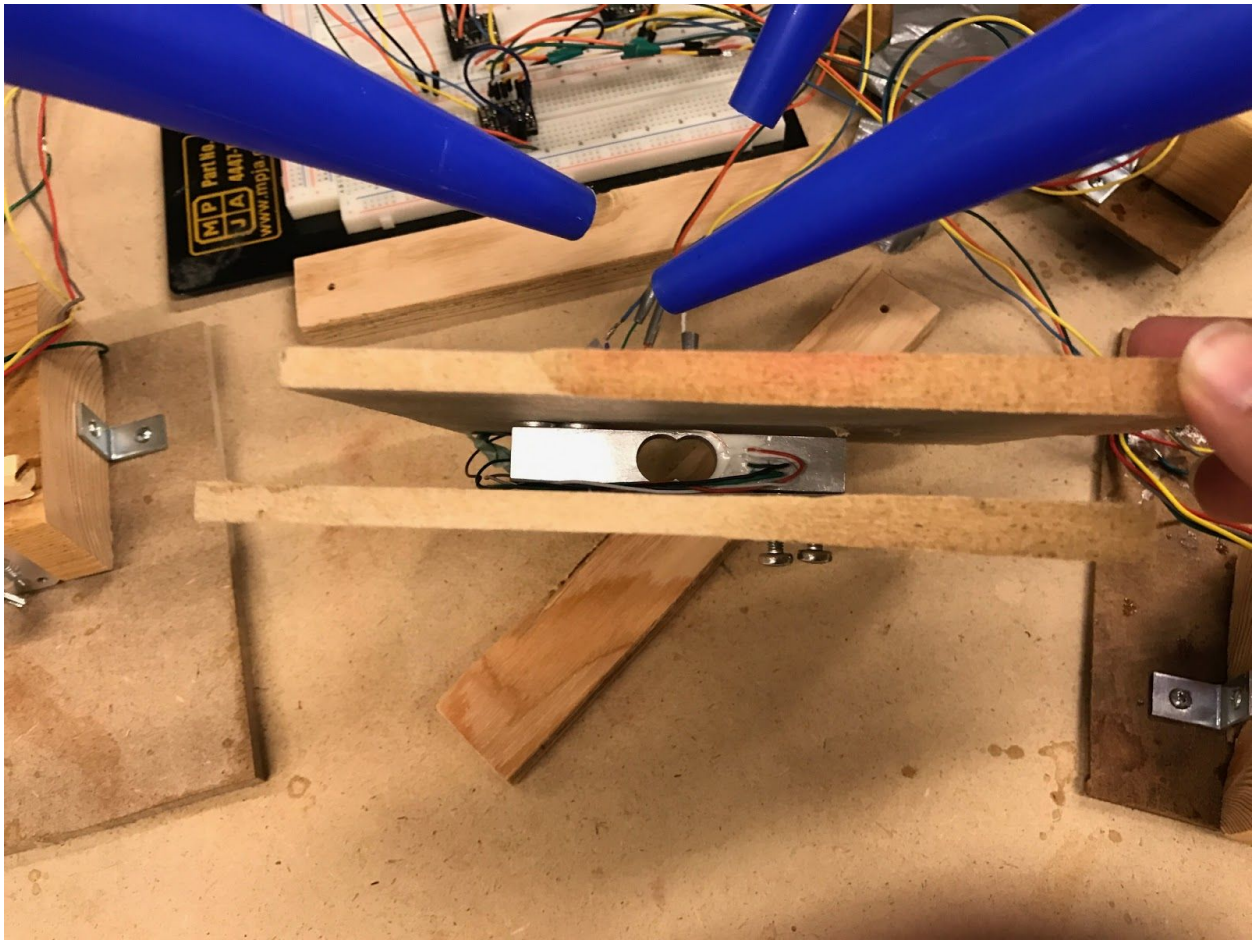
Mechanical Considerations

In total we have just a centerpiece as well as 3 pour designs surrounding the centerpiece. All of this is attached onto one piece of wood for easy accommodation.



In the center of our design we have a weight sensor with a cup attached to it that the liquid from the funnels we attached would fall. We also have the pour designs we came up with in order to pour the liquid from any size bottle successfully. When making and testing the weight sensor we realized that we needed to figure out a way to have a stable centerpiece but a piece that would also work with this weight sensor design. The 0.78 kg micro load cell had a resolution of 2g and works well when we put wieght on it.

The only design aspect is when attaching the weight sensor to the more stable two pieces of wood we had two screws in the bottom sticking out. In order to accommodate for this we needed to build a support of two pieces of wood you see in the background photo that the sides rest on to allow for a correct measurement.



