Robotics I - MANE 4120

Mini-Project 2: Planar robot Arm Motion

Eric Tung

September 28$^{\text{th}}$, 2021

I, Eric Tung, certify that the following work is my own and completed in accordance with the academic integrity policy as described in the robotics I (MANE 4120) course syllabus.

Rensselaer Polytechnic Institute

Contents

1 Abstract

In Mini-Project 2: Planar robot Arm Motion, we implemented concepts such as homogeneous transforms, recursive forward kinematics, geometric inverse kinematics, and differential kinematics to compute orientation solutions to multi-body robot arms and MATLAB software such as `RigidBodyTree` and `checkCollision` to visualize our solutions. Our geometric inverse kinematics algorithm allows us to take any pre-defined path and compute relative homogeneous transforms for our robot to trace out in specific orientations. We also implemented joint constraints that effected how fast our robot could trace out those paths. In subsequent sections, we explore how various factors such as arm length and path orientation effects our optimized trace time.

2 Problem Statement

The goal of Mini-Project 2 was to implement forward and inverse kinematics for an n-link robotic arm. Tracing of a path was achieved through geometric inverse kinematics while creation of a path was achieved via recursive forward kinematics using homogenous transforms. Simulated trace paths were constrained by mechanical limitations of robot joints, which were given to us.

In the modern world, path tracing is imperative to the automotive, manufacturing, and any other automated industry. In order to decrease production time and increase material usage, companies have to figure out how to automate complex, repetitive milling paths, which are ideal for technologies such as n-link robotic arms.

Additional videos and original graphics that complement the following report can be found here!

3 Approach & Results

We implement computational mathematics theory alongside MATLAB algorithms to establish, compute, and visualize the following solutions.

3.1 Problem 1

For problem 1, our goal is to understand how to derive the orientation vector for our robot arm tip, p_T. It is desirable to have the robot oriented normal and to the left of our path. In order to compute this normal path vector, we compute the tangent path vector and cross it with the positive z axis in order to obtain a leftwards normal vector. This cross product can be represented as a matrix multiplication in the form,

$$\vec{v_\perp} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \times \vec{v_{||}}$$

The primary path that we are tracing in Mini-Project 2 is the letter S. Shown below is the S-path as well as the computed normal vectors,

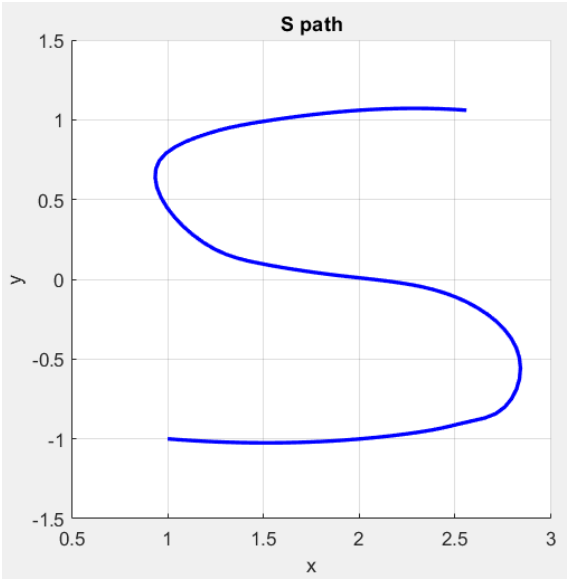

Figure 1: Trace of Path Data

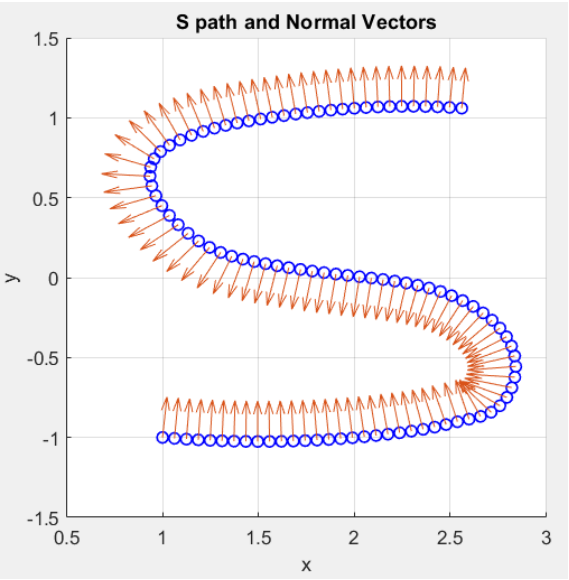

Figure 2: Path Points and Normal Vectors

In order to generate the path length array, λ, we first need to compute the 2-norm of the difference vectors between each point along the path. To compute the total path length array, we aggregate these elements within the path length array until the last element is the total length of the path.

