Pipebot Final Report

Gabrielle Plunket
Lupe Covarrubias
Kantonio Brownlee
Cameron Clark
Jacob Houpy
Dora Rasulova

Table Of Contents

Overview **3**

Tasks Description **3**

System Concept **4**

Design Concept **10**

Overview

A pipette is a biological tool that is used to accurately measure small quantities of liquid to transfer it from one place to another. A pipette is usually controlled manually by a scientist and can be a repetitive process if multiple biological substances are being tested. Our pipetting robot, named Pipebot, is being created to reduce those repetitive motions. Pipebot will use an Arduino kit which will be powered by a battery pack that is attached to a Cartesian robot. A mount is created to hold a pipette in place for an extended period of time. The user will enter a file in G Code that determines the color desired, which is associated with a tip type, and manually turn the top knob of the pipette to choose the volume of liquid. Then, the 3D printed mount will be used to attach a small tip to that pipette that will obtain a specified amount of liquid from a well in a well plate, dispense the liquid into another well on the same well plate, and throw away that tip. To pick up the liquid with the pipette, a button must be pressed at the top of the pipette. The button on the pipette will be controlled by a servo motor with a button pushing effector. To dispense the used tip, a second button must be pressed. A ledge will be mounted at the top of the z-axis so the ledge can push the button that controls the tip and pop it off when the pipette is driven in an upwards motion. A Pixy vision system will allow the robot to know which tip to choose for certain volumes needed, it will scan different colors to decide which pipette tip to acquire. The concepts that will be addressed include vision processing which will be accomplished through the pixy cam and coding, manipulator arm kinematics which will be created from scratch, and coordination which will allow the robot to move from pipette tips to well plate to trash and then be able to repeat this process. Extra parts such as screws, jumper wires, etc. were obtained from the lab.

Tasks Description

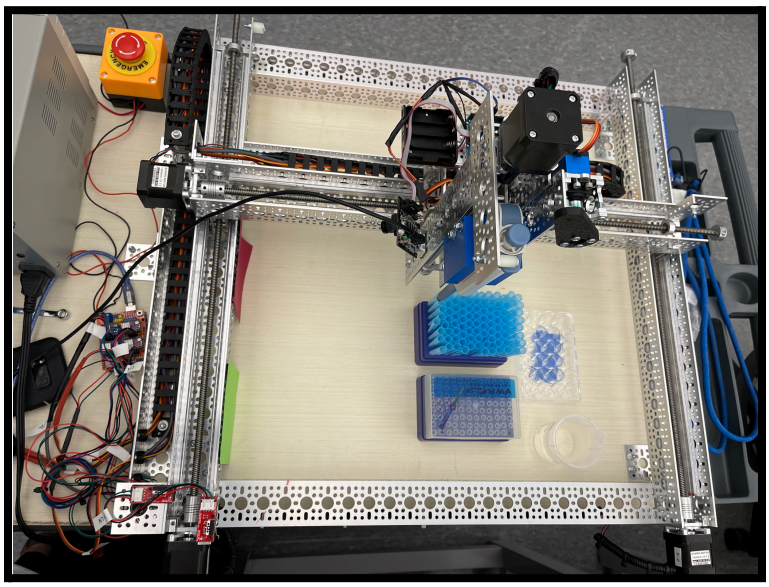


Figure 1. Overview of Entire Pipebot Mechanism

The Pipebot will be responsible for identifying the needed tip for a specified quantity of liquid using a Pixy cam. It will then pick up the correct tip and move toward the liquid in a well plate and draw in the liquid from a push button on top of the pipette which will be driven by a servo motor. The location of the well plate and discard areas will be set in the code as predetermined locations and will not change. The bot will then move to the well plate and inject the amount of liquid it has drawn into a single well. After it has dispensed the liquid, it will then move towards the tip discarding box and discard the tips. Finally, as a precaution, the Pipebot goes through the tip disposal process at the beginning of every execution, whether or not there is a tip on pipette already. Figure 1 is an image of our robot from a bird's eye view showing the main components and props of Pipebot: tips, well plate, pipette, 3D pipette mount, cartesian robot, pixy cam, colored flags detected by the pixy cam, button pushing effector controlled by a servo motor, and ledge to push the button that pops off the tip.

System Concept

The Pipebot will need multiple system components to work in unison to complete multiple tasks. A Pixy camera will be used as a vision sensor to detect different colors between two different pipette tips. A servo motor will be used to press the button on top of the pipette that draws in liquid. A part will be the end effector attached to this motor and it will press the button on the pipette to draw in liquid. We will use an Arduino kit that will connect to a cartesian robot. The cartesian robot is not one from the lab, it was built beforehand. To control the cartesian robot we will use components such as limit switches, stepper motors, lead shaft coupler, dolly kit, CNC shield, motor controller, and Gobilda channels. It will also need batteries for power. The robot as a whole is powered by a TP3005 T model, DC power supply.

