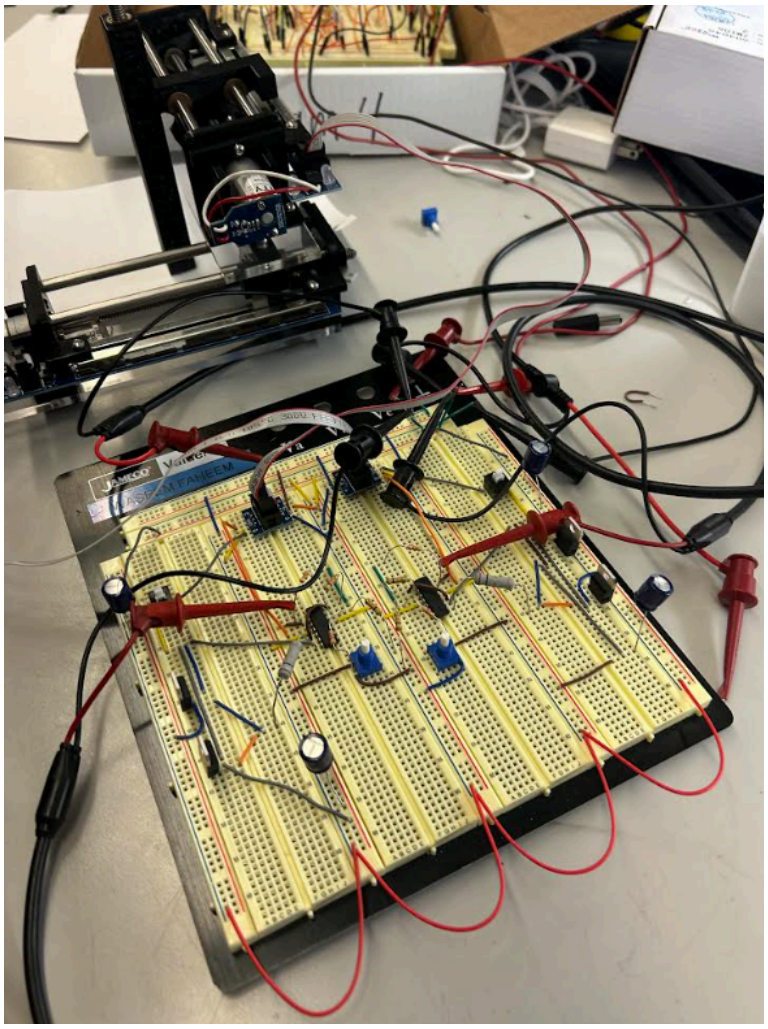
Krishna and Vincent
LAB 10
12/6/2024

PART A:

1. Summarize your measurement procedure and your observations in one or two paragraphs and include a picture of your setup.



First, the circuit is setup as shown. Then, with the probes on the resistors that go into the resistors or the , and all the black probes grounded, with two probes providing power to the rails (in the back), the voltages V_x, $V_{x,set}$, V_y, and $V_{y,set}$ are all measured. Then, using the potentiometers the $V_{x,set}$ and $V_{y,set,}$ voltages are set to 0 V, and then the switches are pressed, and the resulting position of the plotters was shown. Then, with adjusting the values of $V_{x,set}$ and $V_{y,set}$ to different values, the resulting V_x and V_y values were recorded. One thing we observed was some friction with the plotter, so if small changes were applied to $V_{x,set}$ or $V_{y,set}$ at a time, then sometimes the plotter would move negligibly or very little.

2. For each feedback loop (for top axis and bottom axis) calculate loop gain K_{loop}. You may calculate $K_{V\omega}$ from your DC motor characterization (Lab 8). You may assume a gear ratio of G = 40 for the gearbox, a thread pitch of 1.6mm for the leadscrew (to be used to

calculate $K_{\omega X}$) and a linear potentiometer gain of K_{VX}=1 V/cm. Also assume that the motor driver gain is approximately one. In this case, K is set by the gain of the difference amplifier in Figure 4.

First, for the top loop, we use the equation as given: $K_{loop} = \frac{K_{VX}K_{\omega x}K_{V\omega}K}{G}$. We have that G=40, $K_{V\omega} = 1\,V/cm$, and since we have a thread pitch of 1.6 mm = 0.16 cm, then we get $K_{\omega x} = \frac{0.16}{2\pi} = 0.0255$ cm/rad.

Next, we have $K_{V\omega}$ =3.2 RPM=0.335 rad/s, from Lab 8, where we observed that as V increases by 1 V, then the angular velocity increases by 0.335 rad/s.

Finally, our K is calculated from the gain of the difference amplifier. This is a classic configuration, and what we see that if the inverting input is v_1, and the noninverting input is v_2, then the result is $\frac{51.7\,k\Omega}{4.7\,k\Omega} * \frac{47k\Omega}{51.7\,k\Omega}v_2 - \frac{47\,k\Omega}{4.7\,k\Omega}v_1 = \frac{47\,k\Omega}{4.7\,k\Omega}(v_2 - v_1)$, so we get $K = \frac{47}{4} = 10$.