3.2 Problem 2

For problem 2, our goal is to understand the mechanics of our robot simulation. We focus on forward kinematics, taking robot arm orientations to compute a tip path, and inverse kinematics, using a tip path to compute robot arm orientations. Using these two methods, we can visualize arm motions tracing out any path within the feasible range of our robot. For all three-arm analysis, we assume the arm lengths L_1, L_2, and L_3 are 1.5, 1.5, and 0.5 meters, from the origin to the robot arm tip, respectively.

For forward kinematics, we are given arm orientations q_1, q_2, and q_3. With preset robot arm lengths, this should theoretically allow us to compute one solution for the total orientation of the robot.

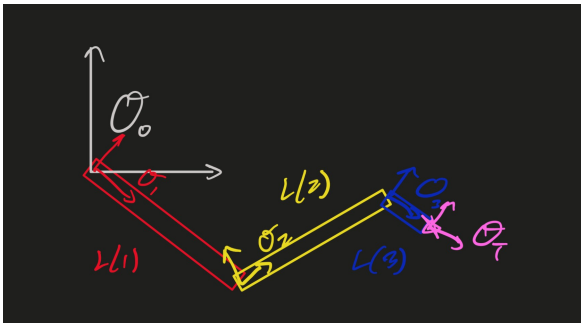

Figure 3: Hand Sketch of Individual robot Arm Basis

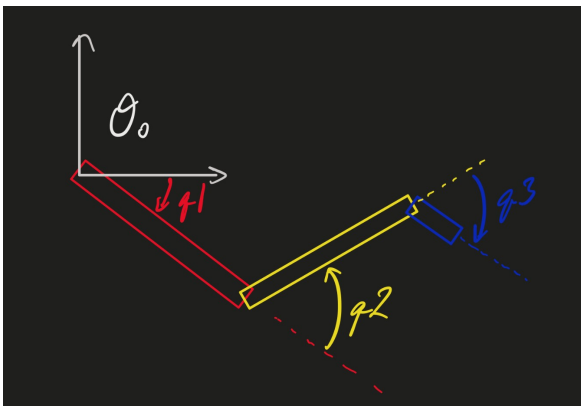

Figure 4: Relative Angles q Between Arms

In order to do this, we compute the transformation matrix from each arm joint to

the next. Between two adjacent arm basis, the transformation between the two can be written as,

$$T_{n,n+1} = \begin{bmatrix} \cos(\theta_n) & -\sin(\theta_n) & 0 & L_n\cos(\theta_n) \\ \sin(\theta_n) & \cos(\theta_n) & 0 & L_n\sin(\theta_n) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We can also see that for the general case, the total transformation from the origin to the tip of the last arm, T_{0T}, can be written as,

$$T_{0T} = \begin{bmatrix} \cos(q_{tot}) & -\sin(q_{tot}) & 0 & D_x \\ \sin(q_{tot}) & \cos(q_{tot}) & 0 & D_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where,

$$\begin{aligned} q_{tot} &= \sum_{i=1}^{n} q_i \\ D_x &= \sum_{j=1}^{n} \left(L_j \cos\left(\sum_{i=1}^{j} q_i \right) \right) \\ D_y &= \sum_{j=1}^{n} \left(L_j \sin\left(\sum_{i=1}^{j} q_i \right) \right) \end{aligned}$$

Assuming we use a three-link robotic arm starting at the origin, we can write this total transformation as,

$$T_{0T} = T_{12}T_{23}T_{3T}$$

Note that because we start at the origin, there is no displacement between p_0 and p_1. As stated previously, we are essentially computing a relationship between the robot arm orientations, q, and the motion of the tip of the robot arm. In matrix form for an n-link robotic arm, the orientation of the tip basis can be written as,

$$\begin{bmatrix} q_T \\ x_T \\ y_T \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} q_i \\ \sum_{j=1}^{n} \left(L_j \cos\left(\sum_{i=1}^{j} q_i \right) \right) \\ \sum_{j=1}^{n} \left(L_j \sin\left(\sum_{i=1}^{j} q_i \right) \right) \end{bmatrix}$$

