CAL POLY

CALIFORNIA POLYTECHNIC STATE UNIVERSITY
CENG - DEPARTMENT OF ELECTRICAL ENGINEERING

EE 471- Vision Based Robotic manipulation

Lab #7: Position based visual servoing with PID control

Presented to:
Siavash Farzan

Presented By:

Rodrigo Menchaca
Giovanni Dal Lago

Nov 18, 2025

INTRODUCTION

Position-Based Visual Servoing (PBVS) extends the earlier work in robot kinematics, velocity control, and camera calibration by closing the loop between perception and motion. Unlike the open-loop behavior implemented in previous labs, PBVS requires the robot to continuously estimate the target's 3D pose and correct its motion in real time. This lab implements a full PBVS pipeline: AprilTag detection, coordinate transformation from camera to robot frame, Cartesian error computation, PID-based velocity control, and conversion of end-effector velocities to joint velocities through the Jacobian pseudo-inverse. The goal is to create a system capable of tracking moving targets and maintaining a desired relative pose with stability and accuracy, capabilities fundamental to autonomous manipulation and robust human-robot interaction. The work also exposes the practical challenges in real-time visual servoing, particularly the sensitivity to calibration and tuning, which directly affect responsiveness, oscillations, and steady-state error.

PROCEDURES

Part 1

System description

Our PBVS implementation closes the loop between camera-based pose estimation and joint-space velocity control of the 5-DOF manipulator. At a high level, the system:

1. Captures RGB-D frames from the fixed RealSense camera.
2. Detects the AprilTag in the image and estimates its 3D pose in the camera frame.
3. Transforms the tag position into the robot base frame using the calibrated camera–robot transformation.
4. Defines a desired end-effector pose as a fixed offset above the tag.
5. Computes the Cartesian position error between the current end-effector pose and the desired pose.
6. Uses a PID controller (running at the same dt as the control loop) to compute a commanded Cartesian velocity.
7. Converts the Cartesian velocity to joint velocities using the Jacobian pseudo-inverse.
8. Sends joint velocity commands to the robot until the error converges or the tag is lost.

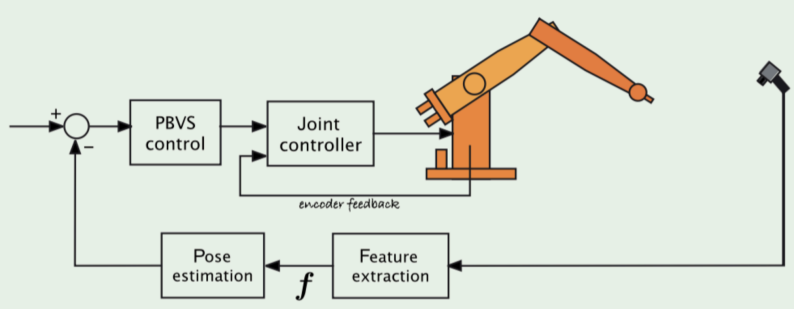

Figure 1: PBVS Control loop

Control equations

The Cartesian PID controller computes the commanded end-effector velocity from the position error between the desired pose (tag position plus offset) and the current pose obtained from forward kinematics. For each axis x,y,z the continuous-time control law is

$$v(t) = K_p\, e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}$$

The real system runs at a fixed control timestep dt, so the PID must be implemented in discrete form:

$$e_k = p_{\text{desired},k} - p_{\text{current},k}, \quad I_k = I_{k-1} + e_k\, dt, \quad D_k = \frac{e_k - e_{k-1}}{dt},$$

$$v_k = K_p e_k + K_i I_k + K_d D_k.$$

Jacobian Mapping

The PID controller produces a desired Cartesian end-effector velocity v=[vx,vy,vz](T). To execute this motion, the robot must convert this Cartesian velocity into joint velocities. The relationship between joint velocities and end-effector velocity is given by the differential kinematics equation $v=J(q)\dot{q}$. Because our manipulator has more joints than controlled task dimensions, the Jacobian J is not square, so we compute the joint velocities using the pseudo-inverse,

$$\dot{q} = J^{\dagger}(q)\, v = J^T (J J^T)^{-1} v.$$

This mapping provides the minimum-norm joint velocity vector that achieves the commanded Cartesian motion. At each control cycle, the Jacobian is evaluated at the current joint configuration, and $\dot{q}$ is sent to the robot's velocity controller. This step is essential for converting visual servoing corrections into executable joint-space commands.

