Robot Devices, Kinematics, Dynamics and Contro
Team 4 : Final Project Report

Joseph Jia Rong Chen, Chun Yin Fan, Abhay Kodapurath Ajay

09 May 2024

1 Introduction

The aim of this project is to utilize the UR5 robot to execute a "place-and-draw" task. In this task, the objective is to draw two parallel lines, each measuring 5cm in length and separated by a distance of 10cm. The robot is trained to locate the initial and final positions of the soft pen and then it autonomously computes the endpoint for the first line and the starting point for the second line. The experiment explores three distinct control strategies - Inverse Kinematics, Resolved Rate, and Jacobian-Transpose - to accomplish the task.

2 Forward Kinematics for UR5

The joint orientation and geometry is represented by,

$$\begin{aligned}
\vec{\omega}_1^T &= \begin{bmatrix}0 & 0 & 1\end{bmatrix}, \quad \vec{\mathbf{q}}_1^T = \begin{bmatrix}0 & 0 & 0\end{bmatrix}, \\
\vec{\omega}_2^T &= \begin{bmatrix}0 & 1 & 0\end{bmatrix}, \quad \vec{\mathbf{q}}_2^T = \begin{bmatrix}0 & 0 & l_0\end{bmatrix}, \\
\vec{\omega}_3^T &= \begin{bmatrix}0 & 1 & 0\end{bmatrix}, \quad \vec{\mathbf{q}}_3^T = \begin{bmatrix}0 & 0 & l_0 + l_1\end{bmatrix}, \\
\vec{\omega}_4^T &= \begin{bmatrix}0 & 1 & 0\end{bmatrix}, \quad \vec{\mathbf{q}}_4^T = \begin{bmatrix}0 & 0 & l_0 + l_1 + l_2\end{bmatrix}, \\
\vec{\omega}_5^T &= \begin{bmatrix}0 & 0 & 1\end{bmatrix}, \quad \vec{\mathbf{q}}_5^T = \begin{bmatrix}0 & l_3 & 0\end{bmatrix}, \\
\vec{\omega}_6^T &= \begin{bmatrix}0 & 1 & 0\end{bmatrix}, \quad \vec{\mathbf{q}}_6^T = \begin{bmatrix}0 & 0 & l_0 + l_1 + l_2 + l_4\end{bmatrix}.
\end{aligned}$$

Twist,

$$\vec{\boldsymbol{\xi}}_{\boldsymbol{i}} = \begin{bmatrix} -\vec{\boldsymbol{\omega}}_{\boldsymbol{i}} \times \vec{\mathbf{q}}_{\mathbf{i}} \\ \vec{\boldsymbol{\omega}}_{\boldsymbol{i}} \end{bmatrix}$$

Forward kinematics,

$$g_{st}(t) = \exp(\xi_1\theta_1).\exp(\xi_2\theta_2).\exp(\xi_3\theta_3).\exp(\xi_4\theta_4).\exp(\xi_5\theta_5).\exp(\xi_6\theta_6).g_{st}(0)$$

where,

$$g_{st}(0) = \begin{bmatrix} R_0 & t_0 \\ 0 & 1 \end{bmatrix}$$

is the initial pose of the end-effector relative to the base(world) frame

3 Algorithm Workflow

To facilitate the "place-and-draw" task, the program is structured into several pivotal functions as described in Figure 1. Initially, the robot is displaced from its default home position to evade singularity issues. Subsequently, depending on whether the program operates in simulation or on the physical robot, the start and end points of the robot's trajectory are determined either at predefined coordinates or through user teaching. Upon the robot's return to the home position, calculations are made to determine the pen tip's position relative to the robot's kinematics, followed by the generation of the robot's path. Then, based on the user's input, a suitable control scheme is chosen to guide the robot along the desired path. Lastly, any motion errors are logged. The overall program flow is illustrated in the following figure. In subsequent sections, detailed explanations of the teach point protocol, pen tip conversion, path calculation, and controller algorithms will be provided.

