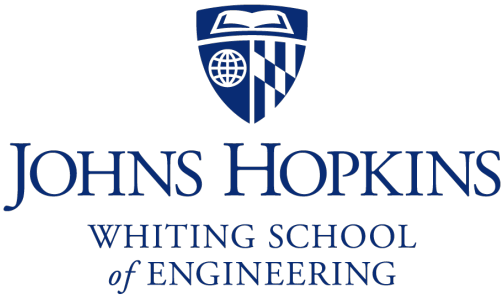
Robot Devices, Kinematics, Dynamics, and Control: Final Project

Spencer Powers
Trevor Schwehr
Kevin Wang

December 15, 2021



Laboratory for Computational Sensing and Robotics

Contents

List of Tables

1 Problem Statement

The objective of this project was to simulate the UR5 robot arm using RViz to perform a move-and-place task. Specifically, this task involves moving the arm from a starting configuration to a target configuration, while pausing above each desired configuration to ensure that the final motion in each instance is strictly vertical. The arm begins and ends the move-and-place sequence in a chosen home configuration.

Three implementations of the move-and-place task are detailed in the following section. The first uses inverse kinematics, the second uses a resolved-rate control, and the third uses transpose Jacobian control. A number of safety checks, detailed in Section 3.2, were implemented to ensure that the arm does not hit the floor or encounter singular conditions.

2 Algorithms

For each of the control methods, we executed a similar algorithm. As both the inverse kinematics function and the forward kinematics function used in dynamic control are computed for the relationship between the base of the robot and the tool frame, we first use a transformation to convert the desired goal for the gripper frame to a desired goal for the tool frame. We did this by left multiplying the desired goal frame by the inverse of the transformation from the tool to the gripper, shown below. This transformation matrix accounts for the 0.1303m length of the gripper, as well as the -90 degree rotational offset around the z-axis from the tool frame.

$$\begin{bmatrix} cos(-\frac{\pi}{2}) & -sin(-\frac{\pi}{2}) & 0 & 0 \\ sin(-\frac{\pi}{2}) & cos(-\frac{\pi}{2}) & 0 & 0 \\ 0 & 0 & 1 & 0.1303 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

After computing the transformed frames, we computed the location of frames at a predetermined height above each of the desired positions. We used these higher frames to compute the location of the "home" frame equidistant from each of the target frames at a reasonable height above the offset frames and the floor of the simulation, as detailed in Section 3.1. Next, we used the user-chosen method for controlling the robot through the sequence of frames for the move-and-place task. The full sequence starts at the home frame, moves to above the start frame, down to the start frame, returns to above the start frame, moves to above the target frame, down to the target frame, returns to above the target frame, and finally returns to the home frame. At each of the desired goal locations (start and target), we computed the error between the actual pose of the robot and the desired pose of the robot as shown below.

$$d_{SO(3)} = \sqrt{Tr\left((R - R_d)(R - R_d)^T\right)}$$

$$d_{\mathbb{R}^3} = ||\mathbf{r} - \mathbf{r}_d||$$

2.1 Inverse Kinematics Implementation

The inverse kinematics trajectory control was implemented through the provided inverse kinematics matlab function ur5InvKin.m. This function calculates all eight possible inverse kinematics solutions for the UR5 for a given frame in the workspace, returning a 6x8 matrix with each column corresponding to one possible joint angle configuration where the robot tool frame aligns with the target frame. After studying the different solutions provided by ur5InvKin, it becomes apparent that only about half are reasonable solutions. It would not be safe for the robot to crash through the floor of the simulation, so all "elbow down" orientations are immediately rejected. The next major consideration is whether the wrist should be oriented "wrist in" or "wrist out", these positions are similar to the "wrist up" or "wrist down" positions shown in [1]. Through trial and error, we found that any distance within 0.3 m of the robot base would cause conflicts if the wrist is oriented away from the robot. We decided to use wrist in for locations within 0.3 m and wrist out for locations farther than 0.3 m. The final decision is whether to use "shoulder left" or "shoulder right", which is fairly arbitrary so we decided on shoulder left. This algorithm is implemented in the *filteredInvKin.m* function. For each frame in the sequence, we use this algorithm to determine the correct joint angles and use *move_joints* function of the UR5 MATLAB interface to move the ur5 through the sequence.

2.2 Resolved-Rate Control Implementation

The discrete-time joint angle update equation for the resolved-rate controller is expressed as follows:

$$\mathbf{q_{j+1}} = \mathbf{q_j} - K\Delta t \left[J^b_{st}(\mathbf{q_j})\right]^{-1} \xi_\mathbf{j}$$

where $\mathbf{q_j}$ is the vector of joint angles at step j, K is the controller gain, Δt is the time step, $J^b_{st}(\mathbf{q_j})$ is the body Jacobian as a function of the joint angles at step j, and $\xi_\mathbf{j}$ are the twist coordinates of the error g_{t^*t} which describes the transformation between the initial configuration and the desired configuration.