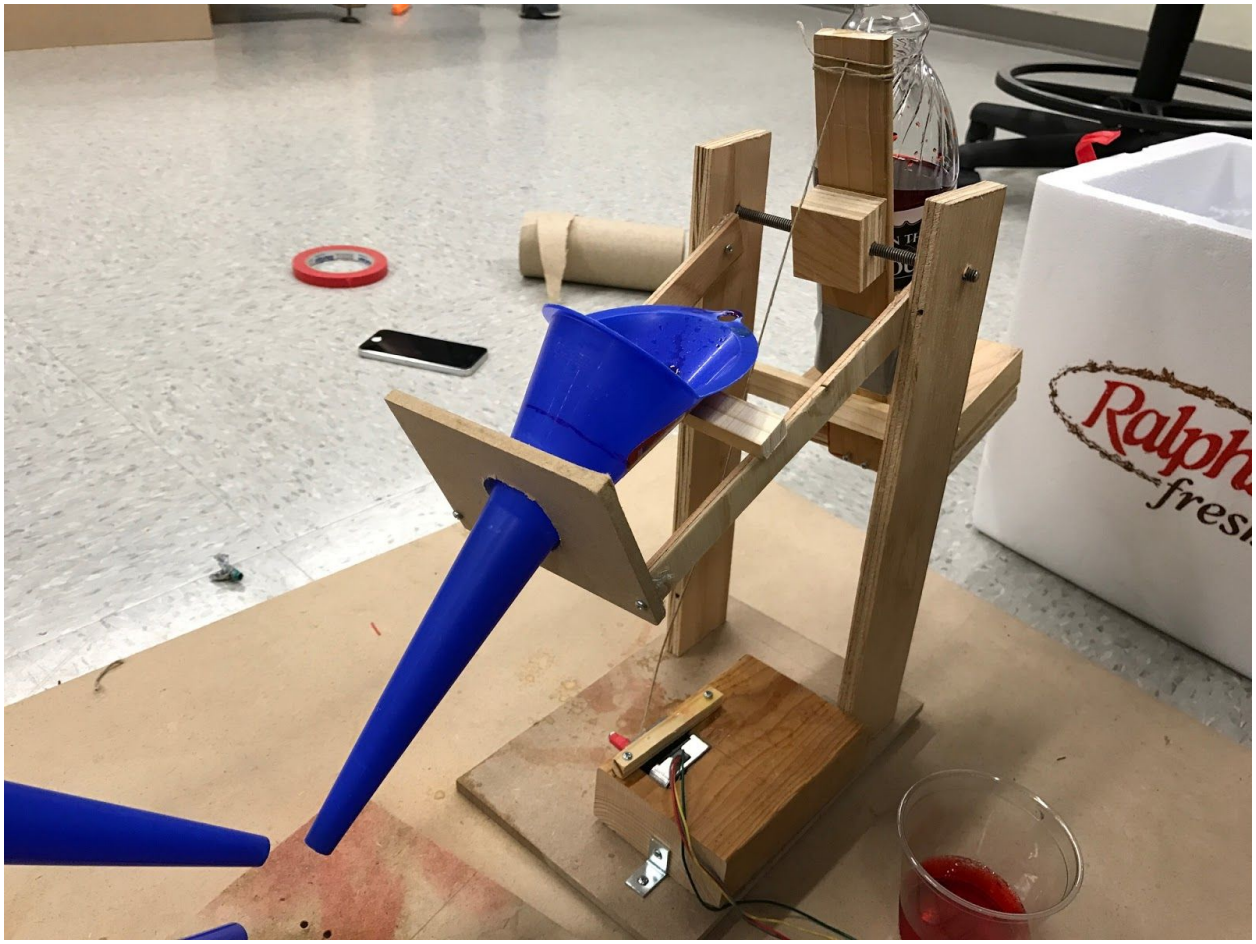
What we have is our 3 pour design surrounding the mixed drink where all four will pour in succession. They are all held up with wooden support on the side through the cups, but attached by a threaded rod specifically cut for it. The actual tilting part of the design is made up of a stand, and lengthwise wooden piece made for certain drink sizes. It is also fitted with seatbelts made of velcro in order to make it so that the drink doesn't fall out when we pour it.

It is attached to the threaded rod in the middle with a small block of wood with a hole through the middle to put through the threaded rod. We also have small wooden supports supporting the tilting structure. We tried to make the design not only functional, but look as sleek as possible.

In the beginning we were seeing problems with the motors not being strong enough to bring the tilting structure to a tilt. We chose a whole bunch of different motors, but stuck with the 12V, 0.35A Stepper Motor from adafruit. We also had to overdrive the motors so that it would be powerful enough to tilt the tilt structure perfectly. The stepper motors will be on the bottle in front of the tilting machine about 5 inches out. A piece of string will be attached from the motor to the wooden panel to allow for tilt as the motor brings the string in. The string is attached by tape to the motor and up throughout the entire quarter was not a bad idea, but it can actually slip off the motor and cause it to not work when the motor spins.

The structures were made of wood. The cups will be measured first and then tilted until a sufficient amount of liquid is poured or the rotation goes to the maximum it can go. We will have a seat belt of sorts attached to the tilting part of our design that will help to keep the bottle in place as we tilt it. We will also have extra 1inch blocks in order to adjust the height of our pour design to accommodate any size length bottle. As our pour design will pour the liquid then it will fall into a set funnel where the cup with the weight sensor is waiting in the middle.

One big area of contention for us was the piece of wood accommodating the funnel. We wanted the funnel to be able to placed easily within the pour design, but we needed it to be stable when we poured our drink. We went through a lot of different failed designs for this to work but asked around for ideas. The version that we finally went with was a mix of two different ideas that were pitched to me and is the best version that worked so far. The funnels themselves are also very long, but we need an angle to be able to pour the liquid.



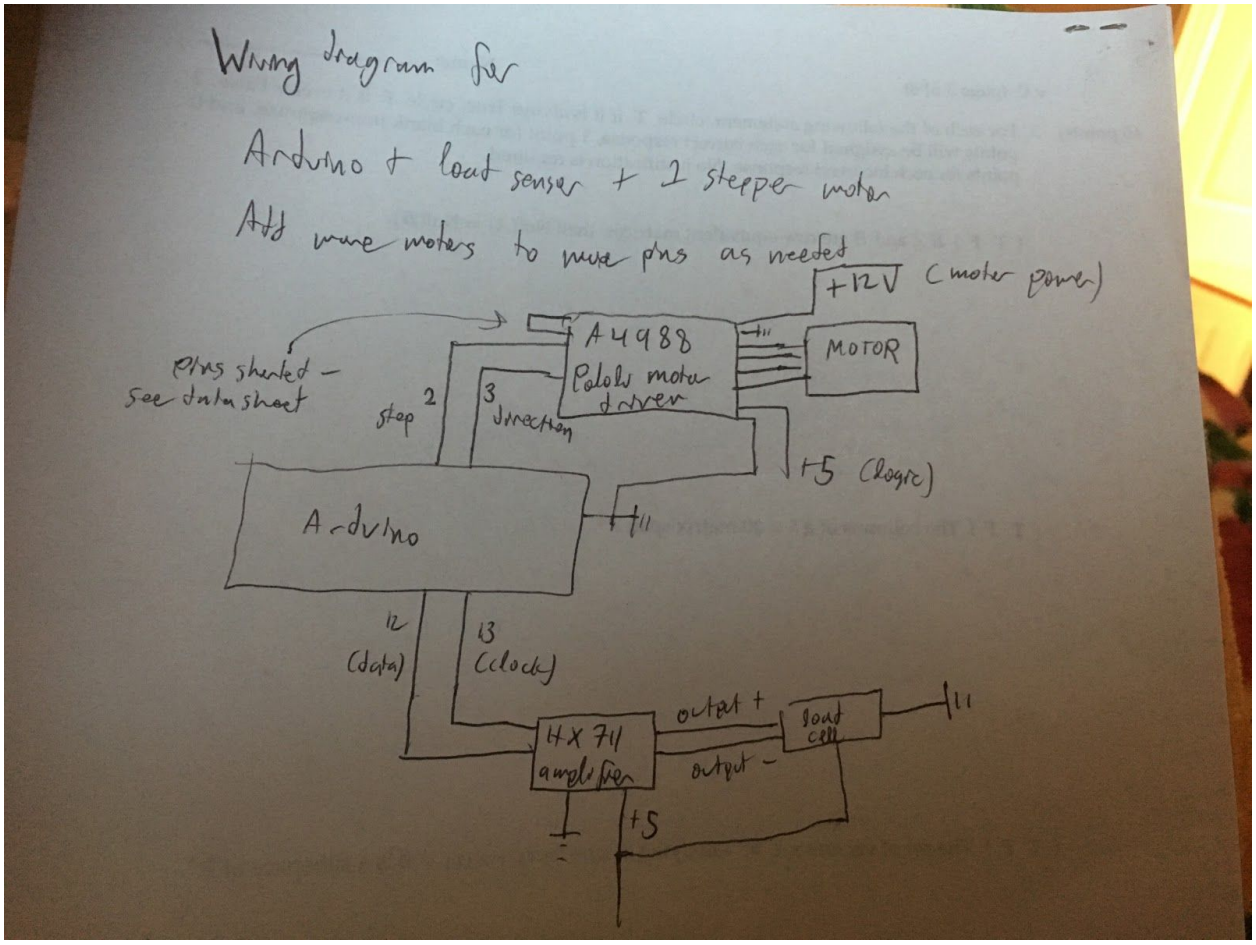
Overall, the mechanical aspect was a huge learning experience and definitely the hardest part of this project, but also the most satisfying to see it work!

