ECE4560 - Homework #11

Due: Nov. 13, 2017

Problem 1. (40 pts) Consider, as usual, the three-link rotational planar manipulator from the previous homeworks, c.f. Figure 1. We have examined it a bit early on in the semester. Let's start to explore more the geometry of its movement, but including the rotation part. $SE(2)$ is nice in that the orientation is given by a single variable, which means that we can pretend that $SE(2)$ is vector-like and then just use the calculus we know (thereby avoiding body and spatial coordinates). This homework problem explores that aspect before we move to the body form of things. Here, you will map velocities in joint space to velocities in the configuration or end-effector space (sometimes configuration space is called the *task space* since all tasks are in terms of group configurations).

Let the link lengths be $l_1 = 1$, $l_2 = \frac{1}{2}$, and $l_3 = \frac{1}{4}$. Further, consider that the joint angles are set to $\alpha = (\pi/3, \pi/4, \pi/12)^T$ and the instantaneous joint velocities are $\dot{\alpha} = (\pi/12, \pi/8, 0)^T$.

(a) Now, let's consider the vector form of the forward kinematics. It will be $(x_e(\alpha), y_e(\alpha), \theta_e(\alpha))^T$. The only difference between this and the earlier version from the homework is the additional orientation coordinate. What is the manipulator Jacobian for this vector function?

(b) For the setup (e.g., the α and $\dot{\alpha}$) described, what is the vector form of the end-effector velocity for the given joint configuration and joint velocity? Will the manipulator rotate clockwise or counter-clockwise? (remember the right-hand rule versus clock direction)

(c) To get a feel for the velocity, plot the manipulator for the specified joint confiuration, then use `velocityPlot` function to plot it. It will plot the linear velocity correctly and it will plot the angular part in the first quadrant if the angular velocity is positive, and in the fourth quadrant if negative. The size does not scale with the angular velocity. This will visualize where the manipulator will move given the joint velocities and in which direction it will rotate. Turn in the plot.

The function `velocityPlot` is available as a homework file and should be added to your $SE(2)$. Actually there will be another useful function in the file called `twistPlot` which you will need later. Make sure to copy both in. They are complete and work.

Code: functions made available as plotCode m-file, to be integrated into your $SE(2)$ class as member functions.

Problem 2. (60 pts) Let's consider again the three-link rotational planar manipulator from the previous homeworks, c.f. Figure 1. The link lengths were specified to be $l_1 = 1$, $l_2 = \frac{1}{2}$, and $l_3 = \frac{1}{4}$. We will investigate further what it means to generate trajectories using different concepts from the class (configuration-space, vector form versus Lie group form).

(a) In one of the last homeworks, you were asked to generte a cubic spline trajectory connecting a pair of inital and final joint angles. $\alpha(0) = (-\pi/6, \pi/4, -\pi/3)^T$ to the final joint angles $\alpha(5) = (0, -\pi/12, \pi/4)^T$. Compute the forward kinematics of these two joint angles to determine the associated end-effector initial and final configurations.

(b) Using the vector form for the configuration, $g = (x, y, \theta)^T$, connect the initial and final configurations with a straight line trajectory. Basically, the straight-line trajectory ignores the Lie group structure and treats the coordinates like independent vector elements. This is done by definiing the following trajectory

$$\left\{ \begin{array}{c} x_e^*(t) \\ y_e^*(t) \\ \theta_e^*(t) \end{array} \right\} = \left\{ \begin{array}{c} x_e^*(0) \\ y_e^*(0) \\ \theta_e^*(0) \end{array} \right\} + \frac{t}{T} \left(\left\{ \begin{array}{c} x_e^*(T) \\ y_e^*(T) \\ \theta_e^*(T) \end{array} \right\} - \left\{ \begin{array}{c} x_e^*(0) \\ y_e^*(0) \\ \theta_e^*(0) \end{array} \right\} \right).$$

For now, break up the trajectory into two segments with the same time duration (one way-point). You will need to apply the inverse kinematics to the trajectory end-points of the segments to get the desired joint-angles (be careful with multiple solutions here!), then spline the desired joint-angles together. What are the spline functions for each joint angle?

(c) Instead of vectorizing the group space and treating it like a linear space, lets try to design a trajectory that is more suited to the Lie group. Using the twist $\xi \in \mathfrak{se}(2)$ associated to the transformation between the initial and final configurations, the trajectory we are going to try to follow is

$$g_e^*(t) = g_e^*(0)\exp(\xi t)$$

Break the trajectory up into two segments along the exponential. You'll need to compute the half-way point along the exponential trajectory, compute the inverse kinematics for the segment end-points to get the desired joint angles, then spline the desired joint-angles together. What are the spline functions for each joint angle?

(d) Plot the joint trajectories, $\alpha(t)$, for the solutions to (b) and (c). Compare them against the joint trajectory from the last homework.

