Homework #7
due Friday 11/17/22 11:59pm

We're going to wrap up the homework sets looking at the condition number of the Jacobian, as well as secondary tasks for redundancies and singularities. After this, it will be all projects!

One final time, we have posted a code package `hw7code` on Canvas. Use the launch files if you are tired of restarting all pieces separately. The package also includes a `matlab` folder, with updated `plotjointstates()` and `plotcondition()` functions. And note, to compute the condition number of the Jacobian, you can compute the SVD and divide the max/min singular values. Or use

```
cond(J)
```

in Matlab. Or in python, use

```
np.linalg.cond(J)
```

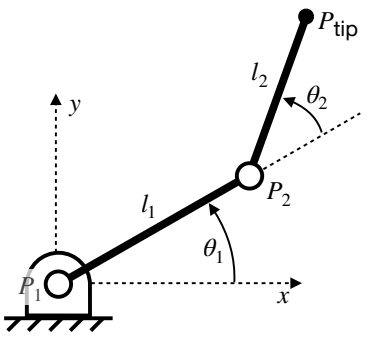
Also one last time, please submit the graphs/plots as well as the relevant code (the `Trajectory` class), together with any other requested information.

Problem 1 (Perfect Jacobian Condition Number) - 15 points:

Recall, the condition number (the ratio of largest to smallest singular value) has to be larger than 1. As discussed in class, it is very hard to have a Jacobian with a condition number of exactly 1. Here we try to construct such a case.

Consider the very simple 2 DOF robot shown to the right.

We will use the standard output space, being the (x, y) coordinates of the tip position P_{tip}. The link lengths are **unknown** and do not have to be equal. Indeed, you can't find a perfect condition number without adjusting the lengths!



(a) How do you have to specify the values $l_1, l_2, \theta_1, \theta_2$ to achieve a condition number of 1 for the Jacobian.

Before anyone starts deriving matrices, think about what this means:

- For the condition number to be unity, all singular values have to be the same. So $S = s\ 1$ is some scale times the identity matrix.
- As $J = USV^T$, the Jacobian has to be $J = s\ UV^T$, i.e. a scaled orthonormal matrix. That means the columns are orthogonal to each other and equal length!
- Place (for now) the tip at coordinates $(x = 0,\ y = 1)$.
- What does that make the first column of the Jacobian?
- What does the second column have to be?
- Where does P_2 (the location of the second joint) have to be to achieve this?
- What does this mean for $l_1, l_2, \theta_1, \theta_2$? Keep in mind, you could have placed the tip anywhere!

Problem 2 (Condition Numbers for the Touch and Go Movement) - 15 points:

Reusing the Problem 5 from last week's set, let us see "how good" the trajectory was. I.e. what where the condition numbers along the trajectory.

Keep in mind, we are looking at the condition number of the Jacobian, which combines translation and rotation. That is, it mixes units. So, to be fair, we should scale the first three rows (in units of m/s) by a length. That way, all rows will be in units of 1/s.

$$\bar{J} = \text{diag}(\frac{1}{L}, \frac{1}{L}, \frac{1}{L}, 1, 1, 1)\ J$$

Which begs the question of what length to use. This should be "an average length", so let's use the arm link lengths of 0.4m. The exact number will clearly change if we chose a different length, but the overall behavior will stay roughly the same.

So please copy over the code from last week. But we'll add the following updates:

(i) To be able to plot, we'll have the code publish the condition number on a new ROS topic under the name '/condition' and with the type 'Float64'. At the top, add

```
# Import the format for the condition number message
from std_msgs.msg import Float64
```

Then, we create a "publisher". In the initialization, in `__init__()`, write

```
# Setup up the condition number publisher
self.pub = node.create_publisher(Float64, '/condition', 10)
```

(ii) Every cycle, i.e. in the `evaluation()` function, compute the condition number. First compute $\bar{J}$. Then the condition number of $\bar{J}$, using either an SVD or `np.linalg.cond(Jbar)`.

(iii) Finally, every cycle, actually publish the number to ROS using

```
# Publish the condition number.
msg = Float64()
msg.data = condition
self.pub.publish(msg)
```

where `condition` is the condition number.

