Robotics Lab 2: Forward and Inverse Kinematics

Authors: Stuart Harley, Kiel Dowdle
Date: 9/17/2021

Abstract

This lab was used as an introduction to forward and inverse kinematics in relation to robotic arms. Forward kinematics involves defining the position of an object as a function of joint angles or speed. Inverse kinematics is the opposite, involving finding the angles and speed to produce a desired position. For this lab, we built and used a 2 DOF (Degree of Freedom) robotic arm to move the end effector to specific locations.

We perform 5 tasks in this lab. In the first two we derived the equations for forward and inverse kinematics for our arm and tested each. The next 3 tasks dealt with moving the arm to specific locations and raising and lowering pencil lead on the end of our arm. In this way we were able to draw straight lines and then combine multiple lines to draw a triangle and a square.

Throughout this lab it was clear that the robot arm was not very accurate in its movements. While our equations for forward and inverse kinematics were correct, the motors were inaccurate, and the arm was flimsy so our actual results for the tasks had significant errors. To reduce the amount of error, we would need to build a stronger robotic arm with higher quality parts that have more precision. Either that or we would need to make the robot control closed loop by checking the actual position of the robotic arm using sensors and adjusting our movements accordingly.

Methods

For all the tasks a piece of paper was taped onto the table. The robot arm was then taped to the table so that the end effector would move across the piece of paper. A piece of pencil lead was taped to the end of the end effector which could be raised and lowered so that the arm could draw on the paper.

When drawing lines on the paper, the arm would not naturally move in a straight line from one point to another. Therefore, when we draw lines in this lab, we program the arm to draw to midway points that fall directly between the end points.

Task 1 (Setup and Forward Kinematics Test)

For task 1, we derived the equation for forward kinematics of a 2 DOF (Degree of Freedom) arm. Using the properties of similar triangles and trigonometry, we derive the equations for forward kinematics for the x and y directions shown below in Image 1.

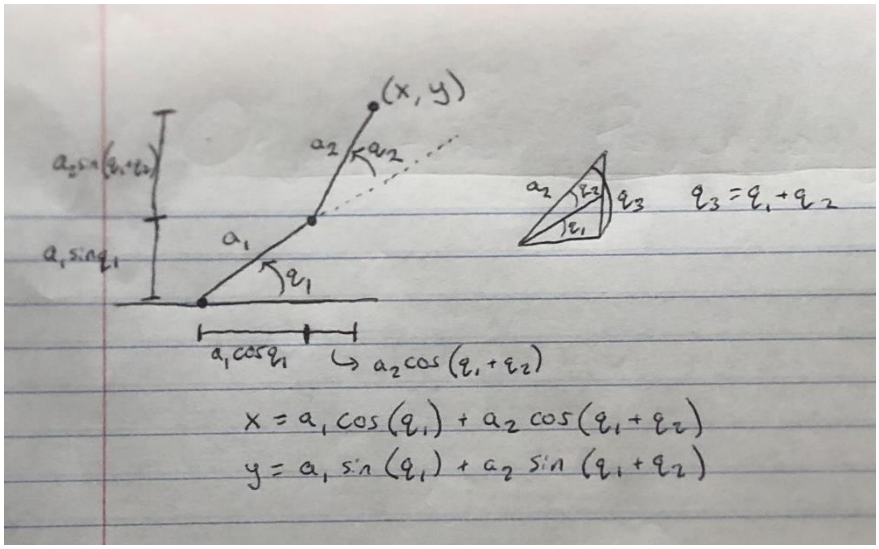

Image 1: Derivation of the Forward Kinematics Equations

To measure the actual location that our robot arm moved to in task 1, a ruler was used to measure the distances in the x and y directions from the point of origin location under the end of servo 1. Example picture below in Image 2.

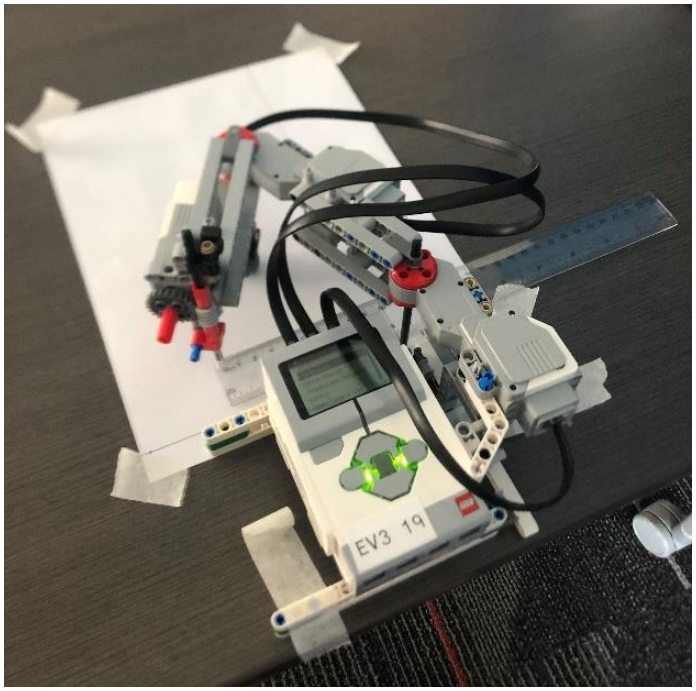

Image 2: Picture of the robotic arm post forward kinematics run

Task 2 (Single Points)

For task 2, we derived the equation for inverse kinematics of a 2 DOF arm. Using the properties of similar triangles and trigonometry, we derive the equations for inverse kinematics for q1 and q2 below in Image 3. Note: because q2 can be either positive or negative, there are 2 sets of angles for each (x, y) location.