By taking the Jacobian of this transformation, we are able to derive an expression for the motion of the tip basis. After differentiating and rewriting in matrix form, we obtain,

$$\begin{bmatrix} \dot{q_T} \\ \dot{x_T} \\ \dot{y_T} \end{bmatrix} = \begin{bmatrix} 1 & \dots & 1 \\ -\sum_{j=1}^{n} \left(L_j \sin\left(\sum_{i=1}^{j} q_i \right) \right) & \dots & -L_n \sin\left(\sum_{i=1}^{n} q_i \right) \\ \sum_{j=1}^{n} \left(L_j \cos\left(\sum_{i=1}^{j} q_i \right) \right) & \dots & L_n \cos\left(\sum_{i=1}^{n} q_i \right) \end{bmatrix} \begin{bmatrix} \dot{q_1} \\ \dot{q_2} \\ \vdots \\ \dot{q_n} \end{bmatrix}$$

This expression will be used in subsequent sections to compute tip motion dynamics from the motion of each of the joints.

For inverse kinematics for three-link robot, we are given the tip path. With preset robot arm lengths, we should able to compute at least one solution for the relative angles between the arms, q, thus the total orientation of the robot. There are multiple perspectives in terms of how to compute these relative angles, which include an algebraic, geometric and iterative method. The orientation and position of the third arm of a three-link robot is determined by the path that the total arm is tracing out. Thus, the only degrees of freedom we have are in the orientation of the first and second arm. The beginning of the first arm must be bound to the origin and the end of the second arm must be bound to the beginning of the third arm. Thus, with constant arm lengths, we essentially have less than 2 points of intersection between the first two arms.

The algebraic perspective of solving inverse kinematics takes us back to explicitly solving for the points of intersection between two circles, drawn out by the first and second arm. Take two circles expressed as,

$$(x - x_1)^2 + (y - y_1)^2 = r_1^2$$
$$(x - x_2)^2 + (y - y_2)^2 = r_2^2$$

Logically, the circles could intersect at one or two points. A single point if the arms can barely brush each other when extended out, or two points if the area swept by the arms overlap.

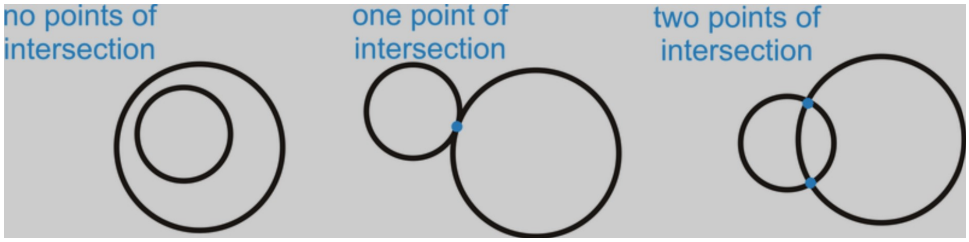

Figure 5: Possible Types of Circle Intersections

There is also the possibility that the circles do not intersect at all because their lengths are too long. However, we ignore this case because the paths we trace out are all within reaching distance. There is also a very small chance that the two circles completely overlap, however that would defeat the purpose of having a three-arm robot because we could reduce it down to a single-arm robot. From coordinates of the intersection points, we could directly solve for the orientation angles, q, of the three arms.

The geometric solution to inverse kinematics would involve a similar procedure, but instead of solving for intersection points, we would be solving for the orientation angles that would allow for the first two arms to touch.

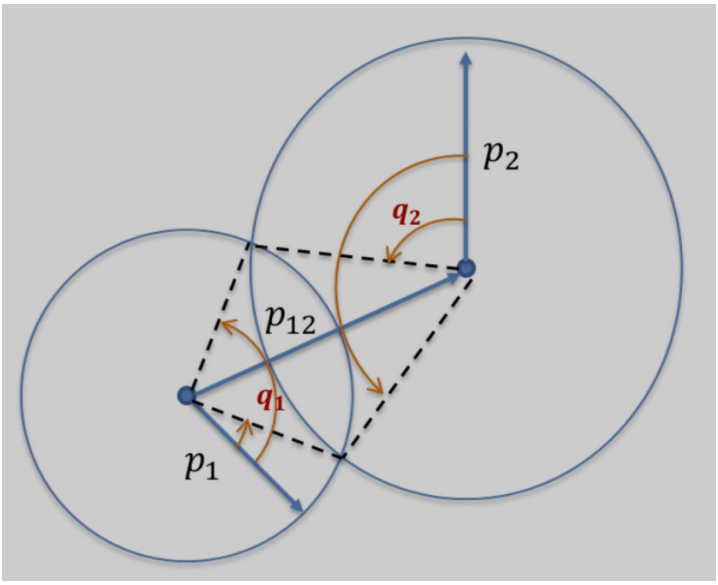

Figure 6: Geometric Approach to Inverse Kinematics

From the above image, we see that if we have two points of intersection between our robot arm traces, we could compute the angle between the first arm and the second arm, q_1. Through further geometry, we can compute q_2 and q_3 as well, which would be the orientation angles of the three arms.

