EN.530.646: Robot Devices, Kinematics, Dynamic and Control Lab 3*

Jin Seob Kim, Ph.D.
Senior Lecturer, ME Dept., LCSR, JHU

Out: 11/29/2022 (Tuesday)
Due: 12/06/2022 (Tuesday) midnight EST

Introduction

The purpose of this lab is two-fold:

Task 1. Implement what you did in lab1 with real UR5 robots. Note that you did only simulation in Lab1, now it is time to play with robots. There is no deliverable in this task. However, each group should schedule a demo with any TAs. Since this is your first time to work with robots, each group have to run the robot with the TAs for the safety reason. **Remember that all students in each group have to be present in the demo session, and perform this task with any TAs. You must not run the robots without supervision of TAs.** Scheduling will be done separately.

Task 2. Implement the forward kinematics and a kinematics-based control of the UR5 robot in Matlab, and visualize it using Rviz. All codes are to be implemented in Matlab and visualized using Rviz (this is *simulation only).*

Note that this lab is a team assignment.

Safety

The robot can be extremely dangerous if it is not manipulated or programmed properly.

1. UR5 Manual
   Please make sure you have read and understand the first chapter, Safety, of the manual which is provided on Canvas.

2. Simulation
   Before executing your program on the robot, the TAs will first check the program in the simulation, where you can see the graphics interface demonstrating the behavior of your code.
   Do not try to run the program before running simulation.

*This document was originally developed by Prof. Noah Cowan.

The First Task: Recapitulation of Lab1 Task

Use one of your Matlab script called lab1_1.m that controls the UR5 to move to a pre-defined sequence of at least 4 different joint configurations. It will use the following API functions to control the robot. Note that there is no deliverable in Task 1.

1. ur5=ur5_interface() initializes an object to interface with UR5. ur5 has membership functions to get data and send command.
2. ur5.move_joints(theta, time_interval) sends the desired robot joint configuration to the robot, the robot will move accordingly after the command is sent.

   theta: the joint angles in [rad] as a 6*N matrix (N goals)

   time_interval: time between arriving goals in [sec].

   For example, "ur5.move_joints(ur5.home,10)" will command ur5 to move to home position in 10 seconds.

   UR5 will move to the target by interpolating the joint angles. So the trajectory of each moving command is approximately a line in the configuration space, not a line in the cartesian space.

   NOTE that if calls to this function are made in rapid succession, the robot will probably not achieve intermediate waypoints, that is, the robot does not cue the locations, it replaces the current target with the new target. Feel free to play around with this e.g. by moving the robot back and forth between two locations with different lag between the calls.
3. theta=ur5.get_current_joints() reads the current robot joint angles.

The Second Task: Introduction

The purpose of this task is to implement the forward kinematics of the UR5 robot in Matlab, and visualize it using R-viz. All codes are to be implemented in Matlab and visualized by using R-viz. Note that this lab is a team assignment.

Deliverables

This laboratory task 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 Homework 6 or 7. You may use results from Homework 6 or 7 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. Finally, the lab report should also *clearly mention the contributions of each member to the assignment.*
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`: 6×1 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 $n = 1, \cdots, 6$. Be careful of sign convention!
  - Output: `gst`: end effector pose, g_{st} (4×4 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`: 6×1 joint space variables vector (see above)
  - Output: `J`: Body Jacobian, J^b_{st} (6×6 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 of MLS. 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 6×6 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 in 6×1 vector or twist coordinate form such that $g = \exp(\hat{\boldsymbol{\xi}})$
- ur5RRcontrol.m
  - Purpose: Implement a discrete-time resolved rate control system. This should iteratively implement the following Resolved Rate control system:

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k - K\, T_{\text{step}} \left[J^b_{st}(\boldsymbol{q}_k)\right]^{-1} \boldsymbol{\xi}_k$$

    where, at each time step, the vector $\boldsymbol{\xi}_k$ would be taken by running `getXi` on the appropriate error matrix as described in class, i.e. ξ_k is such that

$$\exp(\hat{\boldsymbol{\xi}}_k) = g_{t^*t} = g^{-1}_{st^*} g_{st}$$

    This script should terminate when the norm of the twist components $\boldsymbol{v}_k$ and $\boldsymbol{\omega}_k$ 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

    - `ur5`: the ur5_interface object
  - 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 homogeneous 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 $\boldsymbol{q}$, i.e. $\boldsymbol{q} \pm \epsilon \mathbf{e}_i$ where $\mathbf{e}_1 = (1,0,0,0,0,0)^T$, $\mathbf{e}_2 = (0,1,0,0,0,0)^T$, etc. You will then have $g_{st}(\boldsymbol{q}+\epsilon\mathbf{e}_i)$ and $g_{st}(\boldsymbol{q}-\epsilon\mathbf{e}_i)$, and note that, approximately, we have

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

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

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

   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 PS6 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 an element of $se(3)$) 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.

Notes

1. When visually verifying that you got your POE formulation correct first create a frame using "fwdKinToolFrame = tf_frame('base_link','fwdKinToolFrame',eye(4))" then move it to your computed forward kinematics transformation "g" using "fwdKinToolFrame.move_frame('base_link',g)". Then use

"ur5.move_joints" to move the UR5 to the corresponding pose and use the RVIZ simulation to verify your forward kinematics are correct. Repeat for several interesting poses.

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

Submit a single zip file titled "Lab3_TeamNumber_MemberInitials" (Example: Lab3_Team1_AB_CD_EF). This file should be *self-sufficient* to run your entire simulation/code. *Please make sure that you include the files that you created in previous assignments.* Since it is a team assignment, we will not be retrieving files form old submissions of the individual members. Hence again make sure to include ALL THE REQUIRED FILES, otherwise you will lose points. This also gives you a chance to update your code. You also have the option to create new custom Matlab m-files for this assignment, just make sure that they are well documented and all the necessary files are included in the zip.

Make sure your code is clearly documented with sufficient comments to make it easy to understand. Sloppy code will not receive full credit. The TAs will be available during office hours and by appointment to help. Also, there will be a severe deduction of points if submission guidelines are not followed precisely!