EN.530.646: Robot Devices, Kinematics, Dynamic and Control
Final Project

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

Out: 04/25/2023 (Tue);
Due: 05/10/2023 (Wed) midnight EST

Introduction

The objective of this project is to use the UR5 robot to perform a "place-and-draw" task. You will implement a program that allows you to first teach the UR5 the positions of the start and target locations and then automatically complete the place-and-draw operation and several different types of control. The place-and-draw operation will be to use a soft pen attached to the end-effector specifically designed for this purpose, to draw shapes with lines (see Fig. 3 for details). It is essentially the same task as the classical "pick-and-place" task.

The goal is to move the UR5 robot from the start position (where the line begins) to the target position (or stop location where the line ends), both of which will be somewhere on the lab table inside the workspace of the UR5. A pen should be installed on the end of the UR5 using a pen gripper (see Fig. 1 and 2). 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 robot end-effector from the start location to the stop location using the UR5 with a pen.

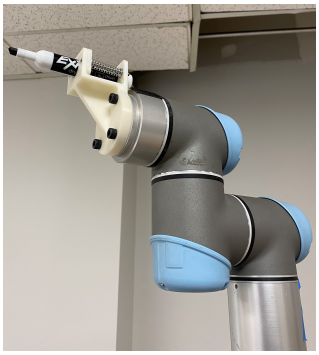
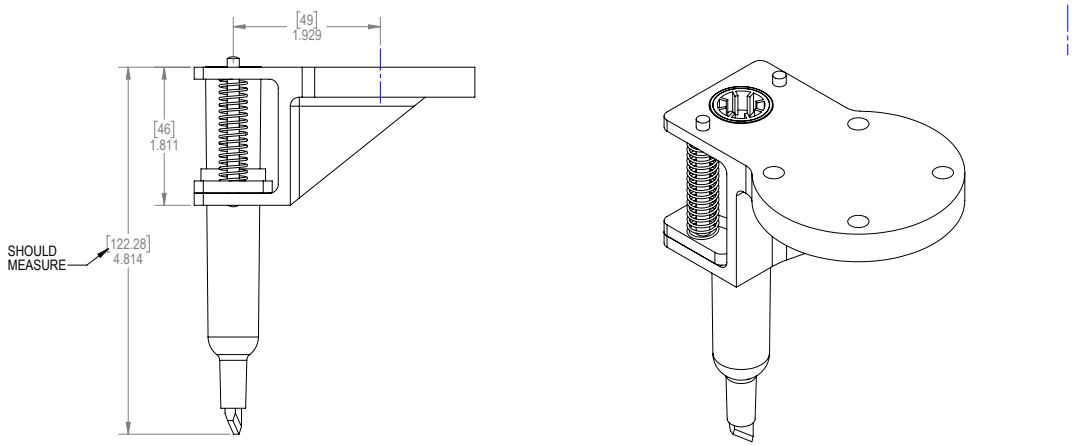


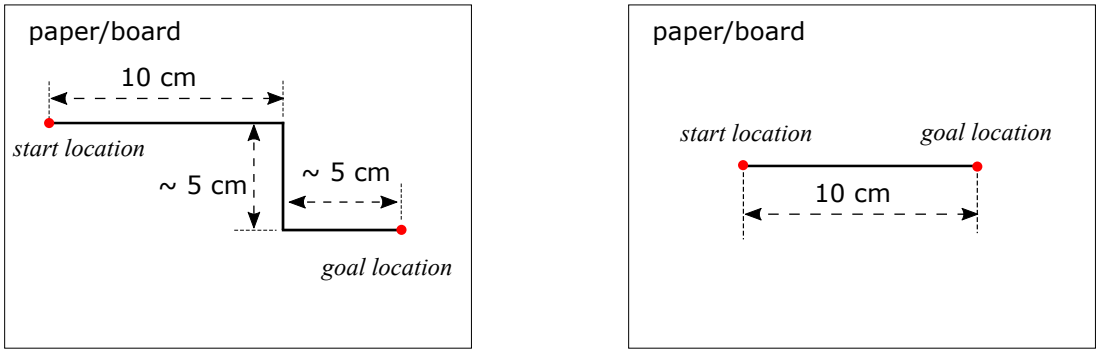
Figure 1: A UR5 with a pen gripper and a pen.



