Feedback Controllers and Visual Servoing

Yuvan Sharma Ziteng (Ender) Ji

Abstract—**In this project, we design and implement control strategies for a robotic manipulator to execute precise trajectories and perform visual servoing. Three control methods are explored: Jointspace PD Velocity Control, Jointspace PD Torque Control, and Workspace Velocity Control. Each controller is theoretically derived, with the PD Velocity controller and Workspace Velocity controller also being implemented on a Sawyer single-arm robot. The robot is tasked with executing predefined trajectories, including straight-line, circular, and polygonal paths. We further integrate a vision system for visual servoing, enabling the robot to dynamically track AR markers in real time. Experimental results are used to compare the performances of the controllers in both trajectory tracking and visual servoing, with comparisons between desired and actual end-effector states. Tuning procedures for the controller gains are also detailed, and the trade-offs in performance between the control methods are analyzed. The findings highlight the effectiveness of feedback control and its applications in tasks requiring high accuracy, adaptability, and real-time adjustments. Potential applications of this work include industrial automation, assistive robotics, and autonomous navigation systems.**

I. Introduction

Robotic manipulators have become integral to applications requiring high precision, adaptability, and robustness in dynamic environments. Central to their functionality are effective control strategies that ensure desired trajectories are followed accurately while adapting to environmental variations. This project investigates three distinct control methods—Jointspace PD Velocity Control, Jointspace PD Torque Control, and Workspace Velocity Control—applied to a robotic arm for trajectory tracking and visual servoing tasks.

Visual servoing in particular represents a potential real-world application of such control methods by integrating real-time feedback from a vision system that detects AR markers to guide the manipulator's motion. This capability is critical for tasks like object tracking, pick-and-place operations, and human-robot interaction.

In this report, we explore the theory and implementation of these control methods, comparing their effectiveness in trajectory tracking and visual servoing scenarios. Through experiments and analysis, we aim to highlight the strengths and limitations of each approach, providing insights into their practical applications in robotics.

II. Method

A. Jointspace PD Velocity Control

Jointspace PD Velocity Control is a feedback control method designed to guide the robot's joints to track desired positions and velocities in joint space. It is a proportional-derivative (PD) controller, meaning it computes the control input (which consists of joint velocities) by combining a proportional term (based on the positional error) and a derivative term (based on the rate of change of the positional error). This approach ensures smooth and stable tracking of the desired trajectory while minimizing oscillations.

Formally, the objective is to compute the joint velocities $\dot{\theta}(t) \in \mathbb{R}^7$ for the Sawyer robot's seven joints such that the actual joint angles $\theta(t)$ converge to the desired joint angles $\theta_d(t)$.

1) Error Definition: The error in joint angles at time t is defined as:

$$e(t) = \theta_d(t) - \theta(t) \tag{1}$$

The derivative of the error, representing the rate of change of the error, is:

$$\dot{e}(t) = \dot{\theta}_d(t) - \dot{\theta}(t) \tag{2}$$

2) Control Law Derivation: The control input for the robot is the joint velocity $\dot{\theta}(t)$. Using a PD control approach, we can define $\dot{\theta}(t)$ as:

$$\dot{\theta}(t) = \dot{\theta}_d(t) + K_p e(t) + K_v \dot{e}(t) \tag{3}$$

where we have $\dot{\theta}_d(t)$ as the feedforward term, which ensures that the robot follows the nominal desired velocity trajectory. $K_p e(t)$ is the proportional term, acting to reduce the positional error $e(t)$, and $K_v \in \mathbb{R}^{7\times 7}$ is the derivative gain matrix, controlling the damping effect. K_p, K_v are usually diagonal matrices, as each joint is considered independently. K_p controls how aggressively the controller responds to positional error. High values may lead to oscillations if K_v is not tuned correctly. In contrast, larger values on the diagonal for K_v make the system more resistant to rapid changes in error.