Our robot code needs to first take input from the user on what tip it should need for the sample. The tip type is associated with the color card attached to the robot. The pixy cam determines the color, the color determines tip type, and this is all done using G-code. Pipebot's code will then initiate it to move towards the well with liquid and the code will tell it to draw in a preset quantity of liquid using the servo motor to push the button at the top. The servo motor is driven by the arduino kit. Once it has moved upwards away from the liquid, the code will then drive the robot to move towards the well plate. At the well plate, Pipebot is programmed to push the button again via servo motor to release the liquid into one of the wells. Once Pipebot has moved away from the well plate, it is coded to dispose of the tip. It is coded to move towards the tip disposal area; once Pipebot gets to this predetermined location it is programmed to drive upwards and press the button on a static ledge that releases the tip into the discard area. The predetermined locations, that are coded into the program, include: well plate site and discard site. After these steps the program can start again. Our code will need to include the pixy libraries

#include <SPI.h> #include <Pixy.h>. We will be using C and C++ and standard C++ libraries for programming the arduino and pixy cam. Some example pseudo code for how the program would look is as follows:

Pipebot starts at (0,0,0) coordinates

Loop:

User enters G code into terminal to determine color pixie cam stops on

Moves towards discard pile and drops off a tip (predetermined location)

Pixy cam:

if(corresponding color is detected)

change y and z axis of arm to fill the tube

else

throw error

I++ (this will move the location of tip up one)

Move towards well with liquid (predetermined location)

Use servo motor to press button to draw in liquid

Move towards empty well (predetermined location)

Press servo motor to drop off liquid

Move towards discard area (predetermined location)

Drive pipette upwards on z-axis and pop tip off

We will be using an Arduino kit with an Arduino IDE to process our code. User interface will be handled by a terminal in which the user enters quantities desired and precoded positions of tips and the liquid determines how the robot maneuvers.

To determine the precoded positions that will be used, the following G-code was used: G94 Y42

```
G94 Z-1
G94 Z0
G94 Y20
G94 X15.4
G94 Y31.8
G94 Z5
G94 Z0
G94 Y37.5
G94 X25.25
G94 Z5
G94 Z5.5
G94 Z5
G94 Z5.5
G94 Z3
G94 X23.5
G94 Z5
G94 Z0
G94 X0
G94 Y42
G94 Z-1
G94 Z0
G94 Y30.8
G94 X15.4
G94 Z5
G94 Z0
G94 Y39.5
G94 X25.25
G94 Z5
G94 Z5.5
G94 Z5
G94 Z5.5
G94 Z3
G94 X23.5
G94 Z5
G94 Z0
G94 X0
G94 Y42
G94 Z-1
G94 Z0
G94 Y0
```

This G-code is only for the red color flag. The code will start by emptying the tip as a safety measure to make sure that a tip was not on the machine at the beginning of the cycle. It then picks up its first tip from the tip tray and moves to the well plate to extract and move the liquid. The code then proceeds to drop that tip and goes to pick up the tip for the second pass. The second pass moves to the well plate and extracts and moves the liquid again. After this pass, it removes that tip and returns to the original zero position.

To run the pixy and servo for machine, the following code was used:

```
#include <Servo.h>
#include <SPI.h>
#include <Pixy.h>

// the main Pixy object - needs to be declared globally
Pixy pixy;
Servo servo;

void setup()
{
  Serial.begin(9600);
  Serial.println("starting..");

  //Start communications with pixy camera
  pixy.init();
  servo.attach(3);
  servo.write(80);
}
long cnt=0;
void loop()
{
  uint16_t blocks, sig, x, y, width, height;
  char buf[80];

  // grab blocks from pixy.
 //optional getBlocks parameter indicates max blocks you want
 // returned - default is camera max (1000)
  blocks = pixy.getBlocks();

  // If there are detect blocks, process them.
  if (blocks)
  {
```

```
      //Here we'll just process the first block with signature 1 (if there
is one)
      for (int i=0; i<blocks; i++)
      {
          //get parameters for current block
          if(pixy.blocks[i].signature == 1){
            Serial.println("red");
            delay(13000);
            servo.write(95);
            delay(5100);
            servo.write(90);
            delay(150);
            servo.write(85);
            delay(150);
            servo.write(80);
            delay(2000);
            servo.write(95);
            delay(1500);
            servo.write(80);
            delay(10000);
            servo.write(95);
            delay(7500);
            servo.write(90);
            delay(150);
            servo.write(85);
            delay(150);
            servo.write(80);
            delay(2000);
            servo.write(95);
            delay(1500);
            servo.write(80);
            delay(1000);
            exit(0);
          }else if(pixy.blocks[i].signature == 2){
            Serial.println("green");
            delay(15000);
            servo.write(95);
            delay(5100);
            servo.write(90);
```

```
            delay(150);
            servo.write(85);
            delay(150);
            servo.write(80);
            delay(2000);
            servo.write(95);
            delay(1500);
            servo.write(80);
            delay(12000);
            servo.write(95);
            delay(7500);
            servo.write(90);
            delay(150);
            servo.write(85);
            delay(150);
            servo.write(80);
            delay(2000);
            servo.write(95);
            delay(1500);
            servo.write(80);
            delay(1000);
            exit(0);
          }

          sig  =pixy.blocks[i].signature;
          x    =pixy.blocks[i].x;
          y    =pixy.blocks[i].y;
          width=pixy.blocks[i].width;
          height=pixy.blocks[i].height;
          sprintf(buf, " sig: %d", sig);
          Serial.println(buf);
      }
  }
  cnt++;
  delay(100);
}
```