To use this controller, the arm starts at an arbitrary non-singular configuration and is given a target configuration. The twist $\xi_\mathbf{j}$ is computed given these two configurations, the initial positional and angular errors are computed, and the controller then enters its main loop. While the two errors are above hard-coded thresholds, namely 2 cm and 1 degree respectively, the joint angles of the next step are computed via the above equation. If the computed angles correspond to a singular configuration, as detected by the three Jacobian-based manipulability measures detailed in Section 3.2, the step is aborted. If the next step does not result in a singular configuration, the arm is moved to the desired new joint angles via the *move_joints* function of the UR5 MATLAB interface.

This loop runs until either both errors fall below their respective thresholds for success or singular conditions are nearly encountered and consequently aborted. Through experimentation, we chose a gain of K = 0.2. This value was determined to be high enough to move at a reasonable pace, but not so high that the system became unstable or yielded too large of step sizes.

2.3 Transpose-Jacobian Control Implementation

The discrete-time joint angle update equation for the transpose-Jacobian controller is similar in form to the update equation of the resolved-rate controller. The update equation is expressed as follows:

$$\mathbf{q_{j+1}} = \mathbf{q_j} - K\Delta t \left[J^b_{st}(\mathbf{q_j})\right]^T \xi_\mathbf{j}$$

where $\mathbf{q_j}$ is the vector of joint angles at step j, K is the controller gain, Δt is the time step, $J^b_{st}(\mathbf{q_j})$ is the body Jacobian as a function of the joint angles at step j, and $\xi_\mathbf{j}$ are the twist coordinates of the error g_{t^*t} which describes the transformation between the initial configuration and the desired configuration.

The inner workings of the transpose-Jacobian controller are identical to those of the resolved-rate controller detailed in the preceding section. The most significant difference is that the Jacobian as a function of the joint angles at step j is transposed instead of inverted in the update equation. In addition, the positional and angular tolerances for success were slightly increased to 5 cm and 5 degrees respectively to keep the run time within reasonable limits, as this particular controller runs significantly slower than the other two. Through experimentation, we chose a gain of K = 0.4. This value was determined to be high enough to move at a reasonable pace, but not so high that the system became unstable or yielded too large of step sizes.

3 Additional Considerations

3.1 Choosing a Home Configuration

It was discovered that the performance of the resolved-rate controller and the transpose-Jacobian controller was dependent upon the choice of the home configuration, meaning the configuration in which the manipulator begins and ends the move-and-place task. If a suboptimal home configuration was chosen, it could lead to either controller nearly encountering singular conditions and aborting.

After experimentation, a reasonably reliable approach was found to be setting the home configuration joint angles to be the average of the start and target configuration joint angles. The resulting joint angles were necessarily wrapped to be in the viable joint angle ranges of $[-\pi, \pi]$ to ensure that the arm was not commanded to move out of its operating range. Finally, the second and third joint angles were slightly modified after the averaging and wrapping process to raise the arm in the positive z direction to ensure that 3D movement was

required to bring the arm from the home configuration to above the start and target configurations.

3.2 Safety Considerations

To avoid moving the arm into singular configurations over the course of trajectories computed by the resolved-rate and transpose-Jacobian controllers, the manipulability of the arm in the proposed next step's configuration is checked prior to movement. Three manipulability measures are computed each step, and in each case, a measure of zero corresponds to singular conditions. The first measure is the smallest singular value of the Jacobian, the second measure is the inverse of the Jacobian's condition number, and the third measure is the determinant of the Jacobian. The manipulability measure closest to zero is chosen from the three, and if it is within a hardcoded tolerance of 0.003, then the planned next step's movement is aborted with a warning message.

In addition, due to the nature of the move-and-place task, the method by which the home configuration was determined inherently prevents the resolved-rate and transpose-Jacobian controllers from generating trajectories that collide with the floor; in order for the gripper to move strictly down to both the start and target locations, the orientation of the gripper facing downwards is fixed across both configurations. Thus, averaging their respective joint angle vectors produces the same gripper orientation at the home configuration, and since there is little to no error in those joint angles as computed in the controller loops, the arm is not incentivized to invert itself through the floor over the course of the trajectory. To ensure the safety of the inverse kinematics approach, we filter out all configurations that would cause the robot to go below the floor of the simulation by rejecting all "elbow down" configurations. Further, the speed of the robot is controlled by choosing an appropriate time step for the movement. We manually chose a time that would allow the robot to complete the move at a reasonable, and safe, speed. This is further checked by the internal warnings of the UR5 MATLAB interface.