So, putting this all together we get $\frac{1*0.0255*0.355*10}{40} = 0.00226 = K_{loop}$

For the other loop, notice that as it is symmetric, and the difference op-amp is set up the same so we will have the gain, we also see that bottom loop will have the same value of $K_{loop} = 0.00226$, so both loops have the same loop gain.

3. Control top and bottom axis using the two trimmers. Set $V_{X,set}$ and $V_{Y,set}$ according to the Table 1 below. After both sliders move to the corresponding positions and stop, note down corresponding V_X and V_Y in Table1. Do you see a difference between the $V_{X,set}$ and V_X? Why? Do you see a difference between the $V_{Y,set}$ and V_Y? Why?

Overall, we see small differences in V_X, and $V_{X,set,}$ but usually on the magnitude of only around 0.1 or 0.2 V. Similarly, the differences between V_Y, and $V_{Y,set}$ are larger, at most 0.6 V, but still small. Most of these differences can be attributed to the motor friction, or the way it was adjusted. When making small changes to $V_{X,set}$ or $V_{Y,set}$, oftentimes this would lead to much smaller changes in the motor, and it generally worked better when $V_{X,set}$ and $V_{Y,set}$ were changed with large swings, to get closer. Nevertheless, we see that they are approximately the same, except for some differences that can be attributed to the motors themselves and how the plotter works.

$V_{X,set}$ (V)	0	0.6	-0.6	1.5	-1.5	3	-3	4.5	-4.5	6	-6
V_X	-0.12	0.5	-0.6	1.4	-1.6	2.8	-3.2	4.3	-4.4	5.9	-6.1

(V)											
$V_{Y,set}$ (V)	0	0.6	-0.6	1.5	-1.5	3	-3	4.5	-4.5	6	-6
V_Y (V)	0.4	0.8	-0.3	1.5	-1.2	3.3	-2.7	4.7	-4	6.3	-5.4

4. Why do the motors slow down as the sliders get close to the positions set by the set voltages?

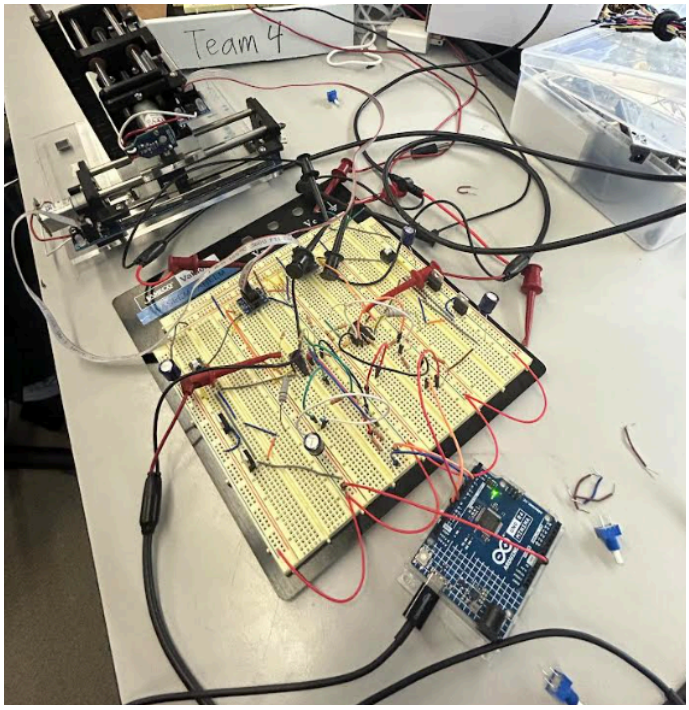
In the context of the closed-loop control test for the analog plotter conducted in this lab, the motors slow down as the sliders approach their target positions due to the reduction in the error voltage (V_e). The error voltage is defined as the difference between the set voltage ($V_{X,\text{set}}$ or $V_{Y,\text{set}}$) and the measured voltage (V_X or V_Y) from the linear potentiometers that monitor the pen's position along the respective axes. As the slider moves closer to its target, the measured voltage approaches the set voltage, thereby decreasing the error voltage (V_e). For instance, if $V_{X,\text{set}} = 3\,\text{V}$, a large error voltage ($V_e = 3\,\text{V}$) occurs when $V_X = 0\,\text{V}$. However, as the slider moves closer, say $V_X = 2.8\,\text{V}$, the error voltage diminishes to $V_e = 0.2\,\text{V}$.

