Robot Devices, Kinematics, Dynamic and Control Final Project (Fall 2024)

Dev Dipak Ghiya Fazil Khan Manyu Garg Roy Siegelmann

1 Introduction

1.1 Goal and Purpose

The objective of this project is to program the UR5 robotic arm to perform a "place-and-draw" task, an adaptation of the classical "pick-and-place" operation. The task involves teaching the robot the start and end positions for drawing two parallel lines of 5 cm each, 10 cm apart, and implementing three programs to automatically execute the operation (as shown).
Note: It is worth mentioning that we have made our algorithms rotation invariant i.e for what ever orientation of two points that we give to the robot it will still draw two parallel lines each of 5cm.

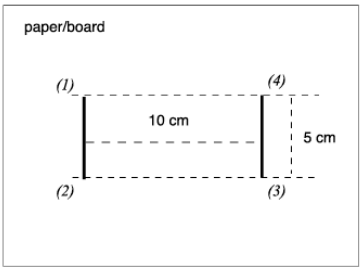
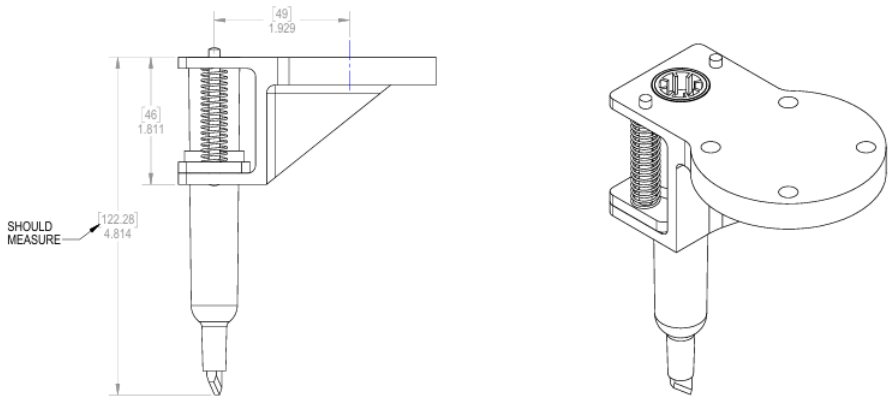


Figure 1: A schematic of two parallel 10-cm long lines

1.2 Drawing Tool

The robot uses a soft pen, attached via a custom-designed pen gripper, to draw the lines on a lab table within its workspace (as shown).

Figure 2: UR5 Robot with pen gripper and a pen



(a) (b)

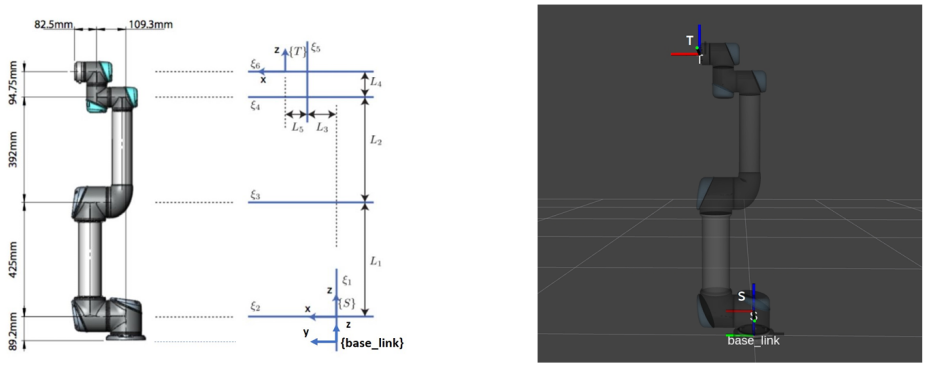
Figure 3: Pen gripper drawing in (a) 2D (b) 3D

1.3 Workflow

1. **UR5 Robot Parameters**
   Firstly, we calculated the DH parameter of the robot as seen in the table below. Utilizing these parameters we are calculating all the thetas of the robot from Link 0 to Link 6.

Link	d(m)	a(m)	alpha(rad)
0	0.089159	0	pi/2
1	0	-0.425	0
2	0	-0.3925	0
3	0	0	pi/2
4	0.1915	0	-pi/2
5	0.09465	0	0
6	0.0826	0	0

Table 1: DH Parameter



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

Figure 4: UR5 Robot Geometry in RVIZ

2. **Forward Kinematics**
   The `ur5FwdKin(q)` function calculates the forward kinematics of a UR5 robot manipulator. It takes a 6×1 joint angle vector q as input and returns the transformation matrix **gst**, which describes the pose (position and orientation) of the robot's end-effector relative to its base frame.