4 Simulation Results

As shown in the following figures and tables, we were able to successfully complete the move-and-place task using all three methods of control. The most precise of the three methods was the inverse kinematics approach. Greater precision could have been achieved with the resolved-rate and transpose-Jacobian method at the cost of a significantly increased run time.

The movie files submitted alongside this report show the entire trajectories instead of the key positions shown below. We would like to note that the videos appear to have an inconsistent frame rate while recording, leading to the trajectories looking choppier than they do in RViz. We tuned the gains to ensure that the arms did not move excessively quickly at any point in the trajectories, and while they may appear to move in bursts due to the inconsistent

recording quality, re-running the simulations shows that our chosen gains do in fact prevent the arm from moving at an unsafe speed.

4.1 Inverse Kinematics Implementation

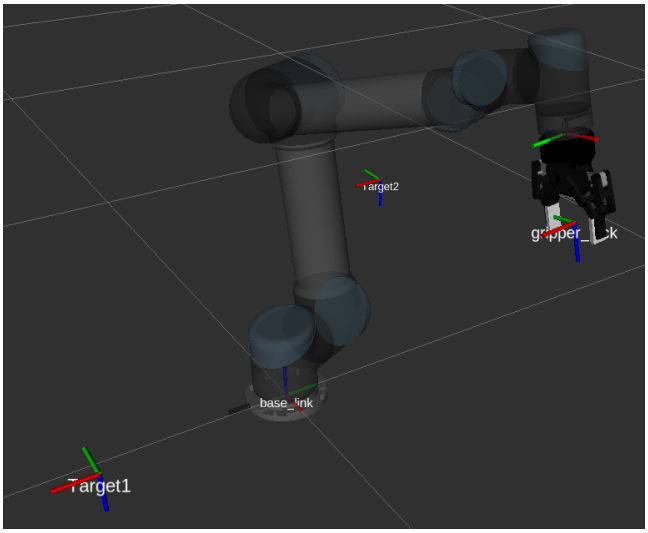


Figure 1: Home configuration.

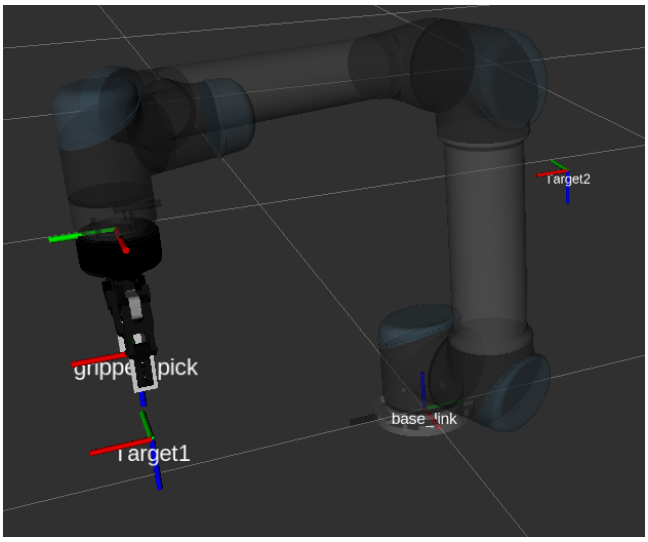


Figure 2: Above starting frame.

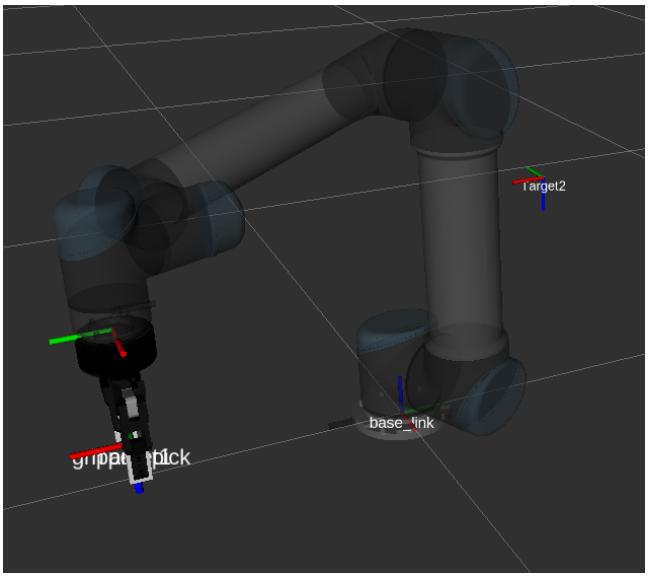


Figure 3: At starting frame.

