ME 530.646 Robot Device, Kinematics, Dynamics and Control Final Project

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

November 30, 2017

Introduction

The objective of this project is to use the UR5 robot to perform the classic pick-and-place operation automatically. You will implement a program that allows you to first "teach"[1] the UR5 the position of the start and target locations and then automatically complete the pick-and-place operation using a gripper and several different types of control.

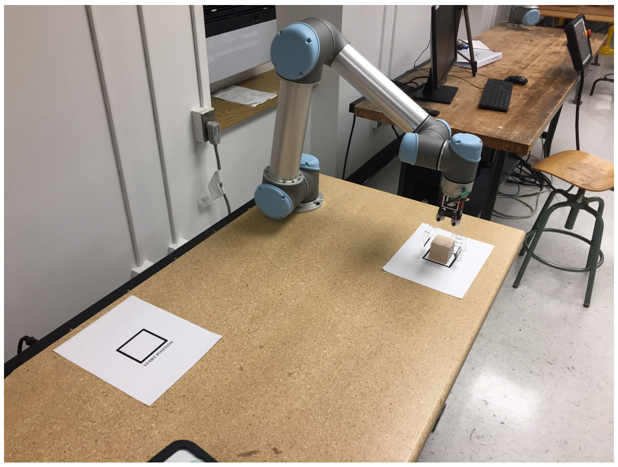


Figure 1: UR5 placing the cube on the target location

The goal is to move a wooden cube from the start position to the target position, both of which will be somewhere on the lab table inside the workspace of the UR5. Grippers will be installed on the end of the UR5 and they can be initialized, opened, and closed as needed using the functions in ur5_interface.m. You have already implemented code to drive the robot to a given position and orientation (pose) in Cartesian space using Resolved Rate Control. Now you need to develop a complete program that moves the cube from the start location to the stop location using the UR5 and gripper.

Minimum Task: Pick and Place

You will need to create three implementations of a pick-and-place task using the API provided in UR5.interface. For the final demonstration the start and target locations will be determined by the TAs, but they will be somewhere on the workbench and within the workspace of the UR5. Obviously this means that the start and target locations cannot be hardcoded and you must "teach" them correctly at the start of your program!

[1]To "teach" is a term used for the process of manually jogging or moving a robot to a desired location and then recording that position and orientation for later use in the program.

The recommended way to do this is use **pause** or another function like **waitforbuttonpress** in Matlab to pause the program while you are manually moving the robot around to the start and target locations. Once satisfied with the robot configuration, a simple button press should record the current frame for later use. Be sure to test your program on several different configurations to be sure it is robust.

Using the frame locations that you "teach", you will plan a trajectory for the robot in Cartesian space, i.e. in SE(3). The process should begin and end with the robot in some "home" configuration that you decide is best. You must implement the control trajectory in three different ways:

1. **Inverse kinematics**. Using inverse kinematics, move the robot along the desired Cartesian path by finding the corresponding path in joint space.
2. **Rate control using differential kinematics** Find the desired velocity (or small, differntial Cartesian offset), and "pull that back" through the inverse of the Jacobian to find the joint space velocity (increment) that moves you in the right direction.
3. **Gradient-based control.** Do the same thing, but now use the transpose, not inverse, of the Jacobian matrix.

Given that there are multiple inverse-kinematic solutions as well as the fact that you can grab the block in one of four orientations, for each method, first start by "teaching" an "optimal" robot joint configuration for picking up the cube by using some form of manipulability measure. Likewise, optimize the gripper orientation when you "teach" it the location of the target.

A "perfect score" on this task is solid B, maybe B+ work

Beyond a B or B+

Apart from the minumum tasks above, you are all welcome to try any interesting idea which involves robot motion using inverse/differential kinematics. You may use available resources or third party libraries (with approval of the instructor or TAs due to safety concerns). Some possible examples maybe:

1. Implementing a "hybrid" control using a manipulator jacobian.
2. Any other creative and impressive ideas

Remember to COMPLETE THE MINIMUM TASKS before becoming adventurous with the further credit!! Contact TAs if you need extra help to setup a special environment.

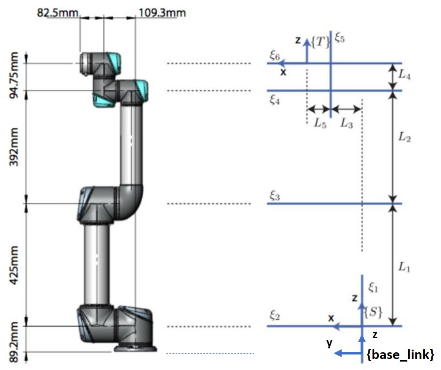
Inverse Kinematics

The function ur5InvKin.m will be provided that calculates all eight possible solutions to the inverse kinematics for the UR5 at a given frame in the workspace. Several of the solutions will not be practical to use and you will need to select the "best" solution to move the robot to during your "Inverse kinematics" control.

