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

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

Out: 12/04/2025 (Thu)
Due: 12/15/2025 (Mon) by midnight ET

Introduction

The objective of this project is to use the UR5(e) robot to perform a "push-and-place" task. You will implement a program that allows you to first teach the UR5(e) the positions of the start location (denoted as (1) in Fig. 1) and then automatically complete the push-and-place operation with two different types of control. The push-and-place operation will be done either without using any end-effector, or by using any end-effector such as a pen gripper (originally to use a soft pen attached to the end-effector: see Fig. 3 and 4 for details). It is essentially the same as the classical "pick-and-place" task.

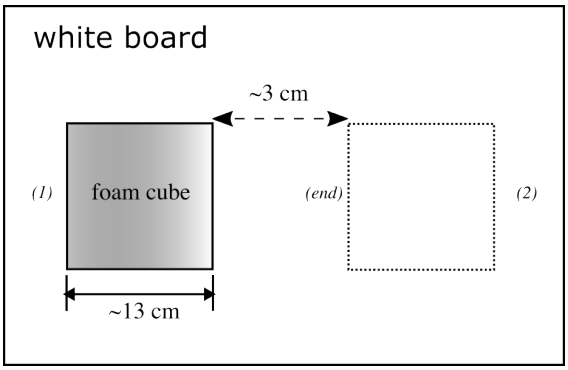


Figure 1: A schematic of the push-and-place task. A foam cube and a white board will be provided.

The goal is to move the UR5(e) robot to gently push a foam cube on a white board about 3 cm. See Fig. 2 for a cube and a white board. The robot is to move from the start position, denoted as (1) in Fig. 1), to push the foam cube by about 3 cm to the right, as seen in Fig. 1. Then the target position, denoted as (2) in Fig. 1), is the new location where the robot moves to push the cube back near the original location.

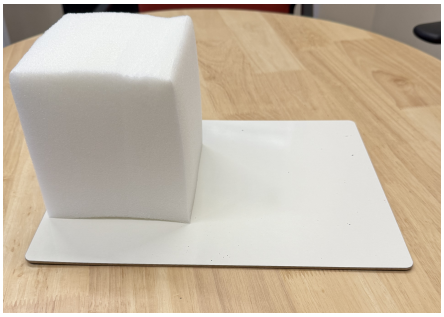


Figure 2: A photo of a foam cube and a white board.

All of which will be somewhere on the lab table inside the workspace of the UR5(e). **Note that (1) is to be taught. However, (2) should be computed based on the last location, denoted as *(end)* in Fig. 1, of the robot trajectory to push the cube.** This means that you will have to compute the end points of the second line in your code so that the robot automatically complete the pick-and-draw task. 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(e).

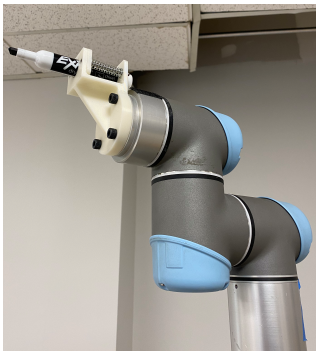
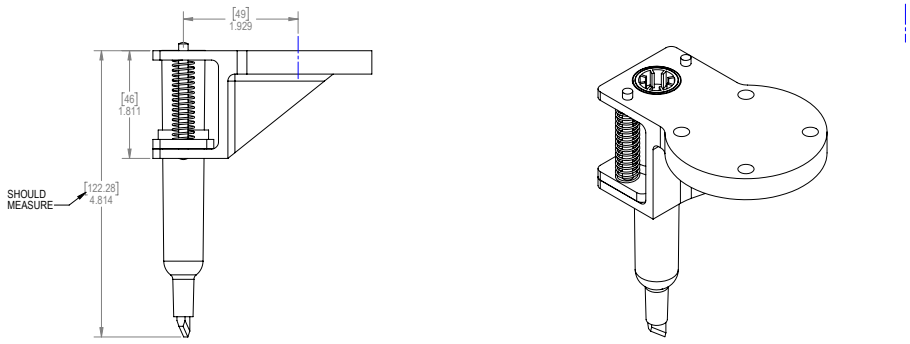


Figure 3: A UR5 with a pen gripper and a pen.



(a) (b)

Figure 4: A pen gripper with a pen: (a) 2D drawing, (b) 3D shape.

Main Task: Push-And-Place

For the final project, you will need to implement a push-and-place task via two interface methods: 1) ROS environment, and 2) MATLAB RTDE environment.