Electrical Considerations

From the Arduino, we only need two pins per motor, and two pins for the weight sensor. That comes out to be 8 pins in all.

We provide power to the motors through the motor drivers. The motors were rated for 12 V and 0.35 A, but that wasn't enough to power our machine---it didn't provide sufficient torque. We attached them to the variable power supply provided on our desks, so we were able to provide each motor instead with roughly 0.5 A, which we found was perfectly fine. The motors heated up, but did not always burn to the touch, and we never damaged anything by doing this.

The motor drivers, as well as the HX711 and the load cells, both also need five volts and ground---so the motor driver accepts two voltage sources, one for driving the motor, and one for logic. The HX711 needs 2 inputs from the load sensors themselves.



Interface

We will be using the arduino uno in order to power the stepper motors to tilt the drinks into the shaker. We will also be using a Serial monitor in order to keep track and test the motors so that they are working properly. THe serial monitor had pre made drink ratios already inputted in.

We will be using 3 stepper motors to efficiently control our stepper motors and will be using the power and ground pins on the arduino. Each motor will have their own motor driver, the Pololu 8-35V 2A Single Bipolar Stepper Motor Driver.

Software

This was programmed in C, through the Arduino IDE.

The code is provided here (Drew wrote it and his beginner programming shows), but here's an outline of what it does. The loop is three functions, pouring_1, pouring_2, and

pouring_3, one for each ingredient. These are called continuously until the system decides we've poured enough for that ingredient, in which case each pouring function has a subroutine (the gohome function) that increases the speed of the motor (the pw parameter) and makes it go in the other direction.

The pouring itself was done amidst checking the derivative. First we draw up the bottle to a reasonable height reasonably quickly---then we change the motor speed (through pw) to become much slower and begin checking to see if we should keep the motor fixed because the pour rate is good, or if we should go forward a little more.

How to run the motor? No libraries. The stepper drivers we got told us to just send pulses of a fixed width through the "STEP" pin for each motor. The longer the width, the greater the torque and less the speed.

How to read the load sensor? In this case, there was a nice library corresponding to the HX711 which works really well.

Testing

Testing was really scary with the bottle pouring stuff and most of the reason we came up with the funnel in the first place---to make sure we could pour reliably. Aside from this, the process was relatively straightforward. Test, see problem, troubleshoot code, check wiring, hope it's not a fundamental problem with design, as it occasionally was sometimes---our motors were too weak initially and Alex came up with the rope idea to get more leverage out of them.

Safety

Tried to keep the liquid from the cups away from the electronics to make sure that nothing shorts if something spills. Tested out the arm first without liquid just in case and then go slowly to see with liquid.

Worse case senario is that the water spills from the cups and hits exposed electrical wires and shorts our project. We found that if we leave it be we only get small spills when the bottle is going back up.

Resuability/Parts required

We used 3 steppers available in lab, Adafruit product ID 324. We also needed a \textit{lot} of scrap wood blocks to fashion into platforms/stands, mounting hardware e.g. screws, bolts, long rods, as well as lots of tape. And of course we needed the motor drivers, load sensors, and load sensor amplifiers.

Unless we spill water on stuff during testing (and we didn't, at least not on the electrical stuff) none of the electrical components ought to be unusable---nothing bent out of shape. The wood will have been cut and to some extent drilled, but they are mostly in nice rectangular blocks and could be reused (unless we decide to finish them).

Specs sheet:

Num	Part
3	pour design (come up through trial and error)
	2 12x12x3/8 inch
	1 10x3/2x1/4 inch
	1 4x4x1/4 inch
	1 3/2x3/2x1 inch
	1 10/2x1/4x3/8 inch
	2 8x3/2x1/2 inch
	1 5x3x1/5inch
	1 3x3/8x1/2 inch
	Miscellaneous wires, capacitors
3	Pololu 8-35V 2A Single Bipolar Stepper Motor Driver
3	10V, 0.5A, 13oz-in Bipolar Stepper Motor
3	multipurpose long funnels
1	5ft of velco strap

Discarded/Possible future Goals

We originally wanted to make four of these things rather than just three, and we also wanted a shaker mechanism. Maybe put a lid on the mixed drinking and use dc motors to spin it around to mix instead of stir, or use a spring attached to the mixed drink to shake it that way

Without a way to adjust the height of our platforms, we would have problems with very short or very long bottles. We ended up using blocks cut to the right dimensions for short bottles to rest their feet on and be put right up to the lip of the funnel, and we thought we'd cut holes through the blocks to slide a rectangular metal bar in through so that the blocks wouldn't fall out, but drilling irregularly shaped holes through thick blocks---that is, roughly rectangular shaped---is really really time consuming, and we didn't implement it throughout.

We also would have liked to put in fancy lights, but didn't have time.

