Lab 2: Jacobian and Inverse Velocity Kinematics

Pre Lab Due Date: 10/5/2021 @ 11:59pm

Code + Report Due Date: 10/12/2021 @ 11:59pm

This assignment consists of both a written Prelab (due Oct 5 at 11:59pm) and the main portion of the lab (code and report). You will submit all parts of the assignment through Gradescope. Late submissions will be accepted per the standard late submission policy, but they will be penalized by 25% for each partial or full day late. After the late deadline, no further assignments may be submitted; post a private message on Piazza *before the submission deadline* to request an extension if you need one due to a special situation such as illness. The Prelab is worth 5 points and must be completed and submitted individually, and the report and code are each worth 20 points. You may talk with other students about this assignment, ask the teaching team questions, use a calculator and other tools, and consult outside sources such as the Internet. **If you make use of outside sources, you are expected to cite them in your report.** When you get stuck, post a question on Piazza or go to office hours!

Note: this (like all future labs) is a partner assignment! You will complete and submit all Prelabs individually, but submitting the main lab individually is not allowed.

1 Prelab

The goal of the prelab is to connect ideas you are learning in class with the actual tasks you will work on in the projects. These assignments are "pen and paper" assignments in that you should not need to write any code, but if you would like to use digital tools to help explain your answers or improve your own understanding of the problems you are welcome to do so.

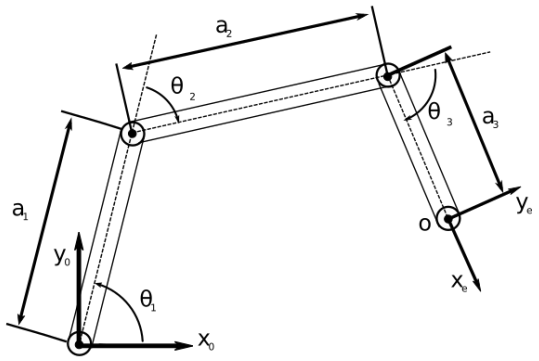

Figure 1: This is a planar robot arm with three revolute joints. Link lengths are specified by the variable a_i, and joint angles are represented by θ_i. The base frame is Frame $o_o x_o y_o$ and the end effector has Frame $o_e x_e y_e$. All angles are positive in the counter clockwise direction, as the arm is drawn, $\theta_1 > 0, \theta_2, \theta_3 < 0$

For the prelab we will once again be working with the same planar RRR robot. All angles are positive in the counter clockwise direction, as the arm is drawn, $\theta_1 > 0, \theta_2, \theta_3 < 0$. The prelab has 4 parts

1. Compute the full manipulator Jacobian of the end effector (position and orientation). In other words, compute J in the equation

$$\begin{bmatrix} \dot{o}_e \\ \dot{\theta} \end{bmatrix} = J \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix}$$

   where o_e is the position of the end effector frame's origin in world frame coordinates and θ is the orientation of the end effector frame as a counterclockwise rotation from the world frame.

2. Choose two configurations and manually determine what the end effector velocity would be if you rotate each joint one at a time. Check to see if your forward velocity kinematics computed with the Jacobian agrees.

3. Compute the inverse velocity kinematics, *i.e.*, determine the joint velocities which will produce a given desired end effector velocity. As with all problems, it's acceptable to define variables and use those in the final solution rather than writing everything out.

4. Consider the system of linear equations written in the form $Ax = b$, where A is (in general) a non-square matrix and x and b are vectors. Describe under what conditions the system has:

   (a) No solution

   (b) A unique solution

   (c) Infinite solutions

   You may express your conditions using concepts from linear algebra.

Submit the prelab on Gradescope. **You must complete and submit your prelab assignment individually**, and your writeup should be clear and concise.

2 Simulation Setup

To update your own private fork of the `meam520_labs` repo for Lab2 please do the following:

```
$ cd ~/meam520_ws/src/meam520_labs/
$ git pull upstream master
```

This is going to look at the public TA repository and grab the code stubs and launch files that we've added since the last lab. Occasionally when running this command, you might encounter a merge conflict. We don't expect that to occur, but if it does, don't hesitate to reach out to the TAs for help.

The algorithms you implement in labs will be stored in the `lib` folder:

```
$ cd ~/meam520_ws/src/meam520_labs/lib
```

since you are building a library of functionality which can be reused in future labs.