1) **ROS Environment:** You will need to create two implementations of a push-and-place task using the API provided in `ur_interface`. You will need to implement both RR and IK control methods. You can use either Python or MALTAB for this task.
2) **Matlab RTDE Environment:** You will need to create implementations of a push-and-place task using the API provided in MATLAB RTDE interface. For this, you will need to implement only RR control method.

For the final demonstration, the start location, (1) in the figure, can be determined by your team, as shown in Fig. 1. Make sure that all necessary locations should be somewhere on the workbench and within the workspace of the UR5(e). Obviously, this means that the start location cannot be hard-coded and you must "teach" it correctly at the start of your program.

You can use `pause` or another function like `waitforbuttonpress`, in both interface approaches, 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.

ROS Environment:

Note that to use freedrive mode, you must first use the function `ur.switch_to_pendant_control()` to allow the pendant to control the robot. Once you are finished using the pendant, you must switch back to ROS control using the function `ur.switch_to_ros_control()`.

So to teach a pose, the workflow would be as follows: call `ur.switch_to_pendant_control()`, add a pause in your MATLAB or Python script, move the robot to your desired position using freedrive, resume your MATLAB script, read the joint angles using `theta=ur.get_current_joints()`, and finally switch back to ROS control using `ur.switch_to_ros_control()`.

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. Here are two different ways to control the trajectory:

1. **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.
2. **Inverse kinematics (IK):** Using inverse kinematics, move the robot along the desired Cartesian path by finding the corresponding path in joint space.

MATLAB RTDE Environment:

Regarding UR5(e) operation, refer to "Control UR5 from Wyman Computers" in RTDE Instruction Manual.

For this environment, you will implement the following control:

1. **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.

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

Extra Task

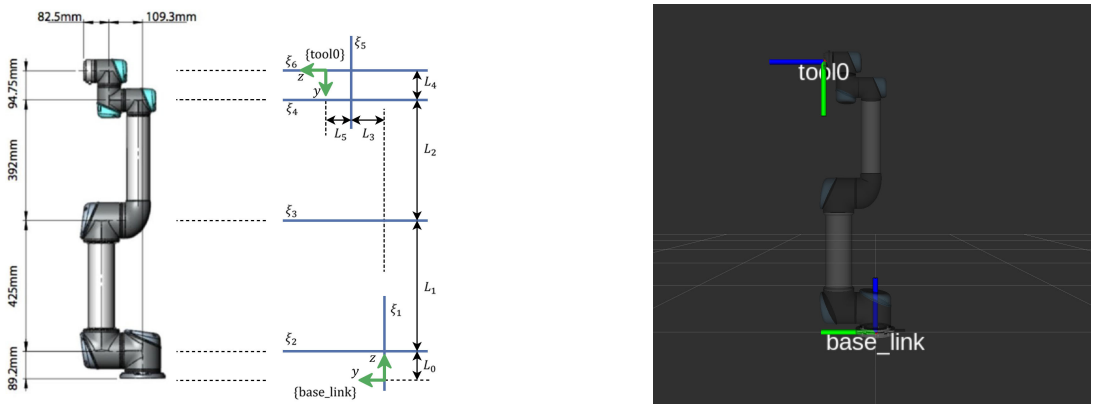
After you complete the main task above, you are all welcome to try any interesting idea which involves robot motion using inverse/differential kinematics. You may use available resources or third party libraries (with approval of the instructor or the TAs due to safety concerns). This will lead to the extra credits. Remember to COMPLETE THE MAIN TASK before becoming adventurous with the further credit!! Contact TAs if you need extra help to setup a special environment.

Inverse Kinematics

The function "urInvKin.m" will be provided that calculates all eight possible solutions to the inverse kinematics for the UR5(e) 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 Kinematic" control. Explain this in your report.

1. Input 1: g_{06} (or T_0^6 as defined in [1]), the 4x4 homogeneous transformation matrix.
2. Input 2: `"ur5"` or `"ur5e"`, indicating the controlled robot type.
3. 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], i.e. from the "base_link" to "tool0" frames. Therefore, you will either need to redefine your forward kinematics to match this convention or 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 if you would like to reuse results from homework sets and previous labs. The frame definitions are illustrated in Fig. 5 for your reference. There is also a Test Function included in the Lab files that will allow you to see the scale of the errors between your forward kinematic implementation and this inverse kinematic implementation. If you would like to reduce your errors, you may update your code to use the same DH parameters as were used in [1]. For the UR5e, the frames are defined the same but the link dimensions are different. Please refer back to Lab 3 for more.