This arduino code is used to sense the color flag that is seen and run servo motor. The motor is first initialized to a position of 80 degrees, which is just above the button to draw and eject liquid. The machine is then run to the position until it sees the red flag and that then triggers the timer for the servo movements. The servo presses the button down to prepare for drawing in the liquid, positioning the servo at 95 degrees. To draw liquid, the servo then steps through the positions 90, 85, and then back to 80 to simulate a human finger drawing the liquid slowly. Then the servo is triggered to eject the liquid after the machine moves to the correct position. If runs this code again for the second tip position as well. There is both a red and green color signature coded for both color flags and the machine can determine between the two flags accurately.

Design Concept

For the cartesian robot we will need a mount that will be able to grab and keep in place a pipette. We will be using a 3D printed mount to hold the pipette in place on the z-axis, shown in Figure 1. The mount was designed in Autodesk Inventor, an angle view is shown in Figure 2. In depth viewings are shown in Figure 3. Figure 3 is a photo of the isometric drawings of each side of the 3D designed printed mount. The printed mount was used instead of a traditional gripper for many reasons. The first reason being that the bending moment on the gripper would cause a lot of wobbliness on the pipette making it less accurate each time it picked up a tip. For aesthetic effect we added the emboss “Pipebot” to the side. It has a concave edge to hold the pipette which is lined with thick double sided tape to keep it in place. There were four holes implemented into the Autodesk Inventor design but they did not print so we had to end up drilling holes through the side, which worked out in our favor. It is 2.0 in x 2.5 in x 2.0 in and printed in the Chevron center’s ABS 3D printer for $36.

Our servo motor powered the end effector that pushed the button that is in charge of drawing in and pumping out liquid.Once the pipette was moved to the correct location the effector arm moved downward onto the button in a motion that was similar to that of a thumb motion and allowed the liquid to be drawn up into the pipette. The pipette then moved into the disposal area and this motion from the effector arm was repeated to dispense the liquid.

A ledge that is mounted on the z-axis was in charge of pushing the second button on top of the pipette that pops off the tips.Once the pipette moved to the correct location for tip disposal the pipette moved upward and the tip ejection button was pressed as it moved into the ledge.

