Manipulation Report

Keaghan Knight

May 1, 2020

1 Introduction

In this assignment, our task was to build a robotic manipulator with two joints. The goal was for the arm to continuously pick up and move soda cans that were placed in front of it. While we did not have access to a lab to physically build the robot, we worked with various tools such as Autodesk Fusion 360, Google Colaboratory, and Google Draw to conceptualize and digitally create such a robot. Furthermore, we found the space in which the theoretical robot could physically reach, by utilizing an inverse kinematics equation.This report will break down the various mechanical designs, along with the mathematics.

2 Mechanical Design

The mechanical design itself went through three main iterations, a very basic sketch using Google Drawing, and two more complex designs using AutoDesk Fusion 360. The first of which uses two 35x35 super chassis, and two 15x16 chassis to form a rectangular base. The mechanical arm is made of a 25x25 chassis for the shoulder and a 15x16 chassis for the elbow. In this design a worm gear is used to move the elbow. The microcontroller sits in the middle underneath the arm, and the battery sits in behind the arm. Finally we chose to use two optimal shaft encoders. One attached to the gear train powering the shoulder, and the other attached to the worm gear powering the elbow. We chose to use the optimal shaft encoder over a potentiometer, since potentiometer is much more limited in terms of range of motion. A detailed diagram in Figure 1 shows our first iteration.

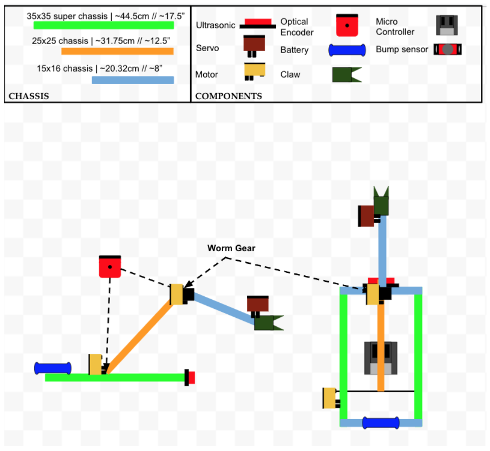
35x35 super chassis | ~44.5cm // ~17.5"
25x25 chassis | ~31.75cm // ~12.5"
15x16 chassis | ~20.32cm // ~8"
CHASSIS
Ultrasonic
Optical Encoder
Micro Controller
Servo
Battery
Bump sensor
Motor
Claw
COMPONENTS
Worm Gear

Figure 1: First Iteration

From the design in Figure 1, we created the first of our Autodesk designs. The gear train and the worm gear are omitted in this design since it was more conceptual, and used as an introduction to using Autodesk. Below in Figure 2, you will see a detailed design of the arm, shoulder, and base. This iteration also