B. Jointspace PD Torque Control

Note: In this section, we mainly use information from section 5.2 from Chapter 4 of A Mathematical Introduction to Robotic Manipulation (MLS) [1].

Jointspace PD Torque Control is a feedback control strategy that computes the torques required at each joint to track a desired trajectory. Unlike velocity control, which outputs joint velocities, this method directly addresses the dynamics of the robot by incorporating the effects of inertia, Coriolis forces, and gravity into the control law. This ensures precise and robust tracking, even in the presence of dynamic effects. The goal is to compute the torques $\tau \in \mathbb{R}^7$ for Sawyer's seven joints such that the actual joint positions $\theta(t)$, velocities $\dot{\theta}(t)$, and accelerations $\ddot{\theta}(t)$ converge to their desired values $\theta_d(t)$, $\dot{\theta}_d(t)$, and $\ddot{\theta}_d(t)$, respectively.

1) Error Definition: The control law uses a Proportional-Derivative (PD) approach to reduce the error in joint space. This controller also uses positional error $e(t)$ and velocity error $\dot{e}(t)$, defined in Equations 1 and 2 respectively.

2) Dynamic model of the Robot: The dynamics of a robotic manipulator are governed by:

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + N(\theta, \dot{\theta}) = \tau \tag{4}$$

where $M(\theta)$ is the inertia matrix, which is positive definite and symmetric. $C(\theta, \dot{\theta})$ is the Coriolis matrix, which accounts for velocity-dependent effects. $N(\theta, \dot{\theta})$ represents gravity and external forces, and τ represents the joint torques.

3) Computed Torque Control Law: The computed torque control law builds in the robot's dynamics to ensure tracking of desired trajectories. The control torque is given by the following equation from *MLS* section 5.3 [1]:

$$\tau = M(\theta)\ddot{\theta}_d + C(\theta, \dot{\theta})\dot{\theta}_d + N(\theta, \dot{\theta}) + M(\theta)(-K_v\dot{e} - K_p e) \tag{5}$$

Here, $\ddot{\theta}_d$ represents the desired joint accelerations, $\dot{\theta}_d$ the desired joint velocities, and θ_d the desired joint positions. K_v represents the velocity gain matrix (positive definite), and K_p is the position gain matrix (positive definite). We note that the control law above can be broken down into two components, the feedforward term τ_{ff} and the feedback term τ_{fb}. The feedforward term τ_{ff} drives the robot along the desired trajectory in the absence of disturbances or errors, and the feedback term τ_{ff} corrects for errors in position and velocity, stabilizing the system and ensuring tracking. The detailed decomposition of each term is provided below:

$$\tau_{ff} = M(\theta)\ddot{\theta}_d + C(\theta, \dot{\theta})\dot{\theta}_d + N(\theta, \dot{\theta})$$

$$\tau_{fb} = M(\theta)(-K_v\dot{e} - K_p e)$$

and we have the complete control law as:

$$\tau = \tau_{ff} + \tau_{fb}$$

4) Error Dynamics: In this section we find the error dynamics. Substituting the control law in Equation 5 into the robot's dynamics in Equation 4, we have the equation:

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + N(\theta, \dot{\theta}) = M(\theta)\ddot{\theta}_d + C(\theta, \dot{\theta})\dot{\theta}_d + N(\theta, \dot{\theta}) + M(\theta)(-K_v\dot{e} - K_p e)$$

Simplifying the equation above gives us:

$$M(\theta)\ddot{e} + C(\theta, \dot{\theta})\dot{e} = -M(\theta)(K_v\dot{e} + K_p e)$$

Since $M(\theta)$ is always positive definite, we get the equation:

$$\ddot{e} + K_v\dot{e} + K_p e = 0$$

This is a linear second-order differential equation describing the error dynamics. The stability and convergence of the error to zero depend on appropriately choosing K_v and K_p.

