EECS 106B Project 1 - Trajectory Tracking with Baxter

Prabhman Dhaliwal   Matthew Hallac   Jaiveer Singh

***Abstract*—In this paper, we investigate the performance of three custom PD controllers against the built-in movement controller on the Baxter 7-DOF robot arm.**

***Index Terms*—PD control, Robot arm, workspace, jointspace**

I. Introduction

Most commercial robot arms come with a preloaded motion controller available for use in navigating the robot arm between user-specified waypoints. By virtue of the fact that a robot arm has a multitude of purposes, the built-in controller must necessarily be adaptable to any such user need. However, this observation might also imply that the pre-included controller accompanying a standard robot like Rethink Robotics' Baxter performs worse than a custom-designed motion controller for a specific use case. In this paper, we explore the idea of tuning custom controllers to operate the Baxter 7-DOF robot arm in specific movement contexts.

II. Methods

For the purposes of this investigation, we considered three separate controllers. Each controller was a Proportional-Derivative (PD) controller, though controllers differ in the space in which each operates. First, we considered a Jointspace Velocity PD Controller; second, a Jointspace Torque PD Controller; and finally, a Workspace Velocity PD Controller.

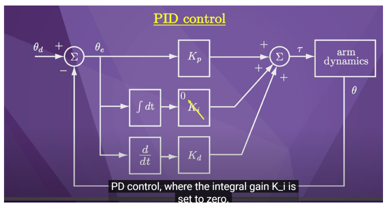

Fig. 1. PID Block Diagram

A. Jointspace Velocity PD Controller

1) Derivation: The Jointspace Velocity PD Controller controls the robot by directly manipulating the velocities for each joint. The control input is defined with a feed-forward term $\dot{\theta}$, and sum the error in position and velocity from the desired trajectory $\theta(t)$ multiplied by k_p and k_v respectively. Error is defined as in equation (1), and the control input is computed in equation (3).

$$e = \theta_{des} - \theta \tag{1}$$

$$\dot{e} = \dot{\theta}_{des} - \dot{\theta} \tag{2}$$

$$\dot{\theta} = \dot{\theta}_{des} + K_p e + K_v \dot{e} \tag{3}$$

The advantage of directly controlling the joint velocities is that it allows Baxter to maintain a set speed, even under strain from external and internal forces. The joint motors are will use as much torque as necessary to maintain the velocities we request. The velocity controllers are concerned with the kinematics of the system more than the actual forces being applied to the system. While a velocity controller does eventually need to command certain torques, the priority is on getting consistent speeds rather than balancing out the external forces on the system with the input torques.

B. Jointspace Torque PD Controller

1) Derivation: The problem statement for the jointspace torque controller can be founded as:

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

where θ represents an n-dimensional vector for the set of joints of the robot. This equation derived in MLS chapter 4 relates our joint angles to our torques. This equation by itself only accounts for direct control using just direct substitution of our angles. If we wish to turn this into the computed torque law used for our control, we must substitute the feedback terms relating our error between the desired and current trajectory. Since M(θ) is positive definite in θ, we can say that:

$$\ddot{\theta_d} = \ddot{\theta} \tag{5}$$

And substituting this in to our open-loop control law, we arrive at the computed torque control law, which incorporates feedforward and feedback! Also note that because the Coriolis terms are small in our usage (because the joint speeds are small), we can remove their influence from this equation.

$$\tau = M(\theta)\left(\ddot{\theta}_d - K_v\dot{e} - K_p e\right) + N(\theta, \dot{\theta}) \tag{6}$$

2) Concept: The jointspace PD torque controller controls the robot by using the computed torque law. As the velocity controllers are better for maintaining consistent speeds, the torque controllers are better for sticking to the desired trajectories. Because torque control directly applies to the motors, we are not relying on any internal control loops to change the velocity, we get complete control over the torques. The torque controller controls the dynamics of the system, and they are balanced out with our inputs according to the computed torque control law. The main terms in our computed torque laws are:

1) Inertia matrix: This matrix accounts for all the inertial forces being applies to our system, and the analogous form of Newtons second law allows this matrix to be multiplied by our acceleration
2) Coriolis terms: We don't compute this in our solution because the library was not present to do it. However, the Coriolis terms are very small in our robot anyway because they are proportional to the rotation rate viewed in a frame of reference. Because our joints do not rotate very quickly relative to a viewer's frame, these terms can be ignored with little effect. If our joint velocities were to increase substantially, then we would not have a proper model for our system because the Coriolis terms would begin to show. Aside from just situations where we want high joint velocities, this could unintentionally come into play if, for example, a singularity occurred in our control matrices that forced the velocity to be extremely high to compensate, and thus breaking the assumption we had for the Coriolis terms.
3) Gravity Vector: Accounts for gravitational energy on our system. This is done with a built in KDL package for the course.