This decreasing error voltage reduces the voltage applied to the motor, as the motor voltage is proportional to the error voltage. Consequently, the motor's speed decreases. When the error voltage eventually reaches zero, the motor voltage becomes zero, causing the motor to stop entirely. This behavior exemplifies the advantages of closed-loop control, as it naturally prevents overshooting and ensures smooth and precise positioning of the sliders without requiring additional braking mechanisms.

Krishna and Vincent
LAB 10
12/6/2024

PART B:

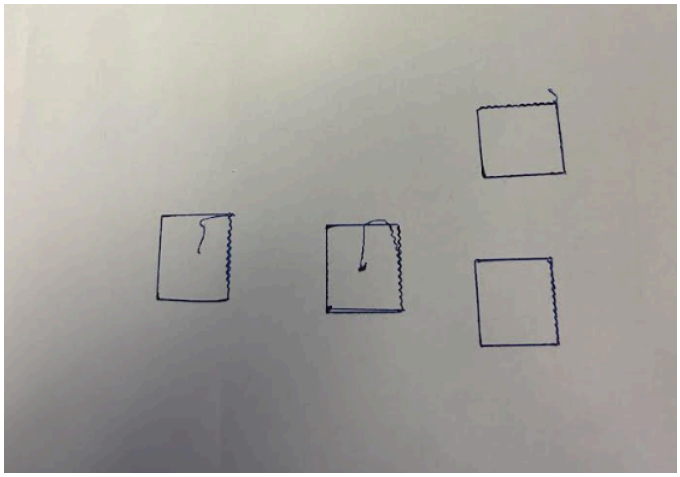
1. Summarize your measurement procedure and your observations in one or two paragraphs and include a picture of your setup.



The setup was updated from part one, including the difference analogs, and then the inputs set up with the arduino. 5 and 6 were sent to the difference op-amp, with 5 to the noninverting input, and 6 to the inverting input. Similarly, 10 went to the noninverting input, and 11 went to the inverting input to work with the difference amplifiers.

Krishna and Vincent
LAB 10
12/6/2024

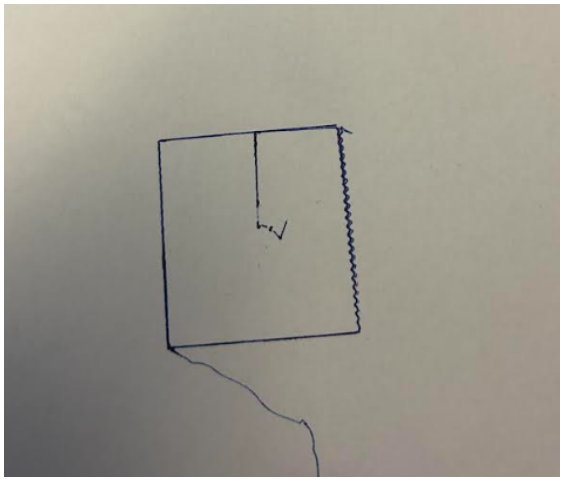
2. Include a picture of the square plotted by the analog plotter. How does this shape compare with the one plotted using the open loop control in Lab 9? Does it look more accurate? Are the corners round or sharp?

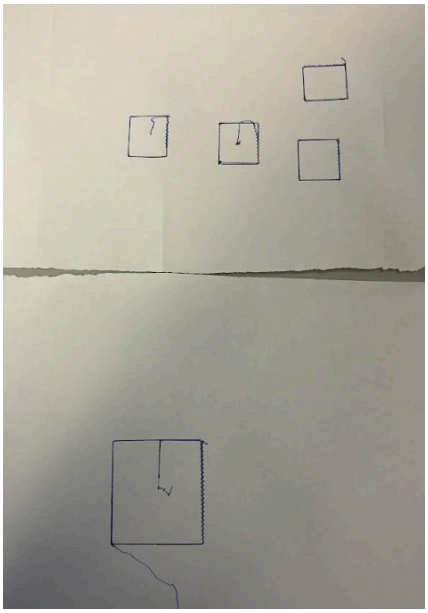


Overall, this square had sharp corners, much sharper than the open loop control square plotted. That one had more rounded corners, and looked more circular in general. This one, apart from the natural squiggle in one direction that was likely caused by the plotter itself's contact being somewhat shaky, looked much more accurate and precise than the other one.

3. How would you double the size of the square? Explain how you chose the PWM voltage and any delay values to achieve this shape. Make sure you consider enough time for the plotter to complete the side of the square. Include a picture of plotted large square in your report along with the modified Arduino code.

Pictures of the larger squares and a comparison photo:





Krishna and Vincent
LAB 10
12/6/2024

The code initially set up two differences: One is with 6/11 to HIGH and then 5/10 to 192, and since high represents 256, then the difference will be -64. The other case is 6/11 set to LOW and then 5/10 to 64, representing a difference of 64, as LOW is 0. So, to double the size of the square, we modified the code to double the differences. Thus, the only change was replacing the 192 to 128, and the 64 to 128. Then, this makes the differences of 128-256=-128, and 128-0=128, so both differences will be doubled, and thus the square will have double the side length. Finally, we just chose a delay of 5000 (5 sec), just to ensure that the plotter would have had enough time to draw each side of the square before the next part of the code ran. This was verified by testing that 5000 was a suitable delay.