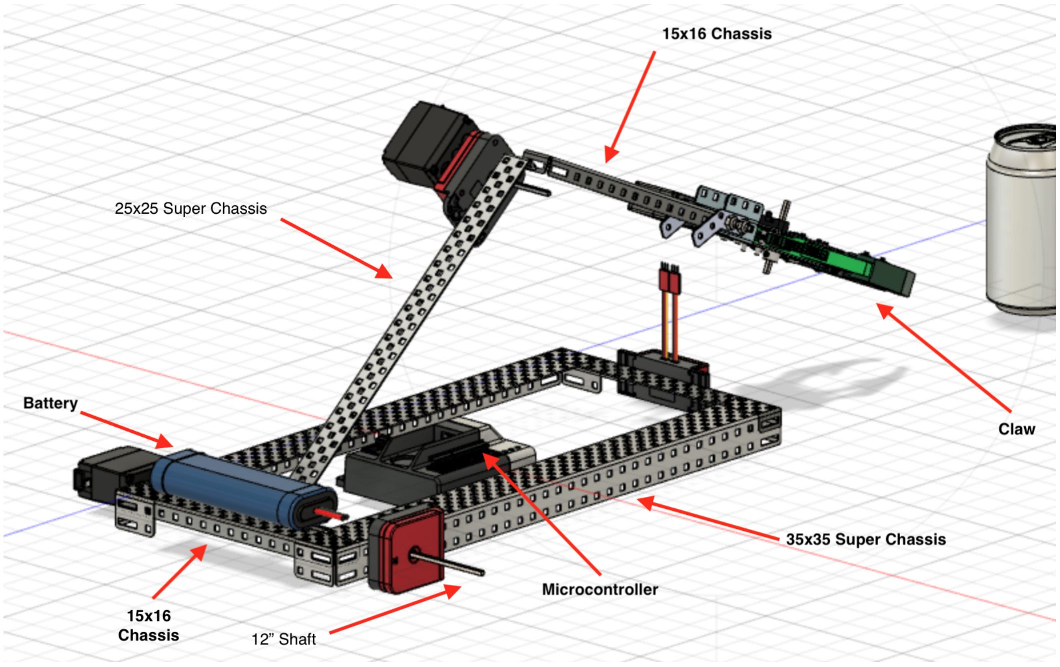

Figure 2: Second Iteration: Structure

featured the various sensors and motors used as seen below in Figure 3. However

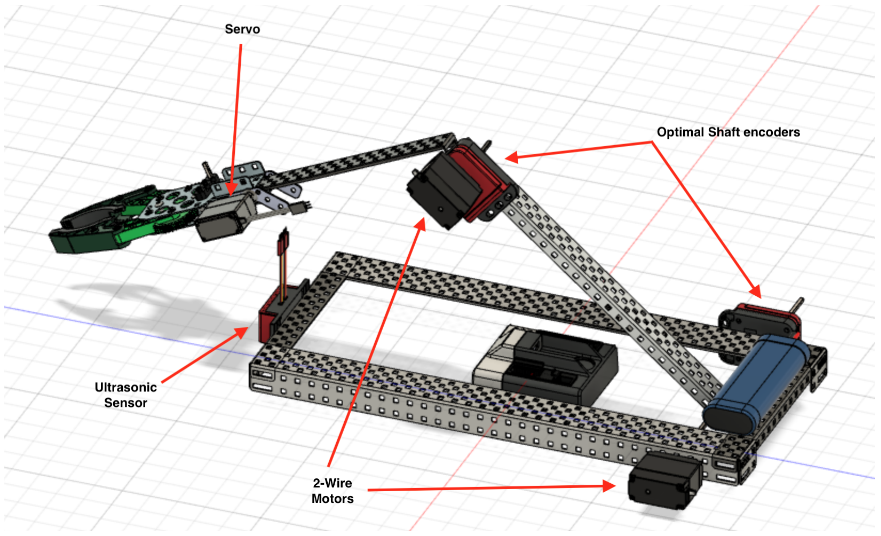

Figure 3: Second Iteration: Sensors

there are some design flaws that were adjusted for in the third and final iteration of the design. One such flaw is the use of the 25x25 chassis and 15x16 chassis for the shoulder and elbow. Since they are in the shape of an 'L' they only provide one point of support for the shaft. Therefore in the final design we decided to use 1x2x1x35 C-Channel for the shoulder and a 1x2x1x15 C-Channel for the elbow. This allows for the shaft to be supported by two point of contact. Another problem we ran into was the arm was set too far back. This meant that the arm was quite limited in how far it could reach out, and potentially not be able to grab cans. The arm was also quite low, and therefore limited physically by the microcontroller and the base. Therefore in the final design we decided to move the arm up, and in front of the microcontroller using two 1x5x1x25 C-Channels to support the 12" shaft. Moving the arm made it no longer limited by the base and thus being able to achieve what we believe to be a full 180°. Finally another minor adjust was to switch the servo powering the claw for a motor, since we did not want to potentially limit the range of motion that the claw can open.The changes to the arm can be seen below in Figure 4.