C. Workspace Velocity PD Controller

1) Derivation: The goal in workspace control is to track our end effector's position, not the joints. Since only jointspace commands can be given to the robot, we need a way to convert from jointspace to workspace, and then a relationship between our workspace and our joint velocities. The steps are as follows:

1) Compute the desired workspace velocity our end effector should perform. It is labeled U^b This term can be composed of a feedforward term and a feedback term that we need to compute.
   a) The feedforward term is the velocity of the desired trajectory in the body frame,V_d^b. Recall that by definition, and in the desired frame:

   $$\hat{V}_d^b = g_d^{-1}\dot{g}_d \tag{7}$$

   In order to convert this equation into something that operates in the current actual frame of the robot we use the adjoint equation:

   $$Ad_{g_{td}}V_d^b \tag{8}$$

   In order to give this command to the robot, we need to change it from the frame of the end effector to that of the robot, using the adjoint of this error configuration. We can use this desired term to give a command of what we want our robot to follow, analogous to giving our robot joint commands to follow a desired trajectory in jointspace
   b) The feedback term is the velocity that minimizes the error between this error frame and the desired frame. The velocity that minimizes this error is exactly

   This velocity was proven to be a twwist in the error configuation:

   $$\hat{\xi_{td}} = \log g_{td} \tag{9}$$

2) Combining the feedforward and feedback terms we can come up with the Cartesian control law, where we only have a Kp that tunes each value in the twist (application onto the translational error and the other 3 which affect the rotational error).

$$U^b = K_p\xi_{td} + Ad_{g_{td}}V_d^b \tag{10}$$

3) Finally we can transform this workspace control input by going back to the spatial coordinates of our robot, and then using the pseudoinverse of the jacobian (which relates joint velocities and SE(3) velocities) to get our jointspace velocities which we use as commands.

$$\dot{\theta}(t) = (J_{st}^s(\theta)^\dagger)u^s(t) \tag{11}$$

D. Tuning procedure by controller

1) Jointspace velocity and Jointspace torque: To tune the jointspace controller, first increase the K_p values uniformly until the system begins to oscillate. Then, increase the K_v values until the oscillation begins to go away. Finally, refine the k-values for each joint individually until they are satisfactory. This method was heavily influenced by the Ziegler-Nichols method. For example, in our controllers, joint 5 was extremely osciallatory even when joints 1-4 were perfect, so we had to individually tune joint 5.
2) Workspace velocity: The workspace velocity tuning was different because there was no K_v term, we did almsot the same thing as the jointspace controllers, except we also examined the jittering in the physical system on the robot and tune our K_p lower or higher to compensate in which direction it was the strongest.

III. Experimental Results

A. Demonstration Video:

YouTube Link

B. Explanation of Experimental Design

In order to verify the performance of each controller, we ran each controller on a series of three different trajectories. The linear trajectory involves the end effector maintaining a constant orientation, Z and Y while it translates in the pure X direction of the robot. The circular trajectory involves the end effector maintaining a constant orientation and Z while translating in a circle in the XY plane. The arbitrary trajectory is one that was initially constructed using the MoveIt controller for Baxter, and varies in all of orientation, X, Y, and Z throughout its path.

The linear and circular trajectories were selected as minimal building blocks of a larger task, with the reasoning that they sufficiently covered many standard paths while also being simple enough to provide robust 'training cases' during tuning and calibration of each controller. The arbitrary trajectory, in contrast, has no simple closed-form solution for its velocity or position, as a consequence of the fact that it is arbitrary. Thus, the arbitrary trajectory is a good ’test case’ against which the controller’s performance can be observed.

The control group in this experiment was the performance against an open-loop controller, since that represents a controller that does not make any attempt to correct for its accumulated error. As expected, each of the three custom controllers performed better than this baseline.