C. Workspace Velocity Control

Workspace Velocity Control operates in the end-effector space rather than the robot's joint space. The goal is to track a desired trajectory defined in the robot's workspace, $g_{sd}(t) \in SE(3)$, which specifies the position and orientation of the robot's end-effector in 3D space. This method combines feedforward and feedback terms to compute the desired spatial velocity $U^s(t)$, which is then converted into joint velocities $\dot{\theta}(t) \in \mathbb{R}^7$ using the pseudo-inverse of the Jacobian $J^{\dagger}(\theta)$.

1) Error Definition: To reduce the error between the current configuration of the end-effector $g_{st}(t) \in SE(3)$ and the desired configuration $g_{sd}(t)$, we define the error configuration:

$$g_{td} = g_{st}^{-1} g_{sd} \tag{6}$$

The error configuration $g_{td}(t)$ describes the transformation required to align the current end-effector pose with the desired pose. Using g_{td} , we compute the body-frame velocity ξ_{td}^b that would reduce this error:

$$\xi_{td}^b = (\log(g_{td}))^{\vee} \tag{7}$$

where the $\vee$(vee) operator converts a twist in $se(3)$ to the $6D$ twist coordinates vector. Since ξ_{td}^b is defined in the body frame, it must be transformed into the spatial frame as below:

$$\xi_{td}^s = \mathrm{Ad}_{g_{st}}\xi_{td} \tag{8}$$

where $\mathrm{Ad}_{g_{st}}$ is the adjoint transformation matrix associated with g_{st}, mapping body-frame velocities to spatial-frame velocities.

2) Control Law Derivation: The desired spatial velocity $U^s(t)$ is computed as the sum of the feedforward term V_d^s (the desired spatial-frame velocity along the trajectory) and feed-back term $K_p\xi_{td}^s$ (a correction term proportional to the spatial error). This gives us the Cartesian control law:

$$U^s = V_d^s + K_p\xi_{td}^s \tag{9}$$

Here, K_p is a 6×6 matrix constructed as below:

$$K_p = \mathrm{diag}(K_x, K_y, K_z, K_{\omega_1}, K_{\omega_2}, K_{\omega_3})$$

where K_x, K_y, K_z are gains for translational error in the x, y, and z directions, and $K_{\omega_1}, K_{\omega_2}, K_{\omega_3}$ are gains for angular velocity components.

3) Mapping to Joint Velocities: To convert the workspace velocity U^s into jointspace velocity $\dot{\theta}(t)$, we use the pseudo-inverse of the spatial Jacobian $J(\theta)$:

$$\dot{\theta}(t) = J^{\dagger}(\theta)U_s$$

where $J^{\dagger}(\theta)$ is the pseudo-inverse:

$$J^{\dagger}(\theta) = J^T(JJ^T)^{-1}$$

4) Summarization - Key Steps in the Control Process:

1) Compute the error configuration:

$$g_{td} = g_{st}^{-1} g_{sd}$$

2) Compute the body-frame velocity from the error:

$$\xi_{td}^{b} = (\log(g_{td}))^{\vee}$$

3) Transform the body-frame velocity into the spatial frame:

$$\xi_{td}^{s} = \mathrm{Ad}_{g_{st}} \xi_{td}^{b}$$

4) Compute the desired spatial velocity:

$$U^{s} = V_{d}^{s} + K_{p} \xi_{td}^{s}$$

5) Map the spatial velocity to joint velocities:

$$\dot{\theta}(t) = J^{\dagger}(\theta) U_{s}$$

D. Tuning Procedure

The tuning procedures used for each controller as described below:

- **Jointspace PD Velocity Control:** We first tuned K_p values, increasing them individually for each joint until the actual joint positions began oscillating about the desired positions. Then, we gradually increased K_v to reduce the oscillations until we found a stable configuration. Our final gain values are below:

$$K_p = [0.08, 0.4, 0.34, 0.3, 0.4, 0.4, 0.6]$$

$$K_v = [0.02, 0.01, 0.02, 0.005, 0.008, 0.008, 0.008]$$

- **Workspace Velocity Control:** Tuning the workspace controller was more intuitive than the joint velocity controller, as modifying K_p for the x, y, z directions directly corresponded to a bigger response in the same direction. We thus tuned these values until the end-effector very closely followed the desired positions. We kept all the angular components of K_p equal, adjusting the magnitude until we achieved a reasonable balance between the position and orientation errors. Our final gain values are below:

$$K_p = [0.08, 1.2, 0.04, 0.2, 0.2, 0.2]$$

III. Experiments

We compared the ability of the PD joint velocity and workspace velocity controllers to follow four different shapes of trajectories: linear, circular, triangular, and square, as well as a visual servoing course consisting of seven individual movements. We also incldue the feedforward joint velocity controller for comparison.

We discuss each experiment and its results in detail below. We use end-effector error to compare the controllers, since the workspace controller cannot be compared to using joint positions or velocities. We only plot results for individual runs as the data between runs for the same trajectory showed minimal deviations.

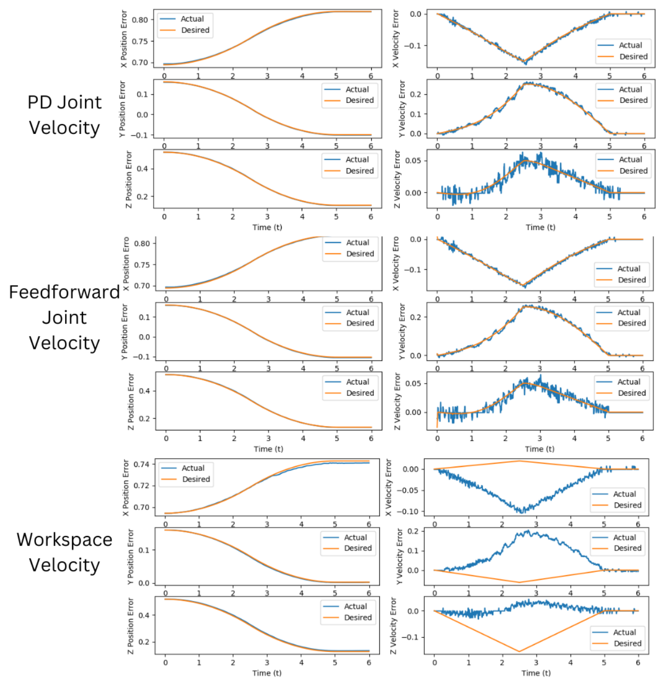

Fig. 1. Comparison between the errors in three controllers for a five second linear trajectory.

We note that for the workspace controller, we removed the movement to the start position (normally done using MoveIt) for the linear trajectory and visual servoing tasks, since we found the z offset between the desired position and actual position reached by MoveIt to be significant. However, we could not do this for the circular and polygon trajectory tasks (since the start point of the trajectory is different from the robot's initial position). This is the reason for the difference between desired and actual z positions for the workspace controller in these tasks.

A. Experiment 1

In this experiment, we recorded the desired and actual trajectory movements for all three controllers for a five second linear trajectory (results plotted in Figure 1) and a ten second circular trajectory of 10 cm radius (results plotted in Figure 2). To ensure fair comparison, all controllers were given the same start and end points.

It can be seen that for the linear trajectory, all controllers are able to follow the desired positions very closely. The PD joint velocity and feedforward joint velocity controllers also track the desired velocity well. We note that the workspace velocity controller is less accurate in this aspect, which is to be expected since there is no feedback term included in the controller to compensate for velocity differences.