Observations and challenges

With the initial untuned PID gains (very small Kp, and Ki=0, Kd=0), the PBVS system displayed slow, inconsistent, and unstable behavior. The robot reacted to the AprilTag movement with noticeable delay, and the end-effector often drifted several centimeters away from the target position before making any corrective motion. Because the proportional gain was low, the controller produced small velocity commands, resulting in sluggish convergence and long settling times.

Small disturbances in the RealSense mounting produced noticeable shifts in the computed robot-frame tag position. Even minor vibrations changed the effective transformation and caused drift.

To solve this we repositioned the camera, closer to the board, repeated calibration, and avoided touching or bumping the setup during testing, the system improved noticeably.

Part 2

Once our initial PBVS control was set we began control tuning to enhance our robot arm functioning. By using the Ziegler-Nichols method we changed our initial Kp,Ki and Kd values into those recommended by such a method. In order to choose our proportional gain, we first set our Ki and Kd to zero and gradually increase Kp until we see a reasonable step response of about 2-4 seconds. Once our new Proportional gain is set, we continue to evaluate our derivative gain (Kd). Keeping Ki = 0 and using our new Kp, we start increasing Kd by a factor of 0.1Kp until we see sufficient damping of overshoot and oscillations within the robot. Once Proportional and Derivative gain are chosen, we start calculating our integral gain by increasing our Ki by a factor of 0.01Kp, we will determine our Ki value until we observe an elimination of our steady state error with minimal oscillations.

Our Design Gains were calculated at:

Kp = 0.9

Ki = 0.009

Kd = 0.025

These chosen gains were able to provide us with a smooth enough movement which resulted in the following Performance shown in Table 1.

Test	Max Error(mm)	Settling Time(s)	SS Errror(mm)	RMS Error(mm)
Step X	80.205	5.037	2.769	35.546
Step Y	102.21	2.453	3.177	37.18
Step Z	172.241	4.068	52.705	55.74
Rectangle(XYZ)	(103.365,185.106,125.492)	28.934(total loop)	(1.830,27.261,1.980)	(38.121,61.071,28.482)

Table 1: Performance data Table

After choosing our newly tuned values for our gains, we ran two different performance tests while logging the following data:

- Time (S)
- Current EE Position (mm)
- Desired EE Position (mm)
- Position Error (mm)
- PID Output (mm/s)

- Joint Velocities
- Tag_detected flag (bool)

We logged these values using two different trajectories, first by moving the end effector through a rectangular 100mm x 100mm motion, this was mainly to record the overall response of the robotic arm as well as any oscillations due to our PID control. The second trajectory done, was 3 separate motions, one to simulate an X direction step response and another two for Y and Z respectively. These motions would give us closer and more detailed values for the calculation of our performance data table for the different X,Y,Z directions. Settling time was about the same for all of our different 3 dimensions, except for our Y dimension. This was likely due to some experimental error at the time of moving the robot in the desired position.