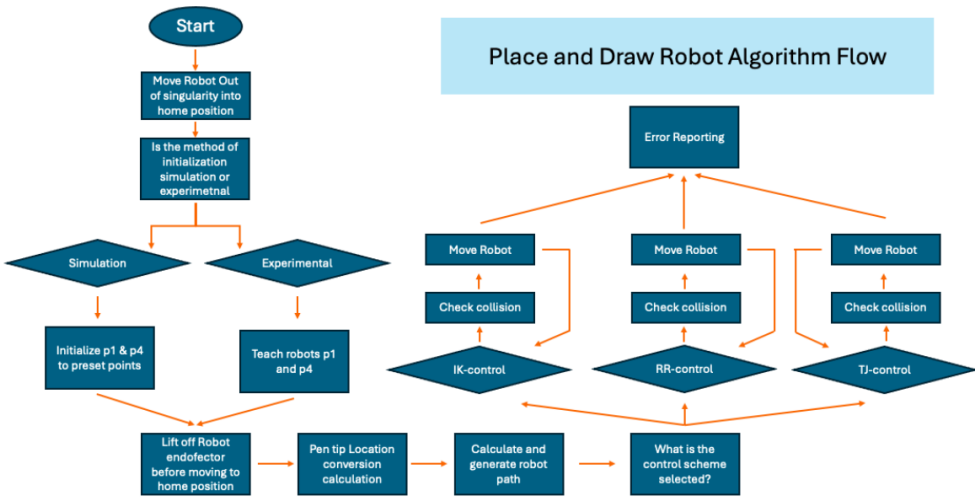

Figure 1: Flowchart for place-and-draw

3.1 Teach Point Protocol

The point of the teach point protocol is to allow users to physically move robot to the preferred start and end position for the pick and draw procedure. This is an useful procedure as it provides a time-efficient method in programming the robot.

3.2 Pen tip location conversion

With the extension of the compressible pen tip, the forward kinematic of the pen tip needs to be derived to achieve accurate drawing. This is especially important because the robot kinematic needs to take in the tilt of robot manipulator during the teach point procedure to ensure it is the pen tip that reaches the desired position instead of the robot end effector. This is achieved by using a simple homogenous transformation g to representation the transformation from tool0 frame to the pen tip frame. Considering that there are not axis of rotation between tool0 frame and pen tip frame, the transformation matrix only involves the positional displacement based on the pen geometry.

3.3 Path Calculation

Considering the transformation between the pen tip and tool0 frame, the path planning begins by employing homogeneous transformation to accurately determine the x, y position of the pen tip on the table surface. Table height sampling involved physically moving the robot to various points on the table and calculating forward kinematics. It was observed that the table exhibits negligible height differences, hence it can be effectively modeled as a flat surface at a consistent height relative to the robot's base. The calculated x, y, position of the pen is concatenated to a height z that considers pen compression. Additionally, for ease of drawing, the pen is oriented perpendicular to the table.

These assumptions enabled the expression of desired start and end points in joint configuration. To ensure that the two lines drawn are consistently 10 cm apart regardless of the distance between the two points, an algorithm is employed to calculate the required direction of the lines. Subsequently, point 2 and point 3 are extended 5 cm in the calculated direction. Once points 1, 2, 3, and 4 are determined, an additional 4 points are added directly above each point to serve as lift-off positions.

Finally, the 8 points are interconnected into a path. Points 1 to 2 and 3 to 4 are set at the table's height, enabling the robot to execute the drawing task effectively.

3.4 Collision Checking

To ensure the safety and working of the robot during movement, safety checks are conducted during robot motion. The safety check whether the robot is in risk of colliding with external obstacles or itself as well as checking for singulatives. The method of checking for collision is by limiting the robot manipulator to operate in a known, safe operating safe. The method of checking for singularity is by finding the determinant of the Jacobian and

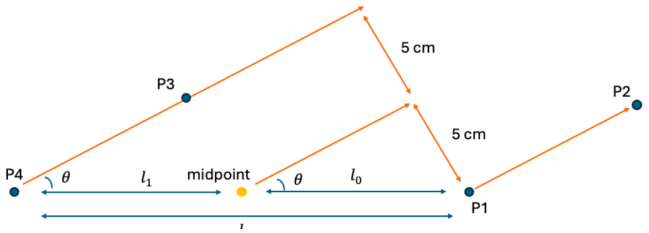

Figure 2: Path calculation

checking if it is close to 0. In this project, a tolerance of 10^{-4} is used to find the singularity. Robot motion will terminate and raise an error when it encounters a singularity or moved outside of the allocated space.

