EECS C106B: Proj1B - Trajectory Tracking

Anuj Raichura
UC Berkeley

Steed Amegbor
UC Berkeley

Tianyue Cheng
UC Berkeley

***Abstract*—The following is a detailed report of the work carried out by team "8am Gangs" for project1B of EECS106B. The objective of this project was to apply concepts from Chapters 2-4 of "Modern Robotics: Mechanics, Planning, and Control" (MLS) to implement closed-loop control with Sawyer, a versatile robotic arm developed by Rethink Robotics.**

For Part B of the project, we focused on incorporating feedback into three different control methodologies, namely a jointspace velocity controller, a workspace velocity controller, and a jointspace torque controller. This involved using various sensors to measure the robot's current position, velocity, and torque, and then using this data to adjust the control signals to achieve the desired movement.

I. Method

Here the problem we're solving refers to controlling the sawyer, which mathematically refers to the following equation:

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

$$\tau = M(\theta_d)\ddot{\theta_d} + C(\theta_d,\dot{\theta_d})\dot{\theta_d} + N(\theta_d,\dot{\theta_d})$$

The first equation states the problem we're solving, and the second refers to the joint trajectory we're keeping track of. Here θ is the set of configuration variables and θ_d is the joint trajectory we wish to track. This open-loop manipulation is not robust since initial configuration for $\theta(0)$ and $\theta_d(0)$ might be different, and we need to incorporate feedback to adjust the position of the sawyer. Here we improve our control strategy by adding a feedback, written mathematically as follows:

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

Here e refers to $\theta - \theta_d$, and K_v and K_p are constant gain matrices.

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

For the torque control, we used the above equation. We got the M, C, and N matrices from libraries that describe the dynamics of the sawyer robot, as in computed torque control. In this controller, we are controlling the joint torques instead of joint velocities. To calculate the control input, we had to reshape the position and velocity error matrices, and then simply used the equation above. We kept the K_p and K_v values separate from the M matrix because that was the easiest way to tune the values.

$$e(t) = \theta_d(t) - \theta(t)$$

For jointspace P_d velocity control we used the above equation. This was the simplest controller because we just directly inputted the amount that the position and velocity were off by into the control input, and multiplied those by K_p and K_v respectively.

$$g_{td} = g_{st}^{-1} g_s d$$

$$\xi_{td}^s = Ad_{st}\xi_{td}$$

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

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

For workspace velocity control, we want to track a trajectory in the workspace as opposed to the jointspace. First, we get the current joint positions and find the gst matrix that represents the current configuration of the end effector. Then, we find the gsd matrix from the target position that is passed into the function. From these, we get a 4x4 homogeneous matrix gtd that is the error configuration between the current and desired positions. To convert this into a velocity twist, we must take the log of the matrix and then use the vee operator over it. This is simply the opposite of converting a twist into a homogeneous matrix, whose formula is $e^{\hat{\xi}}$. Once we have the velocity twist, we need to convert it to the spatial frame using the adjoint matrix, and once we have that we can construct the U_s matrix. The U_s matrix incorporates the K_p value for controller gain and the target velocity in the spatial frame, which is the same as the target velocity in the body frame because the orientation does not change. Once we have K_p, we can convert this into joint velocities by multiplying U_s by the pseudo inverse, which is our final control input.

Parameter Tuning

For tuning our controller: For the workspace controller, we used the formula $2\sqrt{K_p}$ for the K_v value and tuned the K_p value, trying different values in a binary search sort of approach until we came upon 0.02, which seemed to work the best generically. From there, we focused on the individual x, y, and z values of K_p, as it was mentioned in the project spec that changing the first three values of the diagonal matrix corresponds to translational error in the body frame x, y, and z positions. We tested various different values here again in a sort of binary search method that devolved into testing smaller and smaller values until we stopped seeing benefits. However, the scaling factor for the angular velocity, or the last three values of the diagonal K_p matrix, was kept at 0.02 because that is where we saw the best and most consistent results. For the torque control, our process again used the $2\sqrt{Kp}$ formula for the K_v value and also involved us trying both extremely large and small values until we found 18 to work the best. The jointspace controller was again much less of an interesting process, where we played with the K_p value until we found that it started oscillating, omitting K_v from the equation. Then, we tuned the K_v value to a point where it minimized the oscillation as much as possible. The values we ended up with for the jointspace controller were 1 and 0.02 for K_p and K_v respectively.