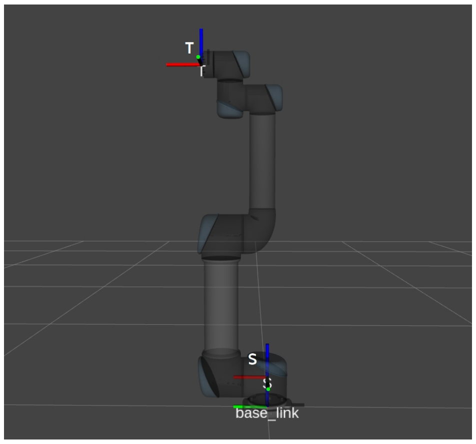
1. Input: g_06 (or T_0^6 as defined in [1]), the 4x4 homogeneous transformation matrix.
2. Output: A 6x8 matrix, each column represents one possible solution of joint angles

The function computes the solutions following the Denavit-Hartenberg convention described in [1]. Therefore you need to correctly define the (constant) transformations from the "0" frame in [1] to the "S" frame and from the "6" frame in [1] to the "T" frame. The "S" and "T" frames are the same convention used in the problem sets and previous labs. They are illustrated again in Figure 2 for reference.

A new RVIZ "urdf" model (ur5.urdf.xacro) of the robot will be provided that includes these frames for clarity. To use it simply replace the old file ur5_urdf.xacro file in catkin_ws/src/universal_robot/ur_description/urdf/ and relaunch RVIZ. Also note that if you move the joints on the UR5 to [0 0 0 0 0 0]' the robot will be will be placed at the Ryan Keating home position.



(a) Description of "S" and "T" frames



(b) New RVIZ model with "S" and "T" frames

Figure 2: Possible solutions to the inverse kinematics for simulated start and target frame described above

Controlling the Gripper

Mechanical Interface

The gripper used in the project is EU-20. The detailed specification of the gripper is available online. In our setup, this gripper is attached to the UR5 end-effector, and controlled by UR5 tool digital output.

API

Use these functions to control the gripper:

1. ur5.init_gripper
   Run this function before sending commands to initialize the gripper.
2. ur5.open_gripper
   Open the gripper
3. ur5.close_gripper
   Close the gripper

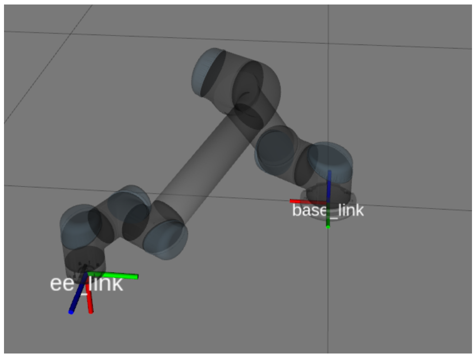
Start and Target Locations

The start and locations will be specified by a box printed on a sheet of paper that is taped to the desk. Try to be sure to accuratly place the cube within the box! You may have to adjust the thresholds on your control algorithms to achieve accurate placement. A copy of the "target box" will be available for testing.

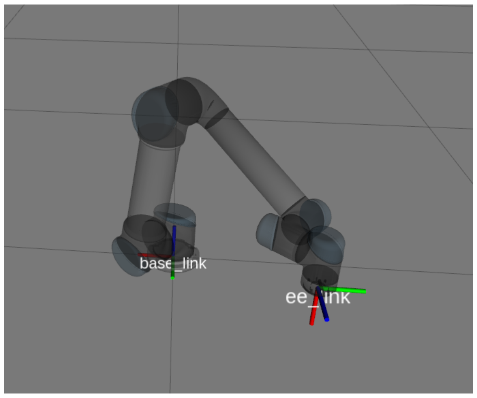
You must test your code in the RVIZ simulation first before running on the real robot. This can be a challenge because the start and target locations in the final demonstration could be anywhere on the table (of course they will be reasonable and well within the workspace). So to test your code in RVIZ use the following start and target frames that were recorded in the setup shown Figure 1:

$$g_{st\_start} = \begin{bmatrix} 0 & -1 & 0 & 0.47 \\ 0 & 0 & 1 & 0.55 \\ -1 & 0 & 0 & 0.12 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad g_{st\_target} = \begin{bmatrix} 0 & -1 & 0 & -0.30 \\ 0 & 0 & 1 & 0.39 \\ -1 & 0 & 0 & 0.12 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

Debug your program with these frames in RVIZ before attempting to run the real UR5.



(a) Possible start pose



(b) Possible target pose

Figure 3: Possible solutions to the inverse kinematics for simulated start and target frame described above

Other Hints

1. You should have a process, or controlled procedure for achieving the desired “pick-and-place” performance.
2. It is recommended to enable some kind of debugging output, or some kind of callback (print the current status when a certain key is presssed), it will expedite your experiment.
3. In real applications there are always a lot of details to consider: the collision between the gripper and the cube, the collision between the gripper and the table surface. Be careful and try not to break the gripper arms! We only have a few spares!
4. Modularize your code.

Deliverables

1. All code should be included in a single project named EXACTLY “**ur5_Final_ Project**”on git-teach (one per each team), with a main script called ur5_project.m. In this script, the user should be able to choose the method of control: IK-based, DK-based or gradient-based. 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.

   ALL code needed to run your program should be in this project (including any code you made for previous labs - It should all be in this single project!).
2. A PDF report specifying your algorithm (workflow), experiment result, workload distribution, and all other important considerations. You need not restate the forward kinematics or the jacobian which was done in previous labs, but any new math (e.g. Gradient control, etc) should be included.
3. Provide a hyper link to a youtube video of a successful “pick-and-place” implementation; it is highly recommended to also include a video of your “extra”. Videos should be edited to be 2-4 minutes.

References

[1] Ryan Keating. UR5 Inverse Kinematics. 2014.