Project code:

```
#define DIR_M1 2 //Green
#define STEP_M1 3 //Orange
#define DIR_M2 4 //Green
#define STEP_M2 5 //Orange
```

```
#define DIR_M3 6 //Green
#define STEP_M3 7 //Orange
#define FORWRD LOW
#define BKWRD HIGH
#include "HX711.h"


HX711 scale;
float calibration_factor = 988; //988 seems to calibrate to measure grams
float val = 0; //readings are incremented to take average
float avgnow = 0;
float avgprev = 0;
float now; //time now
float prev = 0; //previous time step
float deriv = 0;
float beginval;
float currentval = 0;
int pour_begin2 = 0;
int pour_begin3 = 0;
int samp = 3; //number of samples to take for weight
int go = 0; //0 is FALSE, 1 is TRUE

int pw = 15000;

const int minderiv = 2;
const int maxderiv = 10;

int cup = 100; //150 milliliters (grams) per drink.
int state_m1 = 0; //where the motor is
int state_m2 = 0;
int state_m3 = 0;
int dir_m1 = 1; //1 is default for forward
int dir_m2 = 1;
int dir_m3 = 1;
int i = 0; //index
int j; //index
int k = 0; //another index
int l = 0; //fucking indices
int n = 12; //number of steps to turn after turn() is called
int T = 2; //amount of time before we allow the angle to turn
int M = 2; //amount of time before we decide the cup is full
int p = 0; //whoops more indexes
int ii = 0; //another index
```

```
int start = 0;

int request = 0;


float ing1; //proportion of ingredient 1
float ing2; //proportion of ingredient 2
float ing3;
int done_ing1 = 0;
int done_ing2 = 0;
int done_ing3 = 0;

void setup(){
    Serial.begin(9600);
    scale.begin(13, 12); //scale.begin(DATA, CLK)
    scale.set_scale(calibration_factor);

    //(something i still don't understand:
    //i could also write scale.set_scale(1000.f),
    //where 1000 is the calibration factor.
    //the .f changes is into a float. i've never heard of that...
    //but also, if you don't write .f and leave an int,
    //the scale is calculated hella weird! maybe because of floor division?)

    scale.tare();               // reset the scale to 0
    pinMode(DIR_M1,OUTPUT);
    pinMode(STEP_M1, OUTPUT);
    digitalWrite(DIR_M1,FORWRD); //Default direction for winding. set to
BKWRD to unwind.
    dir_m1 = 1;
    pinMode(DIR_M2,OUTPUT);
    pinMode(STEP_M2, OUTPUT);
    digitalWrite(DIR_M2,FORWRD); //Default direction for winding. set to
BKWRD to unwind.
    dir_m2 = 1;
    pinMode(DIR_M3,OUTPUT);
    pinMode(STEP_M3, OUTPUT);
    digitalWrite(DIR_M3,LOW);
    dir_m3 = 1;


  Serial.println("Hi, welcome to VA-11, HALL-A. I'm Jill, your bartender.");
```

```
  Serial.println("What drink would you like? We have ingredients for a
Paloma, a lime margarita, and my favorite, the Tequila Sunrise.");
  Serial.println("But we don't have enough space for all the bottles, so
you'll have to ask my manager to put them in the right place, depending on
your order.");
  Serial.println("If that's okay with you, then by all means, order a drink.
Enter into the Serial Monitor 1 for a Sunrise, 2 for a Paloma, and 3 for the
margarita.");
}

void turnM1(){
    for(j=0; j<n;  j++){
    digitalWrite(STEP_M1,HIGH);
    delayMicroseconds(pw);
    digitalWrite(STEP_M1,LOW);
    delayMicroseconds(pw);
        if(dir_m1 = 1){
      state_m1 += 1;
      }
    else if(dir_m1 = 0){
      state_m1 -= 1;
      }
    }


}

void no_turnM1(){
    for(j=0; j<n;  j++){
    digitalWrite(STEP_M1,LOW);
    delayMicroseconds(pw);
    digitalWrite(STEP_M1,LOW);
    delayMicroseconds(pw);
    }
}

void turnM2(){
    for(j=0; j<n;  j++){
    digitalWrite(STEP_M2,HIGH);
    delayMicroseconds(pw);
    digitalWrite(STEP_M2,LOW);
    delayMicroseconds(pw);
    if(dir_m2 = 1){
```

```
      state_m2 += 1;
      }
    else if(dir_m2 = 0){
      state_m2 -= 1;
      }
    }

}

void no_turnM2(){
    for(j=0; j<n;  j++){
    digitalWrite(STEP_M2,LOW);
    delayMicroseconds(pw);
    digitalWrite(STEP_M2,LOW);
    delayMicroseconds(pw);
    }

}

void turnM3(){
    for(j=0; j<n;  j++){
    digitalWrite(STEP_M3,HIGH);
    delayMicroseconds(pw);
    digitalWrite(STEP_M3,LOW);
    delayMicroseconds(pw);
    if(dir_m3 = 1){
      state_m3 += 1;
      }
    else if(dir_m3 = 0){
      state_m3 -= 1;
      }
    }

}

void no_turnM3(){
    for(j=0; j<n;  j++){
    digitalWrite(STEP_M3,LOW);
    delayMicroseconds(pw);
    digitalWrite(STEP_M3,LOW);
    delayMicroseconds(pw);
    }
```

```
}

void gohome_m1(){
      pw = 500;
      digitalWrite(DIR_M1,BKWRD);
      go = 1;
      dir_m1 = 0;
      while(state_m1 > 0){
      for(j=0; j<n;  j++){
      digitalWrite(STEP_M1,HIGH);
      delayMicroseconds(pw);
      digitalWrite(STEP_M1,LOW);
      delayMicroseconds(pw);
      state_m1 -= 1;
      }
  }
     pw = 15000;
     digitalWrite(DIR_M1,FORWRD);

}

void gohome_m2(){
      pw = 500;
      digitalWrite(DIR_M2,BKWRD);
      go = 1;
      dir_m2 = 0;
      while(state_m2 > 0){
      for(j=0; j<n;  j++){
      digitalWrite(STEP_M2,HIGH);
      delayMicroseconds(pw);
      digitalWrite(STEP_M2,LOW);
      delayMicroseconds(pw);
      state_m2 -= 1;
      }
  }
     pw = 15000;
     digitalWrite(DIR_M1,FORWRD);
}

void gohome_m3(){
      pw = 500;
      digitalWrite(DIR_M3,BKWRD);
      go = 1;
```

```
      dir_m3 = 0;
      while(state_m3 > 0){
      for(j=0; j<n;  j++){
      digitalWrite(STEP_M3,HIGH);
      delayMicroseconds(pw);
      digitalWrite(STEP_M3,LOW);
      delayMicroseconds(pw);
      state_m3 -= 1;
      }
  }
     pw = 15000;
     digitalWrite(DIR_M3,FORWRD);
}


void deriv_check1(){
          if(i == (samp - 1)){
          avgnow = val/(samp*1.0); // average weight over samp samples
          now = millis();
          deriv = (avgnow - avgprev)*1000.0/(now - prev);
          /*Serial.print(avgnow);
          Serial.print(" g, at ");
          Serial.print(deriv);
          Serial.println(" grams/s");
          Serial.println(go);*/
          avgprev = avgnow;
          prev = now;
          val = 0;
          i = -1;   } //calculate the derivative after "samp" samples have
been taken of the weight.
          i++; //increment how many samples were taken.

        if(deriv > minderiv && deriv < maxderiv){ //check derivative
conditions.
          go = 0;
          k = 0;
        }
        else if(deriv < 5 && k < T){
          k++;
        }

        else if(deriv < minderiv){
          go = 1;
```

```
          digitalWrite(DIR_M1,FORWRD);
          pw = 15000;
          dir_m1 = 1;
          k = 0;
        }

        else if(deriv > maxderiv){
          go = 1;
          digitalWrite(DIR_M1,BKWRD);
          pw = 500;
          dir_m1 = 0;
        }
}

void deriv_check2(){
          if(i == (samp - 1)){
          avgnow = val/(samp*1.0); // average weight over samp samples
          now = millis();
          deriv = (avgnow - avgprev)*1000.0/(now - prev);
          /*Serial.print(avgnow);
          Serial.print(" g, at ");
          Serial.print(deriv);
          Serial.println(" grams/s");
          Serial.println(go);*/
          avgprev = avgnow;
          prev = now;
          val = 0;
          i = -1;   } //calculate the derivative after "samp" samples have
been taken of the weight.
          i++; //increment how many samples were taken.

        if(deriv > minderiv && deriv < maxderiv){ //check derivative
conditions.
          go = 0;
          k = 0;
        }
        else if(deriv < 5 && k < T){
          k++;
        }

        else if(deriv < minderiv){
          go = 1;
          digitalWrite(DIR_M2,FORWRD);
```

```
          pw = 15000;
          dir_m2 = 1;
          k = 0;
        }

        else if(deriv > maxderiv){
          go = 1;
          digitalWrite(DIR_M2,BKWRD);
          pw = 500;
          dir_m2 = 0;
        }
}

void deriv_check3(){
          if(i == (samp - 1)){
          avgnow = val/(samp*1.0); // average weight over samp samples
          now = millis();
          deriv = (avgnow - avgprev)*1000.0/(now - prev);
          /*Serial.print(avgnow);
          Serial.print(" g, at ");
          Serial.print(deriv);
          Serial.println(" grams/s");
          Serial.println(go);*/
          avgprev = avgnow;
          prev = now;
          val = 0;
          i = -1;   } //calculate the derivative after "samp" samples have
been taken of the weight.
          i++; //increment how many samples were taken.

        if(deriv > minderiv && deriv < maxderiv){ //check derivative
conditions.
          go = 0;
          k = 0;
        }
        else if(deriv < 5 && k < T){
          k++;
        }

        else if(deriv < minderiv){
          go = 1;
          digitalWrite(DIR_M3,FORWRD);
          pw = 15000;
```

```
          dir_m3 = 1;
          k = 0;
        }

        else if(deriv > maxderiv){
          go = 1;
          digitalWrite(DIR_M3,BKWRD);
          pw = 500;
          dir_m3 = 0;
        }
}

void pour_ing1(){
      currentval = scale.get_units(1); //current val. higher arguments make
the machine run slower.
      val += currentval; //Record scale, add to current "val" for derivative
    if(currentval > cup*ing1 && l < M){ //"debouncing" for cup filled
condition
      l++;
    }
    else if(currentval > cup*ing1){ //cup is filled...
      l = 0;
      gohome_m1(); //so go home.
      state_m1 = 0;
      done_ing1 = 1;
      val = 0;
      //Serial.println("gone_home1");
    }

    else { //cup is not filled
        deriv_check1();
        if(go == 1 && state_m1 < 700){
          pw = 9000;
          turnM1();
        }
        else if(go ==1 && state_m1 > 699){
           pw = 15000;
           turnM1();
        }
        /*if(go == 1 && state_m1 < 1000){ //Try 920 now.
          turnM1();
        }*/
        else{
```

```
          no_turnM1();}

        /*if(state_m1 > 800){
          delay(300);
        }
        if(state_m1 > 840){
          delay(300);
        }
        if(state_m1 > 880){
          delay(400);
        }
        if(state_m1 > 920){
          delay(400);
        }*/

    }
}

void pour_ing2(){
      currentval = scale.get_units(1); //current val
      val += currentval; //Record scale, add to current "val" for derivative
    if(currentval > beginval + cup*ing2 && l < M){ //"debouncing" for cup
filled condition
      l++;
    }
    else if(currentval > beginval + cup*ing2){ //cup is filled...
      l = 0;
      gohome_m2(); //so go home.
      state_m2 = 0;
      done_ing2 = 1;
      val = 0;
      //Serial.println("gone_home2");
    }


    else { //cup is not filled
        deriv_check2();
        if(go == 1 && state_m2 < 700){ //go fast until 700
          pw = 9000;
          turnM2();
        }
        else if(go == 1 && state_m2 > 699){
          pw = 15000;
```

```
          turnM2();
        }
         /*if(go == 1 && state_m1 < 1000){ //Upper limit, until no turn.
          turnM1();
        }*/
        else{
          no_turnM2();
        }

     /*     if(state_m2 > 750){
          delay(300);
        }
        if(state_m2 > 790){
          delay(300);
        }
        if(state_m2 > 840){
          delay(400);
        }
        if(state_m2 > 890){
          delay(400);
        }
        */
    }
}

void pour_ing3(){
      currentval = scale.get_units(1); //current val
      val += currentval; //Record scale, add to current "val" for derivative
    if(currentval > beginval + cup*ing3 && l < M){ //"debouncing" for cup
filled condition
      l++;
    }
    else if(currentval > beginval + cup*ing3){ //cup is filled...
      l = 0;
      gohome_m3(); //so go home.
      state_m3 = 0;
      done_ing3 = 1;
      val = 0;
      //Serial.println("gone_home2");
    }


    else { //cup is not filled
```

```
        deriv_check3();
        if(go == 1 && state_m2 < 700){ //go fast until 700
          pw = 9000;
          turnM3();
        }
        else if(go == 1 && state_m2 > 699){
          pw = 15000;
          turnM3();
        }
         /*if(go == 1 && state_m1 < 1000){ //Upper limit, until no turn.
          turnM3();
        }*/
        else{
          no_turnM3();
        }

     /*     if(state_m3 > 750){
          delay(300);
        }
        if(state_m3 > 790){
          delay(300);
        }
        if(state_m3 > 840){
          delay(400);
        }
        if(state_m3 > 890){
          delay(400);
        }
        */
    }
}



void loop(){
  while(1==1){
    if(Serial.available()){
      request = Serial.read();
      done_ing1 = 0;
      done_ing2 = 0;
      done_ing3 = 0;
      pour_begin2 = 0;
      pour_begin3 = 0;
```