For the more complicated circular trajectory — which requires constant direction changes and additionally maintaining a constant height — we see that both the joint velocity controllers are a little more noisy. In addition, while the magnitude difference is small, we see that the PD joint velocity controller is able to use feedback to stay around the required

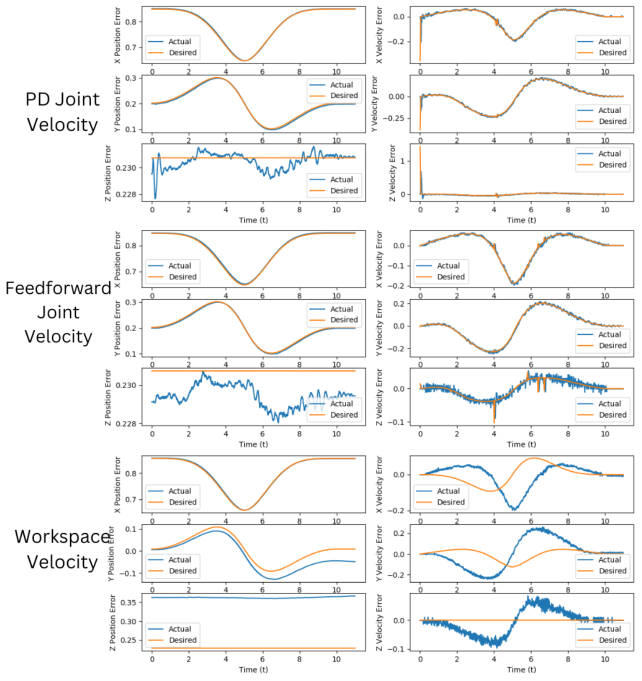

Fig. 2. Comparison between the errors in three controllers for a ten second circular trajectory.

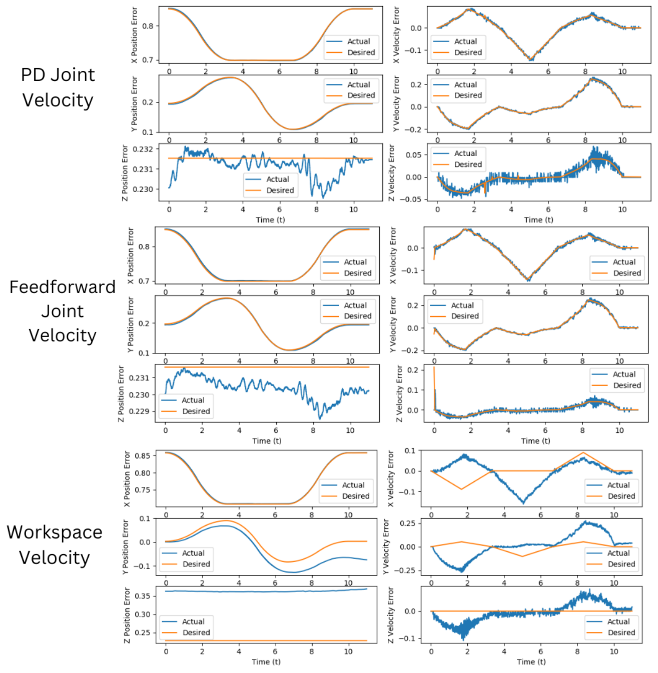

Fig. 3. Comparison between the errors in three controllers for a ten second triangular trajectory.

height, while the feedforward controller starts off lower and is never able to recover. The large difference in height for the workspace controller is explained by the MoveIt discrepancy described in the introduction of Section III.