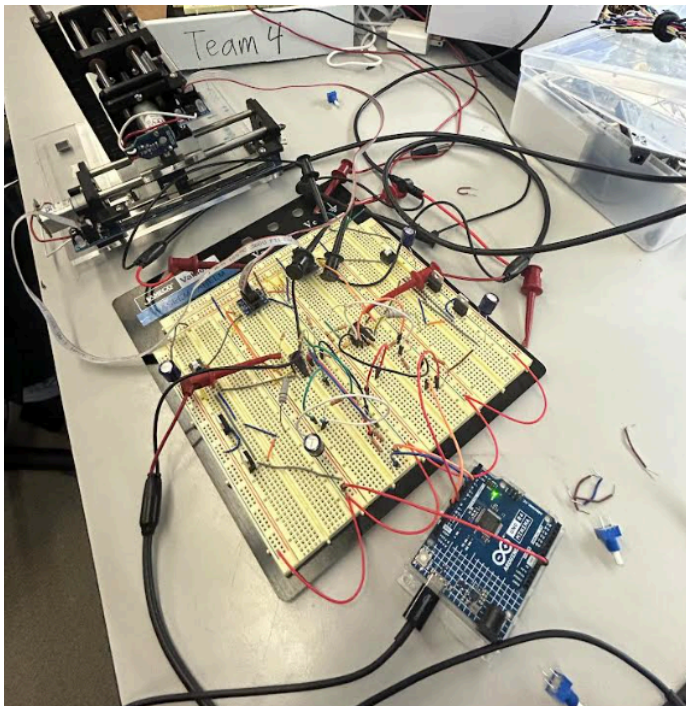
Modified code pasted:

```
/* Setup alternating movement,
by giving enough delay and only changing
the set voltage for one motor at a time.
DOUBLED SQUARE:
*/
void loop() {
//Vy(set) = -6V
  digitalWrite(6, HIGH);
  analogWrite(5, 128);
  delay(5000);
//Vx(set) = -6V
  digitalWrite(11, HIGH);
  analogWrite(10, 128);
  delay(5000);
//Vy(set) = 6V
  digitalWrite(6, LOW);
  analogWrite(5, 128);
  delay(5000);
//Vx(set)=6 V
  digitalWrite(11, LOW);
  analogWrite(10, 128);
  delay(5000);
}
```

Krishna and Vincent
LAB 10
12/6/2024

PART C:

1. Summarize your measurement procedure and your observations in one or two paragraphs and include a picture of your setup.



In Part C, we extended the closed-loop control system for the analog plotter to draw increasingly complex shapes, transitioning systematically from a square to a triangle and finally to a custom intricate design. The system was built upon the principles established in Part B, where PWM signals generated by the Arduino were converted into DC voltages using low-pass RC filters, serving as inputs for the X and Y axis feedback loops. These voltages controlled the precise position of the plotter via error signals (V_e) that minimized positional deviations.

Initially, the Arduino code was designed to draw a square by alternating motor movements along each axis with fixed set voltages $(V_{X,\text{set}}$ and $V_{Y,\text{set}})$. The square was achieved through sequential movements, where each motor's set voltage was toggled between positive and negative values, ensuring precise orthogonal transitions. The delay between movements allowed the plotter to stabilize at each vertex before proceeding to the next. By doubling the set voltage amplitudes in the code, the side length of the square was doubled, demonstrating the flexibility of the control system.

The transition to the triangle design required modifications in the Arduino code to introduce diagonal movements. By simultaneously setting the voltages for both axes, the system enabled diagonal interpolation, allowing smooth transitions along non-orthogonal paths. This refinement involved careful synchronization of PWM signals to ensure that the plotter accurately traced the three sides of the triangle.