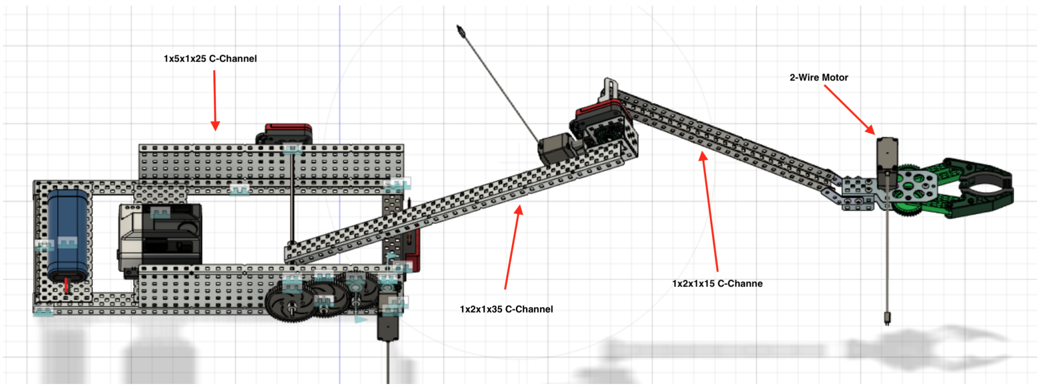

Figure 4: Third Iteration: Structure

The final part to the mechanical design is the gears used to lift the arm. Looking at our arm, we figured it would be very heavy and thus require more torque to lift the shoulder. Therefore we used a 12-tooth to a 60-tooth gear train, three times, which gives a ratio of $60 : 12 \times 60 : 12 \times 60 : 12 = 5 : 1 \times 5 : 1 \times 5 : 1 = 125 : 1$. This means we get 125x the torque and 1/125x the speed. A view of the compound gear train can be found in Figure 5 below. For the elbow we used a worm gear that has a gear ratio of 24:1, which should be enough torque to lift the elbow. A view of the worm gear can be found below in Figure 6.