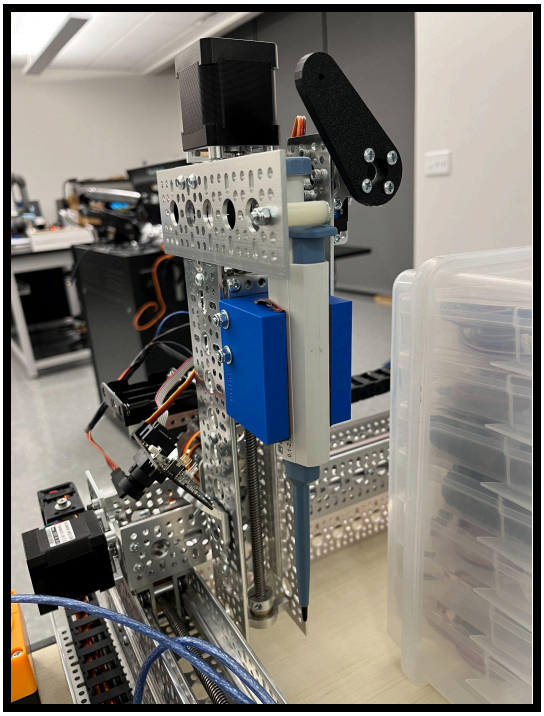


Figure 2. 3D printed pipette mount shown in blue

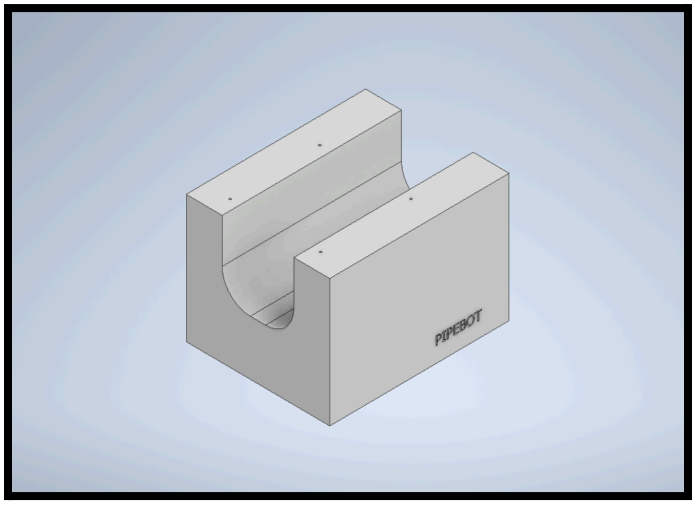


Figure 3. Autodesk Inventor 3D model of Pipebot mount

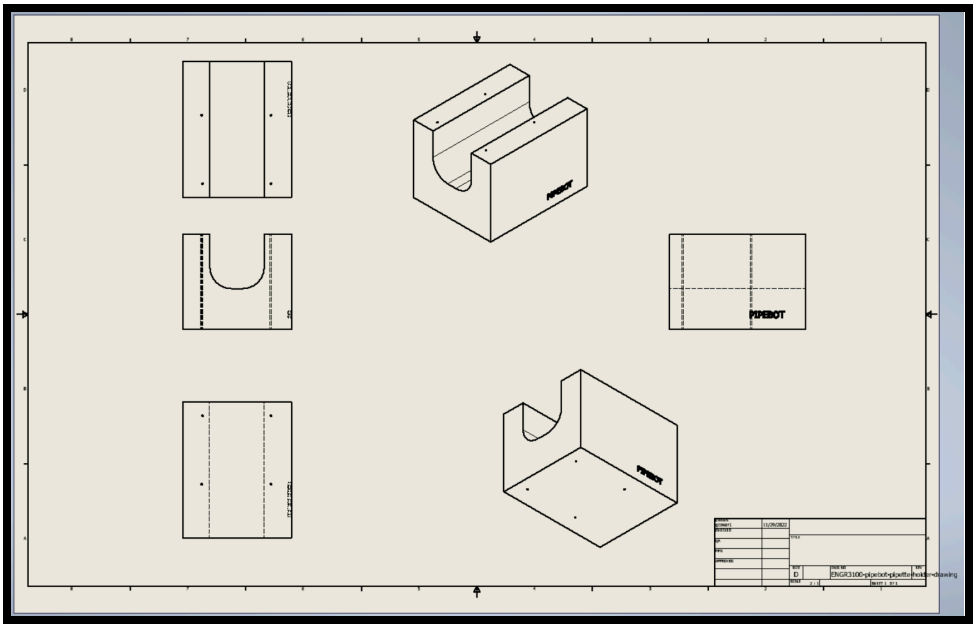


Figure 4. Autodesk Inventor 3D mount model isometric drawings

In the pixy code, we will have the pixy camera recognize different colors and set a signature number to each individual number. The pixy cam will be mounted in a fixed position over the robot and scan for the color where the pipette needs to go. The button at the top of the pipette will be pressed by a 3D printed part that is actuated by a servo motor. It will be powered by an Arduino Uno microcontroller to scan colors and detect the necessary tip size needed, conveyed in Figure 5.

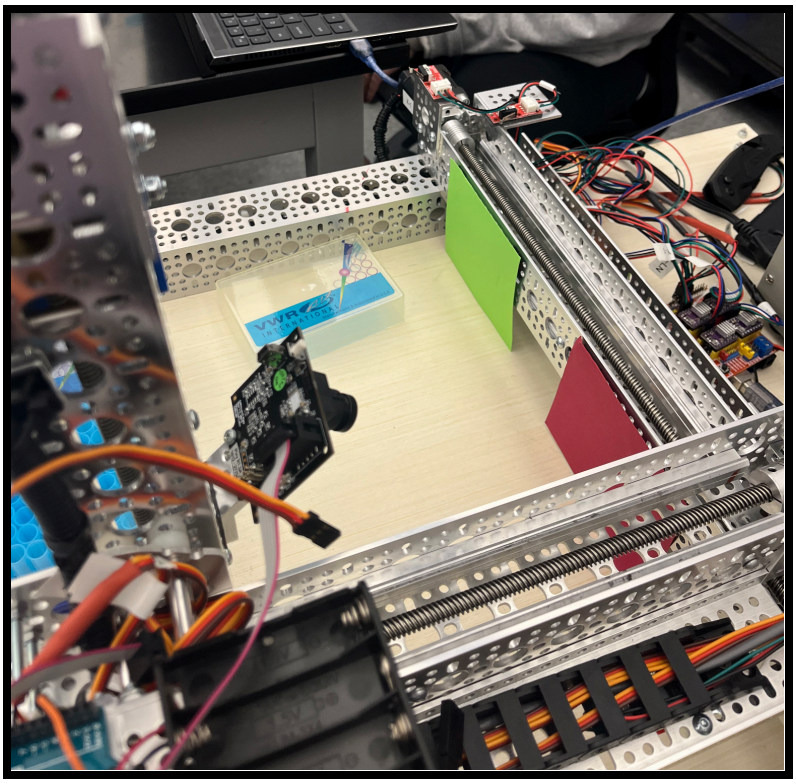


Figure 5. Pixy Camera scanning colors

For the construction of the cartesian robot, we started with a basic design. The team wanted a 3 axis travel, with x, y and z axes. The x and y needed the longest travel so the team decided to use 624 millimeter length channels from Gobilda as a frame from the machine to make it approximately 24 inches square. For the z axis we used a smaller 312 millimeter channel mounted to the traveling part of the y axis. This gave the basic frame of the robot, next we worked on the way that movement would be achieved.

