MEMORANDUM

To:	Charlie Refvem, Professor, Department of Mechanical Engineering, Cal Poly SLO crefvem@calpoly.edu	
From:	Jireh Velasquez jivelasq@calpoly.edu	Aubrey Perez apere276@calpoly.edu
Date:	10/29/2025	
RE:	**ME-428-01**	

Story Time

Our goal for this lab was to integrate a PI controller for Romi's wheels. We began by coding in a standard proportional control that lives within a fifth task. The sole purpose of task 5 is to manage and control the PI controller to update the motor efforts. We also incorporated a new class called PI_control. This class holds methods to create and alter controller objects made in task 5. After much trial and error, ups and downs, we were able to integrate a new task into the task scheduler that uses a new controller class to accept a setpoint given by the user interface task then provide a requested motor effort to the motor/ encoder tasks. We made this setpoint available to be tweaked for both motors in our Bluetooth connection file titled "bluetooth code.py". After getting this to work, we then wrote the code to also allow Kp and Ki to be changed via Bluetooth, allowing full testing and tuning of Romi's two wheels.

Testing

We first wanted to tune Romi's motors while travelling in a straight line at 60,000 ticks/s. Our first few runs were chaotic. For the first one, we only had a Kp of 0.3 and a Ki of 0, while for the second one we tried to see if lowering the Kp to 0.01 would have an effect on the response. The response of both motors during these tests can be seen below in Figures 1 and 2.

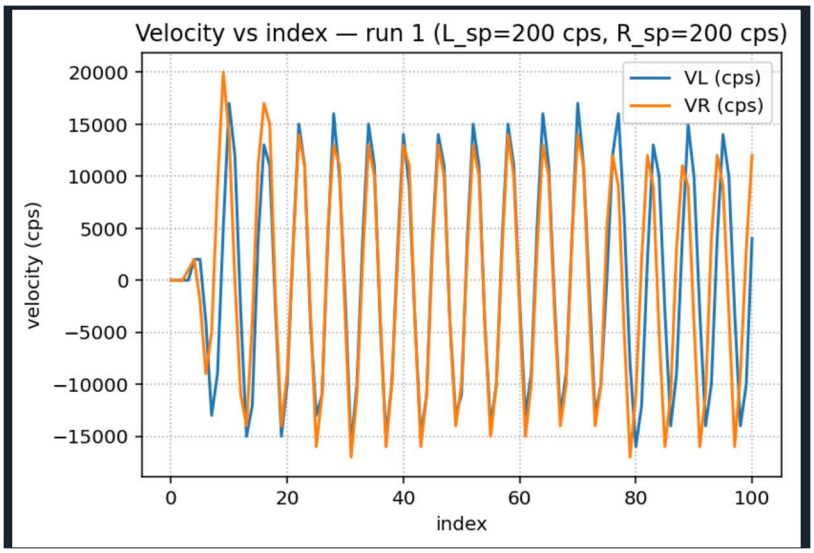

Figure 1. First forward driving test with Kp = 0.3 and Ki=0.