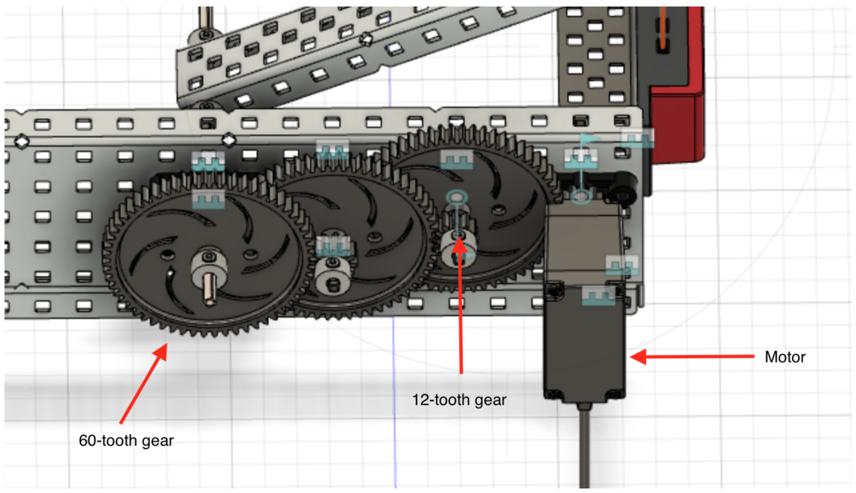

Figure 5: Third Iteration: Compound Gear Train

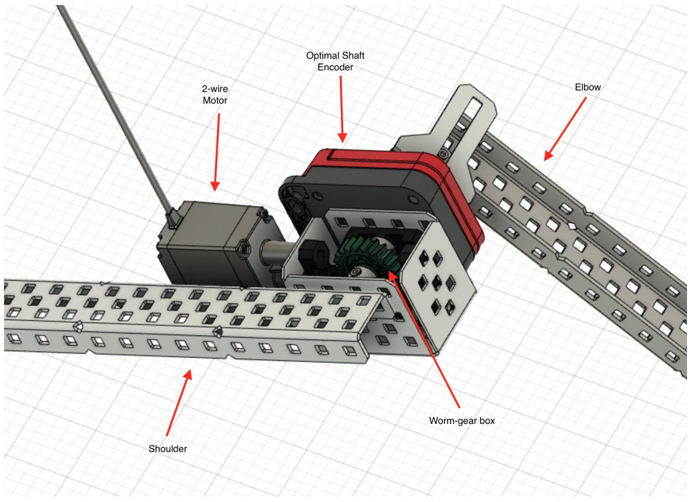

Figure 6: Third Iteration: Worm Gear

3 Inverse Kinematics

The fundamental problem the robot solves, is to move the arms of the robot to a certain angle (θ_1, θ_2) for some given position (x, y). In the Figure 5 below, note you will see a basic geometric interpretation of the problem. Here

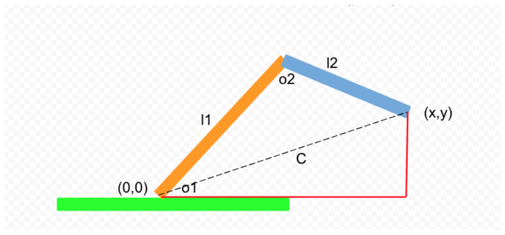

Figure 7: $(\theta_1, \theta_2) = (O_1, O_2)$

l_1 is the shoulder, and l_2 is the elbow of the robot. Solving for θ_2 we find. For any given position of the arm, you can draw a right triangle from the base of the shoulder to the end of the elbow which gives a width of x and a height of y. This effectively creates two triangles as seen in Figure 7. Therefore we can apply the law of cosines to the top triangle in order to solve for θ_2 as seen in the following calculations,

$$\begin{gathered}
c^2 = a^2 + b^2 - 2ab\text{cos}(C) \\
x^2 + y^2 = a^2 + b^2 - 2ab\text{cos}(C) \\
x^2 + y^2 = l_1^2 + l_2^2 - 2l_1l_2\text{cos}(C) \\
x^2 + y^2 = l_1^2 + l_2^2 - 2l_1l_2\text{cos}(\pi - \theta_2) \\
x^2 + y^2 = l_1^2 + l_2^2 + 2l_1l_2\text{cos}(\theta_2) \\
(x^2 + y^2) - (l_1^2 + l_2^2) = 2l_1l_2\text{cos}(\theta_2) \\
\frac{(x^2 + y^2) - (l_1^2 + l_2^2)}{2l_1l_2} = \text{cos}(\theta_2) \\
\theta_2 = \cos^{-1}(\frac{(x^2 + y^2) - (l_1^2 + l_2^2)}{2l_1l_2})
\end{gathered}$$

Now that θ_2 is known, we can solve for θ_1, using the law of sines,

$$\frac{a}{\text{sin}(A)} = \frac{b}{\text{sin}(B)} = \frac{c}{\text{sin}(C)}$$

Or in this case just $\frac{b}{\text{sin}(B)} = \frac{c}{\text{sin}(C)}$ seen by the following calculations,

$$\begin{gathered}
\frac{l_2}{\text{sin}(B)} = \frac{\sqrt{x^2 - y^2}}{sin(C)} \\
\frac{l_2}{\text{sin}(\alpha - \theta_1)} = \frac{\sqrt{x^2 - y^2}}{sin(\pi - \theta_2)} \\
\frac{l_2}{\text{sin}(\alpha - \theta_1)} = \frac{\sqrt{x^2 - y^2}}{sin(\theta_2)} \\
\sqrt{x^2 - y^2}\text{sin}(\alpha - \theta_1) = l_2\text{sin}(\theta_2) \\
\text{sin}(\alpha - \theta_1) = \frac{l_2\text{sin}(\theta_2)}{\sqrt{x^2 - y^2}} \\
\alpha - \theta_1 = \sin^{-1}(\frac{l_2\text{sin}(\theta_2)}{\sqrt{x^2 - y^2}}) \\
\theta_1 = \alpha - \sin^{-1}(\frac{l_2\text{sin}(\theta_2)}{\sqrt{x^2 - y^2}})
\end{gathered}$$