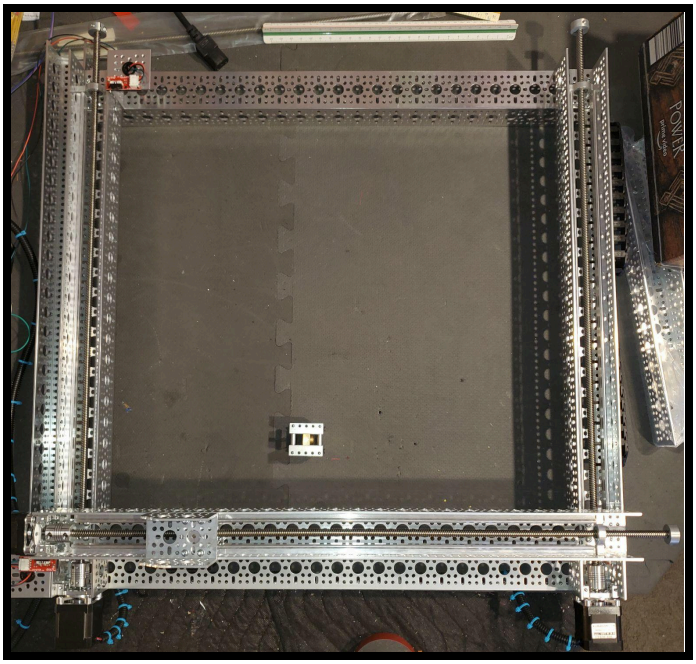


Figure 6. Finished basic frame for the robot

To achieve the movement of the 3 axes, the team decided to use Nema 17 stepper motors. This was because stepper motors allowed for precise positioning as well as reliable and repeatable movement. The reason that the Nema 17 steppers were chosen in particular is that larger steppers would have needed individual power supplies for each motor, thus increasing the cost of the robot.

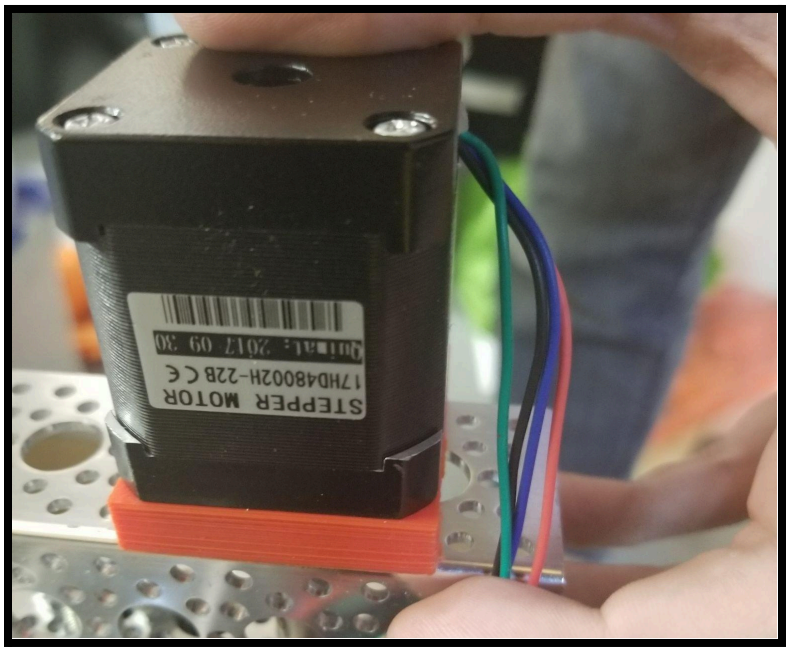


Figure 7. One of the motors used for the robot's movement

To change the rotation movement of the motor into the linear motion desired for the robot, the team decided to use lead screws. The lead screws that we went with were 8 millimeter lead screws for Gobilda. For the x and y axis, a 650 millimeter lead screw was used in order to get the travel required and for the z axis a 300 millimeter lead screw was used. A shaft coupler was used to attach the motor to the lead screw on each axis. The assembly for this is shown in Figure 4.