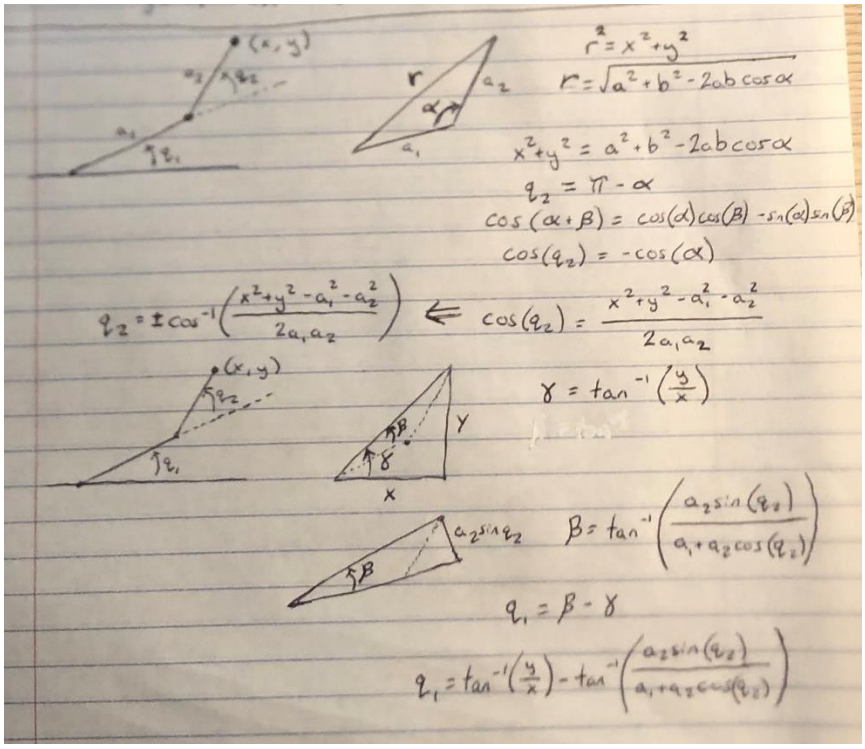

Image 3: Derivation of the Inverse Kinematics Equations

To measure the actual location that our robot arm moved to in task 2, we did the same thing as in task 1. A ruler was used to measure the distances in the x and y directions from the point of origin location under the end of servo 1. Example picture below in Image 4.

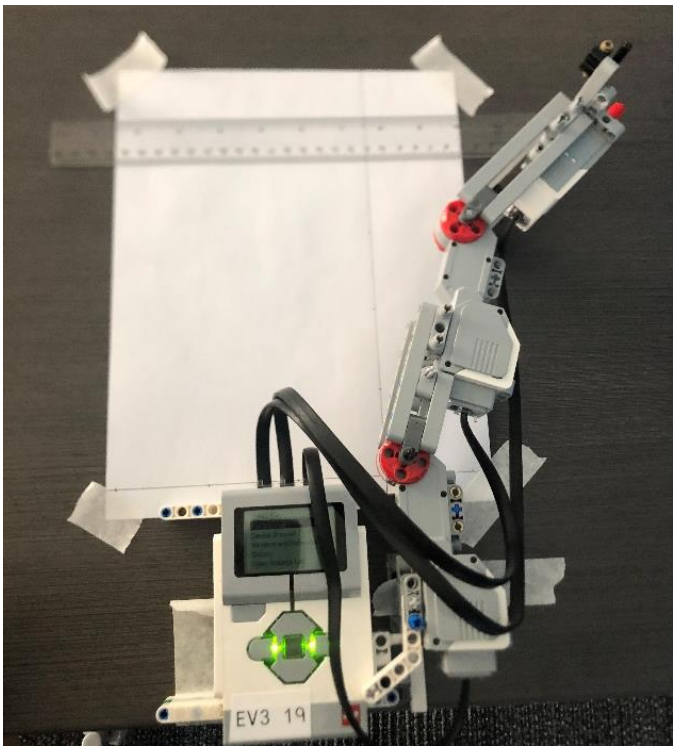

Image 4: Picture of the robotic arm post inverse kinematics run

Analysis

After completing tasks 1 and 2 and recording the results we performed a one-sided T-test to determine if the robot moved as we were expecting based on how far the arm was from the expected value. The null hypothesis for each test was that the robotic arm moved to the expected position. The alternative hypothesis is that the robotic arm did not move to the expected position. In this case we are rejecting the null hypothesis when the p-value of the test is less than or equal to 0.05 ($p <= 0.05$). If we accept the null hypothesis the p-value gives us the probability of the measured mean appearing in a normal distribution of measurements.

Results

Task 1 (Setup and Forward Kinematics Test)

For this task we picked 2 arbitrary sets of servo angles within the range of our robotic arm to test our forward kinematics equation. Using our equation, we calculated the expected x and y coordinates that the robot should move to. We tested these movements 5 times for each set of angles. Results are shown in Table 1. Since the p-value for the first location was 0.0075 and for the second location was 0.0310, we were able to reject the null hypothesis in both cases making the alternate hypothesis stating that the robotic arm did not move to the expected position true for both locations.

	Servo1 = 114°, Servo2 = 124° Expected Location = (-10.77, 2.79)		Servo1 = 70°, Servo2 = -30° Expected Location = (12.42, 18.78)	
	Actual Location	Distance [cm]	Actual Location	Distance [cm]
Trial 1	(-11.1, 4.6)	1.84	(12.2, 18.8)	0.22
Trial 2	(-11.3, 5.0)	2.27	(10.0, 20.5)	2.97
Trial 3	(-10.1, 2.8)	0.67	(11.1, 19.8)	1.67
Trial 4	(-11.0, 3.9)	1.13	(11.8, 19.2)	0.75
Trial 5	(-11.2, 2.8)	0.43	(12.2, 19.0)	0.31
Mean	(-10.94, 3.82)	1.30	(11.46, 19.46)	1.18
St. Dev.	(0.43, 0.90)	0.69	(0.83, 0.62)	1.03
P-value	(0.2142, 0.0317)	0.0075	(0.0308, 0.0349)	0.0310

Table 1: Forward Kinematics experimental and one-sided t-test results

Below in Figure 1 is a box plot with error bars visually showing the distribution of the distance between the expected position and the actual position of the robotic arm.