An important note regarding the linear trajectory is that by the way we constructed the trajectory, the very first motion the robot arm makes is a large and sudden movement to the starting point of the X-oriented line, before slowly traversing along the line in the pure X direction. As a result, many graphs for the linear trajectory display a sudden deviation from the expected function.

C. *Plots of Experimental Results*

The plots of the performances of each of the controllers are included on the subsequent pages. It is important to note that the scales on the vertical axis of each plot was automatically chosen to effectively show the range of the function over the interval. As a consequence, simply looking at the shapes of the graphs without considering the vertical tick markers is likely to lead to an erroneous sense of how effective the controllers are.

For ease of reference, we have linked to the corresponding images below:

- Line Trajectory
  - Open Loop Controller
    - XYZ Position and Velocity Error 2
    - End Effector Angle Error 3
  - Jointspace Velocity Controller
    - Joint Position and Velocity Error 4
    - XYZ Position and Velocity Error 5
    - End Effector Angle Error 6
  - Jointspace Torque Controller
    - Joint Position and Velocity Error 7
    - XYZ Position and Velocity Error 8
    - End Effector Angle Error 9
  - Workspace Velocity Controller
    - XYZ Position and Velocity Error 10
    - End Effector Angle Error 11
- Circle Trajectory
  - Open Loop Controller
    - XYZ Position and Velocity Error 13
    - End Effector Angle Error 14
  - Jointspace Velocity Controller
    - Joint Position and Velocity Error 12
    - XYZ Position and Velocity Error 15
    - End Effector Angle Error 16
  - Jointspace Torque Controller
    - Joint Position and Velocity Error 17
    - XYZ Position and Velocity Error 18
    - End Effector Angle Error 19
  - Workspace Velocity Controller
    - XYZ Position and Velocity Error 20
    - End Effector Angle Error 21
- Arbitrary Trajectory
  - Oopen Loop Controller
    - XYZ Position and Velocity Error 22
    - End Effector Angle Error 23
  - Jointspace Velocity Controller
    - Joint Position and Velocity Error 24
    - XYZ Position and Velocity Error 25
    - End Effector Angle Error 26
  - Jointspace Torque Controller
    - Joint Position and Velocity Error 27
    - XYZ Position and Velocity Error 28
    - End Effector Angle Error 29
  - Workspace Velocity Controller
    - XYZ Position and Velocity Error 30
    - End Effector Angle Error 31

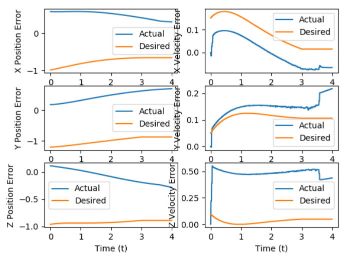

Fig. 2. Open Loop Controller XYZ Error for Line Trajectory

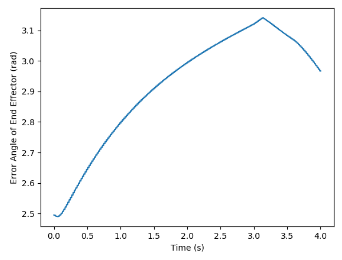

Fig. 3. Open Loop Controller Angle Error for Line Trajectory

IV. Applications

The field of controls has a broad reach and can be applied to many systems and situations. First, the workspace controller is optimal for tracking an object in 3D space because it takes in a workspace coordinate and translates it to joint inputs. In a setting where an arm needs to follow a ball, or some other moving object, the joint controls would not be immediately available, and thus it would make sense to use a workspace

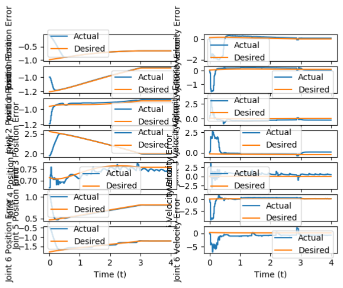
Fig. 4. Jointspace Velocity Controller Joint Error for Line Trajectory

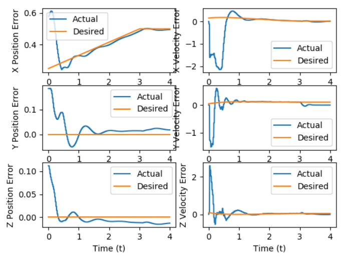
Fig. 8. Jointspace Torque Controller XYZ Error for Line Trajectory