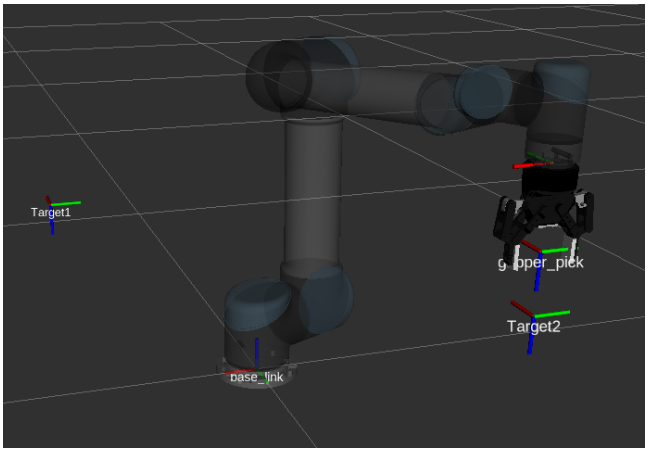


Figure 4: Above target frame.

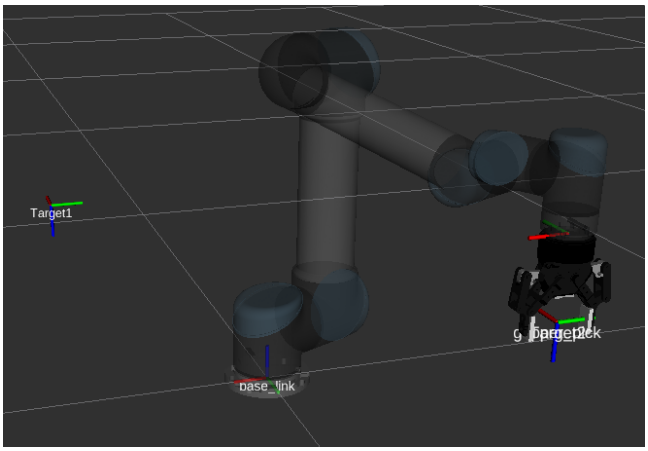


Figure 5: At target frame.

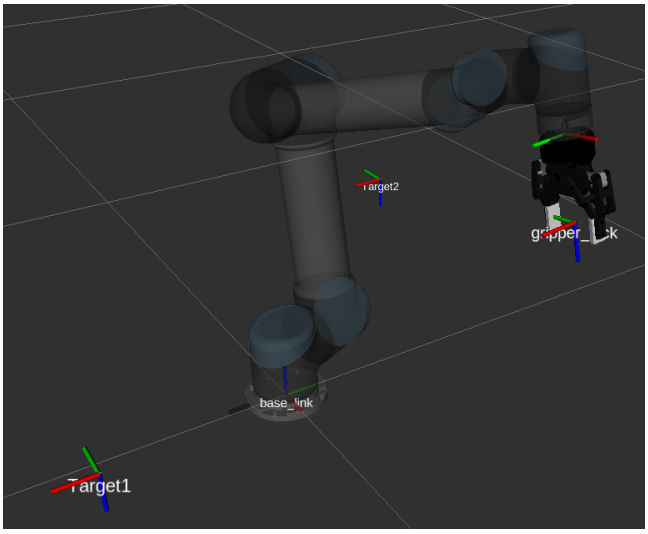


Figure 6: Back to home configuration.

The errors for the inverse kinematics implementation are summarized in the table below:

Desired Frame	$d_{SO(3)}$	$d_{\mathbb{R}^3}$
gst_1	0.0140	0.0015
gst_2	0.0140	0.0011

Table 1: Inverse kinematics implementation error metrics.

4.2 Resolved-Rate Controller Implementation

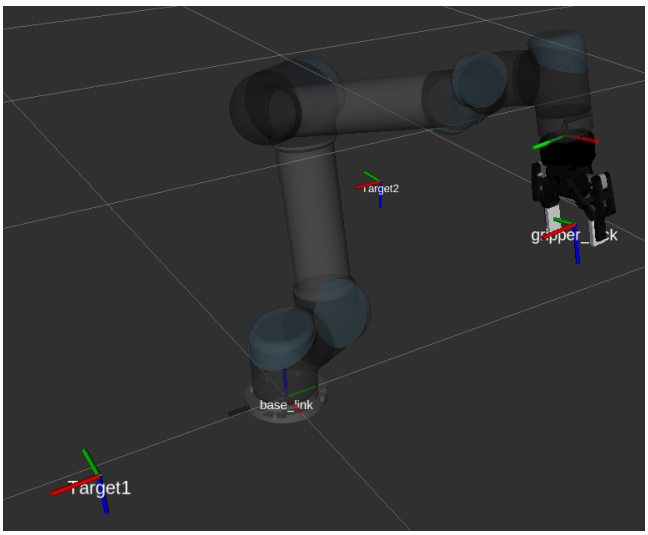


Figure 7: Home configuration.