Moreover, during our experimentation with various control methodologies for Sawyer, we found that one effective tuning method was to adjust the number of waypoints used in the path planning process. We observed that using too many waypoints resulted in an accumulation of errors between each waypoint, leading to the controller becoming stuck in an unrecoverable state.

To mitigate this issue, we experimented with different values for the number of waypoints used in the path planning process. By reducing the number of waypoints, we found that we were able to minimize the accumulation of errors and achieve better control of Sawyer's motion. This was because with fewer waypoints, the robot had to perform fewer corrections and was able to better maintain the desired trajectory.

However, it was important to strike a balance between the number of waypoints used and the accuracy of the path planning. Using too few waypoints resulted in a jagged, irregular motion that was difficult to control. Therefore, we conducted multiple tests to determine the optimal number of waypoints for each controller, taking into account factors such as the complexity of the task and the available compute power.

II. Experiments

We ran experiments with jointspace PD velocity controller, jointspace PD torque controller and workspace velocity controllers. The control group was the moveit controller, and we used it to debug some of the trajectory issues we faced before we tried our controllers on the trajectories in order to isolate the source of the error. The initial experiments we ran were to find if the robot was able to locate and find the position of the AR tag, and after that we moved on to experiments testing our trajectory code in moveit to ensure that it would work on a controller we knew was functioning properly. Then as we worked through the controllers, we tested various K_p and K_v values as well as our code itself to debug and tune values. When testing controllers, we initially used the line as the default because it was the most robust controller and also had applications in the polygon controller, and then tried the circle and polygon trajectories second and third to ensure our controller could suffice in the other applications.

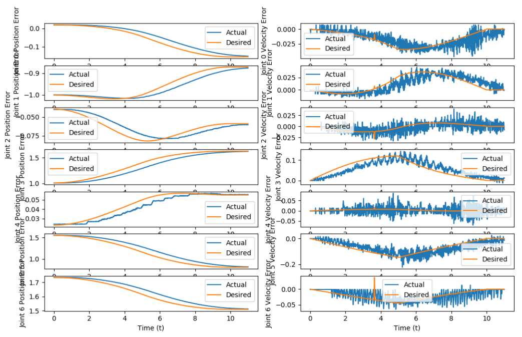

Fig. 1. Jointspace PD velocity controllers moving in a line

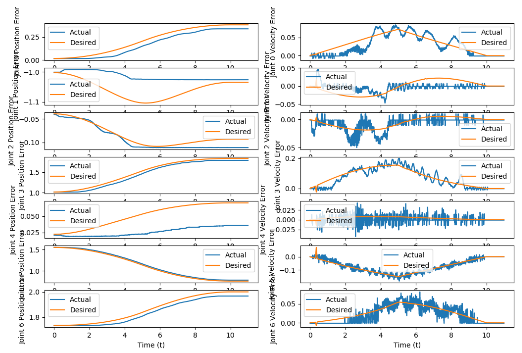

Fig. 2. Jointspace PD torque controllers moving in a line. Though it oscillates a bit while moving but it mostly achieve the desired movement we want.

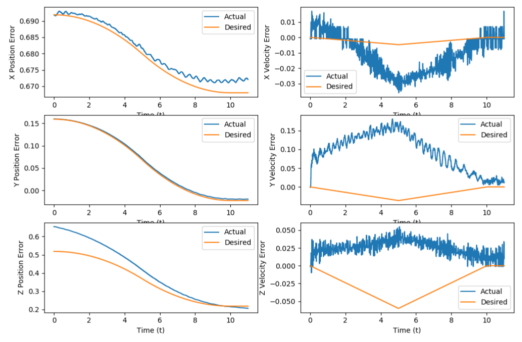

Fig. 3. Workspace PD velocity controllers moving in a line. Similar to Jointspace PD torque controller, minor oscillations but not affecting overall performance.