In the calculations above, α is the angle corresponding the triangle we made with x, and y, and therefore $\alpha = \tan^{-1}\frac{y}{x}$, and the equation becomes

$$\theta_1 = \tan^{-1}(\frac{y}{x}) - \sin^{-1}(\frac{l_2\text{sin}(\theta_2)}{\sqrt{x^2 - y^2}})$$

Finally by plugging in the values for $l_1 = 12.5$" and $l_2 = 8$" from our mechanical design, we find that for some given position (x, y), the angles (θ_1, θ_2) are as follows,

$$\theta_2 = \cos^{-1}(\frac{(x^2+y^2)-(220.25)}{200})$$
$$\theta_1 = \tan^{-1}(\frac{y}{x}) - \sin^{-1}(\frac{12.5 \cdot \sin(\theta_2)}{\sqrt{x^2-y^2}})$$

4 Results

From the angles found above, we were able to calculate the potential solution space of our robot using a python script that mapped all the possible solutions. The script itself consists of a Robot class with two attributes 'arm1' and 'arm2', or (l_1, l_2). The class has two functions. A function 'angle' that returns a pair of angles '(o1, o2)' or (θ_1, θ_2) given some pair (x, y). The function creates the pair from the calculations derived above, and using the attribute passed to the class upon initialization. The second function 'spacemap', maps the possible solutions for some given arm lengths and a range. The function counts the number of solutions from negative range to positive range for x and from 0 to positive range for y. Finally by plugging in the values for our arm and assuming that the base of the shoulder is the origin we can find the following solution space, where the white space corresponds to no solution, the red space corresponds to two solutions, and the blue space corresponds to one solution. Given

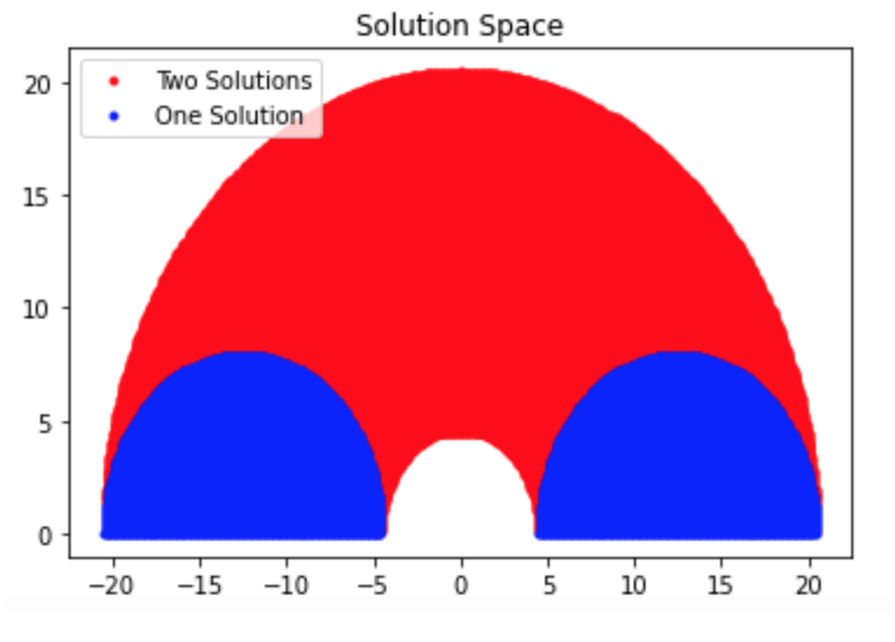

Figure 8: Solution Space for $l_1 = 12.5, l_2 = 8$

our design, we can assume that the arm will be able to reach the entire solution space, because there is no physical limitations. That is the arm is high enough above the frame and microcontroller that fully extended arm would not hit anything. Now if we mapped a solution space using a range of -100 to 100, then we would get the same solution space as -20 to 20. Therefore we know the solution space in Figure 8 is correct and we can come to the following conclusion, the farthest point the arm can reach is 20" in either direction, or (-20, 0) and (20, 0). If we assume a can is the farthest away to the right or the position (20, 0), then the corresponding angle $(\theta_1, \theta_2) = (-10.101038907695248, 26.00575632582788)$. And if a can is in the closest position or (5,0), then the corresponding angle $(\theta_1, \theta_2) = (-20.281649812950196, 167.4878268526627)$. Finally if a can is at the midpoint position or (12.5, 0) then the corresponding angle $(\theta_1, \theta_2) = (-37.32584976988496, 108.66292488494248)$. These angles are found using the Robot classes 'getangle' function.