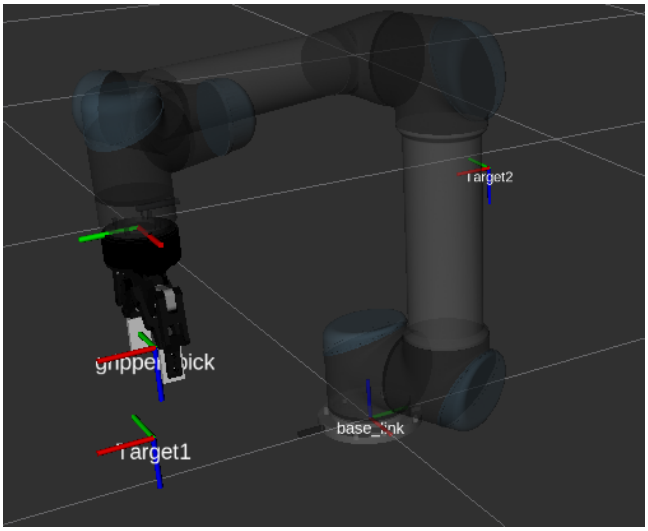


Figure 8: Above starting frame.

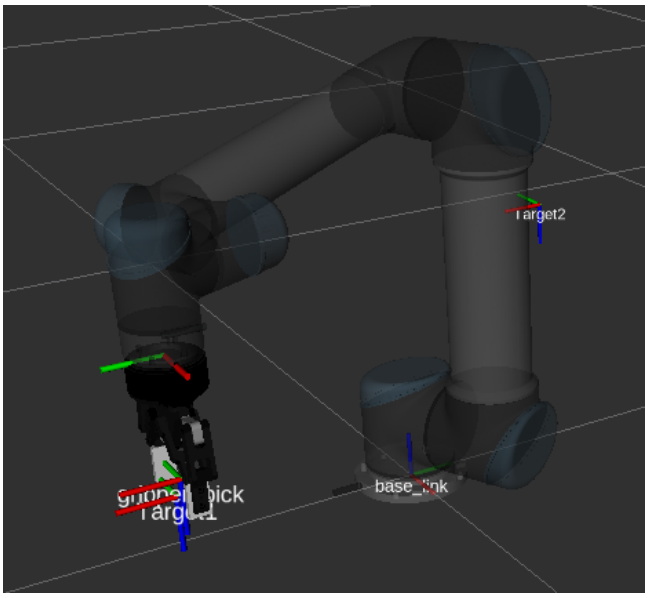


Figure 9: At starting frame.

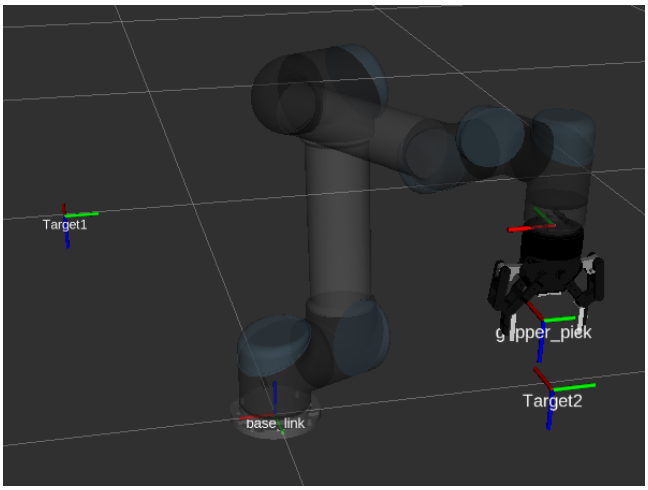


Figure 10: Above target frame.