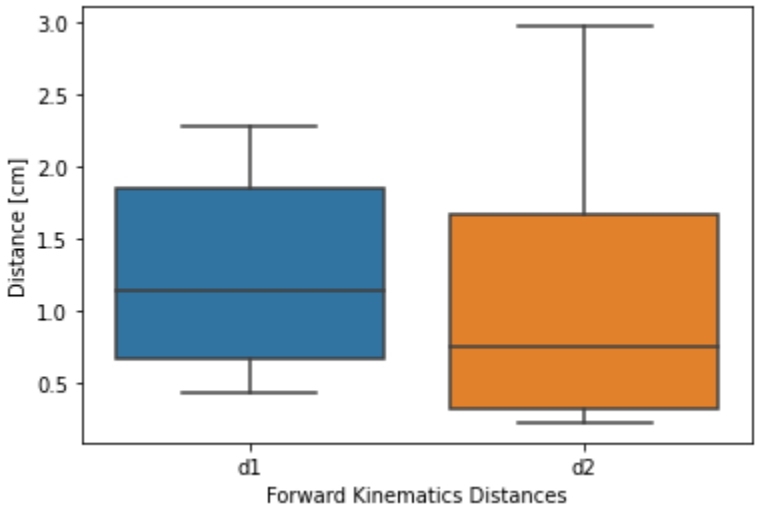

Figure 1: Box plot visualizing the distribution of distances across the trials for task 1. d1 correlates to the (-10.77, 2.79) point and d2 to (12.42, 18.78)

Task 2 (Single Points)

For this task we used the same calculated x and y points from task 1 to test our inverse kinematics equation. Using our equation, we calculated the servo angles that the robot should rotate by to reach those coordinates. We tested these movements 5 times for each set of coordinates. We also verified the angles that our equation was calculating. Results are shown in Table 2. Since the p-value for the first location was 0.0001 and for the second location was 0.0183, we were able to reject the null hypothesis in both cases making the alternate hypothesis stating that the robotic arm did not move to the expected position true for both locations.

	Expected Location = (-10.77, 2.79) Calculated Angles = 114.0°, 124.0°		Expected Location = (12.42, 18.78) Calculated Angles = 70.0°, -30.0°	
	Actual Location	Distance [cm]	Actual Location	Distance [cm]
Trial 1	(-11.1, 5.3)	2.53	(11.7, 19.2)	0.83
Trial 2	(-11.5, 5.0)	2.33	(9.8, 20.6)	3.19
Trial 3	(-11.6, 4.1)	1.55	(12.6, 18.8)	0.18
Trial 4	(-10.9, 5.1)	2.31	(11.4, 19.5)	1.25
Trial 5	(-11.0, 4.6)	1.82	(9.8, 20.8)	3.31
Mean	(-11.22, 4.82)	2.11	(11.06, 19.78)	1.75
St. Dev.	(0.27, 0.43)	0.36	(1.10, 0.79)	1.27
P-value	(0.0113, 0.0002)	0.0001	(0.0254, 0.0233)	0.0183

Table 2: Inverse Kinematics test data

Below in Figure 2 is a box plot with error bars visually showing the distribution of the distance between the expected position and the actual position of the robotic arm.

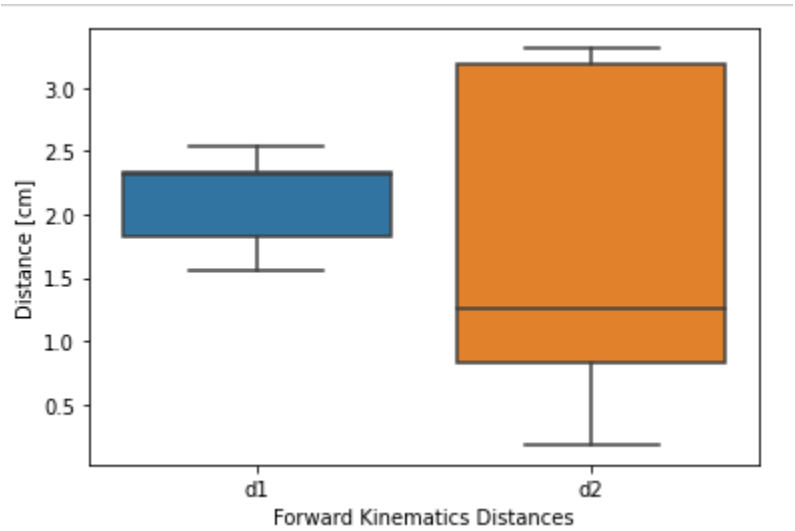

Figure 2 Box plot visualizing the distribution of distances across the trials for task 2. d1 correlates to the (-10.77, 2.79) point and d2 to (12.42, 18.78)

Task 3 (Marking Vertices)

For this task we attached a piece of pencil lead to the end effector or the arm. We picked a triangle that our arm could reach with the corner coordinates (-7, 8), (-13, 8), (-13, 15). We then programmed our arm to move to each of the coordinates and make a mark. The results are shown below in image 5. The dots are the expected locations of the marks, and the circled slashes are the actual results.

