

CS 3630 Project 4

Name: Fei Ding

GT email: fding33@gatech.edu

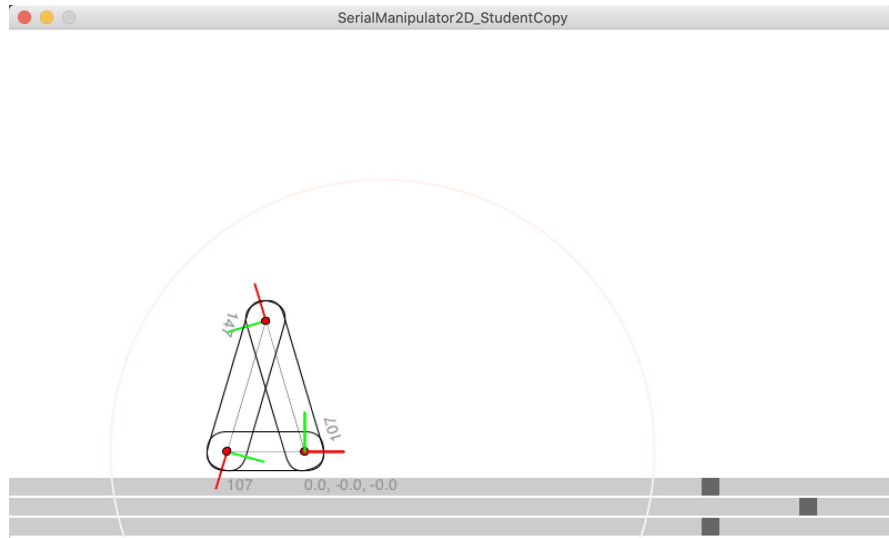
GT username: fding33

GTID: 903444656

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

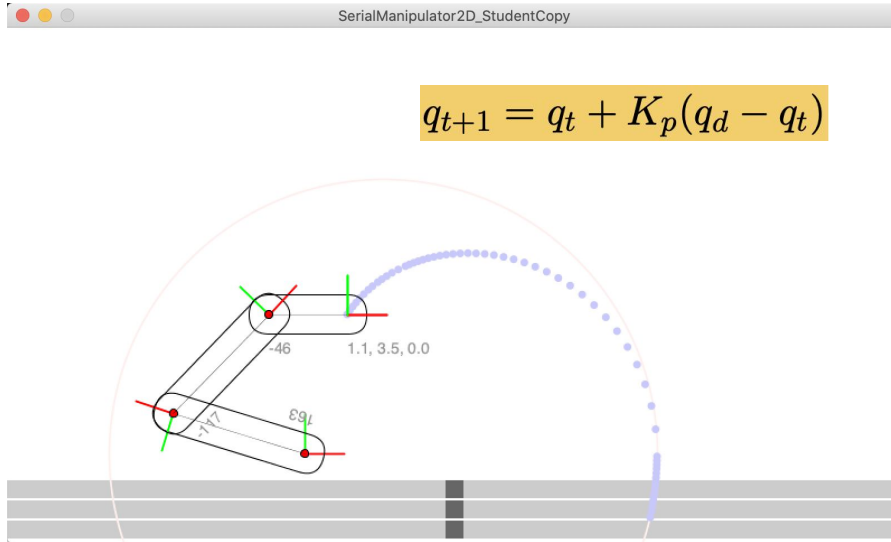
The robot arm in simulation is of RRR type which consists of three fixed-length links rotating in 2-d space. The three joint angles translate to three degrees of freedom in euclidean space: x , y , and the orientation of the end-effector θ . The user can manipulate with it in two ways: specify the joint angles by sliding the squares at the bottom or click on the screen. We attach a coordinate frame to the end of each link and then do forward kinematics as the three joint angles determine the pose of each link and thus where the end effector lands. The other way to control is implemented by the students. After clicking a point in place, that point is inversely transformed to target joint angles. Then we must implement the proportional joint angle controller and the Cartesian controller to make the robot arm moves to desired position and orientation by incrementally changing the joint angles to approximate a curved or straight-line path to the desired point.

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?



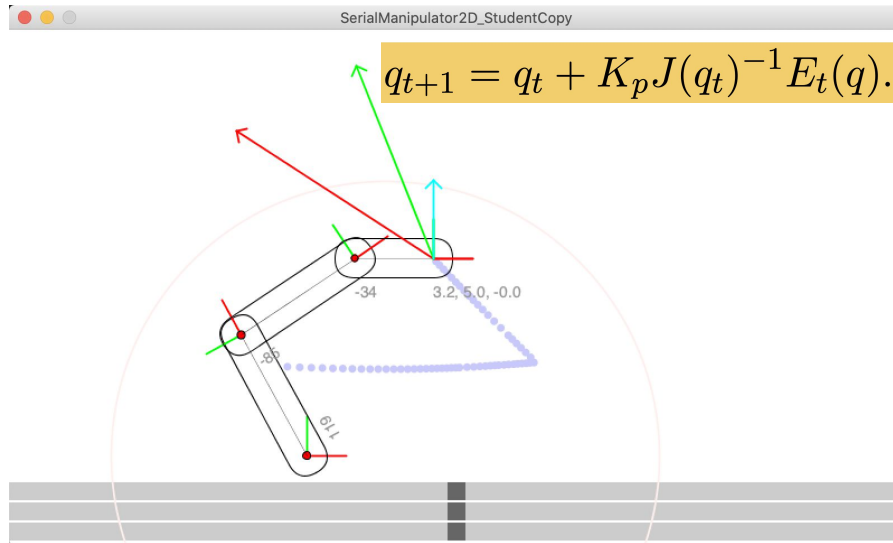
With three revolute joints we must make an isosceles triangle by toggling the three angles. Given two rods are of same length, we can precisely get the three angles using the sine law: 106.6, 146.8, and 106.6. The actual hard part for me was to get these angle values as close as I can with the these slide bars. Also, there is another elbow-down solution, but I did not take that because that would display out of screen.