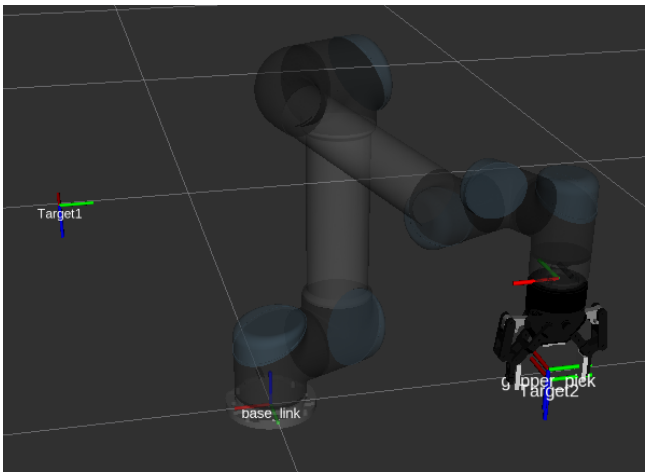


Figure 11: At target frame.

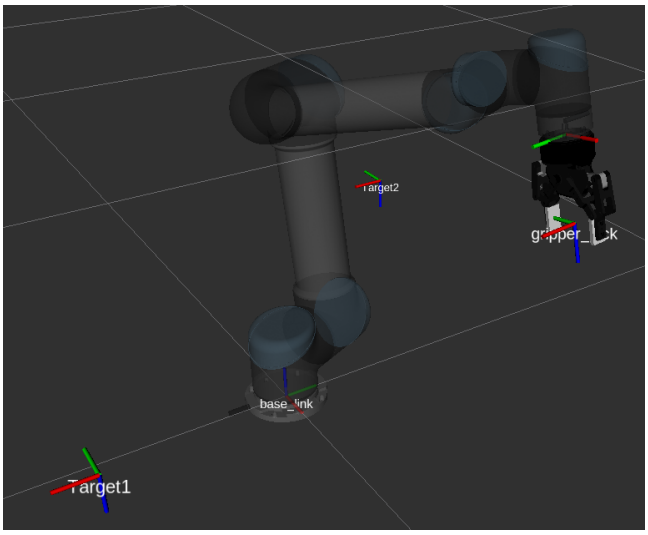


Figure 12: Back to home configuration.

The errors for the resolved-rate controller implementation are summarized in the table below:

Desired Frame	$d_{SO(3)}$	$d_{\mathbb{R}^3}$
gst_1	0.0130	0.0180
gst_2	0.0140	0.0171

Table 2: Resolved-rate controller implementation error metrics.

4.3 Transpose Jacobian Controller Implementation

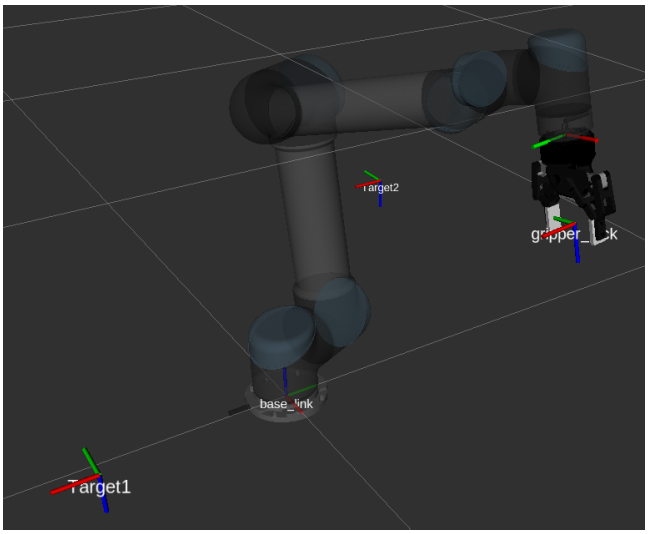


Figure 13: Home configuration.

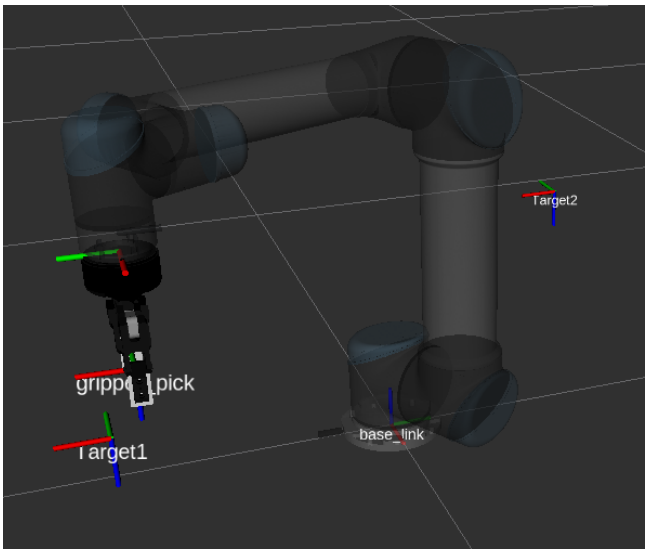


Figure 14: Above starting frame. Note that the gripper frame's position is further from the desired point directly above the target frame because the positional tolerance was increased to 5 cm to keep runtime to a reasonable length.

