ME 530.646 Lab3:
Forward Kinematics and Body Jacobians for the UR5

Noah J. Cowan, Ph.D.
Johns Hopkins University

Due Saturday Nov 11, 2017 23:59:59

Introduction

The purpose of this lab is to implement the forward kinematics of the UR5 robot in Matlab, and visualize it using Rviz. All codes are to be implemented in Matlab and visualized using Rviz.

Deliverables

This laboratory will require two deliverables:

1. A laboratory report (pdf format). Ultimately the report needs to be a single, easy-to-follow PDF. The lab report should contain the forward kinematics calculations and a few screen shots demonstrating that your code works. It doesn't need to be long nor complex. But it does need to be clear and complete. Use the conventions defined in the Mathematica notebook distributed with Problem Set 4. You may use results from Problem Set 4 and use Mathematica to show $g_{st}(0)$, as well as $\xi_i, i = 1, ..., 6$, as well as the final expression. Also, include in your PDF the final expressions for the Jacobian (you probably have to use a list-of-lists due to wrapping). Also include screen shots and explanations as required in Section 3 below.

2. These Matlab *functions*:

   - ur5FwdKin.m
     - Purpose: Compute the forward kinematic map of the UR5. All necessary parameters (e.g. the base twists, gst0, etc) should be defined inside the function.
     - Inputs: `q`: 6x1 joint space variable vector = $[\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6]^T$ where θ_n is the angle of joint n for n = 1,...,6. Be careful of sign convention!
     - Output: `gst`: end effector pose, g_{st} (4x4 matrix)
   - ur5BodyJacobian.m
     - Purpose: Compute the Jacobian matrix for the UR5. All necessary parameters are to be defined inside the function. Again, parameters such as twists and $g_{st}(0)$ (if needed) should be defined in the function.
     - `q`: 6x1 joint space variables vector (see above)
     - Output: `J`: Body Jacobian, J^b_{st} (6x6 matrix)
   - manipulability.m
     - Purpose: Compute a measure of manipulability. Implement all three different types: `'sigmamin'`, `'detjac'`, or `'invcond'`, as defined in Chapter 3, Section 4.4 [1]. This function will return any one of the three measures of manipulability as defined by the second argument (see below).
     - Inputs: There are two inputs to this function:

    * J: a 6x6 matrix
    * measure: a single string argument that can only be one of 'sigmamin', 'detjac', or 'invcond'. Defines which manipulability measure is used.
  - Output: mu: The corresponding measure of manipulability
- getXi.m
  - Purpose: Take a homogenous transformation matrix and extract the unscaled twist
  - Input: g: a homogeneous transformation
  - Output: xi: the (un-normalized) twist such that $g = \exp\{\xi\}$.
- ur5RRcontrol.m
  - Purpose: Implement a discrete-time resolved rate control system. This should iteratively implement the following Resolved Rate control system:

$$q_{k+1} = q_k - KT_{\text{step}} \left[J^b_{st}(q_k)\right]^{-1} \xi_k$$

    where, at each time step, the vector ξ_k would be taken by running getXi on the appropriate error matrix as described in class, i.e. ξ_k is such that

$$\exp \xi_k = g_{t^*t} = g^{-1}_{st^*} g_{st}$$

    This script should terminate when the norm of the twist components v_k and ω_k are less than some threshold. (Note they have different units!) You might want to make it something like 5cm and 15 degrees to start, but as you prefect your code, you should be able to make this tighter. The threshold can be hard-coded in your matlab script, but here you are going to need to play around a little based on other parameters. I suggest you start with a fairly forgiving value, to ensure that your code terminates. Also, the step size, T_{step} can be hard-coded in your script, and may require some experimentation to get a good value that approximates the step size of your script, since the matlab interface is not real time.
    Also, at each step you should check for singularities! If you are close to a singularity the system should ABORT, and return -1.
  - Inputs:
    * gdesired: a homogeneous transform that is the desired end-effector pose, i.e. g_{st^*}.
    * K: the gain on the controller
  - Output: finalerr: -1 if there is a failure. If convergence is achieved, give the final *positional* error in cm.

3. A parent *script* named lab3.m that tests each function above. Each test can and should be very simple as defined below.

   (a) To test ur5FwdKin, define one joint vector and apply the ur5FwdKin function and compute the resulting homogenoues transformation. Use the frame visualization tool introduced in lab2 to place a frame at this location and orientation in RVIZ. Then position the robot at the joint angles corresponding to the same joint vector you defined above (see Notes below for reference). Your visualized frame should be at the end effector of the simulated UR5. Try a couple of poses and take some screen shots.