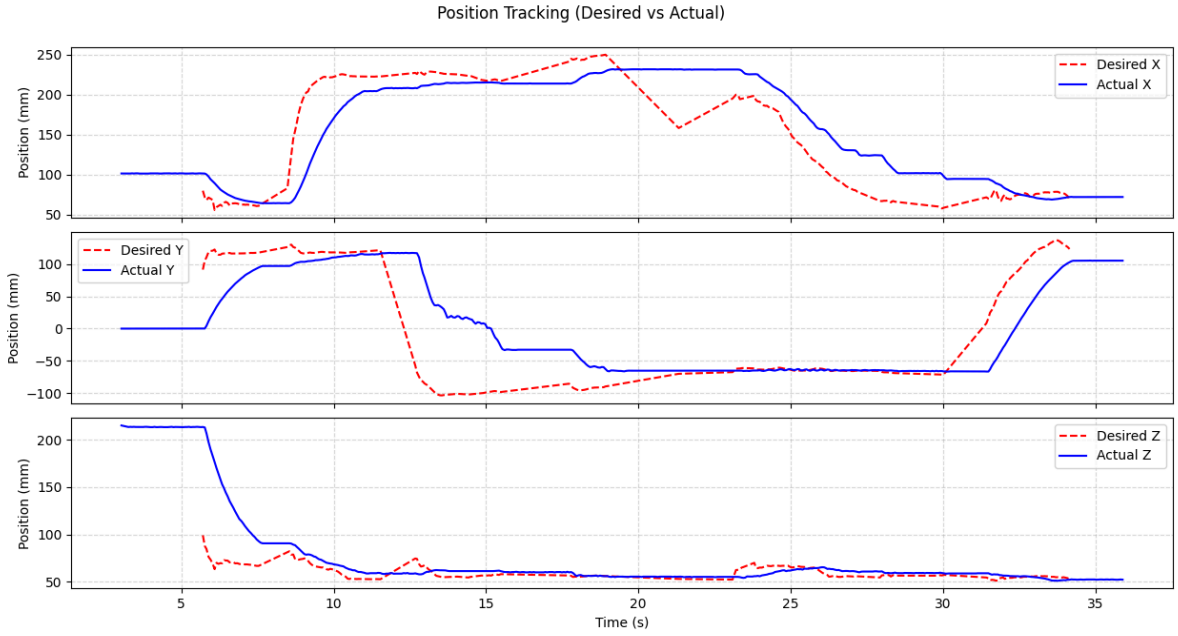

Figure 2: Position Tracking: Desired vs Actual (XYZ)

After collecting our position-tracking data for the X, Y, and Z trajectories, we assessed how well the tuned PID controller allowed the robot arm to follow the desired paths. As shown in Figure 2, the robot was generally able to match the commanded motion, but several tracking errors were still present. In the X-direction, the arm followed the trajectory reasonably well, though some lag and overshoot appeared during faster transitions. The Y-direction showed larger deviations, especially during the downward motion, likely due to unmodeled friction or small experimental inconsistencies. The Z-direction exhibited the greatest offset, with the actual position responding more slowly and never fully matching the desired profile. Overall, the controller provided a functional level of tracking, but the observed delays and axis-dependent errors suggest that additional tuning or compensation strategies may be needed for more precise 3D performance.

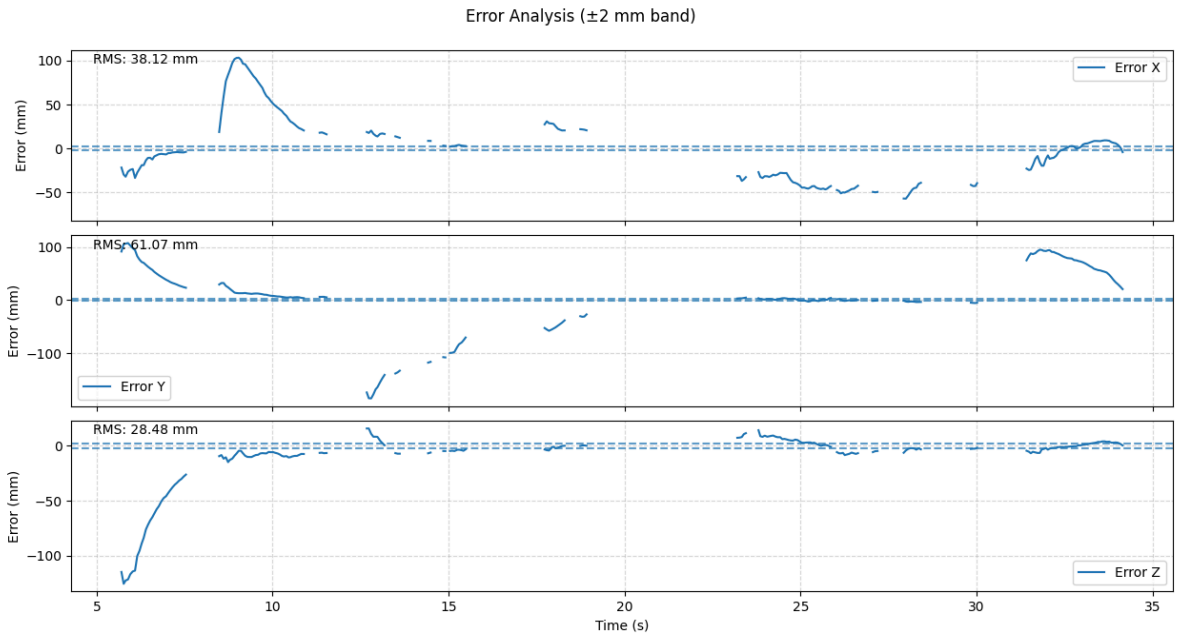

Figure 3: Error Analysis for each dimension

After evaluating the position-tracking results, we performed an error analysis for each individual axis to better understand how our tuned PID controller behaved throughout the experiment. As shown in Figure 3, the X-axis error fluctuated significantly during the initial transitions, reaching peaks well outside the ±2 mm band before gradually settling, resulting in an RMS error of about 38 mm. The Y-axis showed the largest deviations overall, with a consistent negative offset and several large error spikes that contributed to an RMS error of roughly 61 mm. The Z-axis performed slightly better, but still maintained a noticeable steady-state offset throughout most of the motion, giving an RMS error near 28 mm. These results confirm that while the controller produced stable motion, the system was unable to maintain tight tracking accuracy, especially in Y and Z. This indicates that further gain refinement or additional compensation methods may be required to reduce steady-state offsets and improve overall precision across all three dimensions.