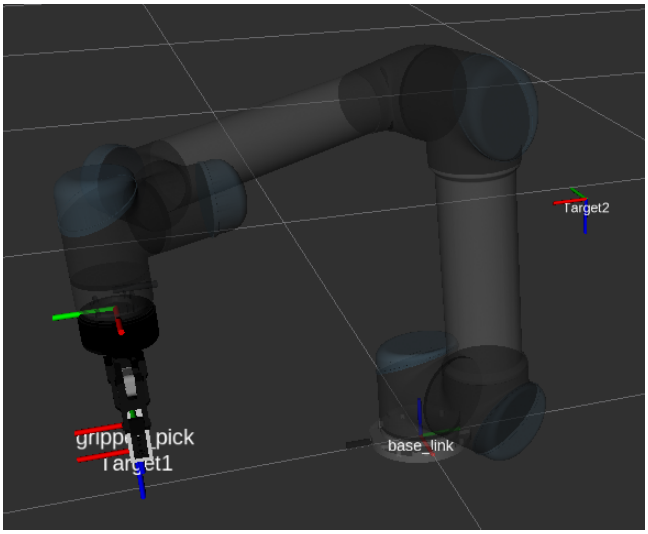


Figure 15: At starting frame. Note that the gripper frame's position is further from the desired point directly above the target frame because the positional tolerance was increased to 5 cm to keep runtime to a reasonable length.

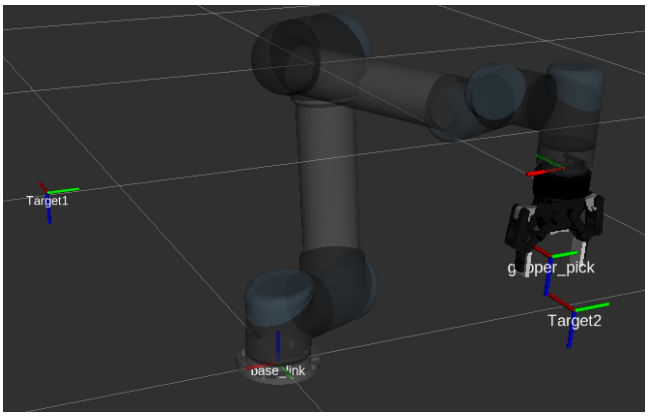


Figure 16: Above target frame. Note that the gripper frame's position is further from the desired point directly above the target frame because the positional tolerance was increased to 5 cm to keep runtime to a reasonable length.

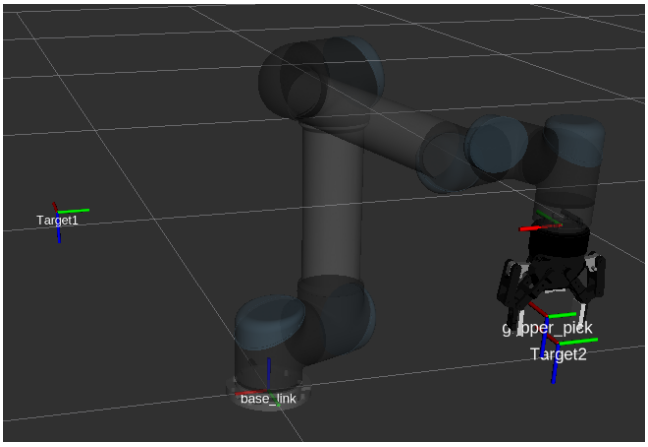


Figure 17: At target frame. Note that the gripper frame's position is further from the desired point directly above the target frame because the positional tolerance was increased to 5 cm to keep runtime to a reasonable length.

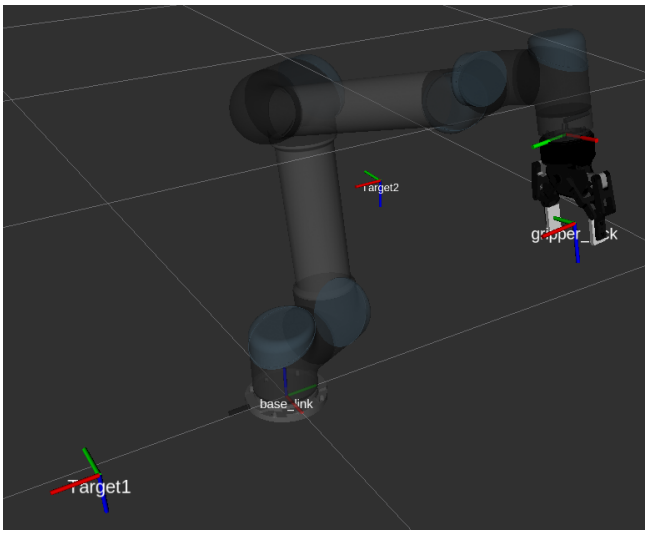


Figure 18: Back to home configuration.