Algorithm 1 Collision Check

Initialize upper and lower joint limits for the allowable robot workspace.
if robot end effector height $<$ table height **then**
    Switch to pendant control and send out error warning.
end if
if any joint is smaller than lower joint limit or greater than upper joint limit **then**
    Switch to pendant control and send out error warning.
end if
if $\det(J) <$ tolerance **then**
    Switch to pendant control and send out error warning.
end if

Algorithm 2 Teach Point Protocol

Switch to pendant control
Input: User input to register robot configuration 1
Wait for user input
Input: User input to register robot configuration 2
Wait for user input
Lift robot manipulator and return to home position

4 Controllers

4.1 Inverse Kinematics Controller

The primary function of the IK controller is to compute the joint angles required to attain a specific end-effector pose in the robot's joint-space. This process involves solving a series of subproblems, as outlined in MLS. Since we are provided with the solver for inverse kinematics, we just need to find the optimal pose determined from this function.

4.2 Resolved-Rate Controller

The Resolved Rate controller is a control scheme that leverages the disparity between the target and current configurations to regulate the velocity of robot arms. In this method, a predetermined path is employed, and the desired velocity, time increment, and current position are used to determine the subsequent desired location of the robot. In addition, a feedforward component is also added to the controller to increase response time and reduce oscillation. Together with the feedback and feedforward component, a stable and effective controller is achieved.

4.3 Jacobian-Transpose Controller

The transpose Jacobian controller utilizes similar concept as the Resolved Rate control with the single difference where the transpose of the Jacobian is used instead of the inverse Jacobian when calculating the distance to the next target location. This change can be justified by interpreting the control scheme as driven by virtual force. The algorithm has the same flow as the Resolved Rate controller.

Algorithm 3 IK Control

Require: g_{desired}: Desired end-effector pose
Require: ur5: UR5 object
Require: δ_x: Change in end-effector pose
Require: δ_t: Time step

Output:
Get current joint angles θ from the UR5 object
Compute current end-effector pose gst using forward kinematics with θ
Compute the error between desired pose g_{desired} and current pose gst, and obtain the corresponding twist ξ_{cur}
Compute the Jacobian matrix J using the current joint angles θ
if $|\det(J)| < 0.01$ **then**
    Set current_err to -1 and raise a singularity error
end if
Use inverse kinematics to compute the new joint angles q_{new} corresponding to the desired pose g_{desired}
Find the joint configuration q_{new} with the minimum error compared to the current joint angles θ
Compute the joint velocity δ as the difference between q_{new} and θ, divided by the time step δ_t
Update the joint angles q_{new} using the computed joint velocity δ and the time step δ_t
Move the UR5 robot to the updated joint angles q_{new}
Compute the positional error current_err as the norm of the translational component of the error between the desired and current end-effector poses
Output current_err, q_{new}, gst, and θ

Algorithm 4 Resolved-rate Control

Require: g_{desired}: Desired end-effector pose expressed in homogeneous representation.
Require: K: Gain of the controller.
Require: $ur5$: UR5 object.
Require: δ_x: Distance.
Ensure: $finalerr$: Final positional error in cm between robot pose and the desired pose. Returns -1 if too close to singularity.

Initialize *condition* to **true**.
while *condition* is **true do**
    Get current joint angle in homogeneous expression.
    Calculate error twist using $g_{st}^{-1} * g_{st}$ and get the twist representation.
    Compute current body Jacobian using joint angle θ.
    if $|J| < 10^{-4}$ **then**
        Set $finalerr$ to -1 and raise singularity error.
    end if
    if positional and rotation error < tolerance **then**
        condition $\leftarrow$ **false**.
    else
        Calculate δ, the additional angular rotation for each joint.
        if largest rotation > maximum allowable distance between speed limit **then**
            Scale δ to equal the maximum allowable distance.
        end if
        Calculate the desired new joint angle and move to the new location.
    end if
end while
Calculate $finalerr$ of the robot manipulator position and return result.