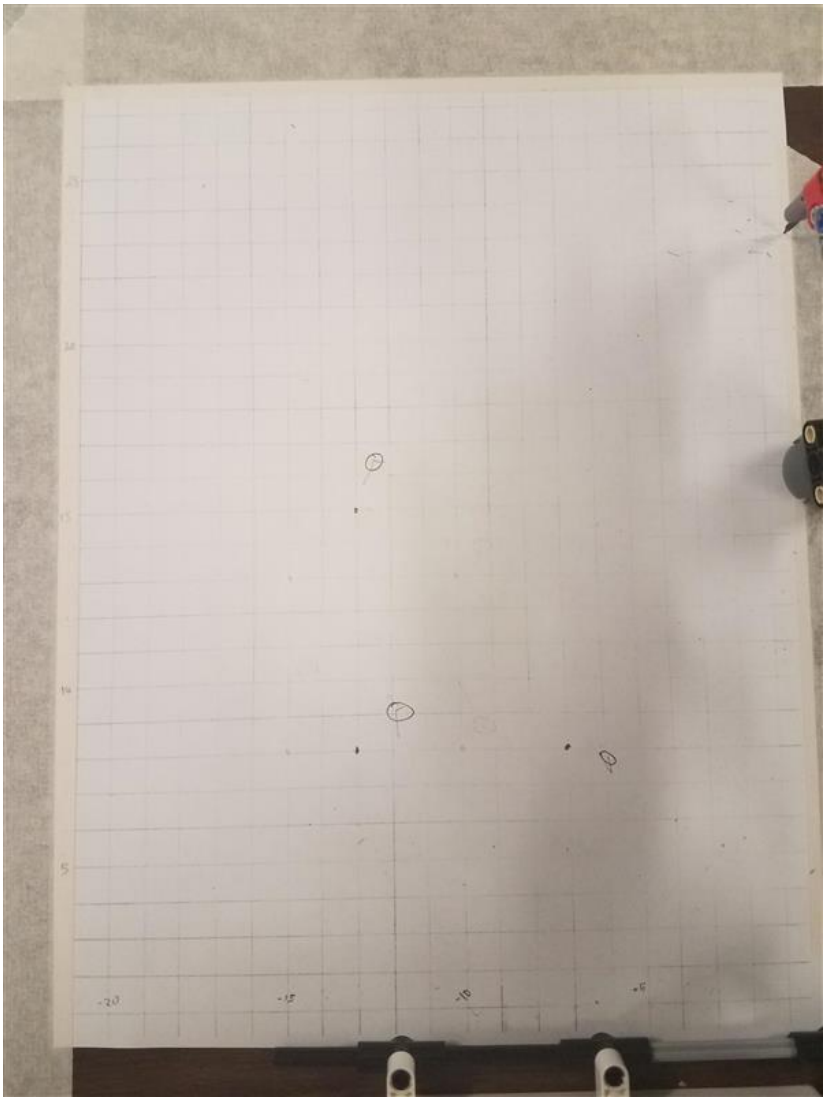

Image 5: Triangle Vertices Marking Results

We also picked a square that our arm could reach with the corner coordinates (-15, 13), (-15, 8), (-10, 13), (-10, 8). We then programmed our arm to move to each of the coordinates and make a mark. The results are shown below in image 6. The dots are the expected locations of the marks, and the circled slashes are the actual results.

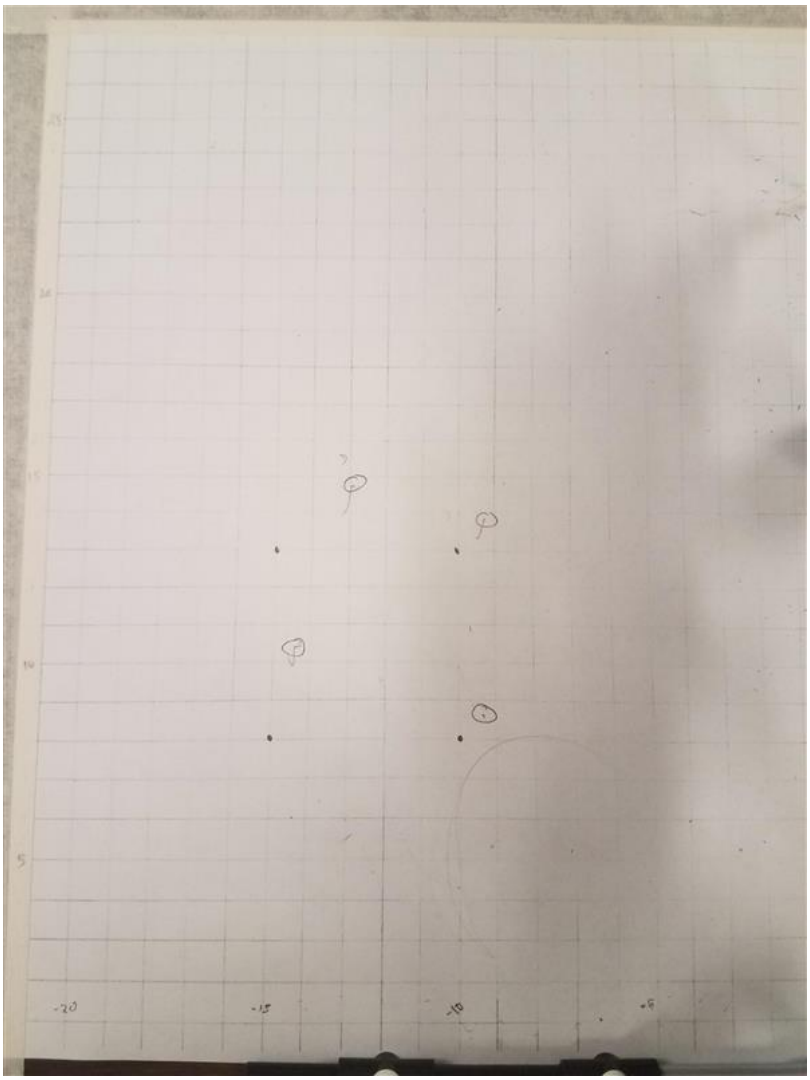

Image 6: Square Vertices Marking Results

Task 4 (Straight Lines)

For task 4, the robot was programmed to draw a long horizontal line from (-2, 15) to (-15, 15). The actual result is shown below in image 7. Video for this task was also recorded.