3 Manipulator Jacobian

In this part of the lab, you will calculate the full manipulator Jacobian of the Panda arm for the end effector.

3.1 Definitions, Dimensions, and Conventions

For full dimensions of the Panda arm, see the Lab 1 handout.

3.2 Implementation

You will need to implement your solution in `calcJacobian.py`. Open the `lib` folder using your preferred text editor, for example:

```
atom ~/meam520_ws/src/meam520_labs/lib/
```

and implement the `calcJacobian` function. You are free to define other functions to modularize your solution. *Note: you will likely need to use some of your forward kinematics functionality from the first lab. Consider adapting your* `FK` *class to add additional methods which expose intermediate information which may be helpful for computing your Jacobian.*

3.3 Testing

We recommend that you test your solution separately from the simulator first. Examine the bottom of `calcJacobian.py` to find a *very minimal* example of a test. To see the output from this simple test, run this script from the terminal, i.e.,

```
cd ~/meam520_ws/src/meam520_labs/lib/
python calcJacobian.py
```

You should perform more extensive testing yourself by adding to the tests done in `calcJacobian.py`. In addition, we've provided a visualization tool which will help you test your solution, similar to Lab1. First, launch the simulator with

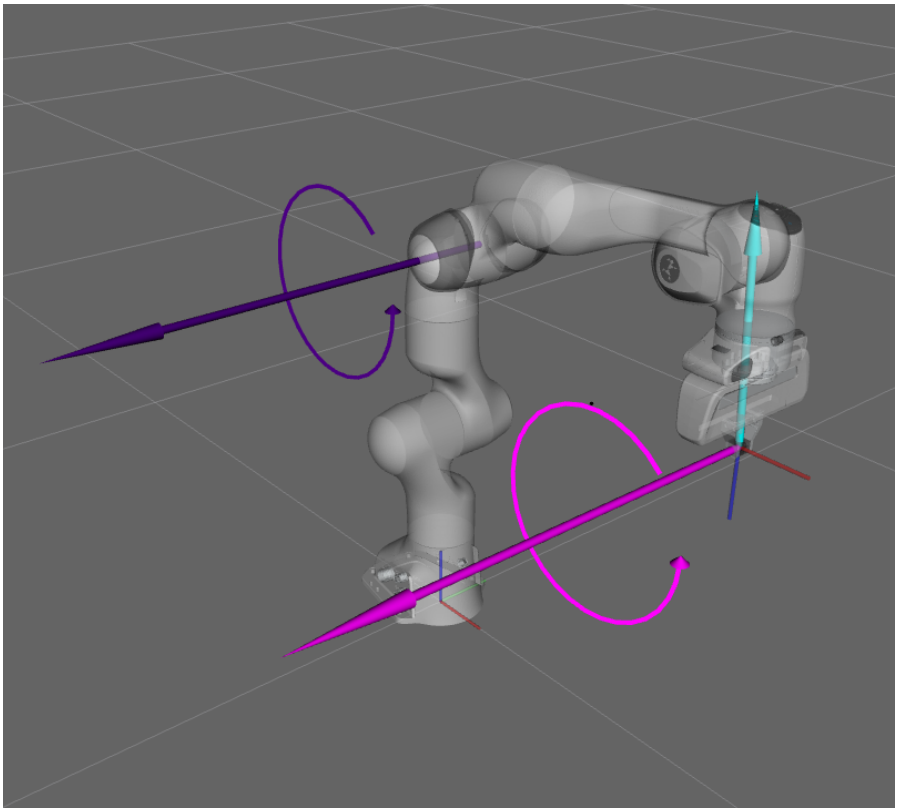

Figure 2: The visualization generated by the `visualize.py` script, showing the end effector linear velocity (in cyan) and angular velocity (in magenta) when moving a specific joint (in purple) at unit velocity for this configuration.

```
roslaunch meam520_labs lab2.launch
```

and then run the test script via

```
cd ~/meam520_ws/src/meam520_labs/labs/lab2
python visualize.py
```

This script will step through several configurations for the robot. For each, it will allow you to iterate through the joints, and moving only that joint at unit velocity. In RViz, the script will visualize the linear velocity (in cyan) and angular velocity (in magenta) of the end effector, as seen in Fig. 2.

4 Inverse Velocity Kinematics

Let's say that you are considering the position and orientation of the end effector, and would like the joint to move with a linear velocity $\mathbf{v}_e^0$ and angular velocity ω_e^0, both expressed in the world frame. The robot is currently in configuration q. What joint velocities should the robot execute to move joint i in the desired direction?