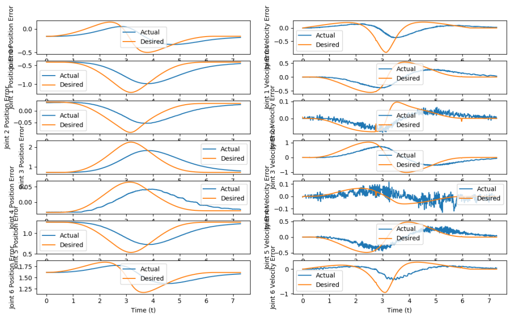

Fig. 4. Jointspace PD velocity controllers moving in a circle. Jointspace PD velocity controllers work pretty smoothly on sawyer. Though the plot might shows minor oscillations but it works pretty well in reality.

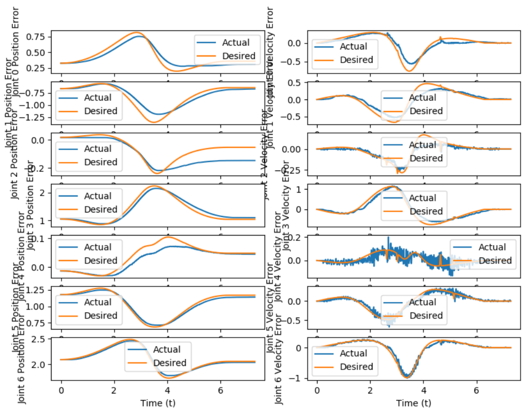

Fig. 5. Jointspace PD torque controllers moving in a circle. It has a relatively smaller oscillations compared with jointspace PD velocity controllers, and apparently performing better in reality.

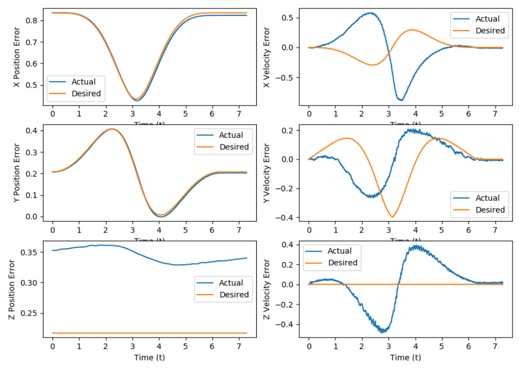

Fig. 6. Workspace PD velocity controllers moving in a circle. This one works the best among all controllers moving in a circle. It has minimum oscillations along the path and outperforms all others.

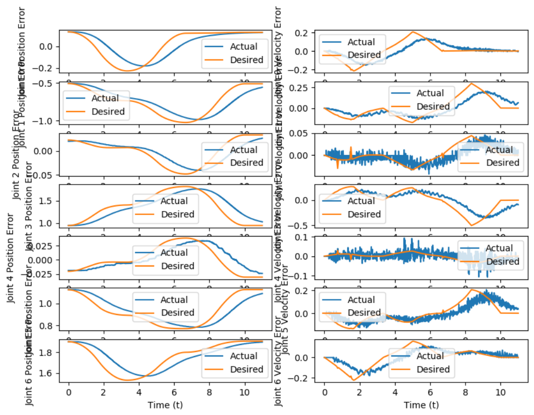

Fig. 7. Jointspace PD velocity controllers moving in a polygon. Though there are some oscillations shown in the graph plotted but pretty smooth in reality.

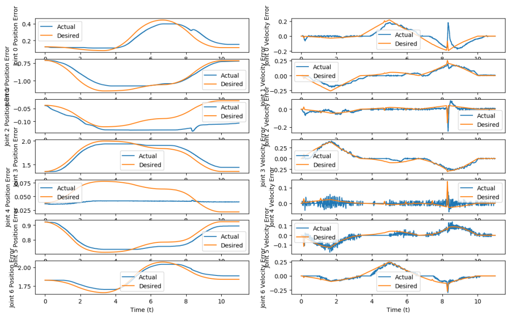

Fig. 8. Jointspace PD torque controllers moving in a polygon. It works a bit smoother and accurate compared with jointspace PD velocity controllers.