Algorithm 5 Jacobian-transpose Controller

Require: g_{desired}: Desired end-effector pose expressed in homogeneous representation.
Require: K: Gain of the controller.
Require: $ur5$: UR5 object.
Require: δ_x: Distance.
Ensure: $finalerr$: Final positional error in cm between robot pose and the desired pose. Returns -1 if too close to singularity.

Initialize *condition* to **true**.
while *condition* is **true do**
  Get current joint angle in homogeneous expression.
  Calculate error twist using $g_{st}^{-1} * g_{st}$ and get the twist representation.
  Compute current body Jacobian using joint angle θ.
  if $|J| < 10^{-4}$ **then**
    Set $finalerr$ to -1 and raise singularity error.
  end if
  if positional and rotation error $<$ tolerance **then**
    condition $\leftarrow$ **false**.
  else
    Calculate δ, the additional angular rotation for each joint.
    if largest rotation $>$ maximum allowable distance between speed limit **then**
      Scale δ to equal the maximum allowable distance.
    end if
    Calculate the desired new joint angle and move to the new location.
  end if
end while
Calculate $finalerr$ of the robot manipulator position and return result.

Figure 3: Drawing pose

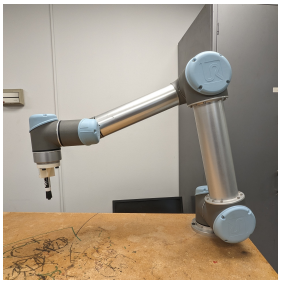

Figure 4: Home pose

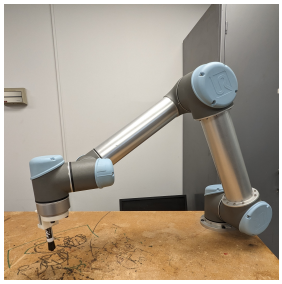

Figure 5: Lift pen pose

5 Results

We performed our tests in simulation and on the physical robot. For simulation, Figure 6 illustrates that the Jacobian-transpose controller exhibits the largest error among all controllers, while the IK-based controller demonstrates the smallest error. Similarly, in terms of rotational error(Figure 7), the Jacobian-transpose controller exhibits a higher error compared to the RR and IK controllers, which approach zero. It can be noticed that rotational errors for RR and IK are close to zero which is expected since there is negligible rotational motion. Figures 8 to 18 show various plots for tests on the physical robot.

6 Extra Credit Task

For the extra credit task, our team came up with a method to enable robot arms to draw out black and white images as shown in Figure 19. To demonstrate the working of our method, out team utilized a cartoon Kirby drawing. The first step is to load and convert the image into binary image as shown in the figures. Next, the skeleton morphological library was utilized to generate the skeleton of the drawing so that each line is shrunk down to single pixel representation. Lastly, to remove the branches of the image, the spur morphological operation was utilized with the resulting images shown below. This series of process successfully reduce the thick and uneven line into a single pixel representation.

To generate the path for the robot, the traveling salesman algorithm was utilized [1]. In essence, we generated a path that arrives and leaves a single point once. In addition, to prevent the robot from drawing undesired lines

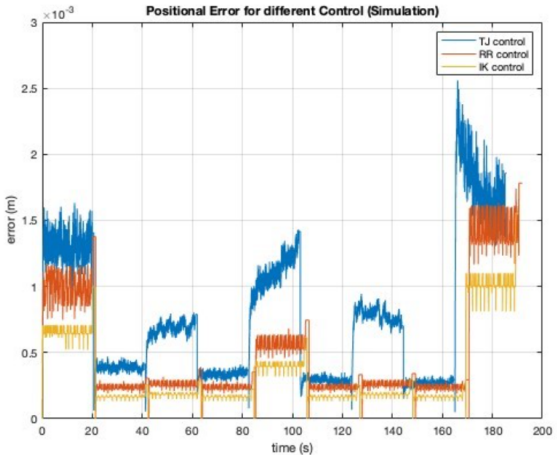

Figure 6: Positional errors(simulation)