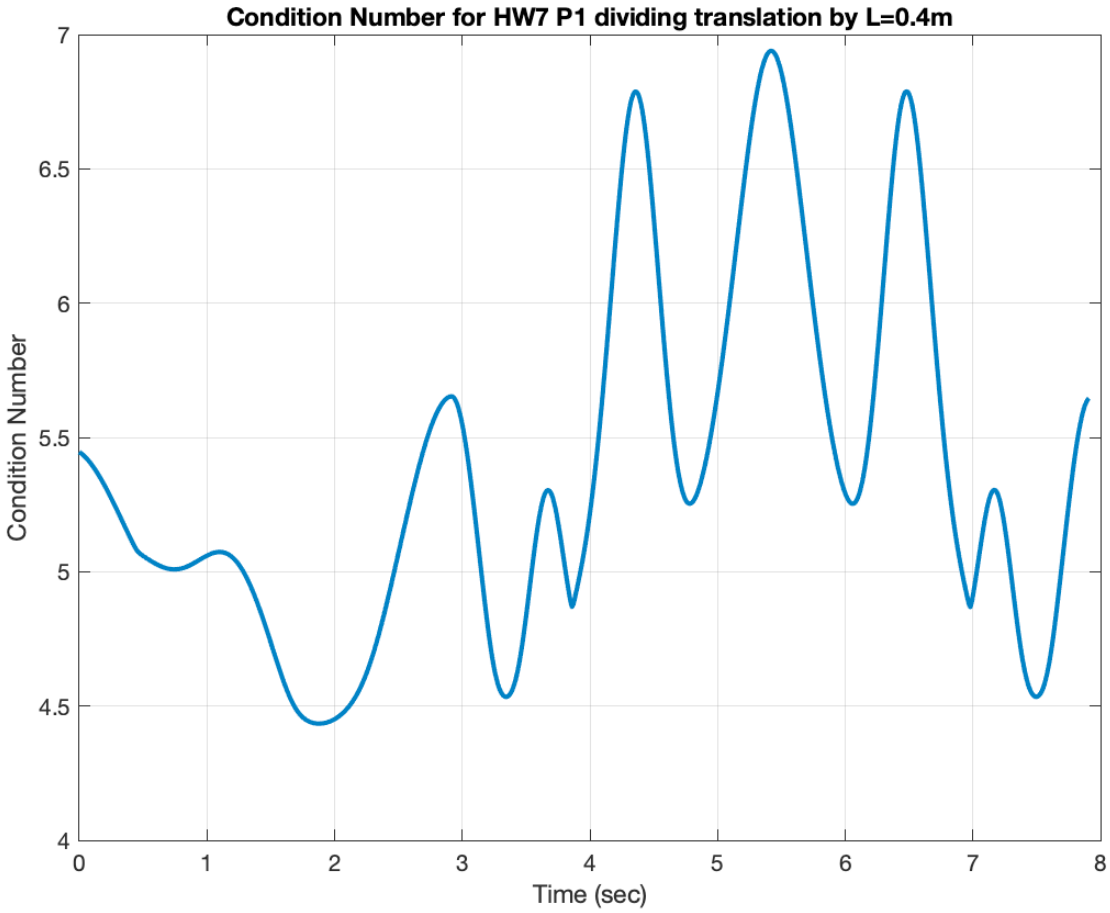
Now, when you run your code, please record a ROS bag using

```
ros2 bag record -o <bag-name> /joint_states /condition
```

which will save both the joint data and the newly computed condition number.

In Matlab, you can continue to plot the joint data using `plotjointstates()`. But also plot the condition number using `plotcondition()`. You should get something like the following, which I would consider this a pretty good trajectory.

Please submit only the plot, unless you changed the trajectory code. Note, it should start at the same value and have the same values at 3s and 8s, being the fixed locations.



Problem 3 (Redundant 7 DOF Robot) – 33 points:

Things get a little more interesting if the arm is redundant. We will use `sevenDOF_Obstacle.urdf` in place of `sixDOF_Obstacle.urdf`. You will notice the URDF adds an upper arm twist, between joint 2 and 3, similar to human arms.

We'll repeat the same tip movements as last week's Problem 5, or equivalently this week's Problem 2, cycling between the two targets. So, please start for the Problem 2 code, though you will need to

- initialize seven rather than six joints (insert an upper arm twist value between joints 2 and 3).
- publish seven rather than six joint names.
- publish seven joint angles.

Make sure the Jacobian is now 6x7.

But with the extra joint, we'll explore four different ways of handling the redundancy. We won't consider the case of just holding the new joint angle at zero - not using the joint would be exactly like the 6 DOF. And note, these are examples of different primary/secondary tasks. Finding "the right task" depends heavily on the situation. Here some work better than others, but that is highly situation-dependent.

Please execute the following four cases:

(a) First, do the simplest possible, using a pseudo inverse in place of the inverse Jacobian. No other changes beyond the above, nothing extra. How does the motion (qualitatively) compare to the motion of the 6R robot? How does having the additional freedom impact the other joints?

(b) Second, add a new primary task to force θ_3 to track $-\frac{1}{2}\theta_1$. To do so, create a seventh task coordinate

$$x_7 = \tfrac{1}{2}\theta_1 + \theta_3 \;\rightarrow\; x_7^{des} = 0$$

Consistently, manually add a seventh row to the Jacobian and a seventh task velocity and task error. We are then again at a fully determined system, 7 DOF and 7 task dimensions.

Qualitatively, what does this do? Better or worse than before?