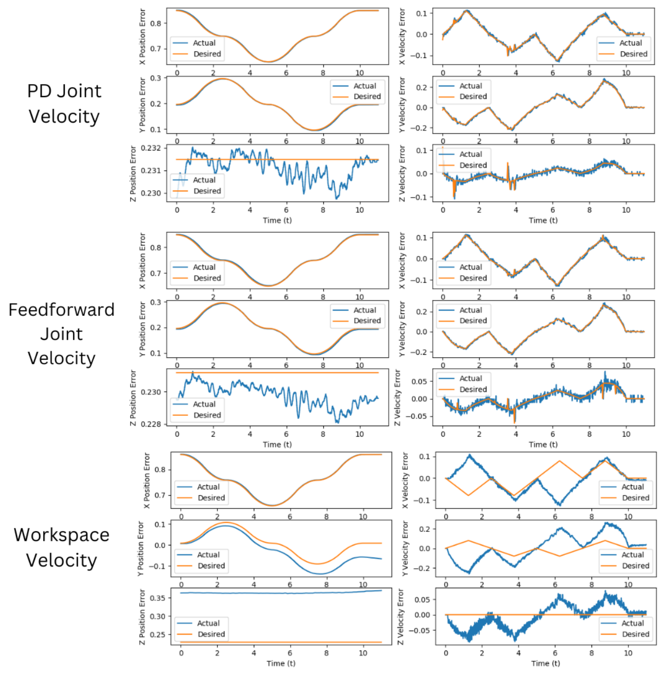

Fig. 4. Comparison between the errors in three controllers for a ten second square trajectory.

B. Experiment 2

In this experiment, similar to Experiment 1, we recorded the trajectory movements of the three controllers for two polygonal trajectories, one triangular and the other square. Once again, to ensure fair comparison, all controllers were given the same center point and thus the same overall polygon to trace. The results for the triangular trajectory are shown in Figure 3, and the results for the square trajectory are shown in Figure 4.

In both cases, trends similar to the circular trajectory (Figure 2) can be seen. The joint velocity controllers track the velocity well but are a little noisy in maintaining the height. The workspace controller is more smooth, and while it does not track the velocity that accurately (because of no velocity feedback), it still achieves the required positions accurately. In addition, for the workspace controller, a gap can be seen towards the end of the trajectory for the y position, which we could not completely remove even after significantly increasing K_p for this dimension compared to the other values.

C. Experiment 3

In this experiment, we created a visual servoing course consisting of seven individual segments to test the ability of the three controllers to accurately follow the movement. To ensure reproducibility and as fair a comparison as possible, we marked each point of the course on the table. The results are plotted in Figure 5, and a video link is available in Section IX.

Overall, all controllers are able to follow the general movement of the AR marker, showcasing their potential to be used for visual servoing and other similar tasks. However, we did