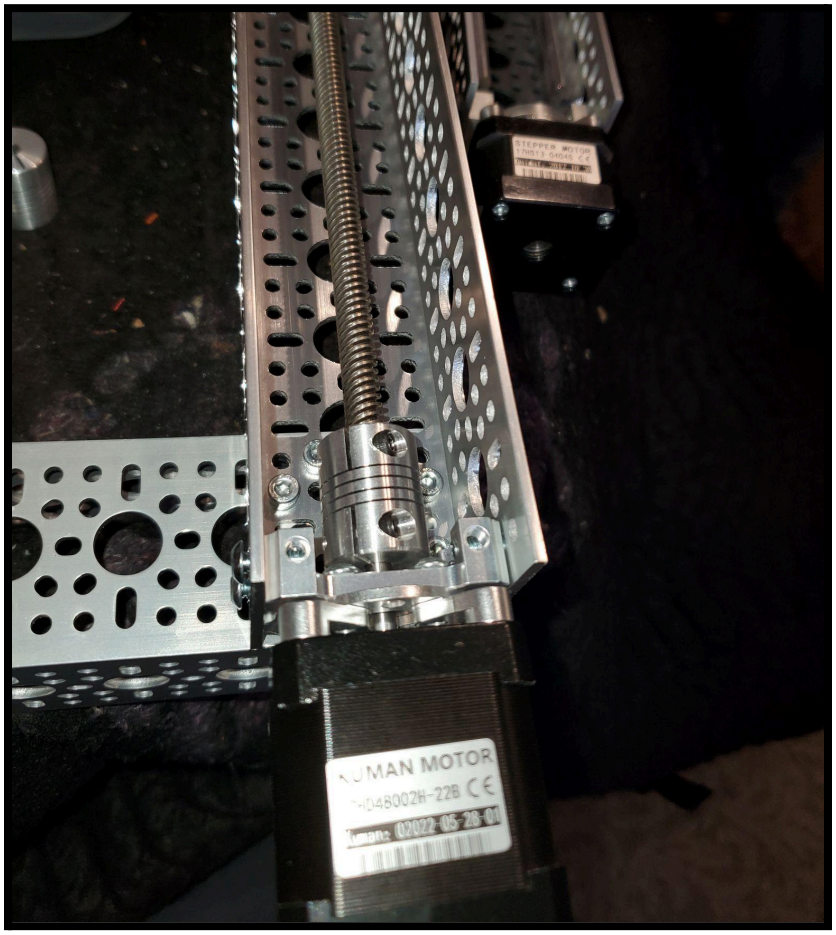


Figure 8. Assembly for rotational motion to linear motion

In addition to the lead screws, we also used the Gobilda v-guide rails cut to size to fit between the motor and a pillow block that holds the lead screws at the end of the channel. The use of the v-guides allowed us to use the Gobilda V-Guide Lead Screw Dolly Kit to achieve a stable and smooth linear motion. The x axis uses two of the dolly kits running in parallel to each other. The y axis rides on the two dolly kits mentioned above, along with having its own dolly kit

that the z axis rides on. And finally, the z axis has a dolly kit that will hold the pipet and move it up or down if needed. In Figure 5 you can see the dooly kits installed for the X and Y axes.

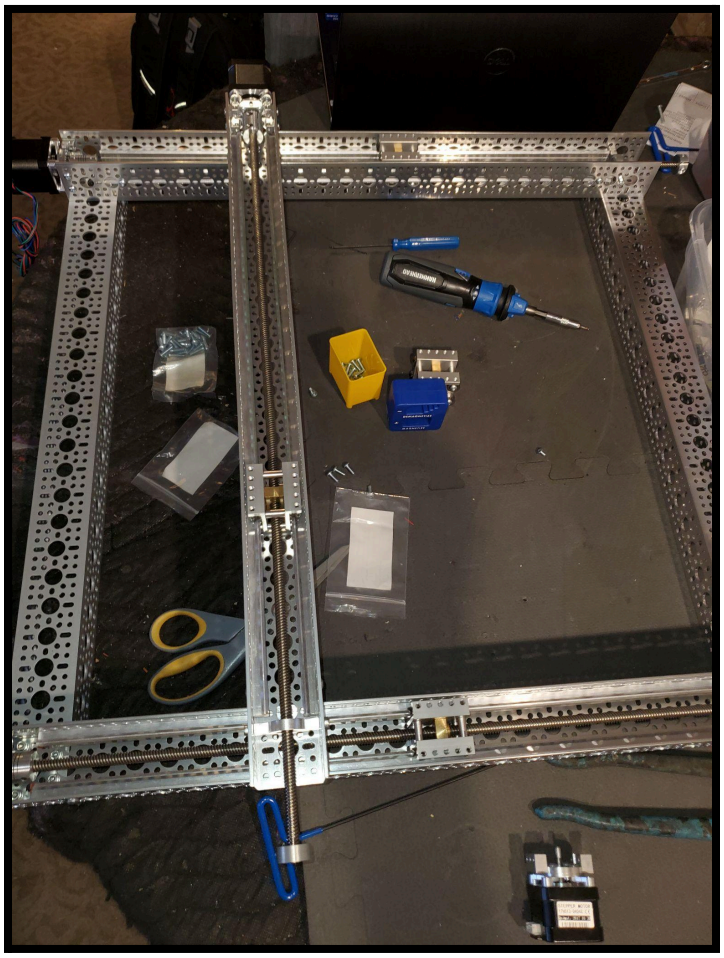


Figure 9. Dooly kits being installed for x and y axis

Next, the team needed to figure out a control system to control the robot. After a little research, the team decided to use an arduino microcontroller with a CNC shield for the control of the stepper motors. The shield has 4 slots to add motor controllers as well as a built- in relay to allow for the external power supply needed to power the 4 motors. The motor controllers that we decided to use were the DRV8824. The reason we chose them is that the DRV8824 motor controllers can take higher current draw and voltages then any of the other motor controllers that the board can handle. Another reason was that the DRV8824 has a potentiometer that allows for the reference voltage (Vref) to be changed, making it better suited for the motors that we were using.