(e) Plot the final end-effector group trajectories, $g_e(t)$, for the solutions to (b) and (c). Since the trajectories you design will be in joint angles, you will have to transform the trajectory using the forward kinematics to get the end-effector Lie group configuration for plotting. This can be done nicely in the appropriate for loop. For (b) and (c) you have the desired trajectories that you can compare against. How well do your trajectories fit the desired trajectories?

(f) Comment on the differences between the three solutions regarding joint-space, vectorized configuration space, and exponential configuration-space. Comment on any challenges associated to computing the trajectories and finding the actual joint angles.

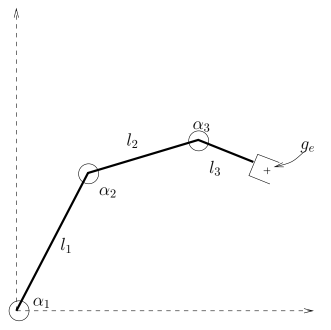

Figure 1: Planar 3R Manipulator.

Note: If you'd like to have more waypoints in the trajectory, then feel free to do so. You can also use Matlab's spline toolbox to handle the multiple waypoints, or you can solve using the matrix worked out in class.

Problem 3. (20 pts) This is a reprisal of the earlier homework problem where you solved the solution to the forward kinematics for the Lynxmotion L6 manipulator using the product of Lie groups. This week the task is to compute the forward kinematics of the Lynxmotion L6 manipulator using the product of exponentials. Show that the same function results. Use the reference configuration to be the straight-up configuration depicted in Figure 2.

Lab Assignments: For the traditional manipulator problems, there are actually two things to accomplish. One is to continue with the `piktul` manipulator and the other is to get started on the `lynx6` manipulator.

Problem 4: Manipulator. (20 pts) Work out the next piktul adventure (Set 1, Module 5), which continues exploring resolved rate trajectory generation. Reading it, you should note that we will be working with the full end-effector configuration in vector form. This is safe since the orientation is only about one axis.

The trajectory should start at the initial joint configuration of:

$$\alpha_i = (1.75, -30, 18, -25, 0.45)^T$$

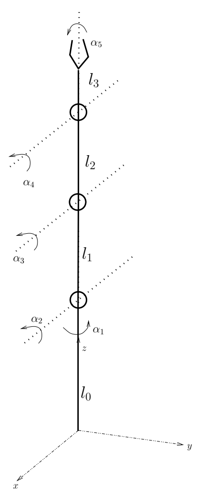

Figure 2: Lynxmotion L6 manipulator.

and this time move at the linear and angular velocity

$$v_{des} = \left\{ \begin{array}{c} -0.4 \\ 2.5 \\ -0.25 \end{array} \right\} \text{inches/sec} \quad \text{and} \quad \omega_{des} = 15^\circ/\text{sec}$$

for 3 seconds. Alternatively, we can think of this as moving at the velocity:

$$\dot{\vec{g}} = (-0.4, 2.5, -0.25, 15)^T$$

where the first three coordinates are the linear velocity and the last coordinate is the angular velocity.

Problem 4: Manipulator. (20 pts) Given the dimensions of the L6 manipultor you computed from prior homeworks and the forward kinematics with the home configuration that you chose, you are to write a program to pick and place a object. To do this you will need to complete the inverskin function at the end of the lynx6 m-file. Make sure that both the forward kinematics and inverse kinematics functions conform to your particular manipulator.

You must turn in the code for the inverse kinematics and the matlab script file for accomplishing the above. Normally, I would give you the code stubs for this set of actions, but I have already given you something similar in previous homeworks. When combined with the recipe given in the Lynx-6 Adventures wiki above plus a few quick changes, you should be good to go.

The initial and final configurations are:

$$g_i = \left[\begin{array}{ccc|c} 0.7071 & -0.7071 & 0 & 5 \\ -0.7071 & -0.7071 & 0 & 5 \\ 0 & 0 & -1 & 0.35 \\ \hline \multicolumn{3}{c|}{0} & 1 \end{array} \right]. \quad \text{and} \quad g_f = \left[\begin{array}{ccc|c} 1 & 0 & 0 & -4.4 \\ 0 & -1 & 0 & 5.5 \\ 0 & 0 & -1 & 0.25 \\ \hline \multicolumn{3}{c|}{0} & 1 \end{array} \right]$$

Good luck.

Problem 4: Turtlebot. (40 pts) Moving to the next module in the Turtlebot adventures, *Sensing*, work out the first three items in the list. These basically start to understand what visual sensor information is available, display it, and perform basic processing of the visual data. The processing emphasis is on the color data.

Problem 4: Biped. (40 pts) Work out the next step in the biped project.

Problem 4: Custom Lab. (40 pts) If you are working on a group project, then your group contact will review the submissions associated to the prior tasks and provide a new set of tasks. You should work out whatever baby steps this group contact assigns the group as opposed to the lab problem.