   The function uses the Denavit-Hartenberg (DH) convention, with the specified DH parameters for the UR5 robot (link offsets d, link lengths a, and link twists α). It iteratively computes the transformation matrix for

each joint using the `DH()` function and multiplies them sequentially to obtain the overall transformation $\mathbf{G}$.

Finally, an additional rotation matrix $\mathbf{R}_{z,\pi}$ (a 180-degree rotation about the z-axis) is applied to correct for a known orientation discrepancy, as referenced from Dr. Noah's recommended paper. The final result **gst** represents the corrected position and orientation of the end-effector.

The function calculates the rigid body transformation from link $i-1$ to link i using:

$$H_i^{-1}(a_{i-1}, \alpha_{i-1}, d_i, \theta_i) = [\mathrm{rot}(\mathbf{e}_1, \alpha_{i-1})\,\mathrm{trans}(\mathbf{e}_1, a_{i-1})]\,[\mathrm{rot}(\mathbf{e}_3, \theta_i)\,\mathrm{trans}(\mathbf{e}_3, d_i)]$$

which expands to:

$$H_i^{-1}(a_{i-1}, \alpha_{i-1}, d_i, \theta_i) = \begin{pmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i \cos\alpha_{i-1} & \cos\theta_i \cos\alpha_{i-1} & -\sin\alpha_{i-1} & -d_i \sin\alpha_{i-1} \\ \sin\theta_i \sin\alpha_{i-1} & \cos\theta_i \sin\alpha_{i-1} & \cos\alpha_{i-1} & d_i \cos\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

We can also calculate, forward Kinematics of a robot determine the configuration of the end effector given the relative configuration of each pair of the adjacent links of the robot. UR5 is open-chain robot consisting of 6 linkages. Using ur5FwdKin.m we were able to calculate the final transformation of the adjacent link frames, then the overall kinematics are given by

$$g_{st}(\theta) = g_{sl_1}(\theta_1) g_{l_1 l_2}(\theta_2) \cdots g_{l_{n-1} l_n}(\theta_n) g_{l_n t}$$

3. **Product of Exponential Formula**
For more detailed geometric description of the kinematics of the individual joints is generated by a twist associated with the joint axis which is represented as

$$\mathrm{gst}(\theta) = e^{\xi_1 \theta_1} e^{\xi_2 \theta_2} \cdots e^{\xi_n \theta_n}\,\mathrm{gst}(0)$$

For a revolute joint, the twist ξ_i has the form

$$\xi_i = \begin{bmatrix} -\omega_i \times q_i \\ \omega_i \end{bmatrix}$$

where $\omega_i \in \mathbb{R}^3$ is a unit vector in the direction of the twist axis and $q_i \in \mathbb{R}^3$ is any point on the axis.

$$\xi_i = \begin{bmatrix} v_i \\ 0 \end{bmatrix}$$

where $v_i \in \mathbb{R}^3$ is a unit vector pointing in the direction of translation.

4. **Point Capturing**
Using the parameters and transformations described above, we captured the first point (g_{p1}) and the fourth point (g_{p4}), which were obtained using the `getXi.m` function. From these points, we also determined the corresponding joint angles of the robot. These points were then used for training to predict the locations of points 2 and 3.

To ensure the line is both straight and rotation-invariant, we applied the cross product to the initial training points, namely points 1 and 4 (included in the code). This step guarantees that the line drawn will be 100% accurate.

5. **Error Calculation**
By applying the error formula below, we calculated the error between the robot end effector's actual starting point and the desired starting point and did same for the target positions which has been discussed in the "Algorithm Implementation" section.

Let $g_d = (R_d, r_d) \in SE(3)$ represent the desired start location, and $g = (R, r) \in SE(3)$ denote the actual start location that the robot end-effector reaches in your simulation. Then, error is

$$d_{SO(3)} = \sqrt{\mathrm{tr}\big((R - R_d)(R - R_d)^\top\big)}$$

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

2 Algorithms Implementations

The following steps outlines the methodology we followed during the implementation of this project:

1. We began by calculating the Denavit-Hartenberg (DH) parameters for the UR5 robot, which are essential for defining the robot's kinematic chain and transformations between frames.
2. Using the DH parameters, we computed the forward kinematics and the Body Jacobian matrix, which is used for controlling the robot's end-effector position and orientation.
3. To ensure the robot operates robustly, even in singular configurations, we implemented a manipulability analysis. This step helps avoid configurations where the robot loses degrees of freedom or control.
4. The getXi function was employed to extract the 6×1 twist vector, representing the spatial velocity of the robot in terms of its linear and angular components. This vector served as an input to the control algorithms for trajectory tracking.
5. Each algorithm was tailored to meet the project requirements for drawing precise lines using the UR5.
6. Through iterative testing and fine-tuning, we optimized the control gain (K) and time step (Δt) to minimize tracking errors and ensure the output lines were smooth and straight. This step was critical for maintaining system stability and performance.

