Homework #4

due Wednesday 10/25/23 11:59pm

I love that our robots are starting to "come alive" in ROS. So I want to keep pushing into motions, velocities, URDFs, and start plotting the data (more precise than just watching the robot dance). But I also appreciate that there have been and are a lot of things going on (in particular the quiz). So we are trying to keep this set appropriately small. If you have any issues, especially related to ROS or Matlab, please let us know!

We have again posted skeleton code in the `hw4code` package on Canvas. As before, download into your robot's workspace source folder, unzip, and rebuild your workspace.

Also, attached at the end of this set are a few notes/hints on data recording and plotting via ROS bags and Matlab. Please take a look, as this is relevant to Problem 4.

As always, we'd like to see your work as well as the answers. This week, that will start including graphs and data plots. Certainly for programming/ROS problems, that means screenshots and code (snippets as appropriate). Please submit any code by printing to pdf, appending at the end of your submission, and declaring the pages in Gradescope.

Problem 1 (Cubic vs Quintic Spline) - 24 points:

Consider only a single DOF of a robot. Your objective is to create a motion that will stop the joint from an initial velocity, returning to the starting location..

More precisely, assume at the initial time of $t_0 = 0$ sec, the joint has a position of 0 rad, a velocity of 1 rad/sec, and an acceleration of 0 rad/sec^2.

You want to bring the robot to a full stop (zero velocity and, if possible, zero acceleration) at the same position of 0 rad at a final time of $t_f = 1$ sec.

(a) Create the motion using a single cubic spline segment.

(b) Create the motion using a single quintic spline segment.

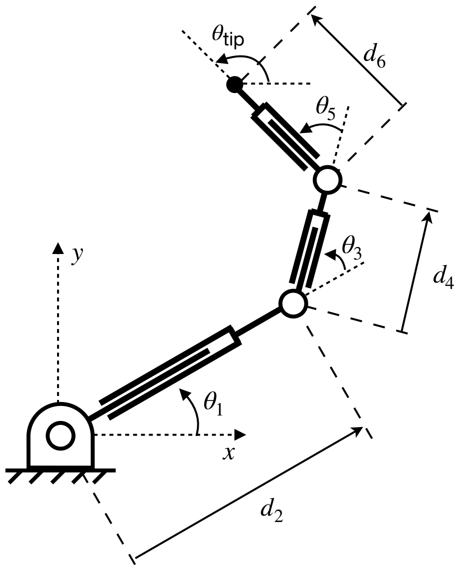
For both cases: What is the maximum position reached during the stopping motion?

Also please plot the positions vs time and the velocities vs time.

You may approach the problem analytically or numerically, whichever you prefer.

Problem 2 (Jacobian of Planar RPRPRP Robot) - 24 points:

Consider the 6 DOF planar RPRPRP mechanism



The joints angles are given as shown. The prismatic displacements are given such that the joint position reports the effective link length between adjacent revolute joints or the tip.

For the following, assume the joint coordinates are

$$q = \begin{bmatrix} \theta_1 \\ d_2 \\ \theta_3 \\ d_4 \\ \theta_5 \\ d_6 \end{bmatrix} = \begin{bmatrix} 45^\circ \\ 3\sqrt{2}\text{m} \\ 45^\circ \\ 2\text{m} \\ 90^\circ \\ 1\text{m} \end{bmatrix}$$

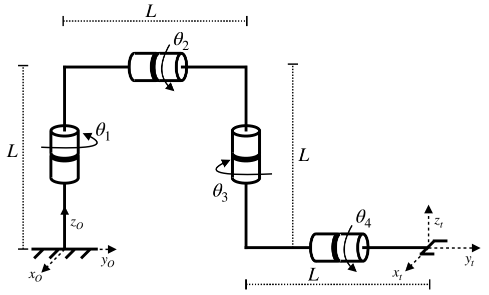
What is the planar 3x6 Jacobian matrix $J(q)$ *in this configuration*, in particular as defined by

$$\dot{x} = \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = \frac{\mathrm{d}}{\mathrm{d}t} \begin{bmatrix} x_{\text{tip}} \\ y_{\text{tip}} \\ \theta_{\text{tip}} \end{bmatrix} = J(q)\ \dot{q} = J(q) \begin{bmatrix} \dot{\theta}_1 \\ \dot{d}_2 \\ \dot{\theta}_3 \\ \dot{d}_4 \\ \dot{\theta}_5 \\ \dot{d}_6 \end{bmatrix}$$

Do not derive the solution analytically. Instead draw the robot (in the given configuration, *to scale*), determine the intermediate and tip positions, and determine/compute the Jacobian *entirely by hand.* Just please show your work.

Problem 3 (URDF for 4DOF Robot) - 24 points:

We return to the 4DOF robot from last week's Problem 2:



(a) Please convert the chain of transforms into a URDF file.

You may want to review `testrobot.urdf` and then start with `fourDOFskeleton.urdf`. Note each

`<link  name="..."> </link>` tag corresponds to a frame.
`<joint name="..."> </joint>` tag corresponds to a transform between frames.

So a URDF joint may be "fixed" (corresponding to our shifts), or "continuous" (moving rotation, corresponding to what we call a joint). If moving, it may include a fixed part which comes first, before the rotation is applied. As such, it can nicely combine two out of steps. A moving URDF joint thus implements the transform

$$T(\text{moving joint}) = T(\text{fixed}) * \begin{bmatrix} & & & 0 \\ & \text{Rot}(xyz, \theta) & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For more information please see the Kinematic Chains handout on Canvas, Section 3 as well as the URDF definitions and tutorials at

`https://wiki.ros.org/urdf/XML`
`https://docs.ros.org/en/humble/Tutorials/Intermediate/URDF/URDF-Main.html`

(b) Test your URDF and make sure it looks right. Do the joints turn the right way? This is likely easiest with sliders driving the joints. Use the GUI command you applied to create the sliders when viewing Atlas.