Splines, Polynomials and MATLAB. Here are some functions that might make you life easier: **polyval**, **ppmak**, **ppval**, **spline**. I believe that some of the spline methods are vectorizable, so you can get all of the joint angles in one polynomial/spline description. You should check out the MATLAB help to learn more about them. Also checking out "help splines" may be of use. Matlab's help is good enough so that you can get a feel for how to manage the homework. If you feel that the help is not clear, feel free to drop by or ask me questions.

Plots. Feel free to use the visualization code I wrote. It's there to help you see what is going on. You can use it as an aid. If you combine it with your existing $SE(2)/SE(3)$ code, then you can visualize everything in Matlab. This is quite useful for testing purposes prior to actually using the Lynxmotion manipulator.

Resolved Rate Trajectory Generation with the Manipulator Jacobian. The body manipulator Jacobian relates the joint velocities to the resulting body velocity,

$$\xi^b = J_{\text{body}}(\alpha)\dot{\alpha}.$$

To get the proper joint velocities given a desired body velocity, the equality must be inverted. If it is full rank and invertible, then this does not pose a problem, as one simply multiplies by the inverse to get

$$\dot{\alpha} = J_{\text{body}}^{-1}(\alpha)\xi^b.$$

If the descriptions are in vector form, then using the body form may not even be necessary. Then the manipulator Jacobian works,

$$\dot{\alpha} = J^{-1}(\alpha)\dot{g}.$$

Certainly, the vector form can be used for $SE(2)$ with $g = (x, y, \theta)^T$. For now the problem involves a square (body) manipulator Jacobian. Later on in class we'll cover what to do when the Jacobian is not square.

Resolved Rate Trajectory Generation with Positions Only. It is possible to apply resolved rate trajectory generation to the position part of the Jacobian only. In this case, only the top two rows for $SE(2)$ and the top three rows for $SE(3)$ would be necessary. In that case, we are considering a restricted version of the Jacobian for the linear (position) component only

$$\Delta p = J_{\text{pos}}(\alpha)\Delta\alpha \tag{1}$$

Using the inverse or the pseudo-inverse (or Matlab's slash operation), the above equation could be used to define a differential equation in α,

$$\dot{\alpha} = J_{\text{pos}}^{-1}(\alpha)\Delta p, \tag{2}$$

which works well when the Jacobian can be written in terms of the observer frame. Now, suppose that we wanted to use instead the **body** manipulator Jacobian, then the relationship is

$$\Delta p = g_e(\alpha) \star J_{\text{pos,body}}(\alpha)\Delta\alpha \qquad \text{or equivalently} \qquad g_e^{-1}(\alpha) \star \Delta p = J_{\text{pos,body}}(\alpha)\Delta\alpha \tag{3}$$

If we define $\nu^b(t) = g_e^{-1}(\alpha(t)) \star \dot{p}^*(t)$ where $p^*(t)$ is the desired trajectory, then the resolved rate differential equation is

$$\dot{\alpha}(t) = J_{\text{pos,body}}^{-1}(\alpha(t)) \cdot \nu^b(t), \qquad \alpha(0) = \alpha_0,$$

with a known initial condition (the current joint configuration). Resolved rate trajectory generation takes this and numerically integrates it to solve to the joint trajectory. The numerical integration can be done with `ode45` or some equivalent method. The actual function that computes the derivative $\dot{\alpha}$ will need to invoke code associated to the pseudo-inverse, to the forward kinematics, and to the manipulator Jacobian. It should also have access to the desired end-effector velocity (from which to generate the instantaneous body velocity).

SE(2) and the Adjoint. The adjoint operation in $SE(2)$ applied to the vector form of the twist, ξ is

$$\text{Ad}_g\xi = \left[\begin{array}{c|c} R & \mathbb{J}d \\ \hline 0 & 1 \end{array}\right], \qquad \text{for } g = (d, R).$$

Generating the Trajectory. Recall that after inversion, the resolved rate trajectory approach contemplates the differential equation

$$\dot{\alpha}(t) = J_{\text{body}}^{-1}(\alpha(t))\xi_{des}^{b}(t), \quad \alpha(0) = \alpha_0,$$

or

$$\dot{\alpha}(t) = J^{-1}(\alpha(t))\dot{g}(t), \quad \alpha(0) = \alpha_0,$$

for the body manipulator Jacobian and the manipulator Jacobian (for the vector form), respectively, with a known initial condition (the current joint configuration). Resolved rate trajectory generation takes this and numerically integrates it to solve to the joint trajectory. The numerical integration can be done with `ode45` or some equivalent method. The actual function that computes the derivative $\dot{\alpha}$ will need to invoke code associated to the (body) manipulator Jacobian. It should also have access to the desired end-effector body velocity.