2.1 Inverse Kinematics (IK-Control)

The Inverse Kinematics function calculates the necessary and chooses the best joint possible angles for point 2 and point 3. This function yields a 6 * 8 matrix as an output for a 4 * 4 matrix input. The theta values are calculated using the DH parameters and Paden-Kahan subproblems as a basis. However, for each angle, we get 8 possible solutions. We choose the best angle using a function defined in the same function file, This function takes two arguments one is a matrix of all 8 solutions and another a reference, the reference in our case for calculating position of point 2 point 1 will be the reference point and for calculating the position of point 3 point 4 will be the reference point 4, From the 8 possible solutions this function selects the solution which is closest to our reference.

To start, the Inverse Kinematics Control function (InvKinControl.m) was made to control the robot. It required input in the form of two 6*1 joint matrices, the start, and desired points. So, the function makes the UR5 robot move to the starting point/ joint angles and then to the final point/ joint angles.

The InvKinControl function is now used to draw the line from point 1 to point 2. Then, the UR5 robot moves to the temporary home position. After this, the robot moves to point 3 and draws a line from point 3 to point 4. The UR5 robot then moves to the home position.

2.2 Resolved-rate Control (RR-Control)

The objective is to implement a discrete-time resolved rate control system for the UR5 robot, the goal is to move the robot from an initial joint configuration q_1 to a desired configuration q_2 while minimizing position and orientation errors.

Methodology

Using the control law:

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

where:

- $K = 10$: Gain constant
- $T_{\text{step}} = 0.1$: Discrete time step
- $\mathbf{J}^b_{\text{st}}(\mathbf{q}_k)$: Body Jacobian at configuration $\mathbf{q}_k$
- $\boldsymbol{\xi}_k$: Error twist vector between the desired and current end-effector poses

The system halts under the following conditions:

- Singularity Detection: If the manipulability of the Jacobian becomes too small.
- Joint Limit Violation: If the second joint exceeds the range $[-\pi, 0]$, i.e., $q_k(2) \notin [-\pi, 0]$.

Implementation

1. **Initialization:** Define velocity/orientation thresholds, gain K, time step T_{step}, and the home configuration.
2. **Move to Initial Position:** The UR5 robot moves to q_1, and the initial rotation and translation errors are computed.
3. **Control Loop:** Joint angles q_k are iteratively updated using the control law until both position and orientation errors fall below the predefined thresholds.
4. **Final Position Analysis:** Compute the rotation and translation errors at the final configuration q_2 to evaluate system accuracy.

The resolved-rate control system successfully drives the robot to the desired configuration while maintaining stability and avoiding joint limit violations.

2.3 Jacobian Transpose Control (JT-Control)

The goal of this section is to implement a discrete-time Jacobian Transpose (JT) control system for the UR5 robot to move from an initial joint configuration q_{start} to a desired configuration q_{target} while iteratively adjusting control parameters for improved accuracy.

Methodology

The Jacobian Transpose control uses the update law:

$$\mathbf{q}_{k+1} = \mathbf{q}_k - KT_{\text{step}}\mathbf{J}_b^\top(\mathbf{q}_k)\boldsymbol{\xi}_k$$

where:

- $\mathbf{q}_k$: Current joint configuration at step k
- K: Gain constant, iteratively increased during execution
- $T_{\text{step}} = 0.05$: Discrete time step
- $\mathbf{J}_b^\top$: Transpose of the body Jacobian at configuration $\mathbf{q}_k$
- $\boldsymbol{\xi}_k$: Error twist vector representing linear and angular velocities

The control gain K is initialized to 0.03 and doubled iteratively every 120 iterations to improve convergence as the control process proceeds.

Implementation

1. **Initialization:** Define velocity/orientation thresholds ($v_{\text{threshold}} = 0.003$, $w_{\text{threshold}} = 5°$), gain K, time step T_{step}, and the home configuration.
2. **Move to Initial Position:** The UR5 robot moves to q_{start}, and initial rotation and translation errors are computed.
3. **Control Loop:** The joint configuration $\mathbf{q}_k$ is updated iteratively using the JT control law until the position and orientation errors fall below their respective thresholds. During execution:
   - K is increased iteratively every 120 steps to improve convergence.
   - The Jacobian manipulability is monitored to avoid singularities.
   - Joint limits for $q_k(2)$ are enforced to stay within $[-\pi, 0]$.
4. **Final Position Analysis:** Compute the rotation and translation errors at the final configuration q_{target} to verify system accuracy.

Key Considerations

- Singularity Handling: Execution halts if the manipulability measure approaches zero.
- Joint Limit Enforcement: Joint angles are monitored to ensure they stay within safe operational limits.

The JT control system successfully moves the UR5 to the desired configuration while dynamically tuning the control gain K, ensuring smooth and precise movement.