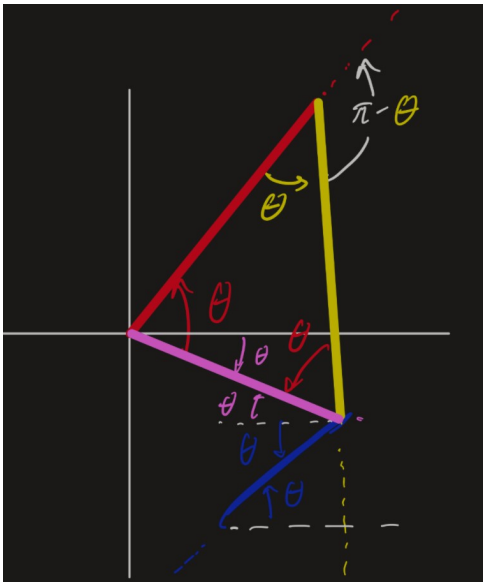

Figure 7: Geometric Reduction of 3-Arm Robot

The iterative solution to inverse kinematics would essentially be solving a minimization problem of the distance from the tip to the desired path. Though this is the most computationally inefficient method of solving the inverse kinematics, we would not have to worry about imaginary solutions resulting from robot arms not intersecting in the algebraic and geometric perspective. By setting a maximum iteration step, we can obtain arm orientation angles as close to the desired path as possible, even if its out of reach.

3.3 Problem 3

For problem 3, our goal is to computationally implement a basic forward and inverse kinematic function for a three-link robotic arm. We decided to utilize the geometric perspective to compute the inverse kinematics, which relied heavily on our understanding of Figure 6.

In our test file, we implemented both a random angle generator as well as a sinusoidal curve. Of course, our functions works very well for the traditional normal orientation of the robot tip. So we further explored the other possible orientations of the robot tip, testing out a tangent orientation,

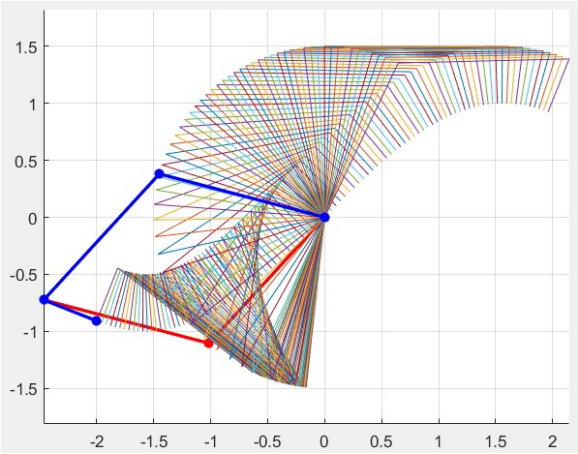

Figure 8: Tangent Sinusoidal Trace

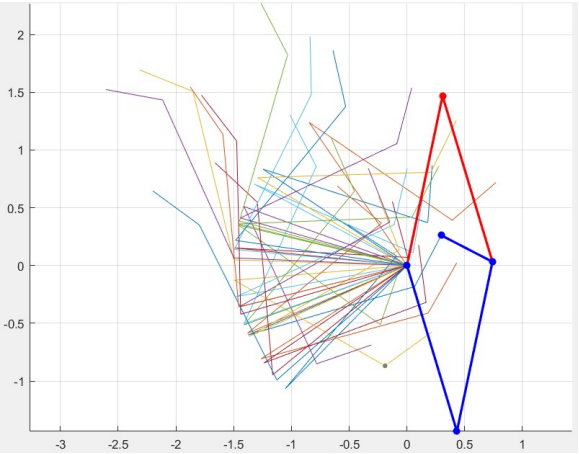

Figure 9: Tangent Random Trace

When we implemented the object `robot` in our forward and inverse kinematic code, we defined the following attributes: `robot.q`, the robot arm orientation vector, `robot.P`, the robot arm zero configuration vector, `robot.T`, the robot arm transformation from the origin to the tip, and `robot.J`, the robot arm Jacobian that allows us to compute the tip velocities from the angular velocities of each individual arm.