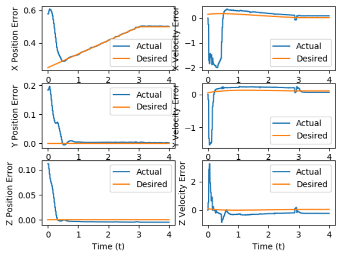
Fig. 5. Jointspace Velocity Controller XYZ Error for Line Trajectory

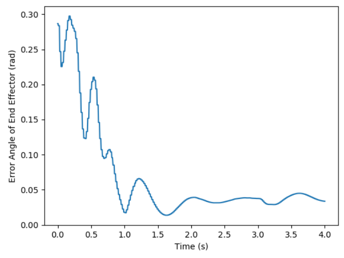
Fig. 9. Jointspace Torque Controller Angle Error for Line Trajectory

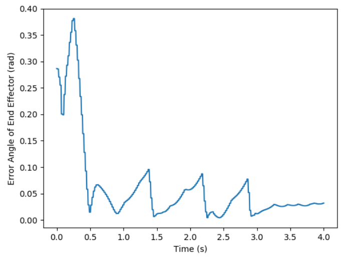
Fig. 6. Jointspace Velocity Controller Angle Error for Line Trajectory

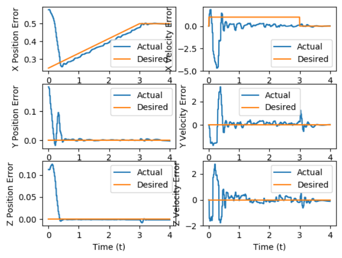
Fig. 10. Workspace Velocity Controller XYZ Error for Line Trajectory

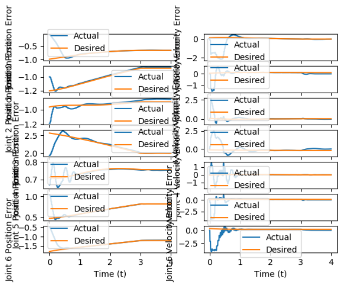
Fig. 7. Jointspace Torque Controller Joint Error for Line Trajectory

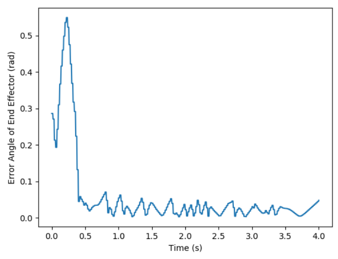
Fig. 11. Workspace Velocity Controller Angle Error for Line Trajectory