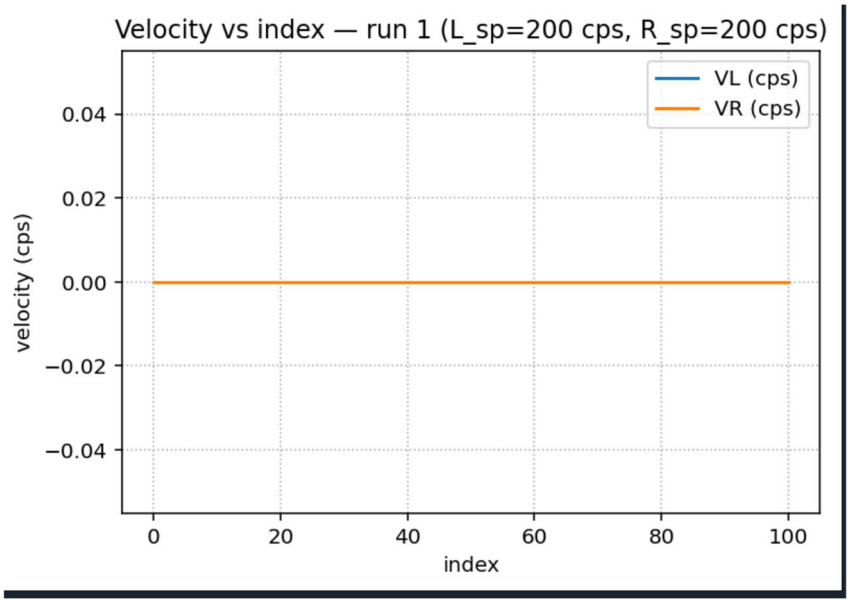

Figure 2. Second forward driving test with Kp = 0.01 and Ki=0.

After a few more tests playing with the Kp value, we finally used a Kp and setpoint value that gave us a reasonable and expected step response. This first reasonable step response can be seen below in Figure 3.

Figure 3. Forward driving test at Kp = 1 and Ki = 0

After getting the good response, we decided to try the same values while having Romi drive forward on the ground rather than being suspended in the air. Running this test gave the best results yet, with curves that had less noise. The response during this test can be seen below in Figure 4.

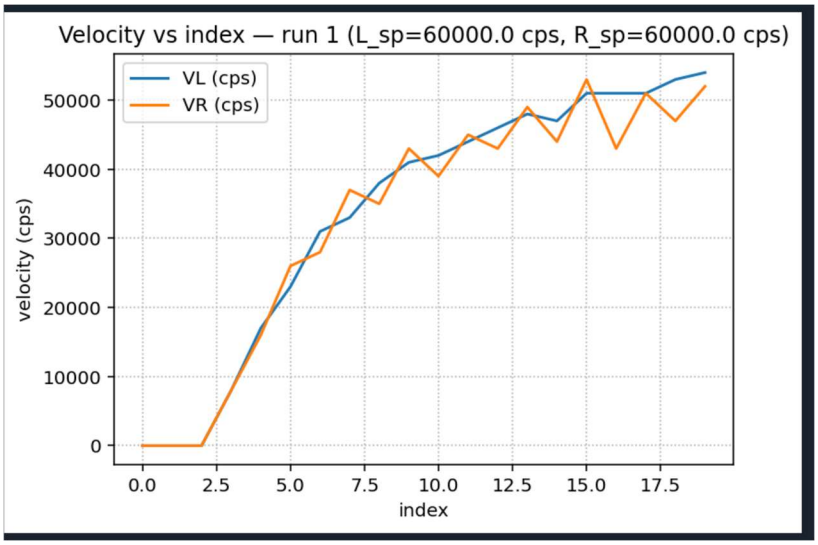

Figure 4. Forward driving on the ground with a Kp=1 and Ki=0

From here, we decided to try out different Ki values. Entering different values of Ki, we saw that there was not much of a noticeable change in the performance of Romi; However, in the graph we were able to see a better overlay between the left and right motor response. As we know that Ki helps keep the system near target value, we upped the Ki to a value of 10 and left it there for most of this lab. The graph of the response for changing Ki to 10 can be seen in Figure 5.