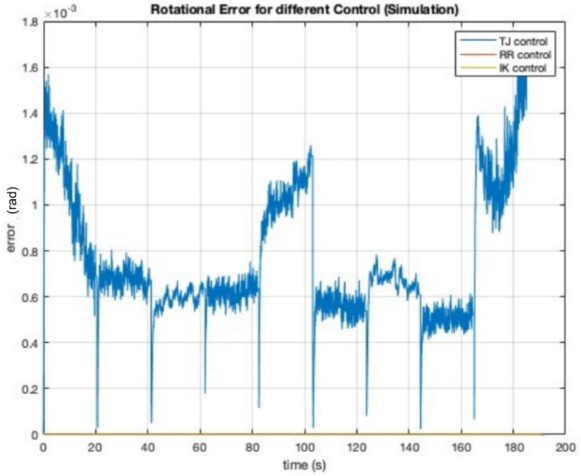

Figure 7: Rotational errors(simulation)

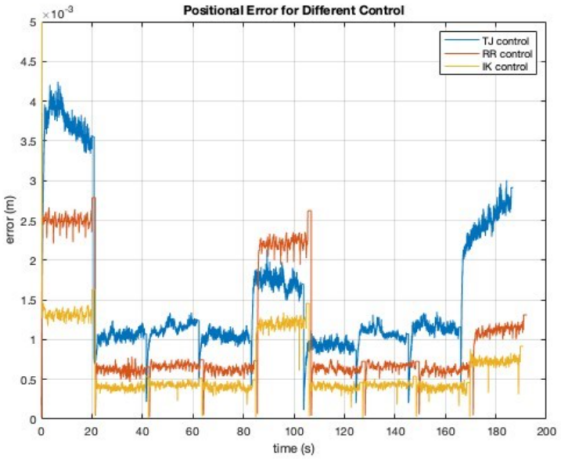

Figure 8: Positional errors

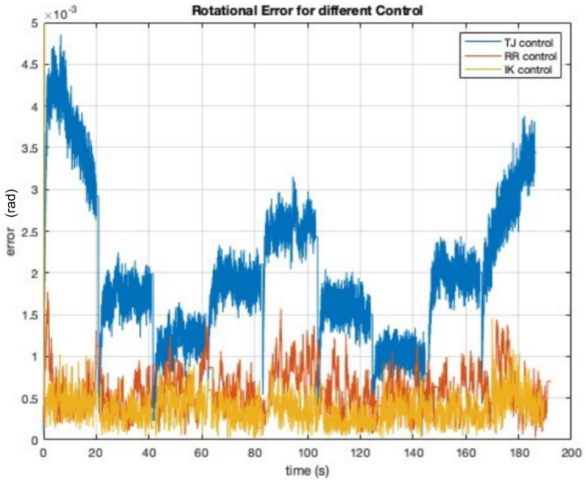

Figure 9: Rotational errors

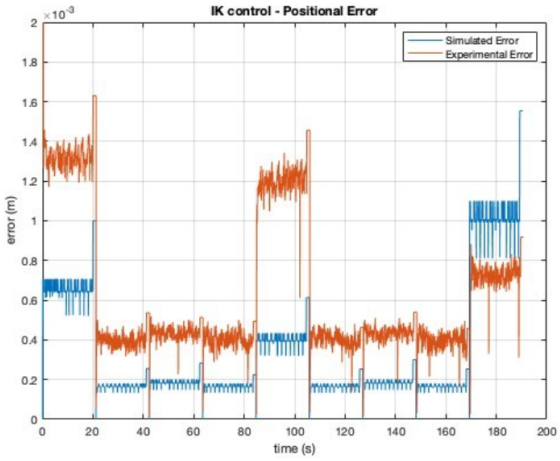

Figure 10: Positional error for IK

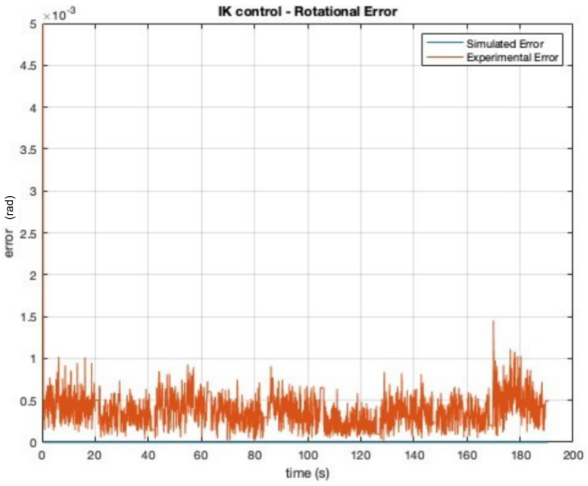

Figure 11: Rotational error for IK

as it travels from two far points, our team have set a cutoff distance that requires the robot to lift of vertically before moving. This resulted in a working path that can effectively draw out the desired image.