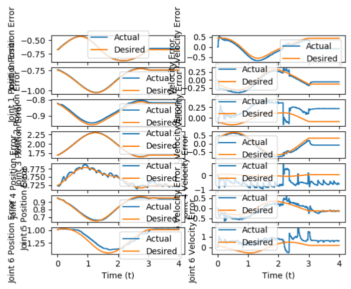

Fig. 12. Jointspace Velocity Controller Joint Error for Circle Trajectory

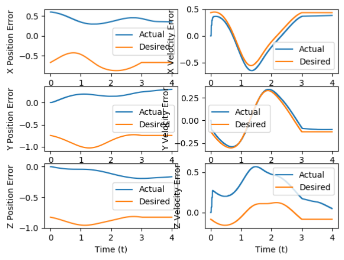

Fig. 13. Open Loop Controller XYZ Error for Circle Trajectory

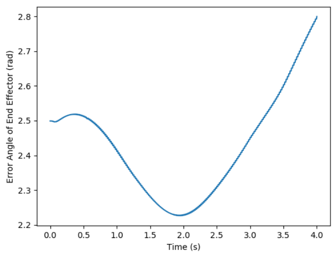

Fig. 14. Open Loop Controller Angle Error for Circle Trajectory

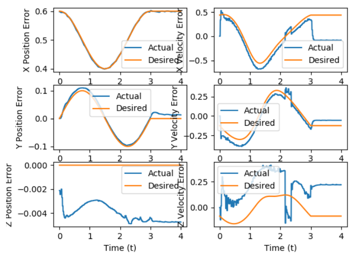

Fig. 15. Jointspace Velocity Controller XYZ Error for Circle Trajectory

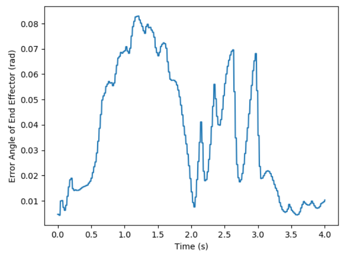

Fig. 16. Jointspace Velocity Controller Angle Error for Circle Trajectory

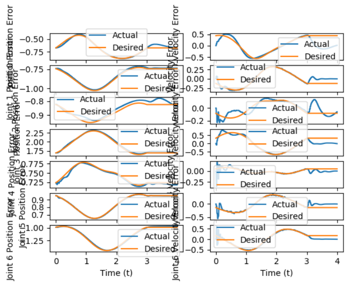

Fig. 17. Jointspace Torque Controller Joint Error for Circle Trajectory

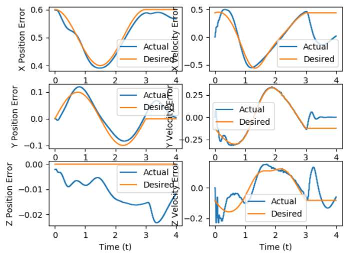

Fig. 18. Jointspace Torque Controller XYZ Error for Circle Trajectory

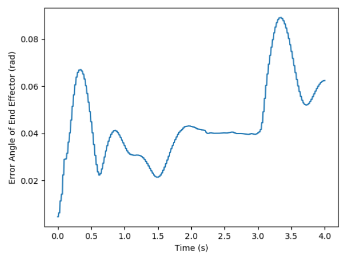

Fig. 19. Jointspace Torque Controller Angle Error for Circle Trajectory

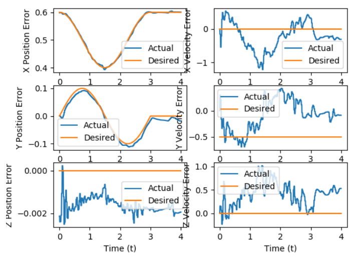
Fig. 20. Workspace Velocity Controller XYZ Error for Circle Trajectory

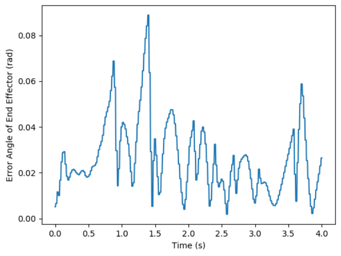
Fig. 21. Workspace Velocity Controller Angle Error for Circle Trajectory

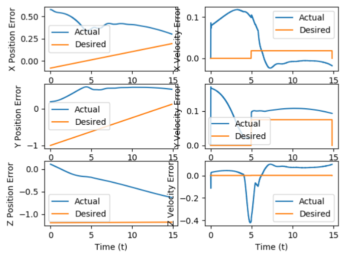
Fig. 22. Open Loop Controller XYZ Error for Arbitrary Trajectory

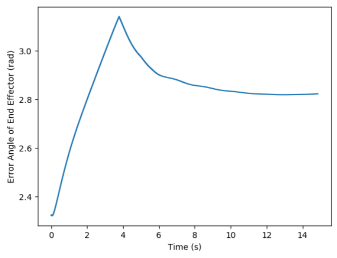
Fig. 23. Open Loop Controller Angle Error for Arbitrary Trajectory

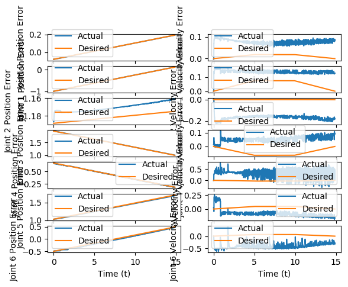
Fig. 24. Jointspace Velocity Controller Joint Error for Arbitrary Trajectory

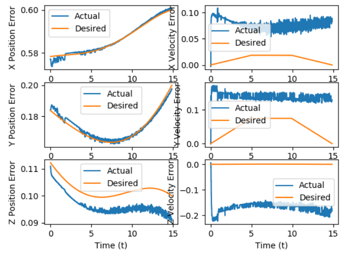
Fig. 25. Jointspace Velocity Controller XYZ Error for Arbitrary Trajectory