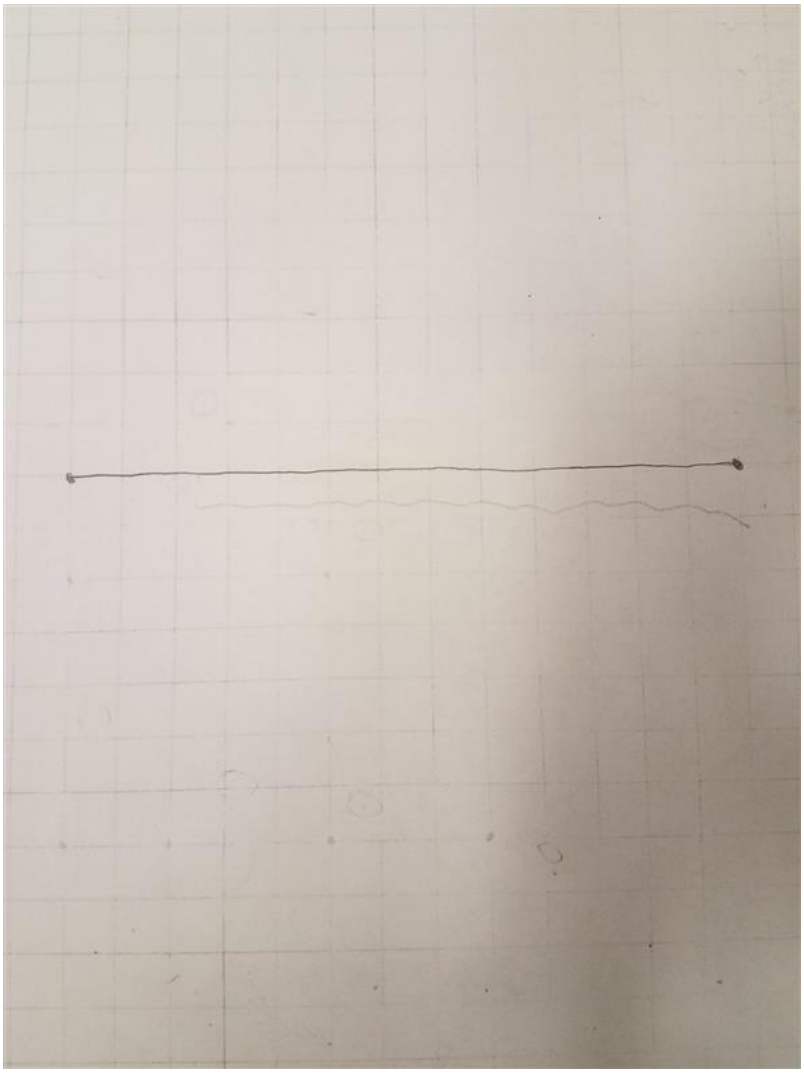

Image 7: Drawing Horizontal Line Results

A long vertical line was also programmed to be drawn from (-7, 21) to (-7, 6). The actual result is shown below in image 8.

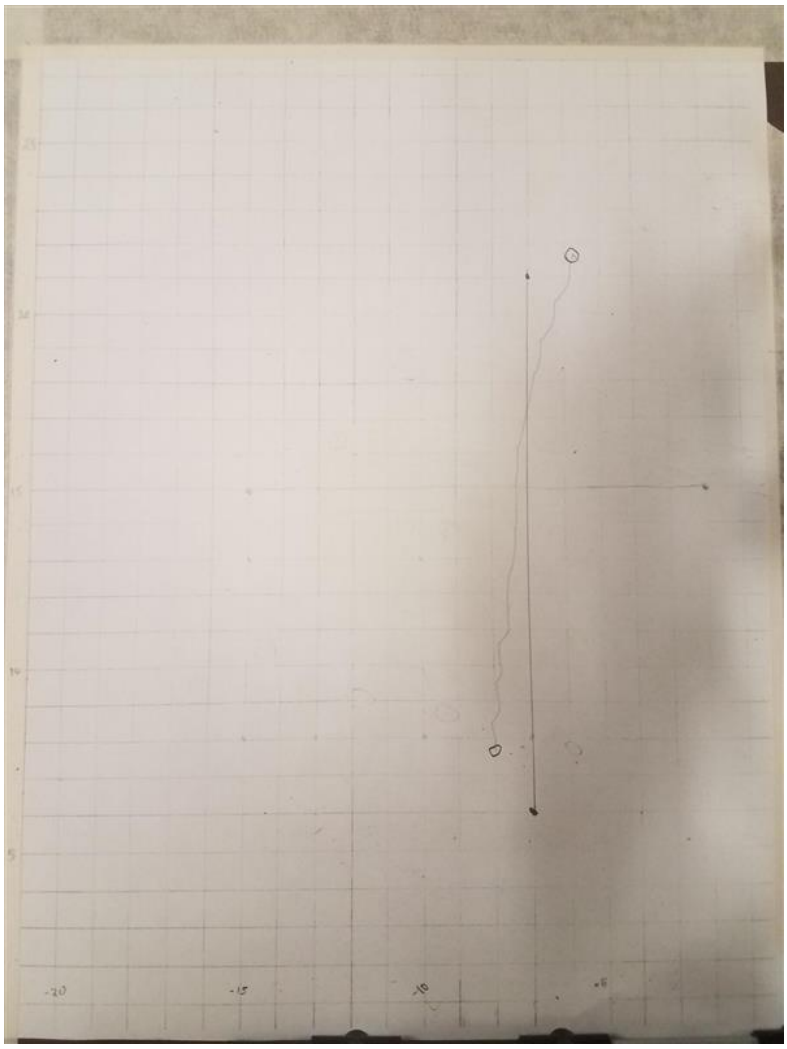

Image 8: Drawing Vertical Line Results

A long diagonal line was also programmed to be drawn from (-16, 9) to (-4, 21). The actual result is shown below in image 9.