(c) Third, return to the regular primary task (remove the additional task created in (b)) and add a secondary task to keep the arm centered in the middle of its joint ranges, as best possible. For this, assume the joints have a range not unlike a human arm:

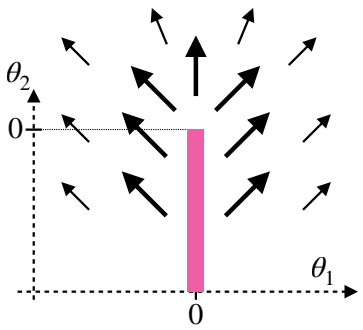
$$\begin{array}{llll} -\pi \le \theta_1 \le \frac{\pi}{2} & 0 \le \theta_3 \le \pi & -\frac{\pi}{2} \le \theta_5 \le \frac{\pi}{2} & -\frac{\pi}{2} \le \theta_7 \le \frac{\pi}{2} \\ -\pi \le \theta_2 \le \frac{\pi}{2} & -\pi \le \theta_4 \le 0 & -\frac{\pi}{2} \le \theta_6 \le \frac{\pi}{2} & \end{array}$$

Note, we aren't explicitly enforcing the limits. But use the secondary task to (try to) drive each joint toward the center of its range, with speed proportional to distance. What is a reasonable gain λ for this?

Also realize this is not being told about the wall. Briefly note the configurations at the two targets - why does this happen?

(d) Finally, ignoring the above range again, let's explicitly use the redundancy to push the elbow away from the wall. I.e. do so within the null-space of the primary task. To simplify, let's consider repulsion in the first two joints. The wall is just below $(\theta_1, \theta_2) = (0, 0)$. Repulse from there, upward with at least 45°:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix} = c_{\text{repulse}}\,\frac{1}{(\theta_1^2+\theta_2^2)}\,\begin{bmatrix}\theta_1\\ \max(|\theta_1|, \theta_2)\end{bmatrix}$$

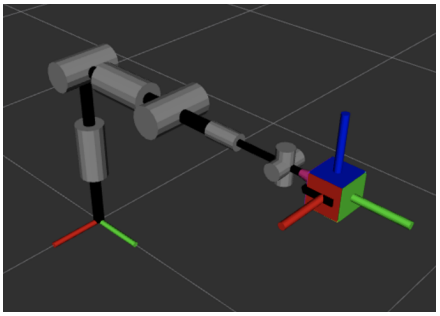


making the repulsive velocity bigger up close. What constant works best? Why, i.e. what happens when c_{repulse} is too small or too large?

For each case, please show/describe the formula you are using for the inverse. Include the code (limited to the inverse kinematics statements) and the joint data graphed as before.

Problem 4 (Handling the Singularity) – 33 points:

To study the infamous singularities, let's continue to use the 7 DOF robot. Though you may want to use the simpler `sevenDOF.urdf` without the obstacle.



Let's imagine you are following a trajectory "in real-time", i.e. that wasn't carefully planned. For example, consider trying to reach for a moving object. Or someone is guiding the robot via joystick or similar. In the end, assume the robot is trying to move the tip along the path

$$p_d(t) = \begin{bmatrix} 0 \\ 0.95 - 0.25\cos(t) \\ 0.60 + 0.25\sin(t) \end{bmatrix} \qquad R_d(t) = I$$

This asks the robot to reach outside its workspace for part of the movement, so I would certainly expect "issues". We'll start at the matched location

$$q_0 = \begin{bmatrix} 0^\circ \\ 46.5675^\circ \\ 0^\circ \\ -93.1349^\circ \\ 0^\circ \\ 0^\circ \\ 46.5675^\circ \end{bmatrix} \qquad p_0 = \text{fkin}(q_0) = p_d(0) = \begin{bmatrix} 0.0 \\ 0.7 \\ 0.6 \end{bmatrix} \qquad R_0 = I$$

Please start with the code from Problem 3, part (a), updating the trajectory to the above.

(a) First do nothing, i.e. keep using only a pseudo inverse approach as in Problem 3 part (a). The pseudo inverse should always provide an answer and this is numerically safe. But what happens? Please describe briefly and submit the joint data plot.

(b) Okay, let's start fixing the problem: use a weighted pseudo inverse. What weight γ should you use? What happens if the weight is too small or too big? Can we get a plot for both?

(c) To help the robot, can we add a secondary task to pull the elbow out of the singularity? How would you set this up? Imagine you want the secondary task to move the elbow to a "elbow up" (say $\theta_4 \approx -90^\circ$) configuration. Please show how you define your $\dot{q}_{\text{secondary}}$. And please also share the final data plot.

And, yes, please also submit the code (inverse kinematics statements only). This simply helps us understand if there are any questions. Thanks.

Problem 5 (Time Spent) - 4 points:

Wrapping the problem sets up, may we ask how much time you spent? Looking back, any improvements that would help next year's class?