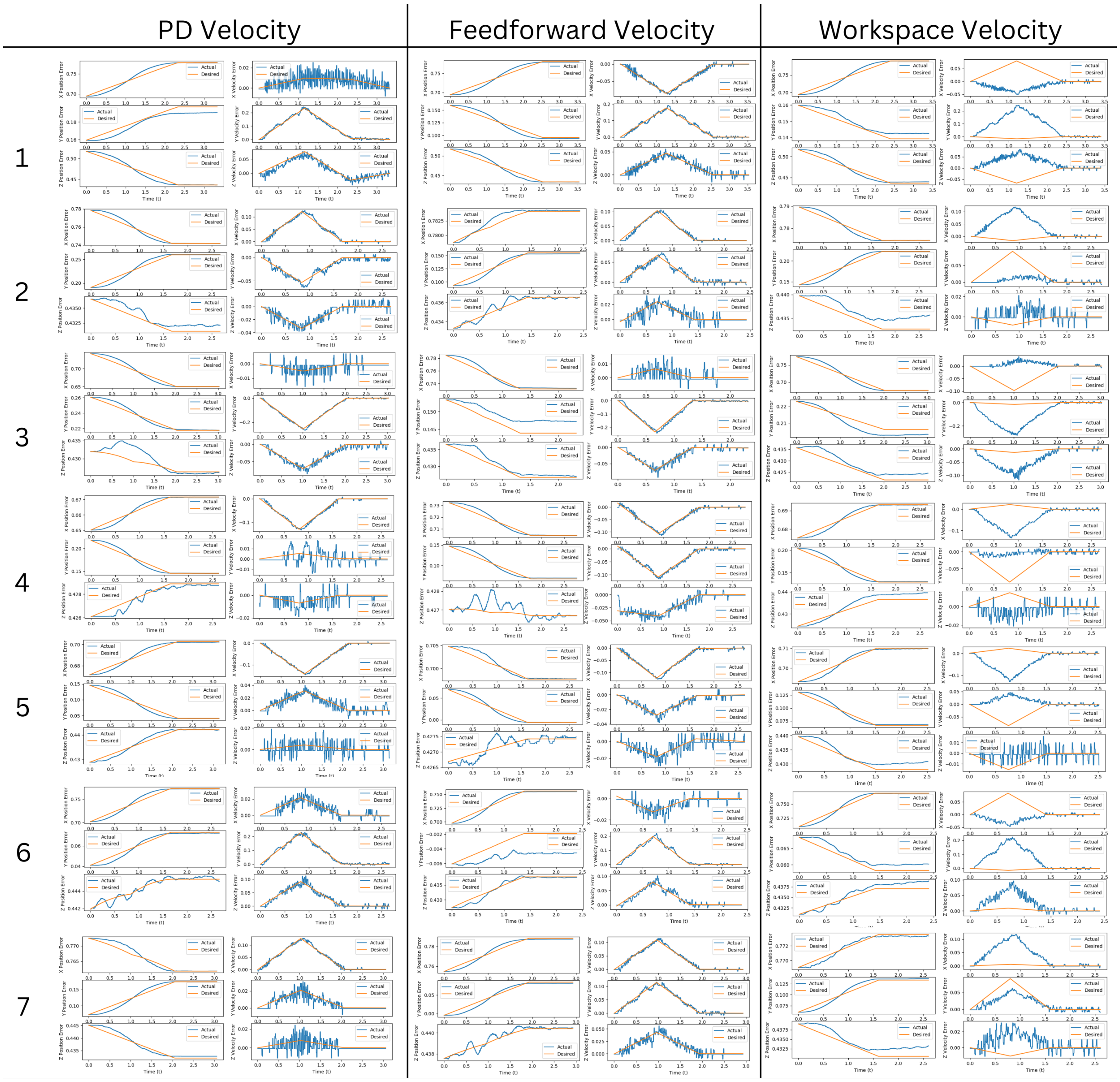

Fig. 5. We tested the three controllers: PD Velocity, Feedforwrad Velocity and Workspace Velocity on a visual servoing task consisting of seven separate segments. The comparison between desired (orange) and actual (blue) trajectories for each segment are plotted above.

notice some trends for each controller. For instance, in addition to being more error prone as the plots show, the feedforward velocity controller was more unpredictable and on more than one occasion made the robot suddenly jerk and veer off course. This follows from the fact that this controller has no sort of state feedback, leading to errors potentially compounding over long trajectories like the visual servoing course used. Additionally, we found the workspace controller to be the smoothest compared to both the joint velocity controllers. As a result, we concluded that this was the best and most stable controller for a visual servoing based task.

IV. Applications

Visual servoing has a wide range of applications in robotics and automation, leveraging real-time visual feedback to guide a robot's motion. One prominent application is in industrial automation, where robots are required to manipulate objects with high precision. For example, visual servoing can be used in pick-and-place tasks in manufacturing lines. By utilizing cameras to detect and track parts, robots can adapt to variations in the position or orientation of objects, enabling them to handle unpredictable environments and improve the efficiency of assembly processes.

Another critical application is in autonomous drones for tasks such as object tracking, delivery, and inspection. Visual servoing allows drones to follow a moving target or maintain a specific position relative to an object or a person. This capability is especially useful in search-and-rescue operations, where drones need to locate and track individuals in dynamic and unstructured environments. Similarly, in agricultural settings, visual servoing can allow drones to monitor and tend to crops, ensuring precise application of water, pesticides, or fertilizers.