4.1 Implementation

You will need to implement your solution in `IK_velocity.py`. Open the `lib` folder using your preferred text editor, for example:

```
atom ~/meam520_ws/src/meam520_labs/lib/
```

and implement the `IK_velocity` method. You are free to define other class methods to modularize your solution. It is highly recommended that take advantage of your implementation of `calcJacobian`.

4.1.1 `nan` inputs

Instead of a decimal number, some entries of both the target linear and angular velocities may be `nan`, this stands for Not a Number. This input means that the particular coordinate of the end effector velocity should be left unconstrained, meaning it can be anything. For example, calling

`IK_velocity(q,v,np.array([np.nan,np.nan,np.nan]))`

would constrain the linear velocity to be `v` while leaving the angular velocity completely unconstrained.

4.1.2 No solutions

If the velocity IK problem has no solutions, i.e. there exists no joint velocity which will exactly produce the given target velocity, then you should choose the joint velocity which minimizes the l^2 norm of the error between the target velocity and the achieved velocity. The term for this sort of solution is the *least squares solution*.

4.1.3 Infinite solutions

If the velocity IK problem has infinite solutions, i.e. many choices of joint velocity will exactly produce the target end effector velocity, then you should minimize the l^2 norm of the joint velocities. The term for this sort of solution is also the least squares solution.

Hint: Take a look at the function `numpy.linalg.lstsq`, which may help in your implementation.

4.2 Testing

To test your inverse velocity kinematics, we provide the demo script `follow.py`, located in

`~\meam520_ws/src/meam520_labs/labs/lab2`

Like in Lab 1, the script has a variety of options for different demos. In `follow.py`, you will be using your inverse velocity kinematics to have the end effector trace out different patterns. Unlike in Lab 1, you will have to implement some of the demos yourself.

Each demo follows a periodic trajectory. All trajectories are centered at the end effector position corresponding to the neutral configuration of the robot. The trajectories are:

- `ellipse`: an ellipse in the y-z with respective semiaxes `ry` and `rz`, traversed with an angular frequency `f`
- `line`: a line of length `L` in the z direction, traversed in an sinusoidal oscillatory pattern with angular frequency `f`
- `eight`: a figure 8 in the x-y plane. The trajectory is computed using a Lissajous curve.

The `eight` trajectory has already been implemented for you, while your job is to implement the `ellipse` and `line` demos, which are controlled by functions of the same name in the `JacobianDemo` class in `follow.py`. Their inputs are the current time `t` and some shape parameters, and the output is the desired position and velocity for the end effector at that instant.

4.3 Trajectory Tracking

For this lab we have implemented a callback function which uses your trajectories and your velocity IK to track the trajectory. The callback function gets called automatically at about 1 kHz with the state of the arm. It then senses the configuration of the arm, uses your trajectory to calculate the desired position and velocity, uses a controller to calculate a desired end effector velocity to track the trajectory, and uses IK velocity to calculate joint velocities, before finally sending the commanded joint velocities to the robot.

You will not need to modify the controller which tracks your trajectories, but if you are curious, it implements a simple proportional controller on the cartesian position of the end effector, with a feed forward term from the velocity of the tracked trajectory:

$$v = v_{des} + k_p(x_{des} - x)$$

That desired linear velocity is then sent to your velocity IK.

4.3.1 Running the code

Once you are ready, relaunch the simulator with:

`roslaunch meam520_labs lab2.launch`

and try running

`python follow.py line`

This script will run the line callback until you press enter. Other options include `ellipse` and `eight`. The simulator tracking the `eight` trajectory can be seen in Fig. 3.