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”?



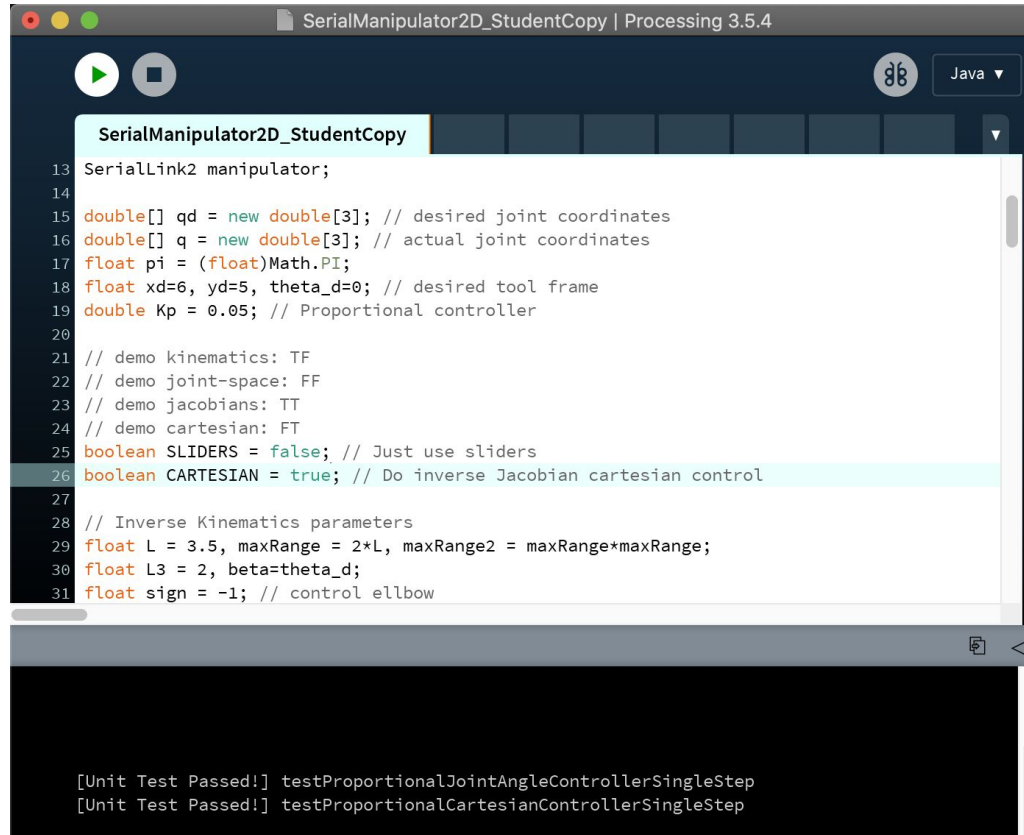
I used the formula listed on the left. To implement given an array of q_d and q_t , I wrote a for-loop to subtract each component of q_t from q_d , multiply that with K_p , and add back to q_t to form that component of q_{t+1} . Also need to notice the error $q_d - q_t$ should be controlled between $[-\pi, \pi]$. The shortest path in joint space corresponds to curve in Cartesian space after transformation. That's because the joint angles, not the Cartesian x-y, are linearly interpolated.

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 the inverse Jacobian here?



I used the formula listed on the left. The inverse Jacobian was provided and I only had to implement $E_t(q)$ which is the error in Cartesian space. Inverse Jacobian will convert it into error in the joint space which becomes compatible to the formula we used for proportional joint angle control. Notice we still have to make angles between $[-\pi, \pi]$. The inverse Jacobian here transforms the Cartesian space back to joint space.

5. Screenshot and paste the result of the unit tests



The screenshot shows the Processing IDE interface. The title bar indicates the file is 'SerialManipulator2D_StudentCopy' and the version is 'Processing 3.5.4'. The code editor displays the following code:

```
13 SerialLink2 manipulator;  
14  
15 double[] qd = new double[3]; // desired joint coordinates  
16 double[] q = new double[3]; // actual joint coordinates  
17 float pi = (float)Math.PI;  
18 float xd=6, yd=5, theta_d=0; // desired tool frame  
19 double Kp = 0.05; // Proportional controller  
20  
21 // demo kinematics: TF  
22 // demo joint-space: FF  
23 // demo jacobians: TT  
24 // demo cartesian: FT  
25 boolean SLIDERS = false; // Just use sliders  
26 boolean CARTESIAN = true; // Do inverse Jacobian cartesian control  
27  
28 // Inverse Kinematics parameters  
29 float L = 3.5, maxRange = 2*L, maxRange2 = maxRange*maxRange;  
30 float L3 = 2, beta=theta_d;  
31 float sign = -1; // control elbow
```

The output window at the bottom shows the results of unit tests:

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

6. Discuss what you have learned from this project.

I learned a handful set of skills in this project. The Java Applet is quite outdated technology but it still facilitated my learning with this cool simulation of RRR robot arms. I learned to perform forward kinematics by manipulating with the slider bars and then predict the end-effector position and orientation. I did manual inverse kinematics by hand-calculate some trigonometry equations. I learned two practical motion control schemes that avoid direct inverse kinematics but use very good interpolations to approximate a path for the robot arms. They are now very intuitive for me to derive. I learned the importance of K_p , the “learning rate”, is critical for the motion control to prevent overshooting and slow convergence. The Inverse Jacobian had taught me transformations and relationships between the joint space and the Cartesian space. Overall, what I have learned here is a good foundation to more advanced robotics topics.