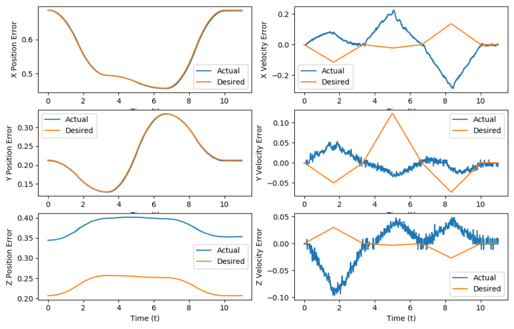

Fig. 9. Workspace PD velocity controllers moving in a polygon. It works the best according to the position as shown in graph, though a bit off in z direction

III. Visual Servoing

We opted to use the workspace controller as it provided a direct means of controlling velocity without the need to manipulate joint angles. Although it delivered satisfactory performance in tracking the xy positions, we encountered an issue with the end effector's orientation, which tended to drift to one side instead of facing down entirely. Nonetheless, the controller's overall performance was reasonable.

However, upon analysis of the plotted data, we observed oscillations in both the desired and actual trajectories. This outcome may be attributed to the slow movement of the AR tag in conjunction with the camera's latency in detecting changes in its position. As a result, the system required almost a second to react to the tag's movement in the opposite direction. Despite this, both the plot and the sawyer exhibited commendable results

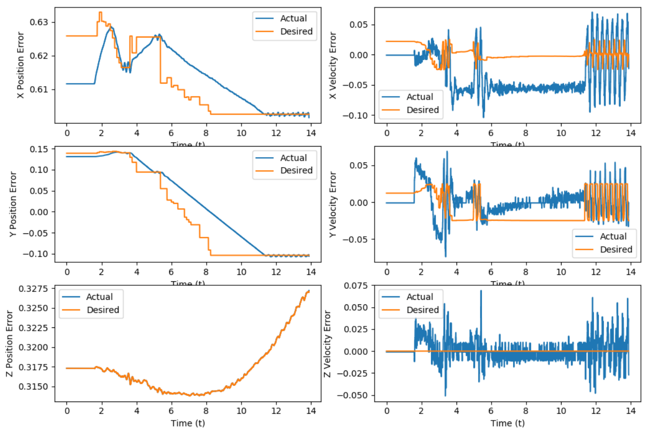

Fig. 10. A working visual servoing with workspace controller

IV. Applications

During our experimentation with various control methodologies, we observed that workspace control was the most effective at controlling the xy position of the end effector. However, it struggled to control the z position accurately, which we attribute to incorrect Kp values. Nevertheless, in situations where only actuation in the xy plane is required or the z axis is less significant, such as lifting and placing larger objects, workspace control is likely the most useful methodology to employ.

Jointspace control, on the other hand, is extremely simple and useful, making it an excellent choice in situations where compute power is limited. It operates by commanding the joints of the robot directly, and as a result, it has the potential to deliver accurate control. However, it may struggle in more complex tasks that require coordinated motion between multiple joints.

Computed torque control is an interesting methodology because it can be highly effective if the inertia, coriolis, and gravity matrices are precise. However, due to its spotty accuracy and the need for gravity compensation, it seemed to us like the least applicable controller. Computed torque control could be used in environments that are sterile and highly precise, where all factors affecting the robot are known and accounted for.

In short, each control methodology has its advantages and disadvantages, and choosing the best one for a particular task depends on various factors such as the complexity of the task, available compute power, and environmental conditions. By understanding the strengths and weaknesses of each approach, we can make more informed decisions about which control methodology to use in a given situation.

References

[1] Murray, Richard M., Zexiang Li, and S. Shankar Sastry. A mathematical introduction to robotic manipulation. CRC press, 2017.

Appendix

Feedback: We think that the learning goals of this assignment were relatively well received. It seemed to be learning control modalities, and applying some of the common formulas on actual robots. However, we thought that way too much time was devoted to finding Kp and Kv values, and something that would be useful could maybe be a range of typical values so we can at least find the range to work in. We also found that there were some problems in the starter code that we spent a lot of time debugging, like making the source frame the right hand instead of the right hand gripper. We also thought that the lab documentation could include more common problems. The main pain point that could also be improved was explaining the reasoning for a lot of the formulas and how they actually contribute to controlling the position, as a lot of the project felt like just implementing formulas given to us. Still, overall, it was a fun project and still a very rewarding experience!

Project Videos

GitHub repo