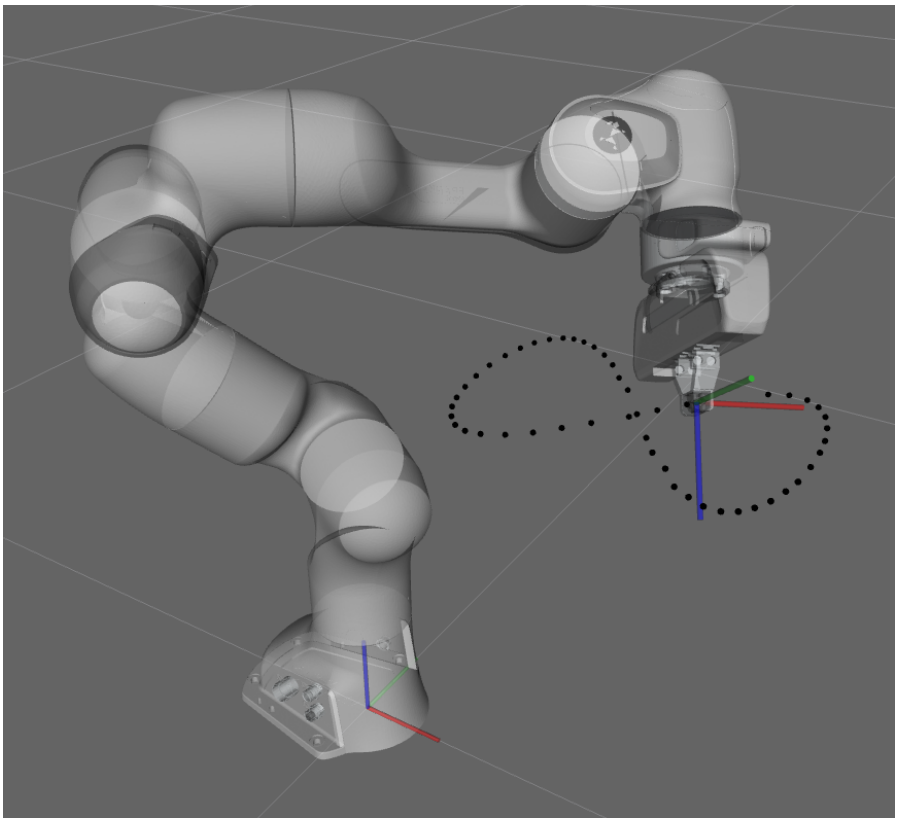

Figure 3: The robot tracking the `eight` trajectory using the `follow.py` script

5 Report

You will submit a written report describing your work in this lab. **The report must be no more than 8 pages.** You are free to use whatever typesetting workflow you are comfortable with, but we highly recommend using LaTeX. Handwritten submissions will not be accepted. If you need to sketch some diagrams by hand, this is okay. Your report should address all of the tasks listed in this section.

5.1 Methods

For the inverse velocity kinematics define the tasks your solutions perform. Describe the approach you followed to solve both the velocity IK problem and to calculate the Jacobian.

5.2 Evaluation

You should extensively test your solutions to ensure their correctness. In this section, document your testing process: how can you be confident that your solutions are free of bugs? Additionally, you should include the screenshots of the robot tracking the `line`, `circle`, and `eight` trajectories using the `follow.py` script.

5.3 Analysis

In this section, you will analyze the performance of your solutions and consider some extensions. Your report should comment on each of the tasks below.

5.3.1 Trajectory Tracking

Does the arm follow the expected trajectories? What happens if you set `kp` to 0 in the trajectory tracking code? Give a justification explaining the behavior you see.

5.3.2 Joint Trajectories

Let the robot track each trajectory in the `follow.py` script for about ten periods of the trajectory. Does the robot follow the same path in configuration space (joint angles) every time the end effector completes a cycle along the tracked

trajectory? If not, what issues might this cause in a practical scenario?

5.3.3 Target End Effector Pose

You can now control the world frame velocity of the end effector. Could you use this ability to move the end effector to a given target position in the world frame? Could you do the same for target orientation? Briefly describe what your approach might look like.

5.3.4 IK Velocity Issues

When might the Panda robot have trouble actually following the target joint velocities generated from your IK velocity solver? Describe how you might be able to modify the IK velocity solver to mitigate the impact of these issues.

6 Submission via Gradescope

- **Group Submission:** Each group should only submit once to Gradescope per Gradescope assignment. While going through the submission please ensure that you **add your group members to the submission**. Failure to do so could impact the grades of all parties involved.
- Please submit `calcJacobian.py`, `IK_velocity.py`, and any other files needed to run your code to the Lab 2 Code assignment on Gradescope. Once submitted, we will run automated tests to grade your code. You may submit as many times as you want, but any attempts to try to figure out what test cases we are running will result in your code getting a 0.
- Please submit your written report as a PDF to the Lab2 Report assignment on Gradescope.

IMPORTANT, please submit the correct thing to each assignment. Don't submit the code to the report assignment. Don't submit the report to the code assignment.