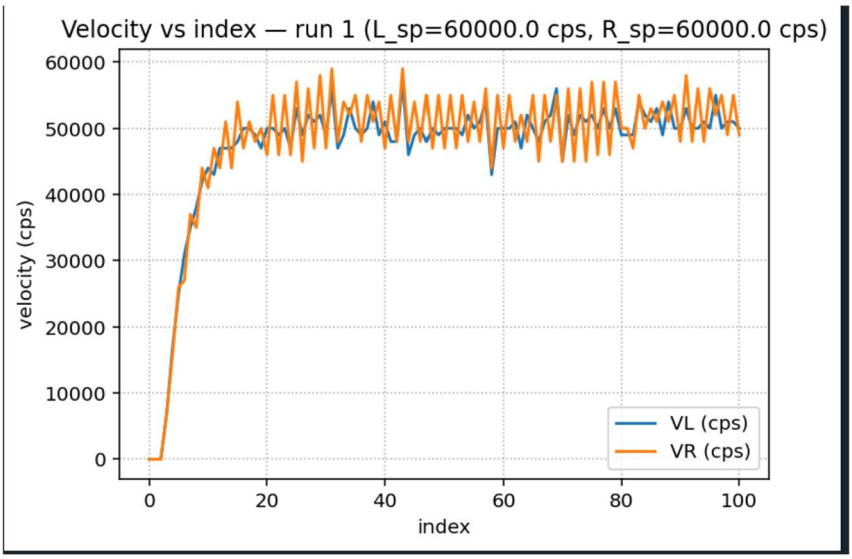

Figure 5. Forward driving on the ground with a Kp=1 and Ki=10.

Through these different trials, we were able to deduce different Kp and Ki values we should have for Romi to drive straight, drive in a circle, and spin. For example, for the set point of 60,000 ticks/s, we found that setting the left motor's Kp to 1, and the right motor's Kp to 2, Romi drives in a straight line rather than turning. The response graph for this example is shown in Figure 6 below.

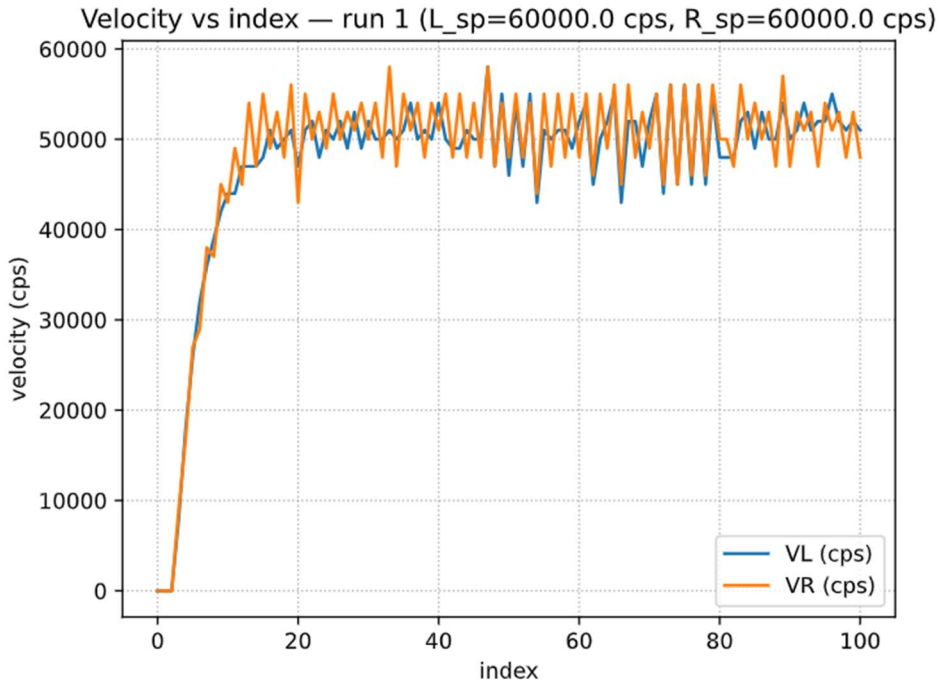

Figure 6. Romi successfully driving forward with Kp_L=1, Kp_R=2, and Ki=10.

Conclusion

In this lab we were able to successfully integrate a PI control loop into our system. There is still some refining to do, but we are able to see noticeable changes to Romi's motors as we change variables in the user interface code. The next steps we want to take in this code is refining any jittering or bugs and adding the battery droop compensation into our control loop.