Krishna and Vincent
LAB 10
12/6/2024

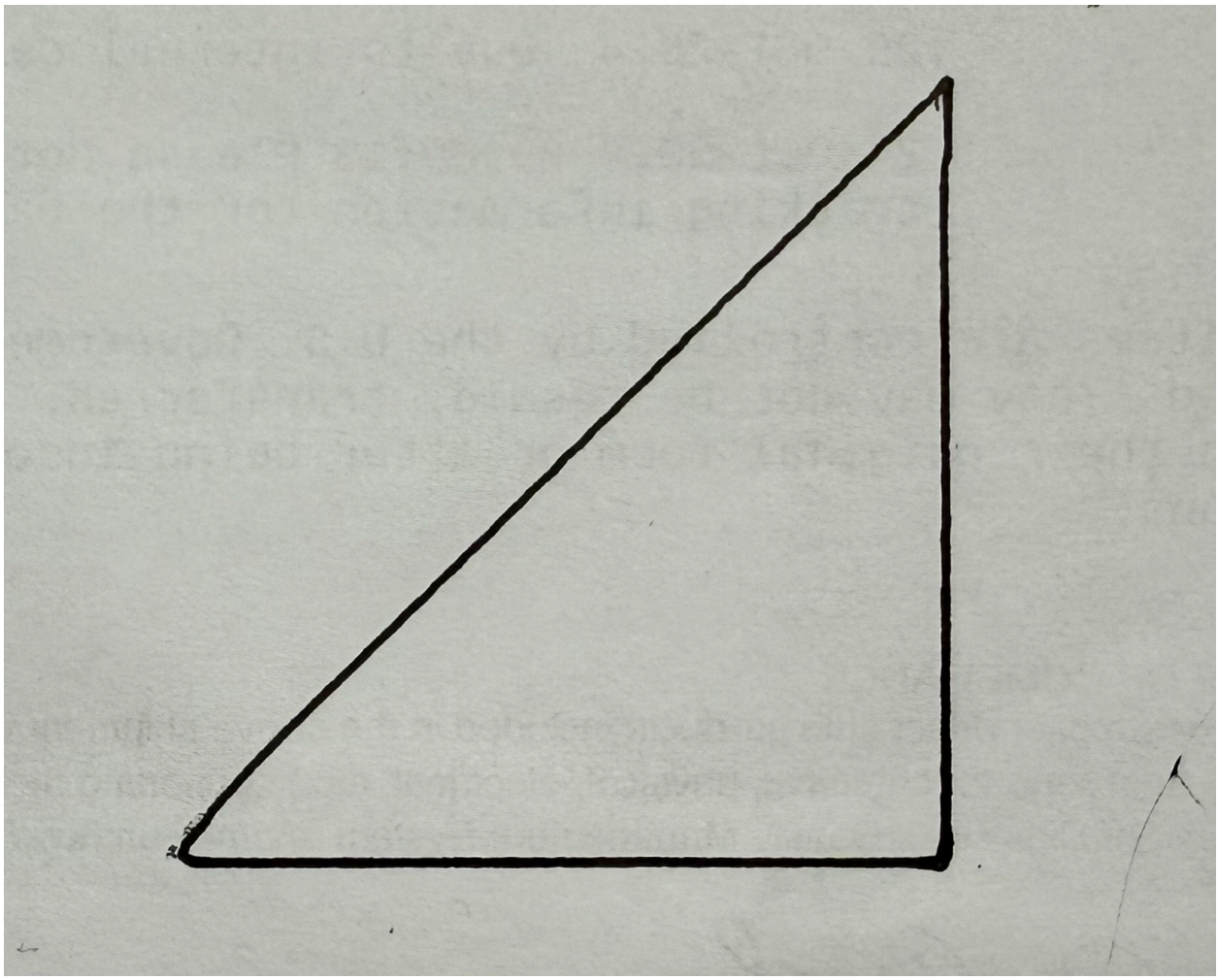
For the custom shape, the system was further enhanced with a coordinate-based approach. Each point of the design was represented as a target coordinate pair, and the Arduino computed incremental movements using a `moveTo` function. This function interpolated the trajectory into small steps, ensuring smooth and precise diagonal paths. Advanced geometric calculations, such as those for semicircles, were introduced to enable the plotter to trace curved and intricate paths. This allowed for the creation of complex designs like a fish outline with detailed features.

We found that using a different plotting structure for diagonals was best suited for our needs, since we needed to account for the fact that the motors would rotate at the same speed for all diagonals.

Overall, the development of the code from simple shapes to complex designs demonstrated the robustness of the closed-loop system. The approach highlighted the importance of incremental refinement in both hardware and software, ensuring accurate and smooth operation of the analog plotter across a range of applications.

2. Include a picture of the Triangle plotted by the analog plotter. Include your code in your report. Add comments for different lines of your code.

Krishna and Vincent
LAB 10
12/6/2024



Triangle Code:

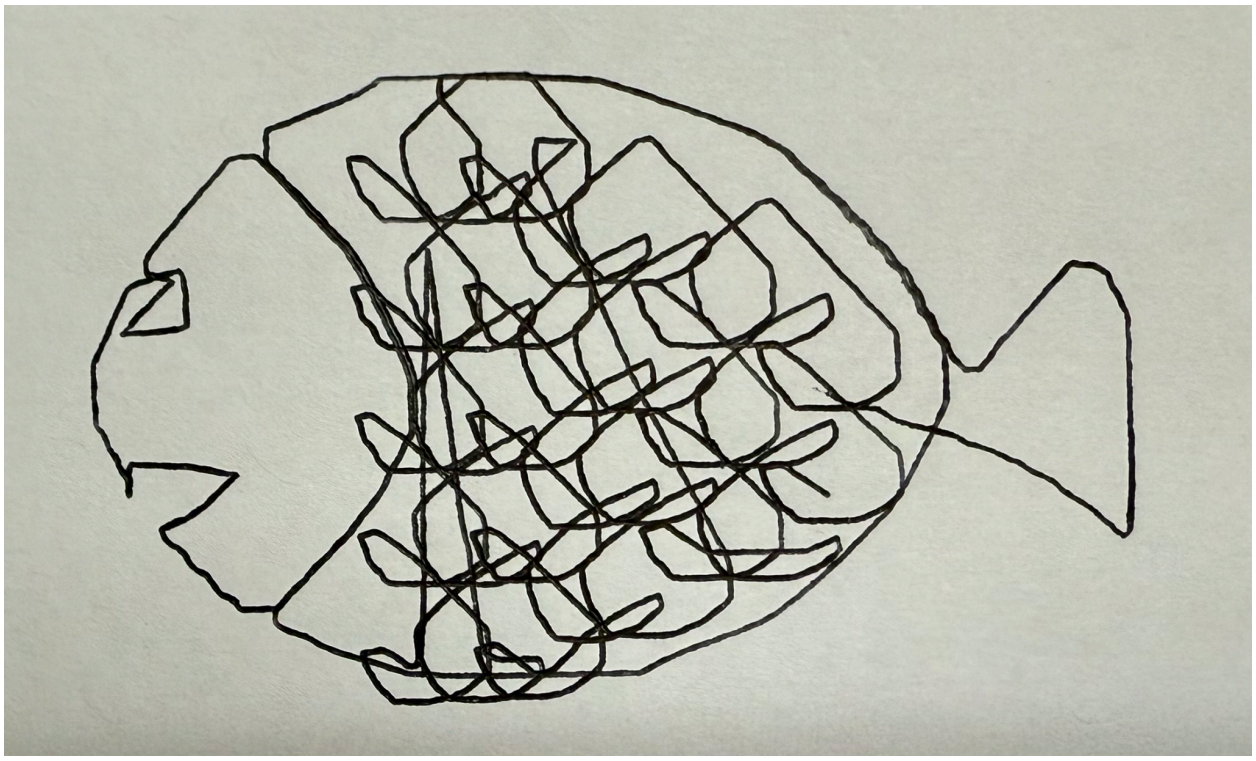
```
void setup() {
//bottom motor
  pinMode(5, OUTPUT); //pwm
  pinMode(6, OUTPUT); //digital = dir
//top motor
  pinMode(10, OUTPUT); //pwm
  pinMode(11, OUTPUT); //digital = dir
}


/* Setup alternating movement,
by giving enough delay and only changing
the set voltage for one motor at a time.
```

Krishna and Vincent
LAB 10
12/6/2024

```
TRIANGLE:
*/
void loop() {
// Move along the negative Y direction
  digitalWrite(6, HIGH);
  analogWrite(5, 128);
// Move along the negative X direction
  digitalWrite(11, HIGH);
  analogWrite(10, 128);
  delay(5000);
// Move diagonally to the positive Y direction and negative X direction
  digitalWrite(6, HIGH);
  analogWrite(5, 128);
  digitalWrite(11, LOW);
  analogWrite(10, 128);
  delay(5000);
// Move along the positive X direction
  digitalWrite(6, LOW);
  analogWrite(5, 128);
  digitalWrite(11, LOW);
  analogWrite(10, 128);
  delay(5000);
}
```

Krishna and Vincent
LAB 10
12/6/2024

3. Include a picture of other shape(s) you plotted using your modified code. At least one shape must be in your report. Add comments for different lines of your code. How did you convert the shape to the Arduino code in the design phase?



When converting the fish design into Arduino code, we began by creating a structured framework for motor control. To achieve this, we defined a struct, ArduinoValues, to encapsulate motor attributes such as speed (speedSignal) and direction (polarity). This allowed us to describe every motor movement explicitly in terms of its control parameters. To track the plotter's position, we introduced global variables, currentX and currentY, which ensured that every movement started from the most recently reached position.

We developed the computeMotorSignals function to interpret coordinates as motor voltages. Each coordinate, representing either the X or Y axis, was scaled to fit within the range of $-5\,\mathrm{V}$ to $+5\,\mathrm{V}$, which was then converted into a corresponding PWM value for motor speed control. This function ensured that each coordinate directly corresponded to a specific motor action. To send these motor signals to the X and Y motors, we implemented the sendMotorSignals function, which used the output of computeMotorSignals to drive the motors with the appropriate polarity and speed.

Krishna and Vincent
LAB 10
12/6/2024

To facilitate smooth transitions between points, we created the moveTo function, which handled both straight and diagonal movements. For diagonal movements, we calculated the distance between the current position and the target point and divided the path into small increments. This interpolation ensured that the plotter moved smoothly along the calculated trajectory without abrupt jumps. For each step, we updated the motor signals and added a delay to maintain stability. After reaching the target point, we updated the current position variables, ensuring continuity in subsequent movements.