   (b) To test Body Jacobian, simply calculate the Jacobian matrix using your function for some joint vector q. Then, compute a central-difference approximation to the Jacobian as follows. Compute the forward kinematics at slight offsets from q, i.e. $q \pm \epsilon e_i$ where $e_1 = (1,0,0,0,0,0)^T$, $e_2 = (0,1,0,0,0,0)^T$, etc. You will then have $g_{st}(q+\epsilon e_i)$ and $g_{st}(q-\epsilon e_i)$, and note that, approximately, we have

$$\frac{\partial g}{\partial q_i} \approx \frac{1}{2\epsilon}\big(g_{st}(q+\epsilon e_i) - g_{st}(q-\epsilon e_i)\big)$$

   To a decent approximation, for a small enough ϵ, you should have that the i^{th} column of the Jacobian is equal to

$$\xi'_i \approx \left(g^{-1}\frac{\partial g}{\partial q_i}\right)^{\vee} \tag{1}$$

Note that the term on the right will NOT be exactly a twist! So you'll want to "twist-ify" it first, i.e. take the skew-symmetric part of the upper left 3×3 matrix before finding the twist coordinate. So, your test function should, for each column, compute the approximate twist in Eq. 1. You will then be able to construct an approximate Jacobian, `Japprox` and compute the matrix norm of the error between `Japprox` and the actual Jacobian, e.g. in MATLAB something like `norm(Japprox - J)`. Print this on the command line in MATLAB.

(c) To test all the different manipulability measures, plot the value of each manipulabilty measurement near a singularity as a function of a joint angle. For instance one of the singular configurations of the UR5 identified in PS4 is when $\theta_3 = 0$. Therefore in MATLAB increment θ_3 from ($-\pi/4$, $\pi/4$), compute the Jacobian matrix at each increment (the other joint angles can be held constant at some random pose- but be sure to avoid other singluarities!), and at each angle plot the values returned from your mainipulability.m function for each of `'sigmamin'`, `'detjac'`, and `'invcond'` mesurement types. Here x axis will be θ_3 values and the y axis are the manipulability measure values. Include these plots in your lab writeup.

(d) To test `getXi`, choose a couple of arbitrary homogenous transforms and show that to machine precision that when you exponentiate the resulting twist (as a 4×4 matrix) using MATLAB's `expm` command, you wind up at the homogeneous transform you started with.

(e) To test your Resolved Rate controller, define a desired configuration and a gain, and then use the command to show that the robot moves to the goal. Pick a second initial condition that is near a singularity (or a goal that is near a singularity) and show that your command terminates and returns -1.

Please organize your code in a project named "Lab3" Use a simple, logical directory structure so that when the TA clones your repository, they can simply launch Rviz, and then run the main script (lab3.m) to test your code.

Notes

1. When visually verifing that you got your POE formulation correct. First use "ur5.publish_frame(g)" to put "your_frame" at the tool ("tool0") position. Then use "ur5.move_joints" to move ur5 around to several interesting positions. Update "your_frame" to verify your forward kinematic is the same with gazebo. A template is provided for the ur5FwdKin() function.

2. Likewise, you should use "ur5.get_current_transformation" to get numeral result and compare it with your result. Use "ur5.get_current_transformation('base_link','tool0')" to get the transformation from the ur5 base to tool tip frame.

Submission Guidelines

Push your Matlab script "lab3.m" and the lab report pdf to a project named EXACTLY "**Lab3**" (four characters) on git-teach. **It is important to name your project and all files correctly so the TAs can find them. If we cannot find your project, we can not grade it!** Include Dr. Cowan and all the TAs as developers. Check to verify that the files you hand in run. If your Matlab functions call custom matlab m-files that you have written (for this course or otherwise) be sure to include *all* necessary files.

DO NOT MAKE YOUR PROJECTS PUBLIC.

Make sure your code is clearly documented with sufficient comments to make it easy to understand. Sloppy code will not receive full credit. TAs will be available during office hours in Wyman to help. Also, there will be a severe deduction of points if submission guidelines are not followed precisely!
Please note: You do NOT have to duplicate any files submitted for Lab1 or Lab2 for this submission. Since each lab builds on the previous labs, the TAs will clone the latest master branch of all of your previous lab

projects when grading and add them to the path. It goes without saying if you find errors in your previous labs fix them and push changes! If you have any questions about this please email the TAs.

References

[1] R. M. Murray, Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Boca Raton, FL, 1994.