Another reason for choosing the shield is that it allows for a clonable axis. This allows for the two motors on the x axis to run in parallel for the design. The shield also allowed for

microstepping. This can come in handy if more precision is needed as the DRV8824 motor controllers can microstep all the way 1/32 of a step.

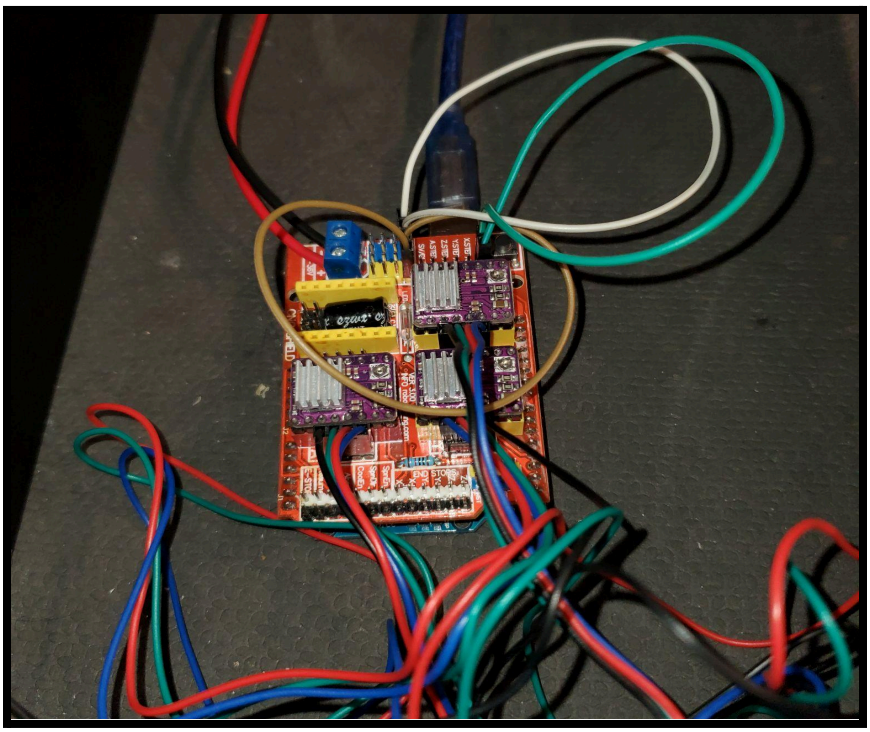


Figure 10. Image of the shield on an Arduino uno

As far as code for the robot, two things are needed. First, an arduino sketch needs to be uploaded in order for the robot to take GRBL code and operate the machine. Next, a file needs to be made in G-code. The G-code will give the robot cartesian points to move to and execute by moving to that point of the 'graph'. A quick test of the X and Y axis of the machine can be seen here: https://youtu.be/DlQrX1cI2SM. As seen, the X axis motors move in parallel and the y axis motor moves the same amount as the X axis.

For sensors on the machine, we have the motors with built-in encoders and the motor controllers that can read them. Each axis also has two limit switches that kill the movement of the machine if it ever overruns the working area. The limit switches can also be used as soft limits for homing the machine. In a cartesian robot, the origin is one of the corners of the machine, in our case the bottom left. So we can move toward the limit switches until the trigger and then back off the switch slightly and then we can be sure that the machine is in the origin position.