(a) (b)

Figure 2: A pen gripper with a pen: (a) 2D drawing, (b) 3D shape.

Main Task: Place-And-Draw



(a) (b)

Figure 3: configurations for different control schemes: (a) for RR and IK; (b) for JT.

You will need to create three implementations of a place-and-draw task using the API provided in UR5.interface. As you can see in Fig. 3, a different task is given to each control scheme. For the final demonstration the start and target locations can 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!

You can 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 (IK):** Using inverse kinematics, move the robot along the desired Cartesian path by finding the corresponding path in joint space.
2. **Resolved-rate control using differential kinematics (RR):** Find the desired velocity (or small,

differential 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. **Transpose-Jacobian control (JT):** Do the same thing, but now use the *transpose*, not inverse, of the Jacobian matrix.

Note that for RR and IK control schemes, once the start and goal locations are "taught" to the program, then your program must calculate the two intermediate locations (corresponding to the corner points in Fig. 3a) so that the robot can draw a line segment from one to another location each, to complete the open-rectangle shaped drawing. For JT, however, once the start and goal locations are "taught" you can directly draw a line segment between them.

In all three tasks, you have to implement the safety check such that the robot does not hit the table surface, joint limits, and so on.

Extra Task

After you complete the main task 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 the TAs due to safety concerns). This will lead to the extra credits. Remember to COMPLETE THE MAIN TASK before becoming adventurous with the further credit!! Contact TA's if you need extra help to setup a special environment.

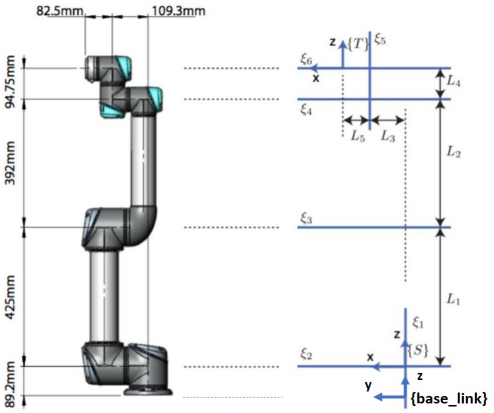
Inverse Kinematics

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. Explain this in your report.

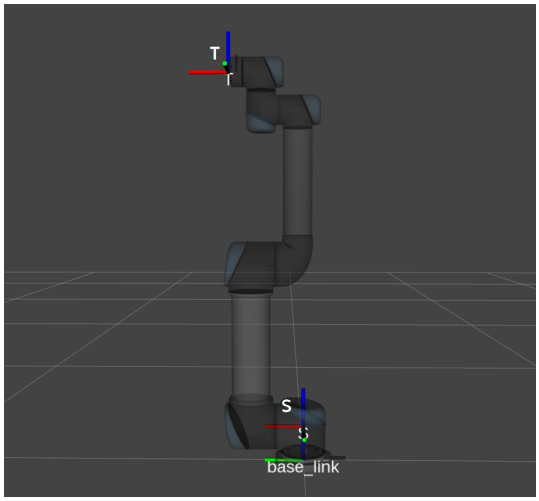
1. Input: g_{06} (or T_0^6 as defined in [1]), the 4x4 homogeneous transformation matrix.
2. Output: A 6×8 matrix, each column represents one possible solution of joint angles.

The function computes the solutions following the Denavit-Hartenberg (D-H) 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 Fig. 4 for reference.

Additionally, if you need, 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 4: (a) Geometry of UR5. (b) The home configuration from the new xacro file.

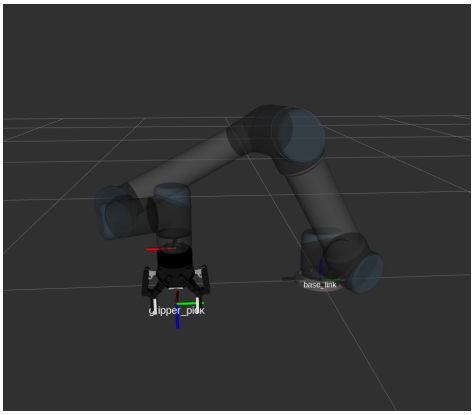
Start and Target Locations

The start locations will be specified near the surface. You must figure out the correct rigid body transformation between target frame and the frame attached to the gripper using the geometry shown in Fig. 2a. It will be a challenge to figure out the appropriate rigid body transformation for the gripper and where the pen is actually located. You may have to adjust the thresholds on your control algorithms to achieve accurate placement. You must test your code in the RVIZ simulation first before running on the real robot. This can also be a challenge because the start and target locations in the final demonstration could be anywhere on the table, i.e., the paper/board could be anywhere on the table (of course they will be reasonable and well within the workspace). Hence, you need to test your code in RVIZ, for which you can use the following start and target frames that were recorded in the setup shown Figure 5:

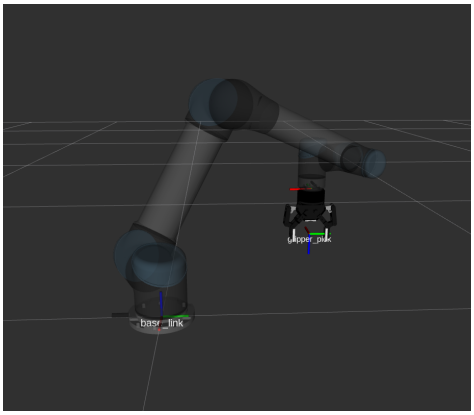
$$g_{st1} = \begin{pmatrix} 0 & -1 & 0 & 0.3 \\ -1 & 0 & 0 & -0.4 \\ 0 & 0 & -1 & 0.22 \\ 0 & 0 & 0 & 1 \end{pmatrix}; \quad g_{st2} = \begin{pmatrix} 0 & -1 & 0 & -0.30 \\ -1 & 0 & 0 & 0.39 \\ 0 & 0 & -1 & 0.22 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Of course, you can change positions as necessary. These frames g_{st1} and g_{st2} are the inputs to "ur5InvKin.m" function. Note that the start and target frames shown in Fig. 5 are NOT for the "gripper_pick" frames which are to be used in the project task. You have to figure out the rigid body transformation between the target frame as the input to "ur5InvKin.m" and the gripper_pick frame.

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



(a) Possible start pose



(b) Possible target pose

Figure 5: 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 "move-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 pressed), it will expedite your simulation.
3. In real applications there are always a lot of details to consider: the collision between the end-effector and the table surface. Be careful and try not to break anything!
4. Modularize your code.

Deliverables

1. Submit a single zip file titled "FinalProject_TEAM#_MemberInitials" (e.g., FinalProject_TEAM2_DL_ZH, if Dimitri and Zhuohong are a team 2) to the gradescope "Final Project", which includes a main script called ur5_project.m. In this script, the user should be able to choose the method of control: IK-based, RR-based, or TJ-based. Check to verify that the files you hand in run. Also make sure that the TA's can set their own the start and the target locations in your main script. Check to verify that the files you hand in run. You also have the option to create new custom Matlab m-files for this project, 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.

   All codes needed to run your program should be in this submission (including any code you made for previous labs!).

2. A PDF report specifying your algorithm (workflow), simulation results, and all other important considerations, as well as workload distribution between team members. This report must be submitted to the gradescope "Final Project: Report". Name the file as "FinalProjectReport_TEAM#_MemberInitials". In particular, report errors between the desired location and the actual location during the simulation (this corresponds to both start and target locations). For example let $g_d = (R_d, \mathbf{r}_d) \in SE(3)$ be the desired start location, and let $g = (R, \mathbf{r}) \in SE(3)$ be the actual start location that the robot end-effector

reaches in your simulation. Then report two errors for this start location as

$$d_{SO(3)} = \sqrt{\mathrm{Tr}\left((R - R_d)(R - R_d)^T\right)}$$
$$d_{\mathbb{R}^3} = \|\mathbf{r} - \mathbf{r}_d\|.$$

Do the same for the target location as well. These errors are applied to all three algorithm cases.

3. Submission should be done by only one member of your team. Hence the report must clearly state team members, with workload distribution as mentioned earlier.

4. We will set up a demo time, so that each team show their final outcome to the instructor and the TAs. Note that even though the demo session for your group is scheduled before the due date of submission, your demo must contain *final outcome* of the project. If you do not succeed in the demo, your final outcome is not considered complete.

References

[1] Ryan Keating, UR5 Inverse Kinematics. 2014