(a) Description of "base_link" and "tool0" frames (b) RVIZ model with "base_link" and "tool0" frames

Figure 5: (a) Geometry of the UR5. (b) The RVIZ Model of the UR5.

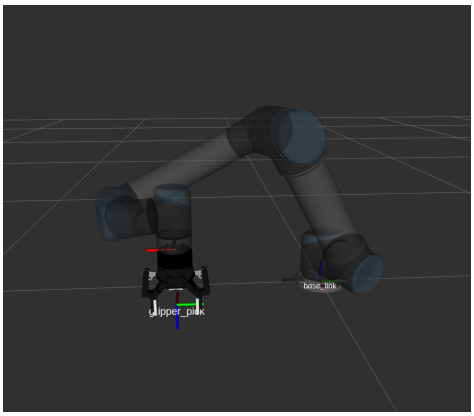
Start and Target Locations

The start and target locations will be specified near the surface. If you are using a pen-gripper, you must figure out the correct rigid body transformation between the "tool0" frame and the "pen_tip" frame attached to the tip of the pen gripper using the geometry shown in Fig. 4a. It will be a challenge to figure out the appropriate rigid body transformation to where the gripper is actually located. Keeping this in mind, you may have to adjust the thresholds on your control algorithms to achieve accurate placement. You must test your code in the RVIZ or MATLAB simulations 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., white board/cube 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 and MATLAB, for which you can use the following sample start and target frames. These are also shown in Figure 6:

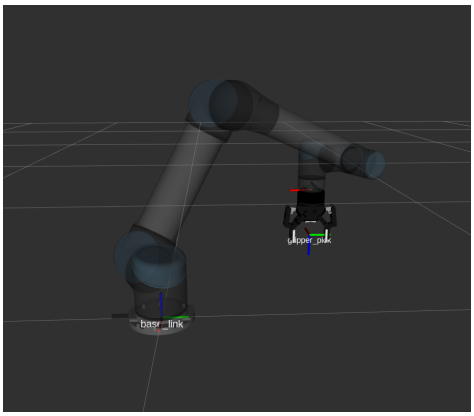
$$g_{st1} = \begin{pmatrix} 0 & -1 & 0 & 0.25 \\ -1 & 0 & 0 & 0.60 \\ 0 & 0 & -1 & 0.22 \\ 0 & 0 & 0 & 1 \end{pmatrix}; \quad g_{st2} = \begin{pmatrix} 0 & -1 & 0 & 0.40 \\ -1 & 0 & 0 & 0.45 \\ 0 & 0 & -1 & 0.22 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Of course, you can change these positions as necessary. These frames g_{st1} and g_{st2} are the inputs to "urInvKin.m" function, which are transformations between the "base_link" and "tool0" frames. Note that the start and target frames shown in Fig. 6 are NOT for the "pen_tip" frame which is to be used in the project task. You must figure out the rigid body transformation between the "tool0" frame and the "pen_tip" frame.

Hint: In the course, you have learned how to compound transformations. To get the position of the pen tip, you must compound the transform between the "tool0" and "pen_tip" frames with the forward kinematics of the robot to get the "base_link" to "tool0" transform. Likewise, when using the "ur5InvKin.m" function, you must convert your "base_link" to "pen_tip" transform to a "base_link" to "tool0" transform as this is what the function expects.



(a) Possible start pose



(b) Possible target pose

Figure 6: Possible solutions to the inverse kinematics for simulated start and target frame described above

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

Other Hints

1. You should have a process, or controlled procedure for achieving the desired "push-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" to the gradescope "Final Project", which includes two folders, one with a main script called ur_project.m for ROS environment and the other with a main script called ur_project_rtde.m for RTDE environment . An example of the file name is:

   "FinalProject_TEAM2_AH_DC_JL_MX", if Annie Huang, Denglin Cheng, Jiacheng Li, and Mengyu Xiao are team 2.

   In the ur_project.m script, the user should be able to choose the method of control: IK-based, or RR-based. Check to verify that the files you hand in run. In the ur_project_rtde.m, the use can execute RR by hitting the return button. Check to verify that the files you hand in run. You also have the option to create new custom MATLAB/Python files for the 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

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

   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. The report must clearly state team members, with workload distribution as mentioned earlier.
4. We will set up a demo time (presumably Monday, Dec. 15, 2025), so that each team shows 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/time 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