2.4 Result and Discussion

From our code implementation, we observed the following:

- IR-Control: Here, it got a smoother line with little deviation.

Points	**1**	**2**	**3**	**4**
Trace Error	6.5e-4	2.06e-4	6.16e-04	2.08e-04
Norm Error	1.73e-4	6.93e-5	5.98e-05	7.11e-05

Table 2: Error Table: IR-Control

- RR-Control: We got a fairly straight line with no error margin during the implementation. We optimized the Gain (K = 10) and delta_t (0.) to our case.

Points	**1**	**2**	**3**	**4**
Trace Error	1.25e-4	4.13e-4	1.87e-4	8.19e-4
Norm Error	4.70e-5	6.44e-5	3.81e-5	7.86e-5

Table 3: Error Table: RR-Control

- JT-Control: In this case, we were getting a jittering motion from the UR5 robot while drawing the desired line motion from P1 to P2 and P3 to P4; we concluded that by hyperparameterizing the Gain (K) and delta_t we could get smoother motion. In lower K and higher values of delta_t, we get a straight line till 4 cm, then the robot moves to the desired point via making a curve (as shown below).

Points	**1**	**2**	**3**	**4**
Trace Error	5.48e-4	0.013	5.27e-4	0.0086
Norm Error	4.35e-5	0.039	1.59e-4	0.0037

Table 4: Error Table: JT-Control

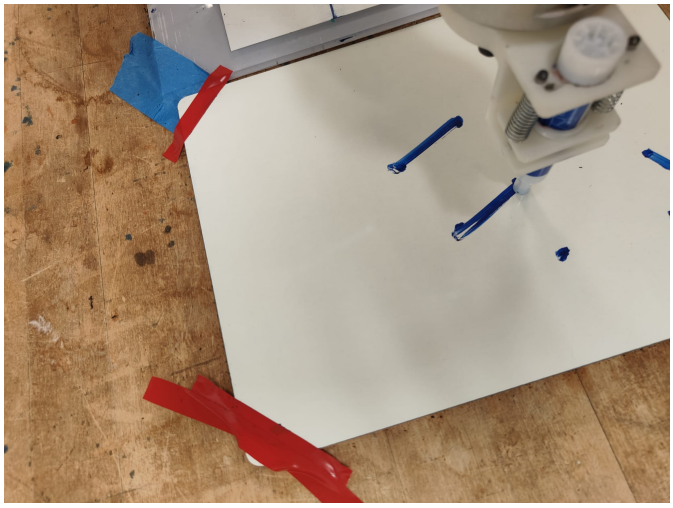


Figure 5: Lines drawn by the UR5 Robot

3 Safety Features

In all the control function files, we ensured sufficient clearances were available so that the robot did not hit the table, human, or any other object in the robot's workspace.

4 Game of Dots and Boxes (Extra Credit Task)

Dots and Boxes is a simple and fun game that can be played with two or more players. At the start of the game, players have an empty grid of dots, where grids can be of any size; in our case, the robot drew a 5 X 5 grid to play with. Usually, a coin is tossed or Rock-Paper-Scissors is played to see which players will go first, but in our case, Robot UR5 starts first.

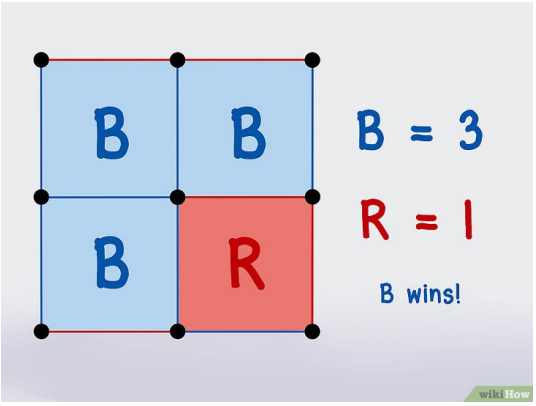


Figure 6: Dots and Boxes

In this game, at each turn, the player drags horizontal or vertical adjacent dots to draw a line. Boxes need four lines/walls to close; making a 4th line will give you a box, therefore earning you a point. The game is played until all boxes are filled with the player's name and initials written inside them. Players must be strategic and aware and should take the time to consider double-crossing their opponents. A double cross is something when you give a short chain of boxes to your challenger and leave them with no choice but to create a long chain for us on the next turn, which ensures that we will win the game.
Note: We were only partially able to complete the implementation of the program for this game.

5 Team Contributions

- IK-Control: Manyu Garg & Fazil Khan
- RR-Control: Fazil Khan & Roy Siegelmann
- JT-Control: Roy Siegelmann
- Final-Script: Dev Dipak Ghiya
- Extra Credit: Dev Dipak Ghiya & Fazil Khan
- Report: Dev Dipak Ghiya & Fazil Khan