3.4 Problem 4

Diving deeper into the mechanics of the inverse kinematics code, the mechanics that I wrote depend heavily on Figure 7. After computing the position and orientation of the third robot arm from the leftwards normal perpendicular constraint, I was able to use vector geometry to numerically determine the colored angles in Figure 7. I was also able to differentiate between the "Elbow Up" and "Elbow Down" orientations based upon my drawings,

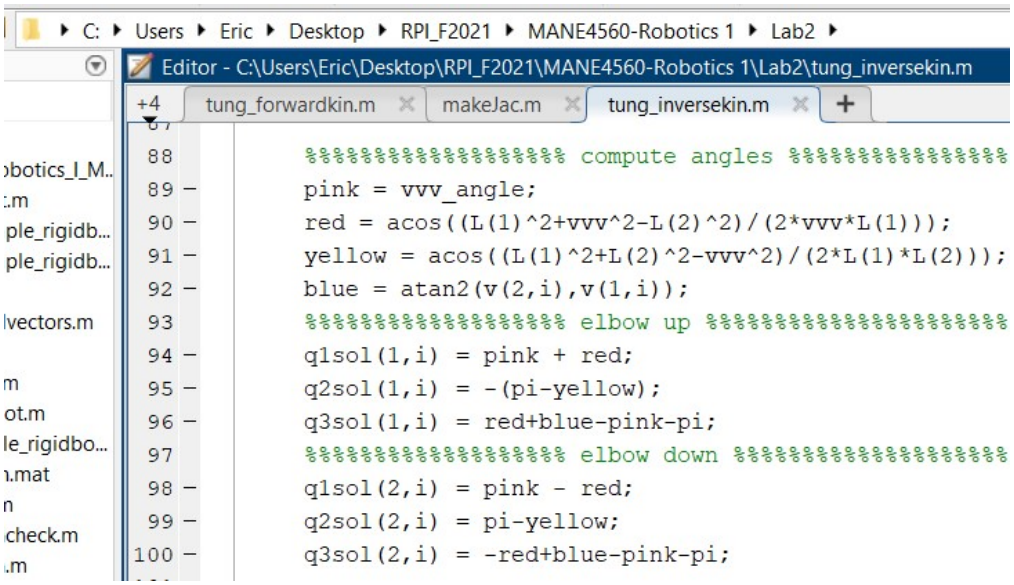

Figure 10: Code Implementation of Colored Angles in `tung_inversekin.m`

Implementing the final version of our inverse kinematic code, we obtained the following plots,

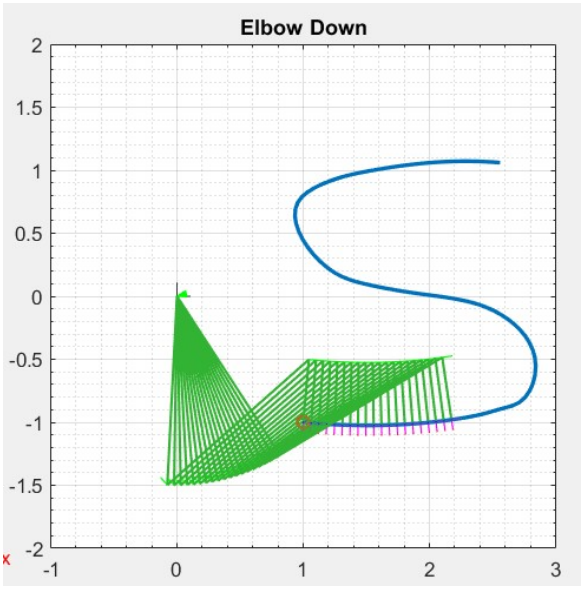

Figure 11: Elbow Down Rigid Body w/ Path Start

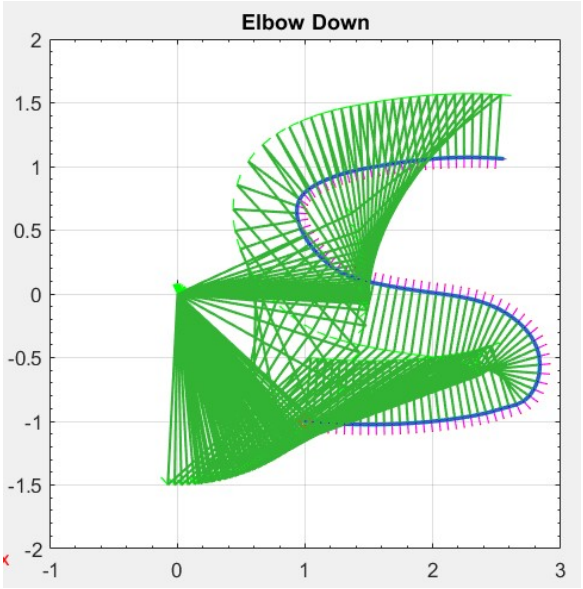

Figure 12: Elbow Down Rigid Body w/ Path End

We also implement a clean, second plotting format with the "Elbow Up" configuration,