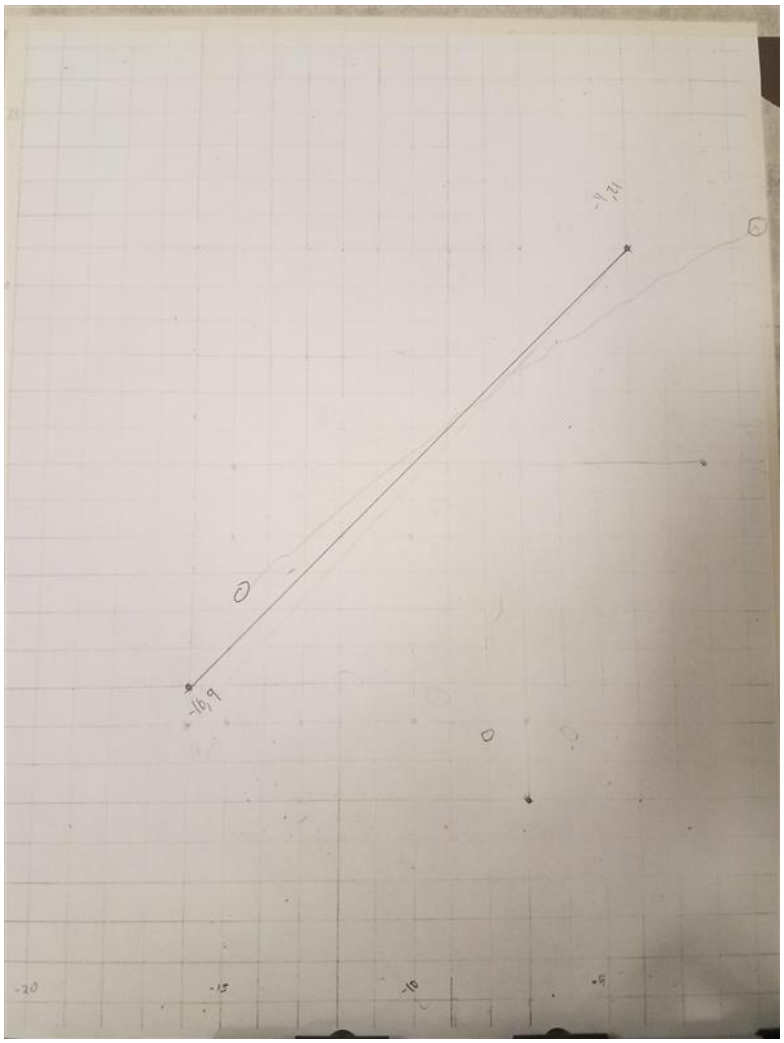

Image 9: Drawing Diagonal Line Results

Task 5 (Draw Shapes)

For task 5, we revisit the square and triangle from task 3. However, in this section we are connecting the points instead of simply marking the corners. Below is the result of the robot drawing the square shown in image 10. The corners of the square are at (-3, 18), (-3, 11), (-10, 11), and (-10, 18). The dark points show the expected square to be drawn, and the circled points show the path that the arm actually drew. Video for this task was also recorded.