When translating the fish shape into code, we broke the design into smaller geometric segments, such as the tail, body, face, and gills. Each segment was represented as a series of target coordinates, and we used the moveTo function to trace these paths. For instance, the tail was drawn by sequentially moving to points such as $(6, 0)$, $(9, -3)$, and back to $(6, 0)$. This modular approach allowed us to systematically convert the shape into manageable sections.

To incorporate curved features, such as the semicircles in the gills, we developed the drawSemicircle function. Using trigonometric calculations, we divided the arc into equal segments and calculated intermediate points along the curve. The function then used `moveTo` to trace the semicircle smoothly. By ensuring precise division of the arc and accurate endpoint alignment, we were able to replicate the curved features of the fish design.

Finally, we integrated all the components into the drawDesign function, which orchestrated the movements for the entire fish shape. This function combined straight paths and curved trajectories to produce a cohesive design. The Arduino's loop function executed drawDesign, ensuring that the plotter followed the sequence to complete the fish. By combining mathematical modeling, geometric segmentation, and careful interpolation, we successfully translated the fish design into Arduino code, enabling the plotter to replicate the intended shape with precision.

Custom Shape (Fish) code:

```
/*
Analog Plotter: Fish Design
*/

// Struct to define motor control attributes
struct ArduinoValues {
    byte speedSignal;  // PWM signal for motor speed
    bool polarity;     // Direction - false for forward, true for reverse
};
```

Krishna and Vincent
LAB 10
12/6/2024

```
// Global variables to track the plotter's position
float currentX = 0.0;
float currentY = 0.0;

// Setup function for pin initialization
void setup() {
    pinMode(5, OUTPUT);  // Y-axis motor PWM pin
    pinMode(6, OUTPUT);  // Y-axis motor direction pin
    pinMode(10, OUTPUT); // X-axis motor PWM pin
    pinMode(11, OUTPUT); // X-axis motor direction pin

    delay(10000);  // Allow time for setup
}

// Function to calculate the distance between two points
float calculateDistance(float x1, float y1, float x2, float y2) {
    return sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
}

// Function to interpret grid coordinates as motor voltage
ArduinoValues computeMotorSignals(float coord) {
    ArduinoValues motorOutput;

    // Convert coordinate to voltage in the range -5V to 5V
    float voltage = coord * 0.5;

    if (voltage >= 0) {
        motorOutput.polarity = false;  // Forward direction
        motorOutput.speedSignal = (voltage / 5.0) * 255;  // Scale voltage
to PWM (0-255)
    } else {
        motorOutput.polarity = true;  // Reverse direction
        motorOutput.speedSignal = ((voltage + 5.0) / 5.0) * 255;  // Scale
for negative voltages
    }

    return motorOutput;
}
```

Krishna and Vincent
LAB 10
12/6/2024

```
// Function to send motor signals and move the plotter
void sendMotorSignals(float xCoord, float yCoord) {
    ArduinoValues xMotor = computeMotorSignals(xCoord);
    ArduinoValues yMotor = computeMotorSignals(yCoord);

    digitalWrite(11, xMotor.polarity);
    analogWrite(10, xMotor.speedSignal);

    digitalWrite(6, yMotor.polarity);
    analogWrite(5, yMotor.speedSignal);
}

#include <math.h>

void drawSemicircle(float startX, float startY, float midX, float midY,
float endX, float endY) {
    // Calculate the radius of the semicircle
    float radius = sqrt(pow(midX - startX, 2) + pow(midY - startY, 2));

    // Calculate the center of the circle
    float centerX = (startX + endX) / 2;
    float centerY = (startY + endY) / 2;

    // Determine the direction of the arc (upper or lower)
    bool isUpperArc = (midY > centerY);

    // Divide the semicircle into 10 equal segments
    for (int i = 1; i <= 10; i++) {
        float theta = (i * M_PI) / 10; // Divide semicircle into 10 equal
angles

        // Calculate the (x, y) point on the semicircle
        float x = centerX + radius * cos(theta);
        float y = centerY + (isUpperArc ? radius * sin(theta) : -radius *
sin(theta));

        // Move to the calculated point
        moveTo(x, y);
    }
```

```
    // Ensure the final point is exactly the endpoint
    moveTo(endX, endY);
}


// Function to move the plotter to a specific target point
void moveTo(float targetX, float targetY) {
    // Calculate whether diagonal interpolation is required
    bool isDiagonal = (fabs(targetX - currentX) > 0.01) && (fabs(targetY -
currentY) > 0.01);

    if (isDiagonal) {
        // Compute path length and number of steps
        float distance = calculateDistance(currentX, currentY, targetX,
targetY);
        int numSteps = ceil(distance / 0.5);

        // Calculate incremental steps for X and Y
        float stepX = (targetX - currentX) / numSteps;
        float stepY = (targetY - currentY) / numSteps;

        // Move incrementally along the interpolated path
        for (int step = 1; step <= numSteps; step++) {
            float nextX = currentX + stepX * step;
            float nextY = currentY + stepY * step;
            sendMotorSignals(nextX, nextY);
            delay(300);  // Reduced pause for smoother motion
        }
    } else {
        // Move directly to the target for non-diagonal paths
        sendMotorSignals(targetX, targetY);
        delay(1000);  // Stabilization delay
    }

    // Update the current position
    currentX = targetX;
    currentY = targetY;
}
```