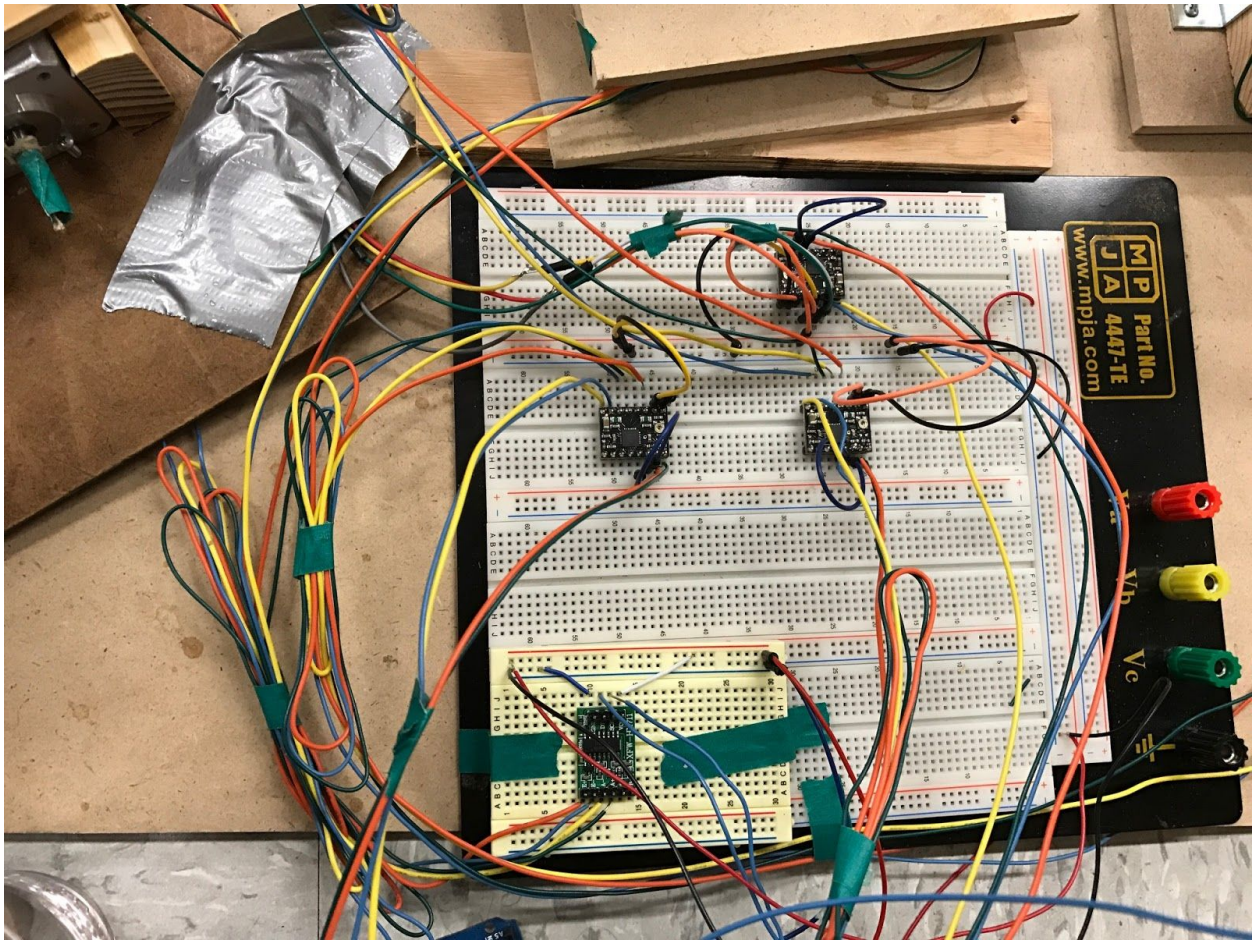
```
      if(request == 49 || request == 50 || request == 51){
       break;
      }
    }
  }
  if(request == 49){
    Serial.println("Request received. One Tequila Sunrise, heading your
way...");
      ing1 = 0.25; //Tequila
      ing2 = 0.66; //Orange Juice
      ing3 = 0.07; //Grenadine
  }

  if(request == 50){
    Serial.println("Request received. One Paloma, coming right up...");
      ing1 = 0.50; //Tequila
      ing2 = 0.50; //Grapefruit Soda

  }

  if(request == 51){
    Serial.println("Request received. Let's see that margarita...");
      ing1 = 0.33; //Tequila
      ing3 = 0.66; //mixer
  }

  /*else{ while(1==1){Serial.println("Ouch, that was the wrong thing. I'm
just going to say this again and again and again.");
        Serial.println(request);}
  }*/

  //Serial.print("Deriv: ");
  //Serial.print(avgnow);
  //Serial.print(" Weight: ");
  //Serial.println(currentval);
 while(done_ing1 == 0 && done_ing2 == 0 && done_ing3 == 0 ){
  pour_ing1();
  }

 if(done_ing1 == 1 && pour_begin2 == 0){
    beginval = scale.get_units(5);
    pour_begin2 = 1;
  }
```