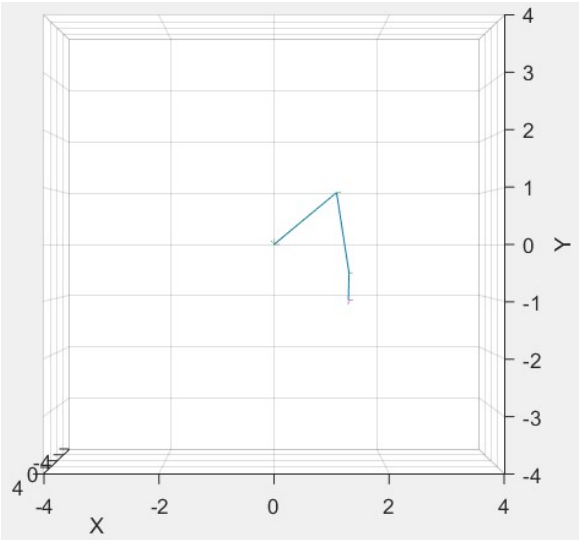

Figure 13: Elbow Up Rigid Body w/ Path Start

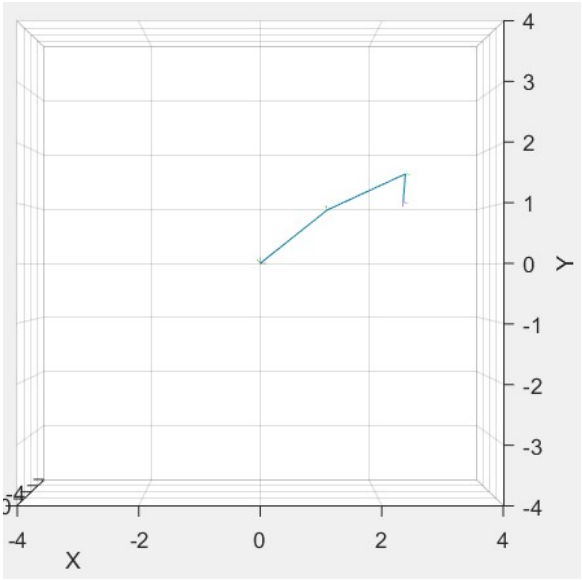

Figure 14: Elbow Up Rigid Body w/ Path End

3.5 Problem 5

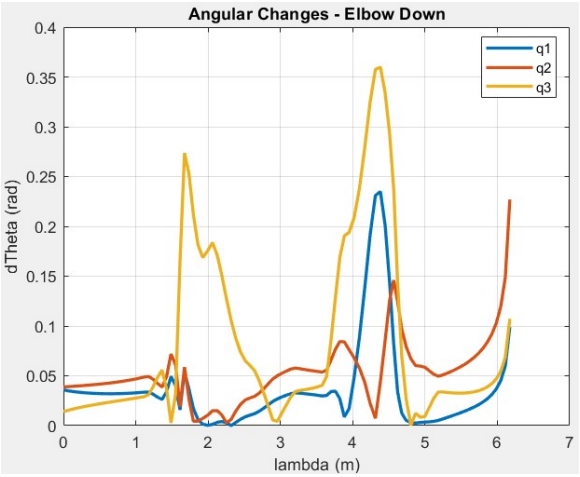

Figure 15: Elbow Down Change in Theta vs Lambda

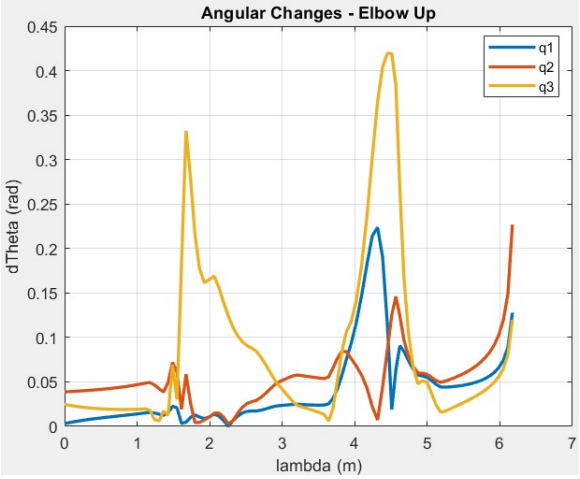

Figure 16: Elbow Up Change in Theta vs Lambda

From the delta lambda data and angular constraints, we are able to compute a total path travel time and create a plot of the angular velocities of the joints, which are just scaled by inverse time relative to the previous plots,