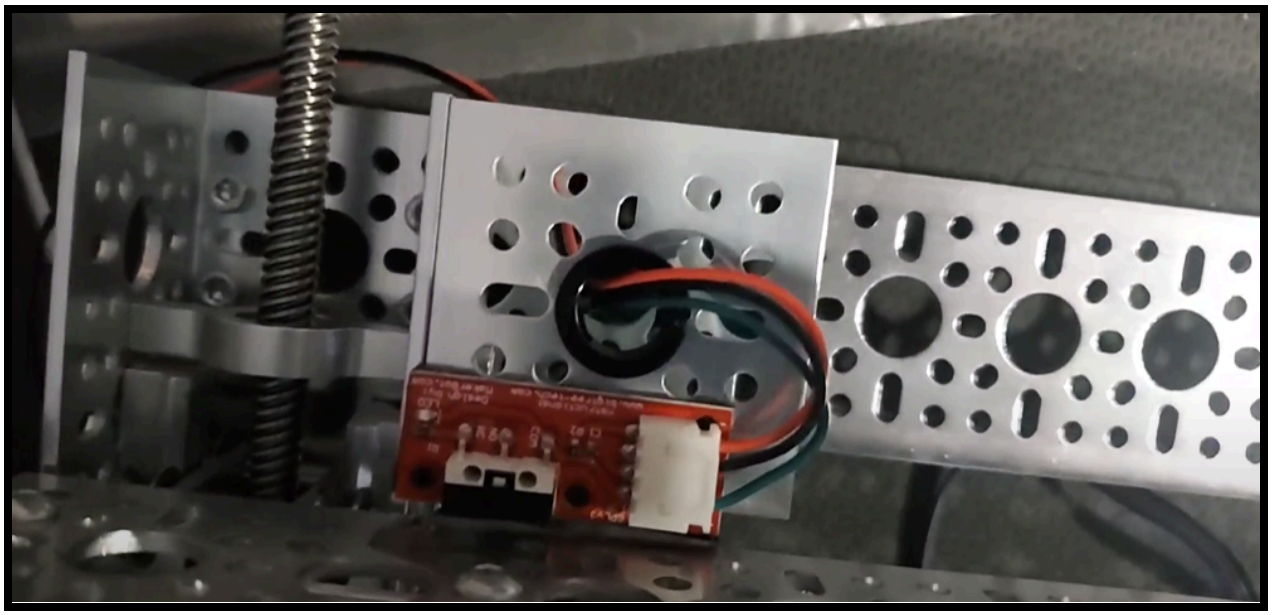


Figure 11. Triggered limit switch

The G-code that will be used will be similar to this example:
G94 X10 Y10
G94 X0 Y0
G94 X20 Y20
G94 X0 Y0
G94 X30 Y30
G94 X0 Y0
G94 X40 Y40
G94 X0 Y0
G94 X47 Y47
G94 X0 Y0

The G94 code will allow the machine to move to the desired position in the x, y, z grid. The code can have additional codes G04 to add a delay to the movement of the machine. The machine can function in imperial inches or mG metric millimeters. This change can be made by using the G20 code for imperial and G21 for metric. The machine will mostly function in the absolute mode; however, it can function in an incremental mode, where each movement adds onto or subtracts from the previous position. This is controlled by the G91 command to switch in incremental commands and G90 to switch the absolute command. Another useful command is the G28 command that will make the machine return to its last set origin. This return would go back to the machine zero; however in order to go to absolute zero, you would have to home the machine using the limit switches on each axis. The G94 command denotes that we want to use linear motion, which is why it is used in the example above to control the movement in the x and y axes.

The overall, important design components consist of: pixy camera for vision processing, cartesian robot for simple movement, 3D printed pipette mount for secure and durable placement of pipette, and servo motor to control the button pushing on top of the pipette, which is very important for drawing liquid in and pushing it out. The entire design of our system is laid out in Figure 12.

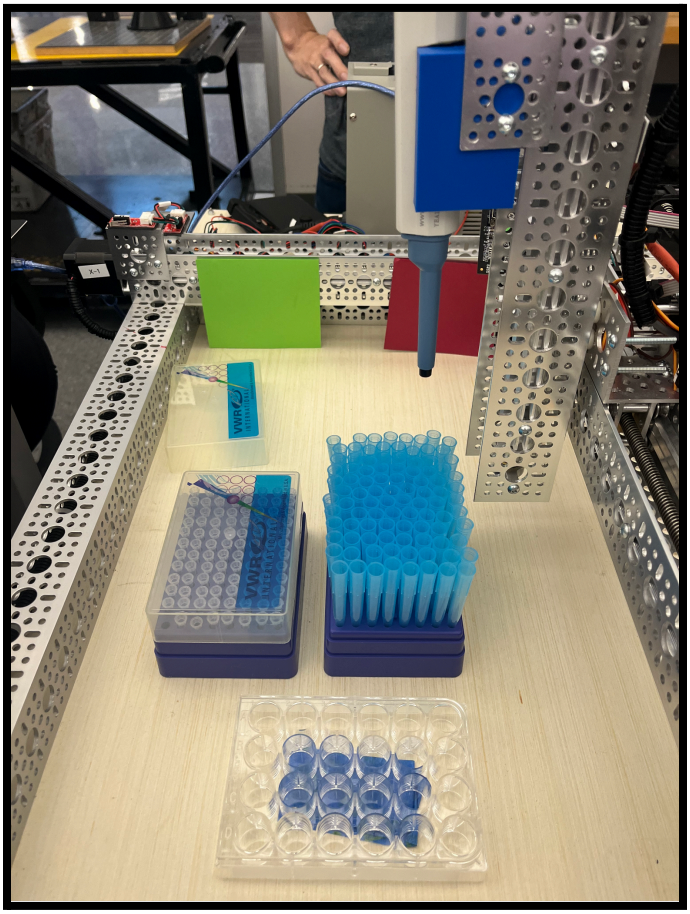


Figure 12. Design in action, picking up a tip