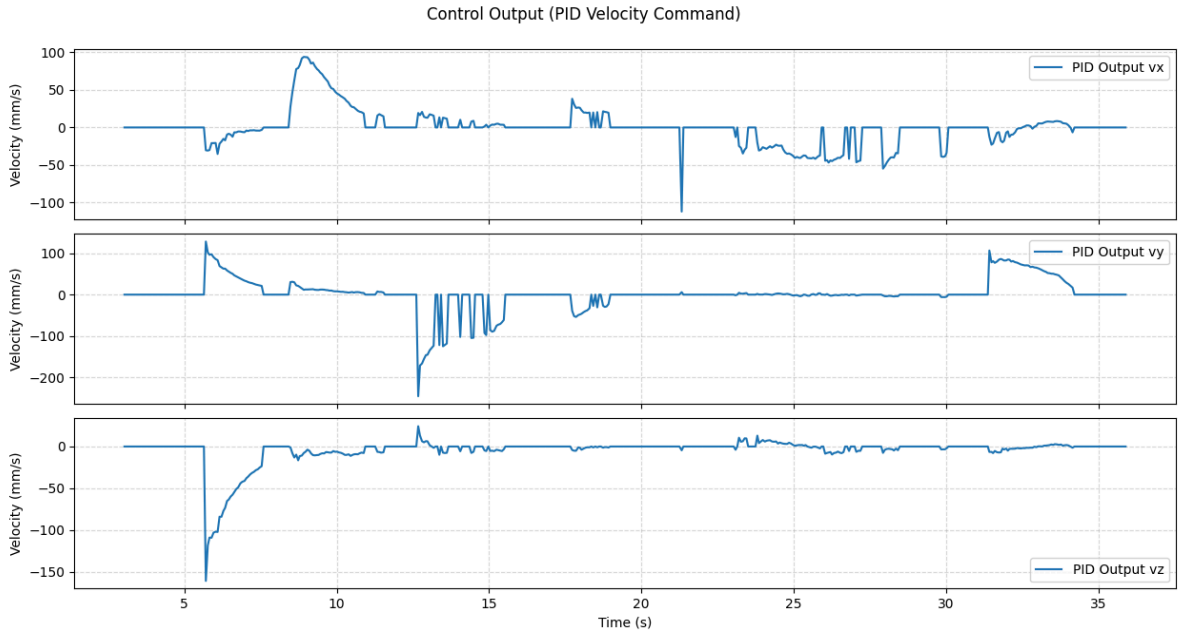

Figure 4: PID control Output

To further understand how our controller responded to the tracking errors, we analyzed the PID velocity commands generated for each axis, as shown in Figure 4. The X-direction output shows several sharp velocity spikes during rapid changes in the desired trajectory, followed by smaller corrective adjustments as the system attempts to reduce steady-state error. In the Y-direction, the controller produced more aggressive negative velocity commands, which aligns with the larger tracking deviations observed in our error plots; these fluctuations indicate that the controller was consistently trying to compensate for undershoot and drift. The Z-direction output behaved more smoothly overall, though a significant initial correction was required before settling into smaller adjustments. These control signals collectively demonstrate that while the PID controller was actively compensating for trajectory mismatches, the magnitude and frequency of its responses suggest that further tuning—particularly in Y—may help reduce large corrective actions and lead to more stable and efficient tracking performance.