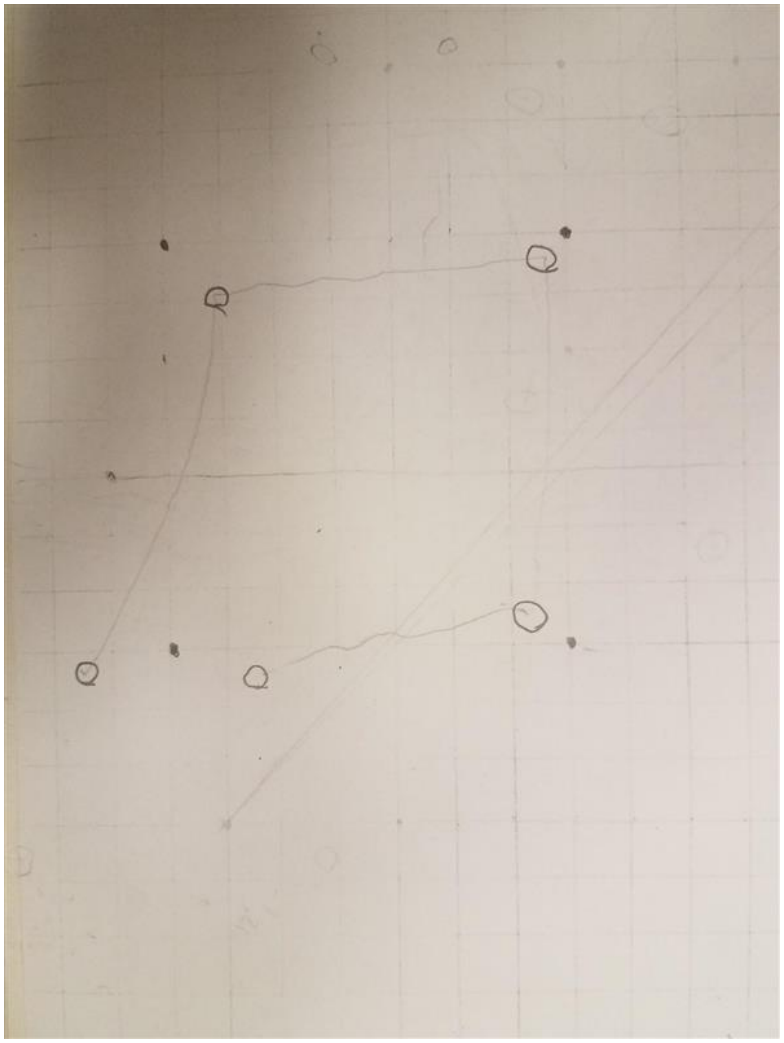

Image 10: Drawing Square Results

Below is the result of the robot drawing the triangle shown in image 11. The corners of the triangle are at (-5, 20), (-5, 10), and (-15, 10). The dark points connected by the straight lines show the expected triangle to be drawn, and the circled points show the path that the arm actually drew.

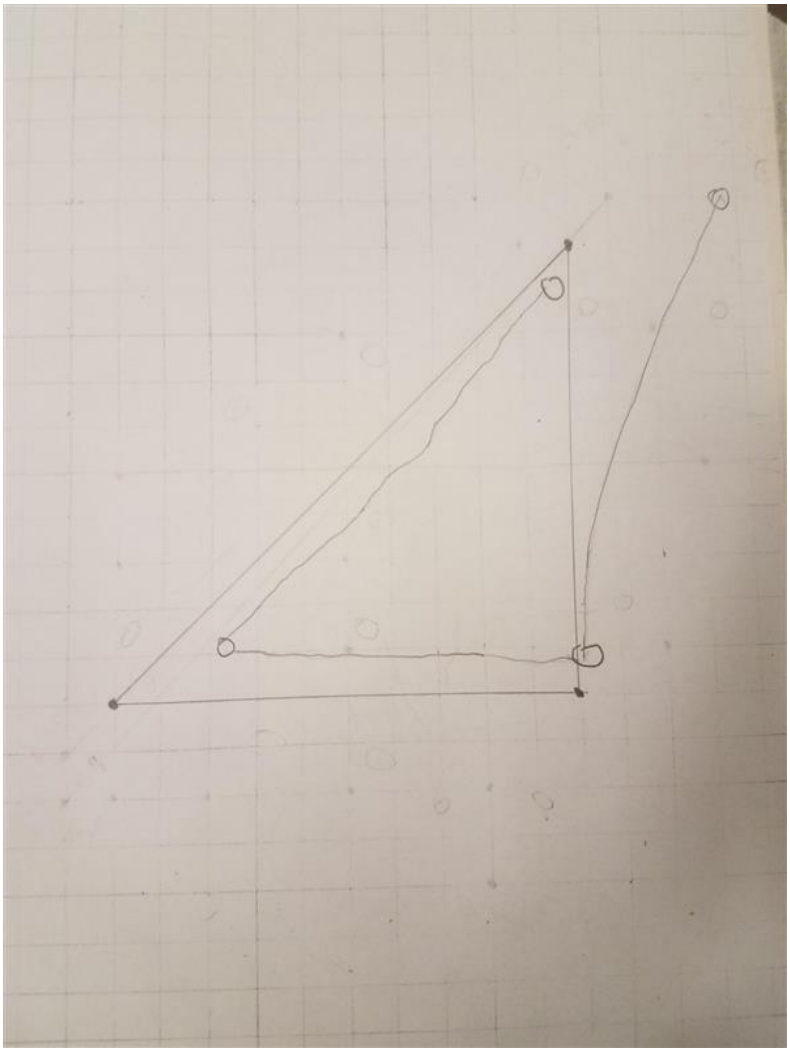

Image 11: Drawing Triangle Results

Discussion

1. Drawing a straight line between two points is non-trivial because the servos do not move simultaneously. This means that when the arm goes between two points, instead of going in a straight line, it will actually move in a path resembling two semi-circles. Therefore, midway points need to be used to make the arm appear to go in a straight line.

2. The hard constraints in our inverse kinematics are places that the robot arm cannot reach. This is due to the arms not being long enough to reach or the fact that the arms cannot rotate a full 360 degrees because they are blocked by the body of the robot. We dealt with these hard constraints by making sure that any points we picked for any of our tasks could be reached by the robot arm.

3. For each desired target location, the analytical method produced two sets of angles. In order to decide with pair to use we checked a few things. First, we checked if the angle for servo 1 exceeded its approximate rotational limit. Second, we checked if the angle for servo 2 exceeded its approximate rotational limit. Lastly, we chose the pair of angles that had the lower servo1 angle. We decided to choose based off the servo 1 angle because servo 2 can rotate further to the left than to the right due to the hard constraints. Therefore, servo 2 generally does not have to rotate right if servo 1 chooses the smaller angle.

4. The movements of the arm were not accurate. This is due to a variety of reasons. The first is motor inaccuracy. The motors don't move the exact correct amount that you specify. The second is the looseness of the arm. The arm is flimsy, so it flexes and bends which causes error. The third is that when resetting the arm to its starting position, the arm is not in the exact same position every time. The arm is not perfectly straight when it gets manually reset so this causes error.

5. The biggest difficulty in working with the robot arm was dealing with the pencil lead and making it draw. When the arm was raised, it was specified to rise until it stalled. However, the motor kept pulling apart the Lego pieces instead of stalling. So, a new pencil arm had to be made that would not fall apart. Also, attaching the pencil lead to the pencil arm was very hard. The lead tended to get pushed up into the arm when it was lowered so it then would not touch the paper. In order to fix this, the tip of a real pencil was cut off and then duct taped to the end of the arm.

