CS 3630 Project 4

Name: Matthew Yang
GT email: mattyang@gatech.edu
GT username: myang349
GTID: 903447357

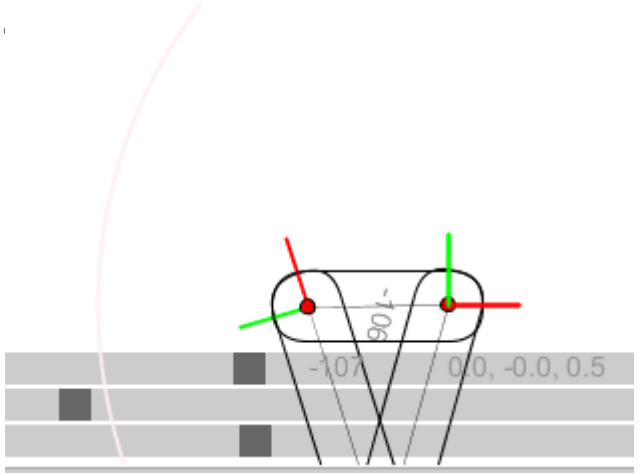
1. Give a short overview of how the simulator works. (Your explanation does not have to be very detailed at this point)

The simulator simulates a three-link robot arm. The user can only control three things. These are: the angle from frame0 to frame1, the angle from frame1 to frame2, and the angle from frame2 to frame3.

Then, the simulator computes the correct location/orientation of the robot arm by using the forward kinematic map. The forward kinematic map gives the position and orientation of the end-effector frame as a function of the joint variables.

Ultimately, this means that the simulator works by taking the three angles mentioned above as user input, and then calculating the corresponding robot arm.

2. Screenshot and paste the robot arm with the end effector frame and base frame aligned. How did you achieve this? What were the main difficulties in achieving the goal?

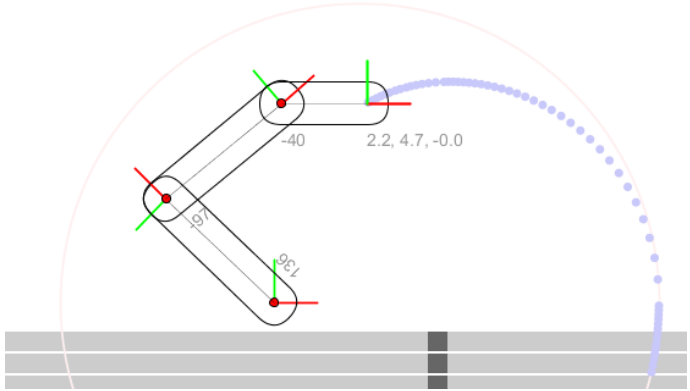


I achieved this by first controlling joint 2 and joint 3 so that the ORIGIN of the end effector frame overlapped with the ORIGIN of the base frame. From there, I controlled joint 1 to adjust the orientation of the end effector frame. This works because changing joint 1 when the origins are overlapped does not change the position of the end effector frame, only the orientation.

The main difficulty in achieving the goal was understanding how the controls worked. For me, it would be more intuitive if the slider went from [0, 360] instead of [-180, 180].

3. Screenshot and paste the robot arm with the joint-space control scheme. Make sure to include the trail. What did you do to implement this? Why does the trail look “arcsy”?

<Insert Screenshot here.>



To implement the joint-space control scheme, first I calculated the error using the equation on L11, Slide 14, which is $e_t = q_d - q_t$. To find the error for each joint angle I used a for loop to perform element-wise subtraction.

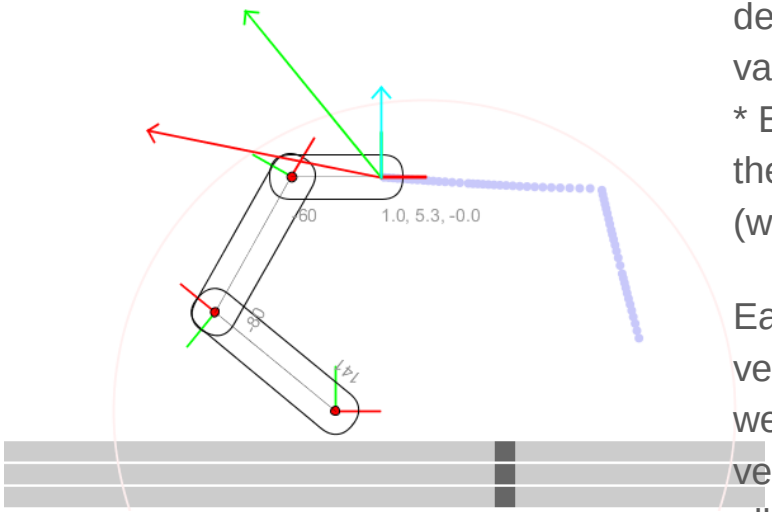
Next, I used the feedback law on the same slide, which is $q_{t+1} = q_t + K_p(q_d - q_t)$, again doing element-wise operations. Lastly, returning q_{t+1}, or “res”, results in the desired movement.

Note: during both steps, I used theta = (theta + pi) % 2pi – pi to ensure the angles were within the bound [-pi, pi].

The trail looks “arcsy” because our proportional feedback control is done in joint space, but we are visualizing it in cartesian.

Basically, the shortest path should be a straight line. But the shortest path (a line) in joint space (in which we are operating) causes a curved path in cartesian space.

4. Screenshot and paste the robot arm with the cartesian control scheme. Make sure to include the trail. What did you do to implement this? What is the role of th e inverse Jacobian here?



To implement this, first I called the given inverseJacobian3 method, using q as the param. Next, I calculated the position error by subtracting the tool coordinates from xd, yd, and I set the angle error = tool.theta() – theta_d or theta_d – tool.theta() depending on which subtraction result had the lowest absolute value. Lastly, I performed matrix multiplication w/ inverseJacob * Error, multiplied by Kp (proportional control), and updated via the formula

$$q_{t+1} = q_t + K_p J(q_t)^{-1} E_t(q)$$

(where res = qt+1)

Each column i of the manipulator Jacobian represents the velocity corresponding to a change in joint angle qi. Therefore, we can multiply the inverse Jacobian by any given end-effector velocity, to get the corresponding joint space velocities. This allows us to use proportional control as before, following a gradient-descent like algorithm.

5. Screenshot and paste the result of the unit tests

```
[Unit Test Passed!] testProportionalJointAngleControllerSingleStep
[Unit Test Passed!] testProportionalCartesianControllerSingleStep
```

6. Discuss what you have learned from this project.

I have learned about the Processing application and gotten a good chance to brush up on my (semi-forgotten) Java skills.

Also, since I've been a bit behind in class, this project really helped solidify my knowledge about how to do both Joint-space Motion control and Cartesian Motion Control, and the relationship between the two.

By reading over the pre-written project code, I also learned how various algorithms/methods such as forward kinematics, Jacobian calculation, and QR factorization can be implemented.

It was also really cool to see something that I learned in Calculus 3 (Jacobian and derivatives) being used for the first time in a computer science class.

Overall, this project was a lot less painful than the previous ones, and I think I have a solid understanding of how RRR robot arms work.