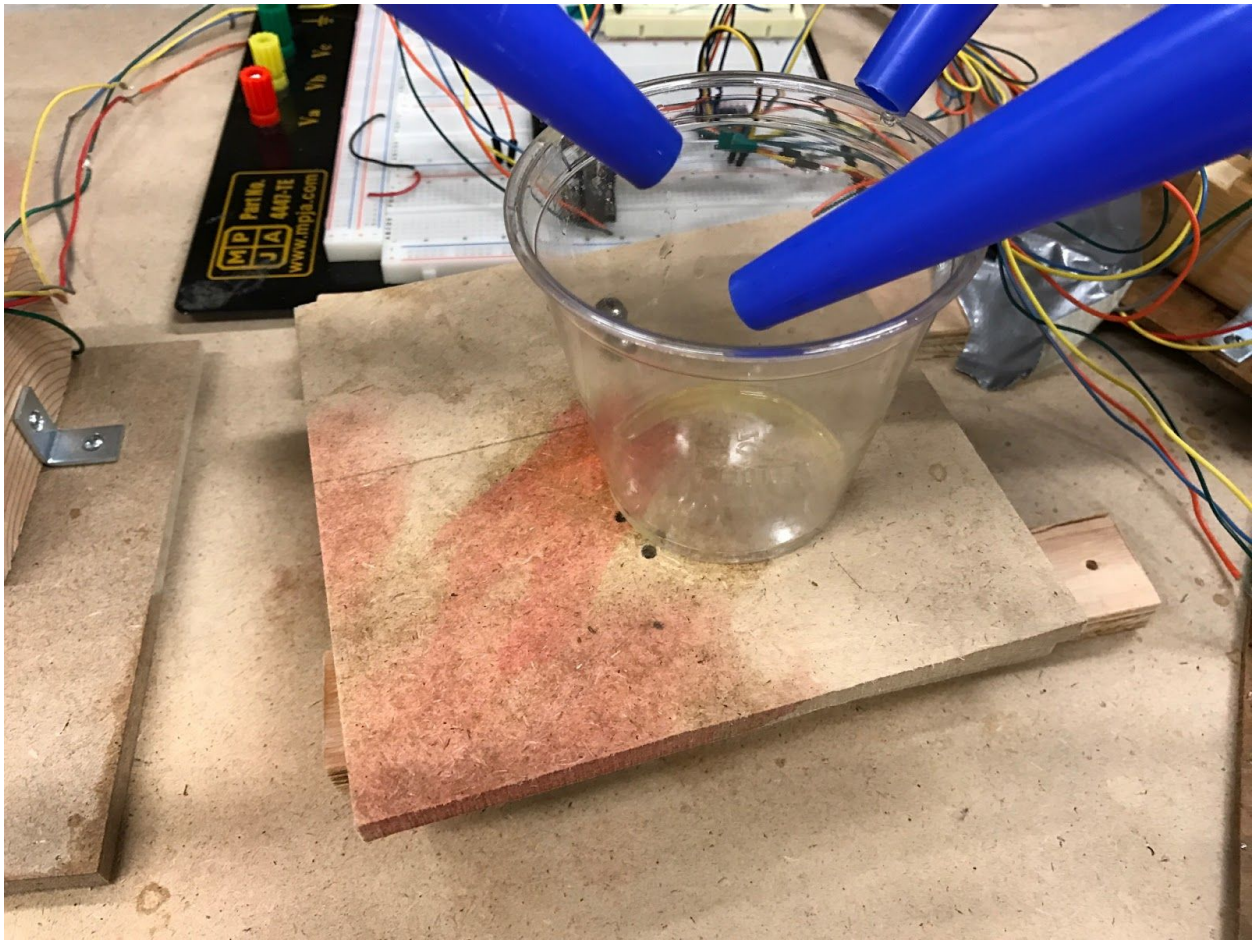
```
  while(done_ing1 == 1 && done_ing2 == 0){
  pour_ing2();
  }

  if(done_ing1 == 1 && done_ing2 == 1 && pour_begin3 == 0){
    beginval = scale.get_units(5);
    pour_begin3 = 1;
  }

  while(done_ing1 == 1 && done_ing2 == 1 && done_ing3 == 0){
  pour_ing3();
  }

  Serial.println("---------");
  Serial.println("Thanks for your business. If you like, you can order
another drink...");
}
```