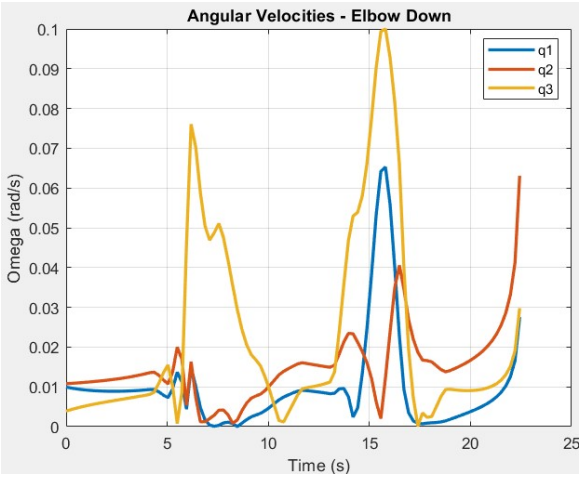

Figure 17: Elbow Down Omega vs Lambda

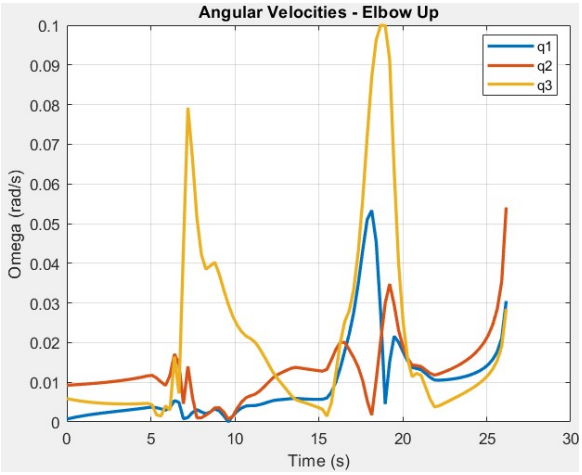

Figure 18: Elbow Up Omega vs Lambda

With each arm constrained to limit of 1 radian per second, we can see that the angular velocities for all the joints are below the constraint in previous figures. The speed of both of these orientations are limited by the angular constraint of the third joint. The "Elbow Down" and "Elbow Up" orientations were completed in 22.4 seconds and 26.2 seconds, respectively.

For the following observations, we are going to focus on the "Elbow Down" orientation. With regards to the lengths of the robot arms, I ran a simple test of increasing the length of the first robot arm, L_1, to 2 meters. Through simulated this 33% increase in a single robot arm length increased the trace time by almost 37% (from 22.4 seconds to 30.7 seconds). When looking at the Jacobian between the tip trace motion and the angular speeds of our arm joints, we see that almost all of the terms increase in magnitude with an increase in a single arm length. Thus, while keeping our angular constraint constant, this would force our robot arm to travel to its next position in a larger amount of time to compensate for a larger arm length.

With regards to the positioning of the letter S, I ran a test of shifting the entire S curve up, down, left, and right. I also obtained the plots of joint angle, q, versus time.

