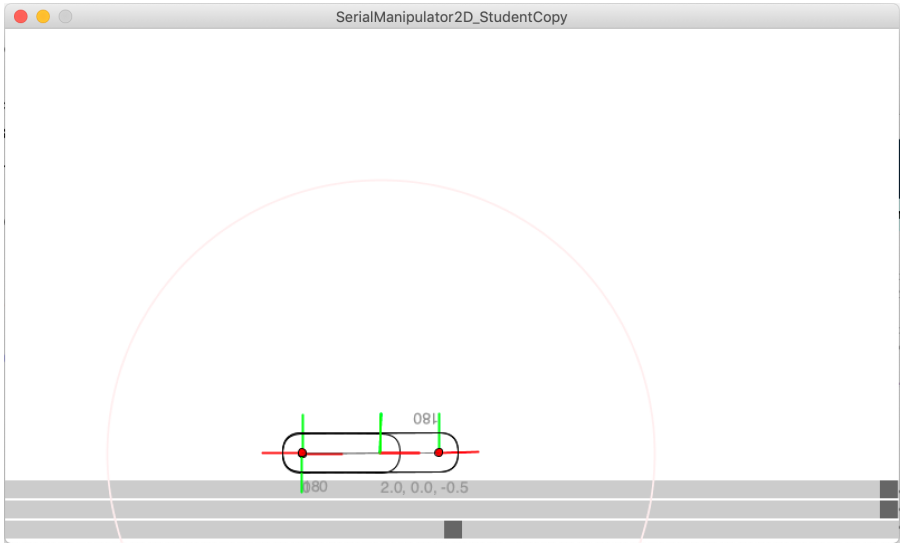
# CS 3630 Project 4

Name: Yiyang Wang
GT email: ywang3420@gatech.edu
GT username: ywang3420
GTID: 903408141

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

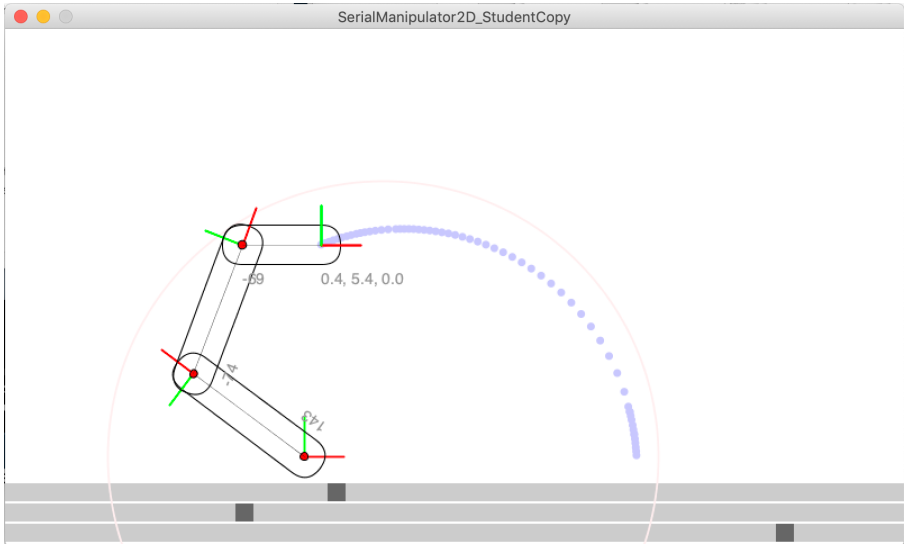
The simulator works by drawing robot links that have poses(x, y, $\theta$) relative to each other.  By scrolling the sliders, the theta of the corresponding link changes and therefore changes the poses of the links. The simulator has multiple supporting java classes to help it create links and change their positions.

# 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?



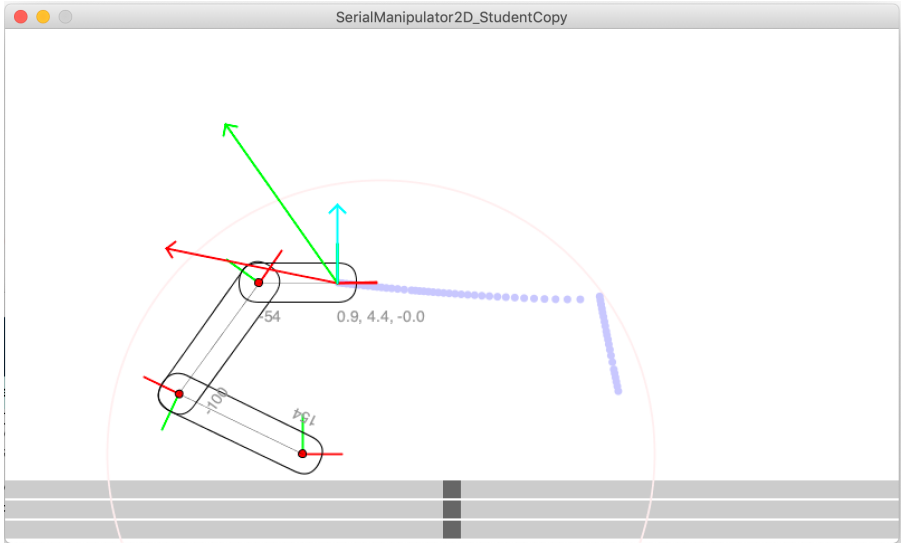

I achieved this by moving the slider to change the theta for joint 2 and joint 3 to 180 degrees. The difficulty in doing this is when moving the sliders to explore how to align them and when two links overlaps, it is hard to tell which frame is the end effector frame.

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”?




First I calculate the joint space error using equation et = qd – qt. I then calculate the next joint angle position using the equation: res = qt + Kp * et. The trail is “arcsy” because the joint space error calculates the difference between two joint angles, so it is like drawing circles. There is also a non-linear kinematic map to convert Cartesian space to joint 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?




I first calculate the inverse Jacobian using the inverseJacobian function in SerialLink2. I then calculate the position error Et by subtracting the x, y, theta in desired pose of end effector to those in current pose of end effector. Finally I use the equation: res = qt + Kp * inverseJacobian * Et. The role of the inverse Jacobian is to calculate the joint space velocities corresponding to a given end-effector velocity, so that we could eventually achieve a desired trajectory in Cartesian space.

# 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 learn how to apply the equations for joint-space control and Cartesian control in class. By visualizing the two controls, I could vividly see their difference in trajectory and how they handles them. I could see how differently the two controls change when the poses of the end-effector changes. For example, the change of the arrows at end-effector in Cartesian control and how they are always tangential to circle of joint velocities.