The errors for the transpose-Jacobian implementation are summarized in the table below:

Desired Frame	$d_{SO(3)}$	$d_{\mathbb{R}^3}$
gst_1	0.0145	0.0484
gst_2	0.0369	0.0500

Table 3: Transpose-Jacobian controller implementation error metrics.

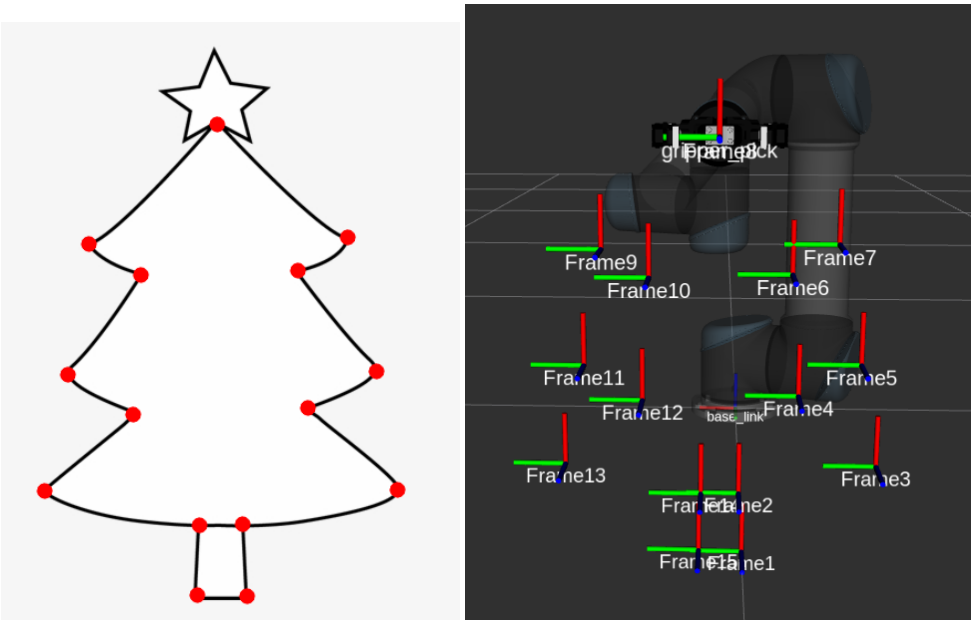
The slightly larger errors exhibited above are consistent with the choice to slightly increase the positional and angular tolerances for this controller due to excessive runtime. The tolerance values are discussed in Section 2.3.

5 Extra Task

For our extra task we decided to control the robot to trace the outline of a Christmas tree. This is an important task in real life scenarios such as the robot being used for painting and decorating in a commercial setting or applying some kind of adhesive or welding in an industrial setting, or potentially even hanging lights on a Christmas tree. In each of these scenarios, the orientation of the robot is not as important, but the precise positioning is.

To compute the points needed to trace the outline of the tree, we found a suitable image of the tree and hand picked the corner points for the robot to travel to, as shown in 19a. We decided to orient the tree in the xz plane starting at the point $[0, 0.3, 0]'$. To do this, we needed to rotate the points by 180 degrees

about the y axis, scale the pixels to be in meters, and translate the pixels by the distance of the first frame to the starting point. We chose a default orientation that would have the z-axis of the gripper parallel to the y-axis of the world frame. A screenshot of our results can be seen in 19b.



(a) Christmas tree frames (b) Christmas tree RVIZ simulation

Figure 19: Extra task: Christmas Tree

6 Contributions

6.1 All

- Collectively spent vast majority of the time on the main file and debugging all functions. This work was equally split among the team members and realistically constitutes the most significant contributions by each member.
- Collectively wrote this report

6.2 Spencer Powers

- Wrote *computeError.m*
- Wrote *ur5RRTransposeControl.m*
- Took screenshots and screen capture videos

6.3 Trevor Schwehr

- Wrote *filteredInvKin.m*
- Completed the extra task
- Wrote *findOptHome.m*

6.4 Kevin Wang

- Found offsets between tool frame and gripper

References

[1] Ryan Keating. Ur5 inverse kinematics, 2014.