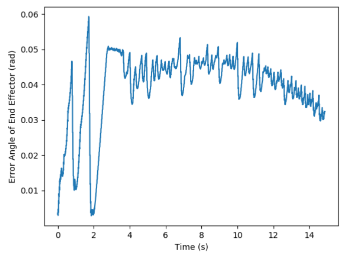
Fig. 26. Jointspace Velocity Controller Angle Error for Arbitrary Trajectory

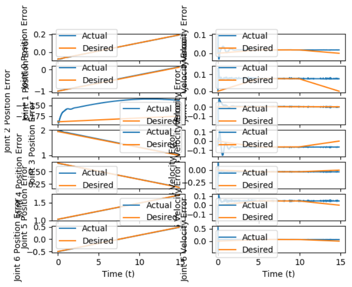
Fig. 27. Jointspace Torque Controller Joint Error for Arbitrary Trajectory

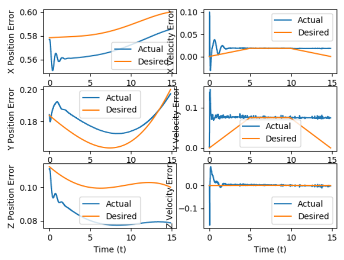

Fig. 28. Jointspace Torque Controller XYZ Error for Arbitrary Trajectory

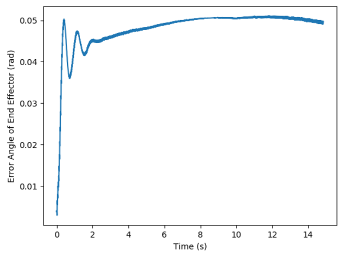

Fig. 29. Jointspace Torque Controller Angle Error for Arbitrary Trajectory

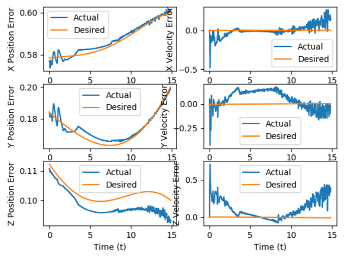

Fig. 30. Workspace Velocity Controller XYZ Error for Arbitrary Trajectory

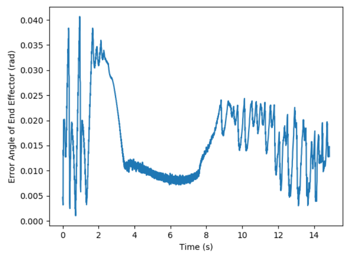

Fig. 31. Workspace Velocity Controller Angle Error for Arbitrary Trajectory

controller. This is because the tracked object is naturally moving in the XYZ workspace, and so this type of controller will be able to rapidly recompute the necessary angles to send to the motor.

The jointspace torque controller would be best used for a situation where the motors have torque limits, and could potentially break or malfunction if the wrong amount of torque is applied. This is especially true for cheaper motors. By controlling the torques directly, it is possible to put constraints on the torque inputs so the robot will choose a path that minimizes strain. Additionally, this adds an element of human safety, since we can effectively impose safety torque limits on the robot.

Finally, the jointspace velocity controller is best for slow moving joints and situations where the robot needs to maintain a certain speed. This is especially useful in situations with human-robot interaction, where rigid motions and unexpected jerks can surprise people. It is important for the robot to move in a way that is legible, and in a way that humans can perceive, and constraining the velocity of the joints to human-perceptible speeds is essential to enabling effective collaboration.

V. Bibliography

References

[1] R. Murray, Z. Li, and S. Shankar Sastry, *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

VI. Appendix

A. *Future Improvements*

One way the started code could be improved would be by improving the plots produced when our own controllers were used. When tuning our controllers, we attempted to follow the Ziegler-Nichols method, but struggled to find the oscillatory k values and the frequency of the oscillations, and our method of tuning had to be modified. Also, for the workspace controller, we were confused about the usage of the body_jacobian_pinv function in utils, and found that it did not work as expected. When we used np.linalg.pinv with ee106b_baxter_kdl, we were able to get the behavior we desired.

B. *Code Repository*

GitHub Link

C. *Demonstration Video*

YouTube Link

D. *Bonus*

We believe the learning goals of this assignment were to show the complexities of taking mathematical theory and translating it into functional code on real machines. We also learned the nuances involved with tuning controllers, and how difficult it can be to tune controllers to work perfectly in every circumstance.