

## CMPUT 312 – Lab 3 Report

**Group members names & ccids:** Justin Valentine (jvalenti), Yash Bhandari (yashaswi)

### 1) Path Planning:

Code in `Driver.draw_line` in `driver.py`

For small changes in the arm angles ( $x$ ), we can estimate the corresponding change in the end effector position ( $y$ ) by multiplication with the Jacobian:

$$J\Delta x \approx \Delta y$$


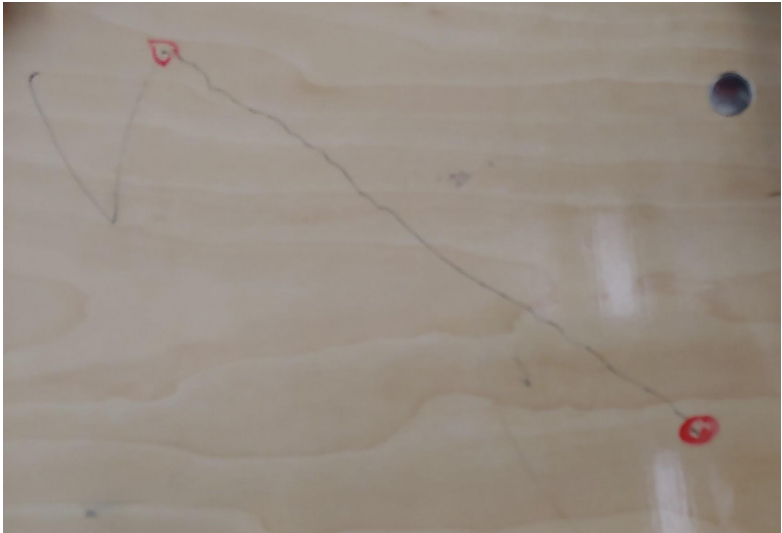
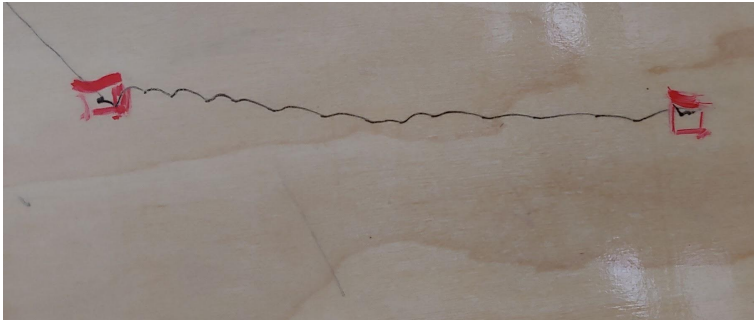
We can solve for the change in the arm angles  $\Delta x$  by inverting the Jacobian:

$$(a) \quad \Delta x \approx J^{-1}\Delta y$$

Our strategy to draw a line from  $y_i = (y_{i1}, y_{i2})$  to  $y_f = (y_{f1}, y_{f2})$  is as follows:

1. Use inverse kinematics to move the end effector to  $y_i$
2. Fix  $\Delta y = (y_{f1} - y_{i1}, y_{f2} - y_{i2})$
3. Compute the Jacobian at the current location
4. Find the change in  $x$  required to move the end effector in the direction of  $\Delta y$  using equation (a)
5. Scale  $\Delta x$  so that  $|\Delta x| = \text{step\_size}$  and move the actuators by  $\Delta x$ . We chose  $\text{step\_size} = 5^\circ$ .
6. Repeat steps 3-5 until the end effector is close enough to the desired end point.

On the next page are a few sample lines drawn using our program. The lines that are generated are fairly jagged and appear to be composed of many small arcs. These small arcs are the individual steps from step 5. This could be mitigated by adjusting the relative speeds of the joints so that they match  $\Delta x$ . For example, if  $\Delta x = (2, 1)$ , the first joint should move at twice the speed of the second joint.