Supplementary Material

Contained in this section are code screenshots for tasks 1-5. Videos of tasks 4 and 5 are also included in the submission.

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import (Motor, TouchSensor, ColorSensor,
                                 InfraredSensor, UltrasonicSensor, GyroSensor)
from pybricks.parameters import Port, Stop, Direction, Button, Color
from pybricks.tools import wait, StopWatch, DataLog
from pybricks.robotics import DriveBase
from pybricks.media.ev3dev import SoundFile, ImageFile

import math
import time

ARM1 = 12.8
ARM2 = 10.5
DEG_PER_SEC = 15

# Helper Methods
def task_one(angle_one, angle_two):
    motor1.run_target(DEG_PER_SEC, angle_one)
    motor2.run_target(DEG_PER_SEC, angle_two)


def task_two(x, y):
    move_to(x, y)


def task_three(x, y):
    lift_pen_arm()
    move_to(x, y)
    drop_pen_arm()
    mark()


def task_four(x, y, direction):
    lift_pen_arm()
    move_to(x, y)
    drop_pen_arm()
    if direction is "horizontal":
        horizontal(x, y, 10, -1)
    elif direction is "vertical":
        vertical(x, y, 10, -1)
    elif direction "diagonal":
        horizontal(x, y, 10, -1)
```

```
def task_five(x, y, shape):
    lift_pen_arm()
    move_to(x, y)
    drop_pen_arm()
    if shape is "square":
        square(x, y, 7)
    elif shape is "triangle":
        triangle(x, y, 7)


def square(x, y, d):
    vertical(x, y, d, -1)
    y = y - d
    horizontal(x, y, d, -1)
    x = x - d
    vertical(x, y, d, 1)
    y = y + d
    horizontal(x, y, d, 1)


def triangle(x, y, d):
    vertical(x, y, d, -1)
    y = y - d
    horizontal(x, y, d, -1)
    x = x - d
    diagonal(x, y, d, 1)


def horizontal(x, y, d, s):
    for i in range(0, d):
        x = x + s
        q1, q2 = two_dof_inverse_kinematics(x, y)
        motor1.run_target(10, q1)
        motor2.run_target(10, q2)
        time.sleep(1)


def vertical(x, y, d, s):
    for i in range(0, d):
        y = y + s
        q1, q2 = two_dof_inverse_kinematics(x, y)
        motor1.run_target(10, q1)
        motor2.run_target(10, q2)
        time.sleep(1)

def diagonal(x, y, d, s):
    for i in range(0, d):
        y = y + s
        x = x + s
        q1, q2 = two_dof_inverse_kinematics(x, y)
        motor1.run_target(10, q1)
        motor2.run_target(10, q2)
        time.sleep(1)
```

```
def move_to(x, y):
    angle_one, angle_two = two_dof_inverse_kinematics(x, y)
    motor1.run_target(DEG_PER_SEC, angle_one)
    motor2.run_target(DEG_PER_SEC, angle_two)


def two_dof_inverse_kinematics(x, y):
    q2 = math.degrees(math.acos((x*x+y*y-ARM1*ARM1-ARM2*ARM2)/(2*ARM1*ARM2)))
    q2neg = -math.degrees(math.acos((x*x+y*y-ARM1*ARM1-ARM2*ARM2)/(2*ARM1*ARM2)))
    q1 = math.degrees(math.atan2(y, x) - math.atan2(ARM2*math.sin(math.radians(q2)), \
        (ARM1+ARM2*math.cos(math.radians(q2)))))
    q1neg = math.degrees(math.atan2(y, x) - math.atan2(ARM2*math.sin(math.radians(q2neg)), \
        (ARM1+ARM2*math.cos(math.radians(q2neg)))))
    if q1 > 140 or q1 < 40:  # Approx Rotational limits of servo1
        q1 = q1neg
        q2 = q2neg
    elif q2 > 150 or q2 < -45:  # Approx Rotational limits of servo2
        q1 = q1neg
        q2 = q2neg
    elif q1 > q1neg:  # Angle of arm1 compared and picks the lowest
        q1 = q1neg
        q2 = q2neg
    return (q1, q2)


def drop_pen_arm():
    pen_arm.run_until_stalled(50)


def lift_pen_arm():
    pen_arm.run_until_stalled(-50)


def mark():
    time.sleep(2)
    motor2.run_time(20, 400)
    time.sleep(2)
    motor2.run_time(-20, 400)
    time.sleep(2)

# Running the Tasks
ev3 = EV3Brick()
motor1 = Motor(Port.B)
motor2 = Motor(Port.C)
pen_arm = Motor(Port.D)

motor1.reset_angle(90)
motor2.reset_angle(0)
lift_pen_arm()

# Task 1: Test Forward Kinematics
task_one(114, 124)

# Task 2: Test Inverse Kinematics
task_two(-10.77, 2.79)
```

```
# Task 3: Mark Square Vertices
task_three(-15, 13)
task_three(-15, 8)
task_three(-10, 13)
task_three(-10, 8)

# Task 3: Mark Triangle Vertices
task_three(-7, 8)
task_three(-13, 8)
task_three(-13, 15)

# Task 4: Drawing Horizontal, Vertical, and Diagonal Lines
task_four(-2, 15, 'horizontal')
task_four(-7, 21, 'vertical')
task_four(-16, 9, 'diagonal')

# Task 5: Drawing a Square and a Triangle
task_five(-3, 18, 'square')
task_five(-5, 20, 'triangle')

lift_pen_arm()
```