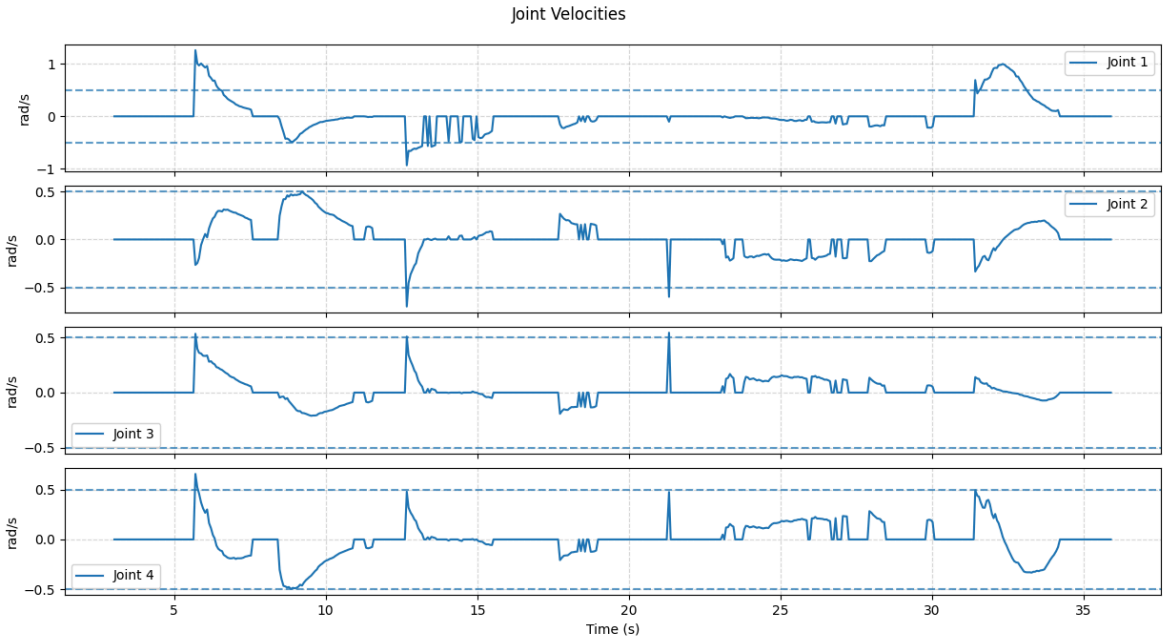

Figure 5: Joint Velocities vs Time

We examined the joint velocities throughout the experiment, shown in Figure 5. Joint 1 demonstrated the largest range of motion, with several noticeable spikes during the initial transitions and near the end of the trajectory, which corresponds to the more demanding movements in the X-direction. Joints 2, 3, and 4 showed smoother behavior overall, though each still exhibited brief bursts of corrective velocity whenever the end effector deviated from the desired path. These short, rapid adjustments are consistent with the PID control effort observed earlier and indicate the system compensating for accumulated error. While all joints stayed within their expected velocity limits, the presence of frequent small oscillations—especially around the midsection of the trajectory—suggests that additional fine-tuning of the gains could help reduce unnecessary joint activity and yield more efficient, smoother arm motion.