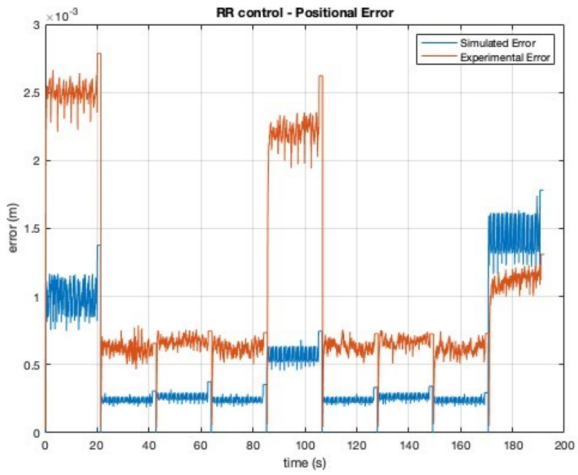

Figure 12: Positional error for RR

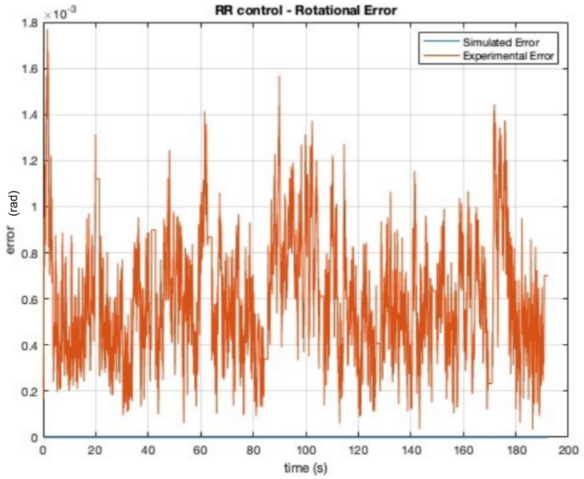

Figure 13: Rotational error for RR

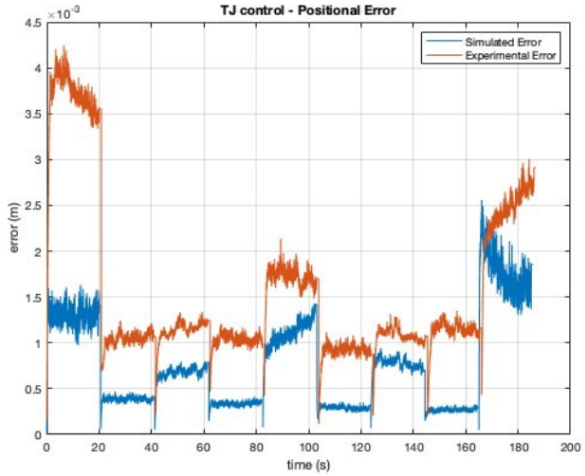

Figure 14: Positional error for TJ

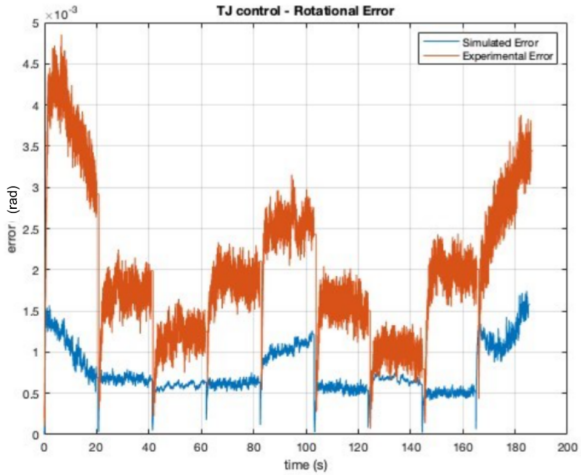

Figure 15: Rotational error for TJ

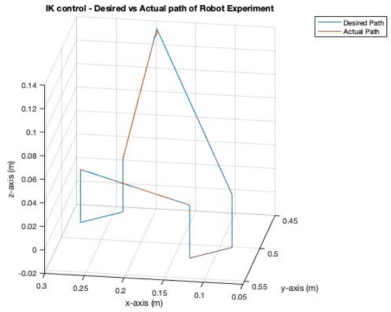

Figure 16: Desired vs. actual trajectories for IK

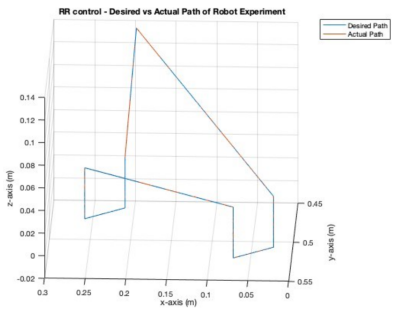

Figure 17: Desired vs. actual trajectories for RR

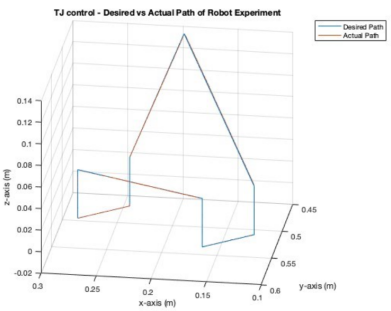

Figure 18: Desired vs. actual trajectories for TJ

7 Conclusion

In conclusion, this project aimed to explore the application of different control strategies - Inverse Kinematics, Resolved Rate, and Jacobian-Transpose - in executing a "place-and-draw" task using the UR5 robot. The task involved drawing two parallel lines, each 5cm in length and separated by a distance of 10cm.