Krishna and Vincent
LAB 10
12/6/2024

```
// Function to draw a fish
void drawDesign() {
    // Tail
    moveTo(6, 0);
    moveTo(9, -3);
    moveTo(9, 3);
    moveTo(6, 0);
    // Body and Face
    moveTo(5, 2);
    moveTo(3, 4);
    moveTo(0, 5);
    moveTo(-3, 5);
    moveTo(-6, 4);
    moveTo(-4, 2);
    moveTo(-3, 0);
    moveTo(-4, -2);
    moveTo(-6, -4);
    moveTo(-8, -2);
    moveTo(-7, -2);
    moveTo(-6, -1);
    moveTo(-8.5, -1);
    moveTo(-8, -2);
    moveTo(-9, 0);
    moveTo(-8, 2);
    moveTo(-7, 2);
    moveTo(-7, 1);
    moveTo(-8.5, 1);
    moveTo(-7, 2);
    moveTo(-8, 2);
    moveTo(-6, 4);
    moveTo(-4, 2);
    moveTo(-3, 0);
    moveTo(-4, -2);
    moveTo(-6, -4);
    moveTo(-3, -5);
    moveTo(0, -5);
    moveTo(3, -4);
    moveTo(5, -2);
```

```
    moveTo(6, 0);
    // Gills and Scales
    moveTo(5, 2);
    moveTo(3, 4);
    moveTo(0, 5);
    moveTo(-3, 5);
    drawSemicircle(-3, 5, -2, 4, -3, 3);
    drawSemicircle(-3, 3, -2, 2, -3, 1);
    drawSemicircle(-3, 1, -2, 0, -3, -1);
    drawSemicircle(-3, -1, -2, -2, -3, -3);
    drawSemicircle(-3, -3, -2, -4, -3, -5);
    moveTo(-3, 5);
    moveTo(-2, -5);
    drawSemicircle(-2, -5, -1, -4, -2, -3);
    drawSemicircle(-2, -3, -1, -2, -2, -1);
    drawSemicircle(-2, -1, -1, 0, -2, 1);
    drawSemicircle(-2, 1, -1, 2, -2, 3);
    drawSemicircle(-2, 3, -1, 4, -2, 5);
    moveTo(-2, 5);
    moveTo(-1, 5);
    drawSemicircle(-1, 5, 0, 4, -1, 3);
    drawSemicircle(-1, 3, 0, 2, -1, 1);
    drawSemicircle(-1, 1, 0, 0, -1, -1);
    drawSemicircle(-1, -1, 0, -2, -1, -3);
    drawSemicircle(-1, -3, 0, -4, -1, -5);
    moveTo(-1, -5);
    moveTo(0, -4);
    drawSemicircle(0, -4, 1, -3, 0, -2);
    drawSemicircle(0, -2, 1, -1, 0, 0);
    drawSemicircle(0, 0, 1, 1, 0, 2);
    drawSemicircle(0, 2, 1, 3, 0, 4);
    moveTo(0, 4);
    moveTo(-1, 4);
    drawSemicircle(1, -4, 2, -3, 1, -2);
    drawSemicircle(1, -2, 2, -1, 1, 0);
    drawSemicircle(1, 0, 2, 1, 1, 2);
    drawSemicircle(1, 2, 2, 3, 1, 4);
    moveTo(1, 4);
    moveTo(2, 3);
```

```
    drawSemicircle(2, 3, 3, 2, 2, 1);
    drawSemicircle(2, 1, 3, 0, 2, -1);
    drawSemicircle(2, -1, 3, -2, 2, -3);
    moveTo(2, -3);
    moveTo(3, -3);
    drawSemicircle(3, -3, 4, -2, 3, -1);
    drawSemicircle(3, -1, 4, 0, 3, 1);
    drawSemicircle(3, 1, 4, 2, 3, 3);
    moveTo(3, 3);
    moveTo(4, 2);
    drawSemicircle(4, 2, 5, 1, 4, 0);
    drawSemicircle(4, 0, 5, -1, 4, -2);

    delay(30000);  // Pause after drawing
}

// Arduino main loop
void loop() {
    drawDesign();  // Execute the drawing sequence
}
```