In the field of assistive robotics, visual servoing plays a crucial role in enabling robots to interact with humans in a natural and intuitive manner. Robots equipped with visual feedback can assist elderly or disabled individuals with daily tasks, such as fetching objects or preparing meals. Extending the concept further, by recognizing and tracking human gestures or facial expressions, these robots can also adapt their actions to the user's needs, providing personalized and safe assistance.

Visual servoing is also highly applicable in surgical robotics, where precision and adaptability are paramount. Robots guided by visual feedback can assist surgeons in minimally invasive procedures, ensuring accurate tool placement and reducing the risk of errors. This technology enhances the surgeon's capabilities and improves patient outcomes by providing real-time tracking of surgical instruments and anatomical structures.

In autonomous vehicles, visual servoing enables dynamic obstacle avoidance and path following. By using visual input to track road markers, pedestrians, and other vehicles, autonomous cars can navigate complex environments safely. This application is particularly relevant in urban areas, where unpredictable scenarios demand rapid and precise adjustments.

These examples highlight the versatility and transformative potential of visual servoing across diverse domains. Its ability to combine real-time perception and control makes it an important and relevant skill for intelligent and adaptive robotic systems.

V. Difficulties During Implementation

During the implementation of our control framework, we encountered two significant challenges that impacted the efficiency and accuracy of our robotic system. These challenges were primarily related to the complexity of tuning parameters in the Computed Torque Controller and the comparative difficulty of tuning Jointspace PD Velocity Control versus Workspace Velocity Control.

A. Tuning Challenges in the Computed Torque Controller

One of the difficulties we faced was the process of tuning the proportional (K_p) and derivative (K_v) gains in the Computed Torque Controller. Since the computed torque method incorporates the full system dynamics — including the inertia matrix, Coriolis forces, and gravitational effects — the tuning process was more complicated. Because the system is dynamically coupled, the effects of K_p and K_v are not immediately intuitive; changing one parameter may have nontrivial consequences on multiple joints due to interactions in the inertia matrix.

Despite extensive tuning of these parameters, we continued to observe extremely poor performance in tracking the desired trajectories. This issue persisted even after verifying the correctness of our implementation and ensuring that the inertia, Coriolis, and gravity terms were properly computed. We thus left out the torque controller from our final analysis due to time constraints.

VI. Intuition in Tuning: Workspace Control vs. Jointspace PD Velocity Control

Another challenge we encountered was the relative difficulty in tuning Jointspace PD Velocity Control compared to Workspace Velocity Control. In our experience, tuning workspace-based controllers was more intuitive because adjustments could be made in a Cartesian frame, where parameters correspond directly to physical motions in space (e.g., translational and rotational adjustments in x, y, z). This made it easier to reason about errors and correct them by modifying control gains.

In contrast, Jointspace PD Velocity Control operates in the robot's joint space, where the relationship between gain values and end-effector behavior is nonlinear and dependent on the kinematics of the manipulator. A small change in a joint's gain can produce unexpected results in the workspace due to the coupled nature of robotic joint movements. This lack of direct interpretability made it relatively more difficult to fine-tune the control parameters.

VII. Improvement and Feedback

We found that the feedforward joint velocity controller implemented in the starter code had a bug. Since this is a jointspace controller, it should have the variable `self.is_jointspace_controller = True`.

However, this was not set in the starter code, resulting in plots that make the feedforward controller seem very inaccurate when that is not the case.

VIII. Bibliography

[1] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*, Boca Raton, FL: CRC Press, 1994, pp. 193–199.

IX. Appendix

Our video with demonstrations for each controller can be found **here**. Our github repository can be found **here**. The repository also includes additional plots for end-effector angle errors, but we did not identify any clear trends across the three controllers. Therefore, we excluded them from this report.