Through experimentation, it was observed that each control strategy presented unique advantages and challenges in achieving the desired task. Inverse Kinematics provided precise control over the end-effector position,

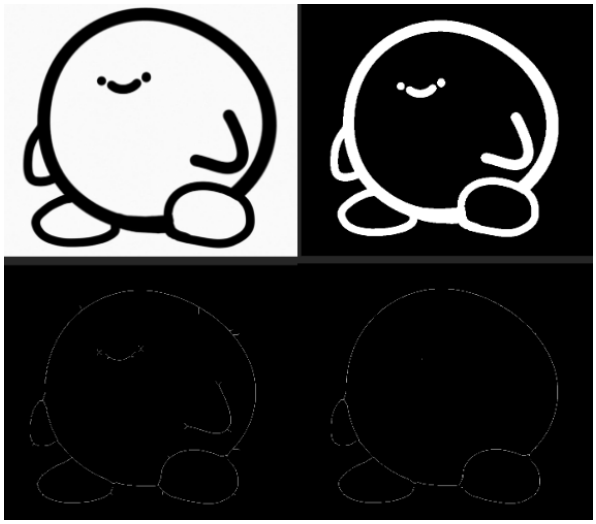

Figure 19: End-effector trajectory calculation

Figure 20: Portrait drawn by UR5

enabling accurate drawing of the lines. Resolved Rate offered a smooth and continuous motion, albeit with some limitations in terms of accuracy. Jacobian-Transpose demonstrated effectiveness in adapting to changes in the environment, ensuring robustness in task execution.

Additionally, we also used IK controller to successfully draw an input image using UR5 by estimating waypoints from the image and using traveling salesman algorithm to calculate the path.

8 References

1. https://www.mathworks.com/matlabcentral/fileexchange/71226-tspsearch

9 Contributions

Joseph and Chun developed the algorithms and tested them on the physical robot. Abhay assisted with IK control and prepared the report.