5 Conclusion

Since their inception, robotic arms have had a revolutionary effect on the manufacturing industry. A simple task for us can be automated using gear trains, and kinematic functions. Our concept of such a robot was built using 12 tooth to a 60 tooth gear train for a 1:25 gear ratio. We found the inverse kinematic equations for some angles (θ_1, θ_2) given some position (x, y). Using these equations and a python script we were able to map a potential solution space of a full 180°. Since we had no lab, there was no true way to build and test our robot, however I do think there may be some potential problems that would arise. One problem I think we would run into is the arm would be limited by the length of wires of the various sensors. For example, when the arm is fully extended the wires for the motor that powers the claw may be too short to reach the micro controller. However if all the sensors and motor were able to reach the microcontroller at the extremes of the arm, then I do believe that our robot would have the full 180° range of motion. By studying our own biology we can create robots such as this arm, that can automate real world problems.

6 Appendix

```
import math
import matplotlib.pyplot as plt
import numpy as np

# Establish a Robot class instance
class Robot:
    def __init__(self, arm1, arm2):
        #Store values for arm lengts
        self.l1 = arm1
        self.l2 = arm2
    def angles(self, coord):
        #Split passed coordinates into separate variables
        x = coord[0]
        y = coord[1]

        #Given a set of coordinates, calculate the corresponding angles for the two arms
        #If given an error, return "Not Possible"
        try:
            num1 = x**2 + y**2 - self.l1**2 - self.l2**2
            den1 = 2*self.l1*self.l2

            o2 = math.acos(num1/den1)

            num2 = self.l2 * math.sin(o2)
            den2 = math.sqrt(x**2 + y**2)
            alpha = math.atan(y/x)

            o1 = alpha - math.asin(num2/den2)
            o1 = math.degrees(o1)
            o2 = math.degrees(o2)

            return (o1, o2)
        except:
            return "Not Possible"

#Function to map out the space of what's possible to reach
def space_map(length1, length2, test_value):
    x1_list = []
    x2_list = []
    y1_list = []
    y2_list = []

    A = Robot(length1, length2)

    #Run through test value range and create map
    for x in np.arange(-test_value,test_value,.1):
        for y in np.arange(0,test_value,.1):
            output = A.angles((x,y))
            if output != "Not Possible":
                a,b = output
                if x < 0:
                    #Map single possibilities in x < 0
                    distance = math.sqrt((x+A.l1)**2+(y-0)**2)
                    if distance < A.l2:
                        x2_list.append(x)
                        y2_list.append(y)
                    else:
                        x1_list.append(x)
                        y1_list.append(y)
                else:
                    #Map single possibilities in x > 0
                    distance = math.sqrt((x-A.l1)**2+(y-0)**2)
                    if distance < A.l2:
                        x2_list.append(x)
                        y2_list.append(y)
                    else:
                        x1_list.append(x)
                        y1_list.append(y)

    plt.plot(x1_list, y1_list,'r.', label='Two Solutions') #Plot 2 possibilities spaces
    plt.plot(x2_list, y2_list,'b.', label='One Solution') #Plot single possiblity spaces
    plt.title("Solution Space")
    plt.legend()
    plt.show() #Show plot

def get_angle(length1, length2, coordinate):
    A = Robot(length1, length2)
    print(A.angles(coordinate))

#Plot solution space for robot w/ arm lengths of 12.5 & 8
space_map(12.5, 8, 40)

#Return needed solution to given coordinate
get_angle(12.5, 8, (12.5,0))
```