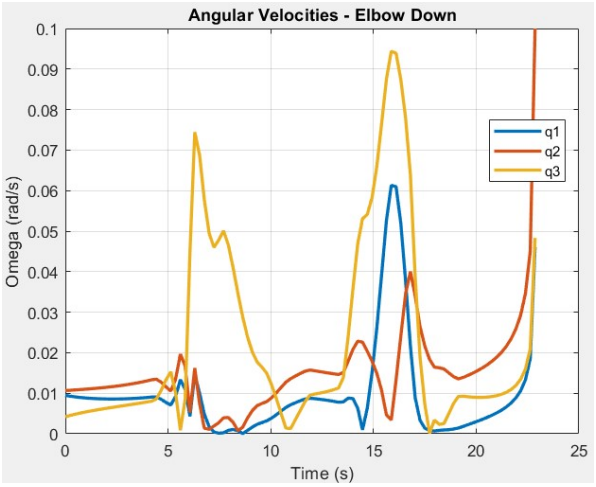

Figure 19: S-curve 4 cm Up, $t_f = 22.9$

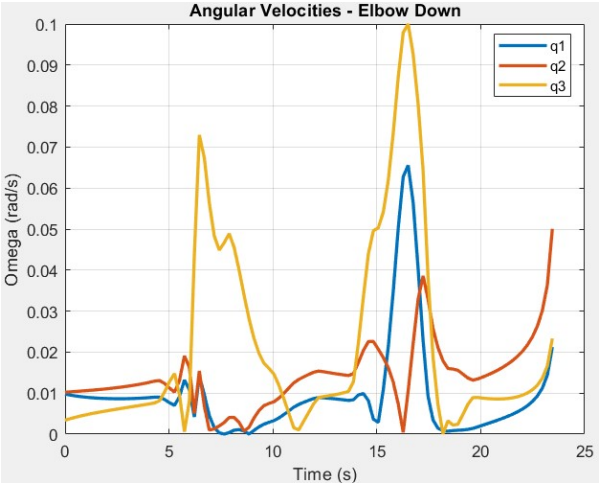

Figure 20: S-curve 4 cm Down, $t_f = 23.4$

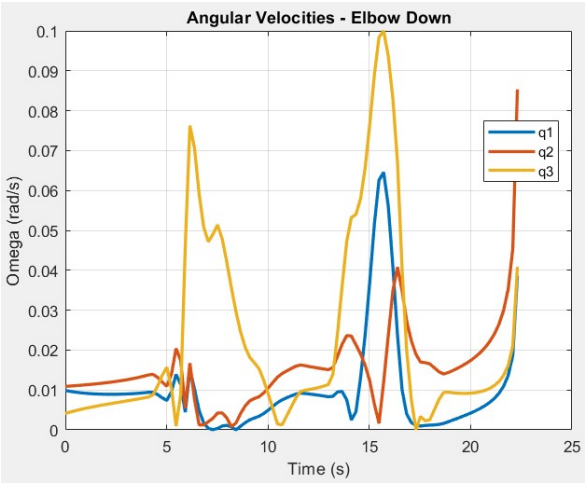

Figure 21: S-curve 2 cm Right, $t_f = 22.3$

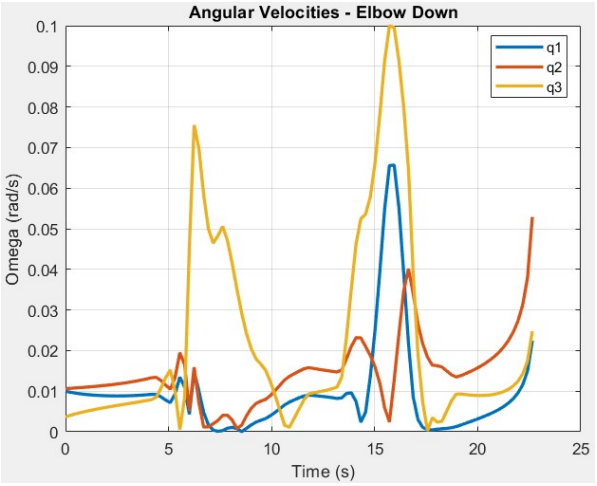

Figure 22: S-curve 2 cm Left, $t_f = 22.6$

From our initial observations, we see that shifting the S curve to the right decreases our travel time. Conceptually, the farther away the S curve is to the right, the less all of the joints have to turn in order to trace the path. Thus, with a lower change in angle per step in lambda, we obtain a smaller trace time and a larger maximum path speed with the same constraint in our angular velocity.

After further observations, we see that most of the angular velocity plots have shown the third joint, q_3, to be the constraining joint. This can be seen with the yellow curve being the only curve to approach 0.1 radians per second, our pre-defined constraint. However, when shifting the S curve up, our constraining joint now becomes the second joint, q_2, as demonstrated with the red curve in Figure .

4 Conclusion

This report has derived and highlighted the mechanics of forward and inverse kinematic algorithms for implementation in n-link robotic arms. These computed orientation solutions executed as expected, which allowed us to explore other possible orientations and configurations. The objectives of this projects were to implement these algorithms for paths of various shapes, lengths, and mechanical constraints, whilst allowing us to understand the benefits and detriments to using specific algorithms.

In the future, more dynamic orientation vectors for our robotic tip can be implemented alongside more n-links. Moreover, a stronger focus on collision detection and individual homogeneous transforms would allow us to simulate mechanics ideal for free-space travel objectives such as multi-axial milling. Further optimization of path tracing is essential to further improving humanities need for more dynamic, universal, and complicated manufacturing methods. Implementing these algorithms in a three-dimensional case would further improve the applicatio of our studies.

5 References

(1) Wen, J. T. (2021). "MANE 4560 - Mini-Project 2: Planar robot Arm Motion" Rensselaer Polytechnic Institute