Line Start (x,y) cm	Line End (x,y) cm	Path
10,0	5,10	
15,0	0,15	
12,15	0,15	

## 2) Uncalibrated Visual Servoing (UVS):

The code for visual servoing is contained in visual\_servoing.py

[Demo video](#)

### Method For Initializing Jacobian:

(1) Move Joint 1 through  $\theta_1$

(2) Read visual change of the camera coordinates  $(u, v)$  with respect to  $\theta_1$ :

$$J_{1,1} = \frac{\partial u}{\partial \theta_1} \approx \frac{\Delta u}{\Delta \theta_1}, J_{2,1} = \frac{\partial v}{\partial \theta_1} \approx \frac{\Delta v}{\Delta \theta_1}$$

(3) Move Joint 2 through  $\theta_2$

(4) Read visual change  $(u, v)$  with respect to  $\theta_2$ :

$$J_{1,2} = \frac{\partial u}{\partial \theta_2} \approx \frac{\Delta u}{\Delta \theta_2}, J_{2,2} = \frac{\partial v}{\partial \theta_2} \approx \frac{\Delta v}{\Delta \theta_2}$$

### Analysis of method:

There is error associated with our measurements of  $(u, v)$  because our tracking point jumps around, we mitigate this partially by averaging 20 tracking points across 2 seconds everytime we measure  $(u, v)$ . However the average is biased towards some directions depending on lighting conditions, this introduces error into our jacobian approximation. There is also intrinsic error in the method we use to approximate the jacobian. We used linear approximations to find our jacobian indices so we are ignoring all other terms in the Taylor expansion. For small theta this approximation should be close to the true value because the movement is approximately linear.

### Method For Visual Servoing:

(1) Solve for motion  $(\Delta x)$ :

$$(\bar{y} - y_i) = J_i \Delta x$$

(2) Normalize and scale joint movement:

$$s = \mu(\Delta y) \frac{\Delta x}{\|\Delta x\|}$$

(3) Move robot through  $s$ :

$$x_{i+1} = x_i + s$$

(4) Find change in end effector position:

$$\Delta y = y_{i+1} - y_i$$

(5) Update Jacobian (Broyden update)

$$J_{i+1} = J_i + \gamma \frac{(\Delta y - J_i \Delta x) \Delta x^T}{\Delta x^T \Delta x}$$

### Analysis of method:

Let  $x = (\theta_1, \theta_2)$  be the joint angles of the robotic arm. Let  $y = (u, v)$  be the pixel coordinates of the end effector in the image frame. And let  $\bar{y} = (u, v)$  be the pixel coordinates of the goal in the image frame.

In our implementation we found  $\gamma = 1$  worked well as a scaler on the Broyden update.

We also found that scaling  $\frac{\Delta x}{\|\Delta x\|}$  by a function of the distance to the goal  $\mu(\Delta y)$  gives us the behavior that as we get closer to the goal, the movements become smaller. We implemented this by using a simple piecewise function  $\mu(\Delta y) = \{10 \text{ if } \Delta y \geq 30, 5 \text{ if } \Delta y < 30, 2 \text{ if } \Delta y < 15\}$ . We found that this behavior gave us more reliable results, then just using the value  $\Delta x$  we solved for in (1).

We found that subdividing the path was not necessary for achieving reliable visual servoing. However, if we wanted to make our method more robust we could implement this feature. One way of doing this could be by reinitializing the jacobian along our path after a certain distance has been covered.

### 3) Visual Servoing In 3D:

In order to perform visual servoing in 3d, you would need a second camera. The simplest set up would be for the second camera to be perpendicular to the first – the first camera would capture the xy plane, while the second camera would capture the yz plane. When finding the coordinates of the end effector, we would use the x and y coordinates from the first camera and the z coordinate from the second camera.

Instead of a  $2 \times n$  Jacobian (where  $n$  = the number of degrees of freedom the robot arm has), we would use a  $3 \times n$  Jacobian. When initializing the Jacobian, we use the second camera to measure the z coordinate and set  $J_{3,i} = \frac{\Delta z}{\Delta \theta_i}$ .