(c) Eventually, position the joints as given last week at

$$\theta_1 = \frac{\pi}{4} \qquad \theta_2 = -\frac{\pi}{3} \qquad \theta_3 = \frac{\pi}{4} \qquad \theta_4 = \frac{\pi}{6}$$

either using the sliders, or a stripped-down/simple trajectory program to be precise.

While still in RVIZ, look at the left side in the "Displays" window, expand the "TF" item, and find the "tip" frame. Read out the position and orientation. Note that RVIZ placed the fixed frame in world, so you are seeing the coordinates of tip w.r.t. world.

The orientation is given in quaternion form. Convert to a rotation matrix. Does this match the rotation matrix you got last week?

You may use any method for the conversion. For example, SciPy in Python can convert the four quaternion elements (X, Y, Z, W):

```
from scipy.spatial.transform import Rotation
R = Rotation.from_quat([X, Y, Z, W])
print(R.as_matrix())
```

Please submit the coordinates (position, orientation as quaternions and rotation matrix) as well as a screen shot of RVIZ in that state. Please also submit the content of the URDF file.

Problem 4 (Plotting Data for Last Week's Animations) - 24 points:

To start looking at real data, we revisit last week HW#'s Problems 4 and 5, plotting the resultant joint movements (position and velocity of all joints).

For both problems, following the attached visualization notes (ROS bag recording and Matlab plotting),

(a) Rerun your code, while storing a ROS bag.

(b) Run for at least one period (2π seconds in P4, 6 seconds in P5).

(c) Stop the ROS bag recording.

(d) Plot the contents in Matlab.

Submit the resultant plots.

Clearly the graphs for P4 should be smooth as the trajectory was smooth, the graphs for P5 should be piece-wise linear with velocity discontinuities.

No code submission required, just the plots please.

Problem 5 (Time Spent) - 4 points:

Approximately how much time did you spend on this homework set? Was it appropriate to offset the quiz?

Data Visualization: ROS Bag Recording, Matlab Plotting

Regarding data visualization, ROS has a number of debugging features. For example, try

```
rqt
```

and look under Plugins → Visualization → Plot. This is the same as running

```
ros2 run rqt_plot rqt_plot
```

This acts like an oscilloscope, monitoring signals in real-time. Try adding topics, for example:

```
/joint_states/position[0]
/joint_states/position[1]
/joint_states/velocity[0]
```

Don't forget to hit the '+' button. You can pause, or change parameters to expand the time window.

But to record, review, and analyze past data and keep things consistent (and hopefully easy), we will store/plot the joint data using ROS bags and Matlab.

ROS Bag Recording

ROS bags can record and play back messages being passed via topics, in particular our `/joint_states` topic commanding the motions. Further, Matlab can read/plot the data. To record a bag:

1. In Linux, in a separate terminal, change directory (`cd`) to a data folder. This should be outside the workspace's source files, ideally in a shared data folder. For example

   ```
   cd ~/robotws/data
   ```

   Using a shared folder will make the transfer plotting much easier! See the InstallingROS2.pdf documentation on creating appropriate links.

2. Before you run your Python code (specific to each problem, which will publish `/joint_states`), run the command

   ```
   ros2 bag record /joint_states
   ```

   This starts recording (waiting for your code to begin sending). The bag will be named by the current time. You choose a different bag name use

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

3. Run your Python code (from another terminal/elsewhere). Eventually let your code finish, ctrl-C to kill it, or just let it run.

4. Stop the ROS bag recording (using ctrl-C), creating the bag.

To confirm the bag's contents, try

```
ros2 bag info <bag-name>
```

The bag itself is a folder (named `<bag-name>`), structured as

```
<bag-name>                        entire bag folder
<bag-name>/metadata.xml           summary information
<bag-name>/<bag-name>_0.db3       raw data
```

ROS allows the raw data to be split across multiple files, which we will *not* do.

Matlab Plotting

Setup

1. The following utilities require Matlab releases **R2022a** or following. If you have an older version, please consider updating.

2. To read ROS bags, Matlab uses the 'ROS Toolbox'. This should be part of the complete Caltech installation. But to make sure, type in Matlab:

   ```
   help ros2bag
   ```

   You should see a normal help message. If Matlab doesn't have this function, select 'Add-Ons' → 'Get Add-Ons' and search for 'ROS Toolbox'.

3. I will assume you have a shared folder that is visible to both guest (Linux) and host computers. Within this folder, if necessary, create a `data` sub-folder. This will serve to collect the bags (or other data) and make them immediately available to Matlab. If you are not using a shared folder, you will need to transfer the bag folders manually.

4. Copy the Matlab function `plotjointstates.m` from the `matlab` subfolder of the `hw4code` package to this shared `data` folder. Or place it in any folder that will be on Matlab's path.

Plotting

5. In Matlab (on the host), change directory to the shared `data` folder. To plot the bag file contents, run `plotjointstates()`. If you do not specify a bag file, the latest (newest) will be used. If you specify a joint, only that joint will be plotted.

   ```
   plotjointstates();
   plotjointstates('latest', 'all');
   plotjointstates('bagname, jointnumber);
   plotjointstates('bagname, 'jointname');
   ```

6. Update the title so it's easier to track what is what

   ```
   title('This is my plot')
   ```

7. If you want to do/plot other things, `plotjointstates()` can also output the joint data:

   ```
   [t, pos, vel] = plotjointstates();
   ```