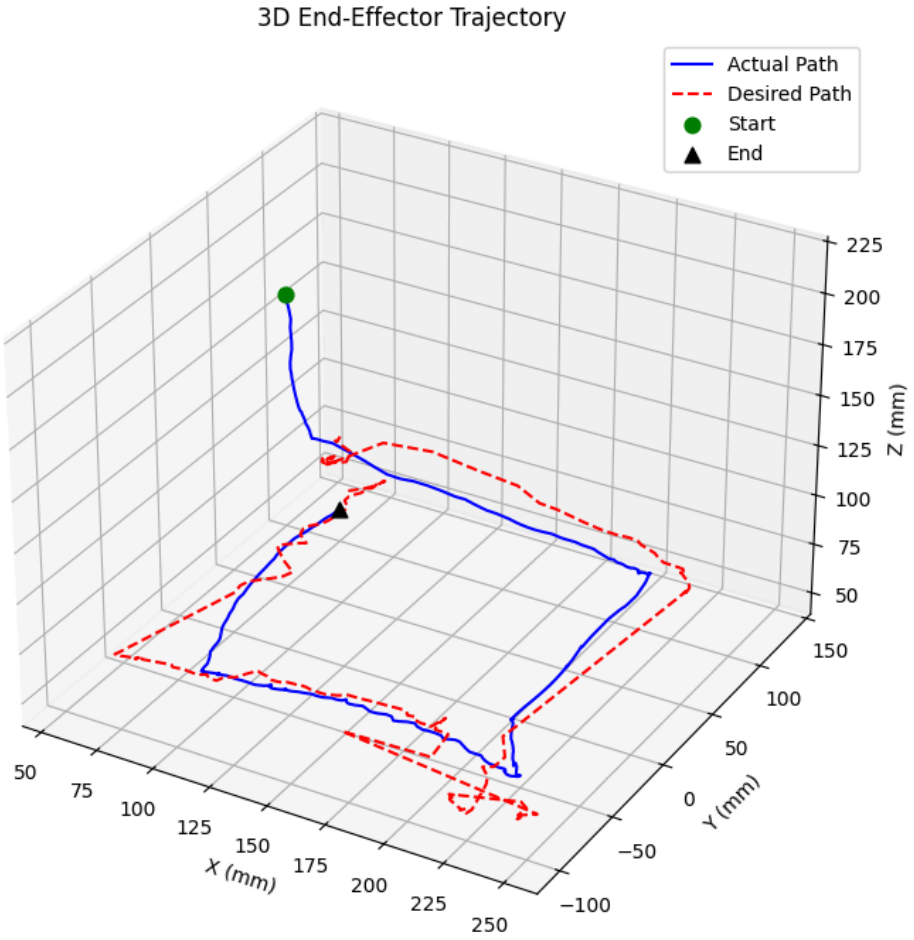

Figure 6: 3D End-Effector Trajectory

To visualize the combined motion of the robot arm, we plotted the full 3D end-effector trajectory shown in Figure 6. The desired path forms a smooth rectangular loop with a slight vertical transition, while the actual trajectory reveals noticeable deviations in all three dimensions. The largest discrepancies occur along the Y- and Z-directions, where the end effector tends to undershoot corners and drift away from the commanded path. The initial descent and final ascent also highlight some instability in vertical control, consistent with the earlier error and PID output plots. Despite these differences, the robot is still able to capture the overall shape of the trajectory, completing the full cycle and returning near its final target point. Although the motion of the robot is not within the desired position, it does still follow a rather straight rectangular path. The discrepancies shown in figure 6, are more than likely due to an experimental error due to some erroneous human movement of the april tag rather than a control issue, the slight oscillations seen in one of the motions in figure 6 are more than likely due to a PID control matter. Further tuning of our PID control output would allow us to minimize these oscillations even further.

CONCLUSION

Overall, this lab gave us a much better understanding of how Position-Based Visual Servoing actually works when everything—vision, kinematics, and control—has to come together in real time. By combining AprilTag detection, the camera-to-robot transformation, PID control, and the Jacobian pseudo-inverse, we were able to build a full PBVS system that could track a moving tag and try to keep the end effector at a desired offset. Even though the system wasn't perfect, it was clear that the main pieces of the pipeline were working as intended, and the robot was able to follow both the rectangular path and the step movements with reasonable behavior.

Tuning the PID controller helped improve stability, but our results showed that getting accurate tracking in all three directions is still tough. The gains we chose reduced some of the big oscillations we saw in the beginning, but we still had noticeable errors—especially in the Y and Z axes. These steady-state offsets and jumps in the RMS error suggest that the controller wasn't fully optimized, and that factors like calibration accuracy, lighting, and even small mistakes in moving the AprilTag by hand had a bigger impact than we expected. The joint velocity plots also showed that the robot had to make a lot of quick corrections, which lines up with the tracking errors we saw earlier.

Even with these challenges, the lab gave us a clearer picture of how visual servoing behaves in practice. It showed how sensitive the system is to noise and timing, and how the PID gains directly affect responsiveness and stability. It also reinforced how important it is to keep the control loop timing consistent with the PID timestep, since any mismatch affects the controller's behavior immediately.

In the future, we could improve the results by spending more time on calibration, trying different gain values for each axis, or filtering the camera measurements so the controller isn't reacting to every small fluctuation. More advanced controllers or adaptive methods would probably help too. But overall, our PBVS system met the main goals of the lab, and it successfully demonstrated how real-time visual feedback can guide robot motion.

Appendix

Equations/ methods used for measuring of performance metrics:

$$e_x[k] = x_{\text{desired}}[k] - x_{\text{current}}[k]$$

Error Calculation

$$T_{s,X} = t[k] - t_{\text{step}}$$

Settling time calculation

$$e_{\text{SS},X} = \frac{1}{N_{\text{win}}} \sum_{k \in \text{window}} e_x[k]$$

Steady state Error

$$\text{RMS}_X = \sqrt{\frac{1}{N} \sum_{k=1}^{N} e_x[k]^2}$$

RMS Error Calculation

Video Links:

Step Response Video:
https://youtube.com/shorts/rYscE_-QcJM?feature=share
Rectangle Motion Video:
https://youtu.be/GqZ9yAM0chw