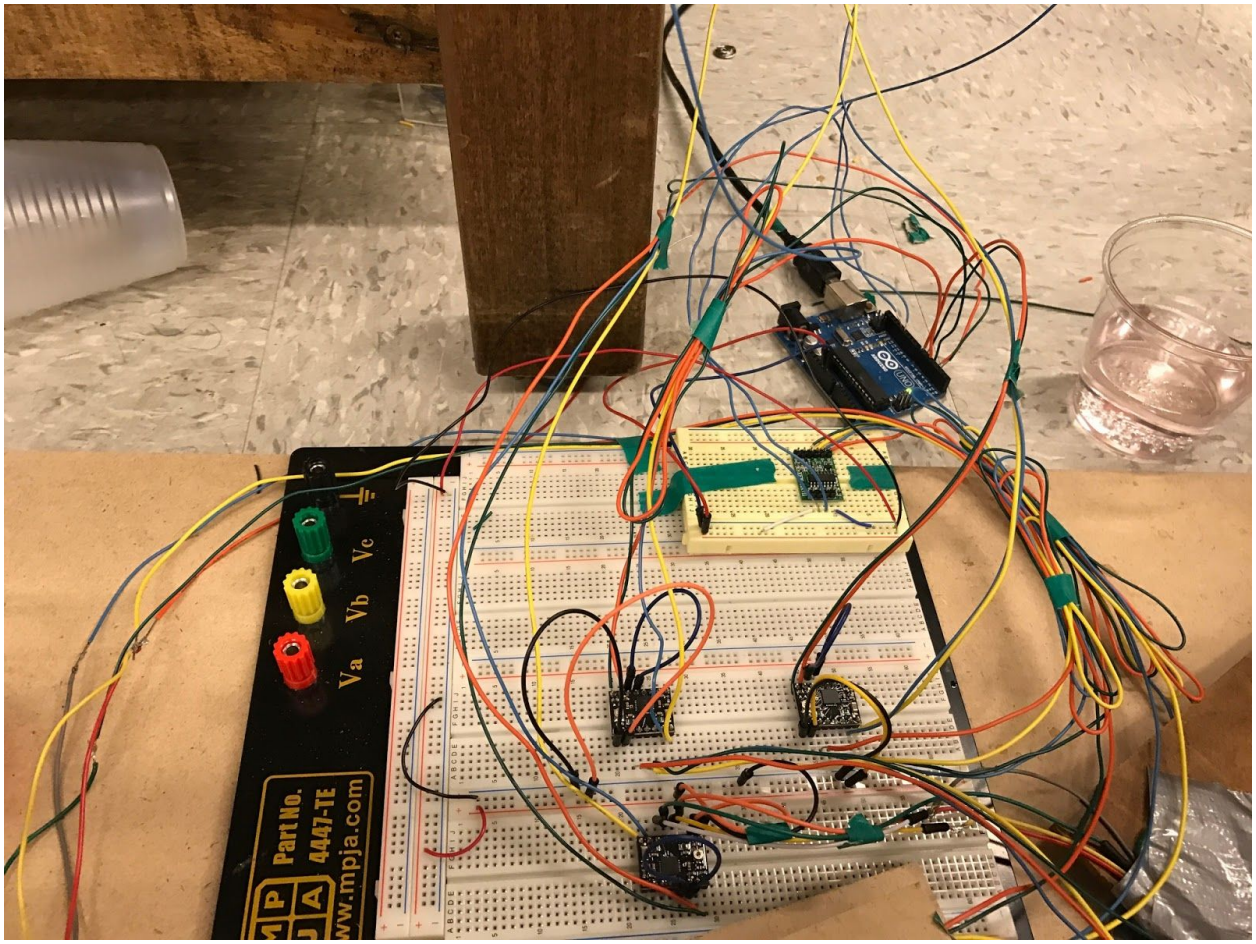
More photos of the set up:

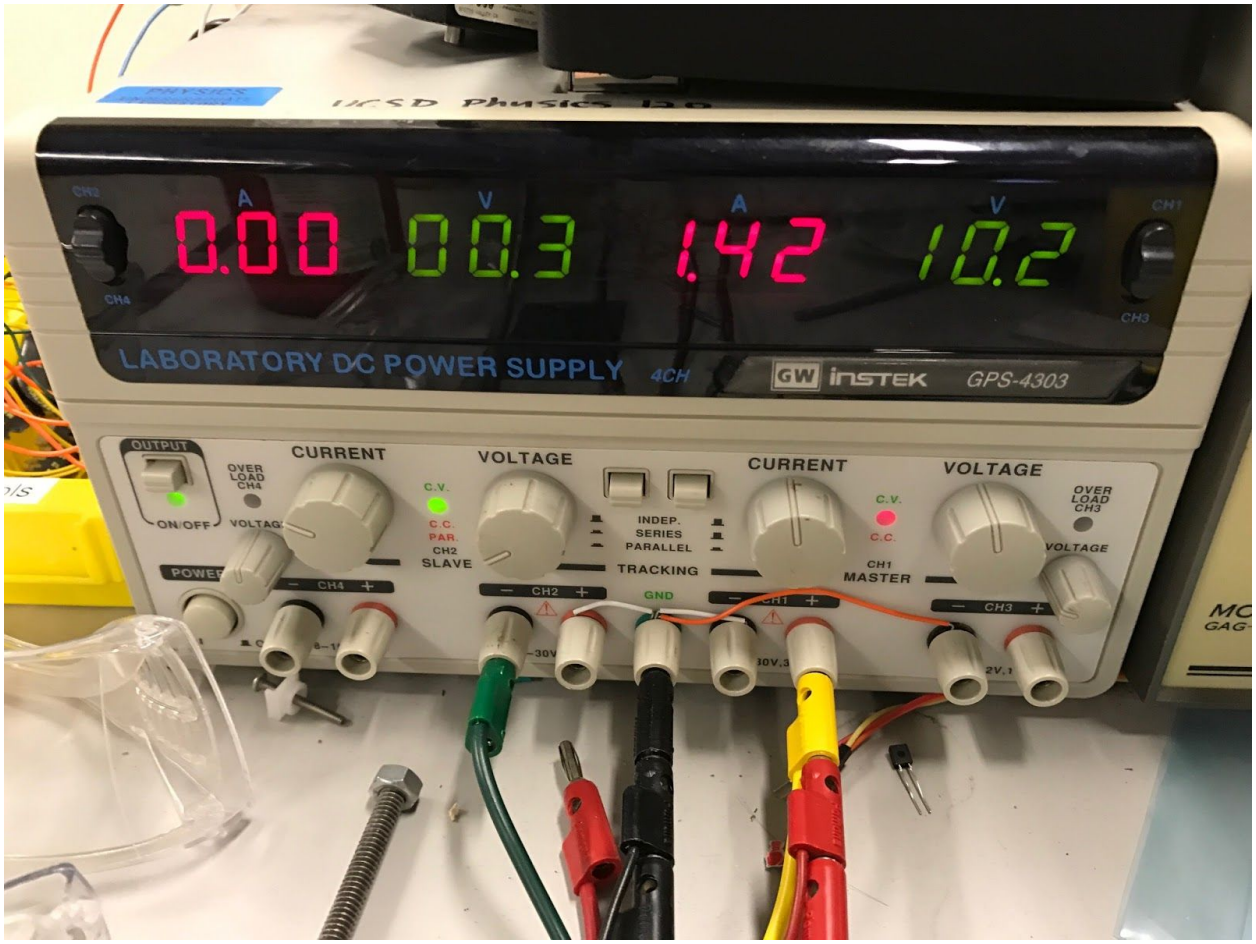
… See next page


nhs
resh fare


MPJA
Part No.
4447-TE
www.mpja.com
Vb
Vc


MPJA
Part No.
4440-rz
www.mpja.com


Vc
Vb
Va
Part No.
4447-TE
www.mpja.com


A
0.00
V
00.3
A
1.42
V
10.2
CH2
CH4
CH1
CH3
LABORATORY DC POWER SUPPLY
4CH
GW INSTEK
GPS-4303
OUTPUT
ON/OFF
OVER LOAD CH4
CURRENT
VOLTAGE
C.V.
C.C.
PAR.
CH2
SLAVE
INDEP.
SERIES
PARALLEL
TRACKING
GND
CURRENT
VOLTAGE
C.V.
C.C.
CH1
MASTER
OVER LOAD CH3
VOLTAGE
POWER
CH4
CH2
CH1
CH3

WIRE JUMPER KIT
JE10
Part No.
6407-TE
www.mcpja.com
Vc
Vb
Va