Vertical Profiler Simulator and Control Algorithms

Dwayne Wallace

Abstract

The goal of the simulator was to test different control algorithms, like PID loops, in a controlled environment instead of on hardware. A secondary goal of this project was to help decrease iteration time to find a solution sooner. The simulator would also double as a place to get rough values for tuning the chosen control algorithm.

The Simulator

The simulator uses kinematics to find the sum of the forces applied to the vertical profiler. 3 main forces are affecting the vertical profiler. We have gravity, the buoyant force, and drag. Gravity, we simply use the mass of the vertical profiler and the acceleration due to gravity to find that force.

$$F_g = m * g$$

Then there is the buoyant force. This is the force that the vertical profiler controls to move through the water. As a baseline, we use the neutral buoyancy force; this force will be equal to the force of gravity. We get the output from the control algorithm as a float value from -1 to 1. The buoyancy engine delta force is found next; it is the maximum force that the buoyancy engine can put out, and we can use the float value from the algorithm to get a buoyancy force to control the vertical profiler.

$$F_b = -F_g + (rho * g * (Bouyancy\ Engine\ Volume / 2) * Control\ Algorithim\ Output$$

The last piece of the force from drag, currently, we are estimating it with the drag force equation. We are using a drag coefficient of a cylinder with a cross-sectional area of 0.007 square meters. In all of our calculations, we are using a density of 1023 kg/m^3 for rho, as that will be the density of the pool we will be competing in.

$$F_d = \frac{1}{2} * rho * v^2 * C_d * A$$

Then the net force applied to our vertical profiler is simply the sum of the forces

$$F_{net} = F_g + F_b + F_d$$

Using the force and the known mass of the vertical profiler, we can find the acceleration of the vertical profiler. Then, using Euler's method, we find our velocity and position.

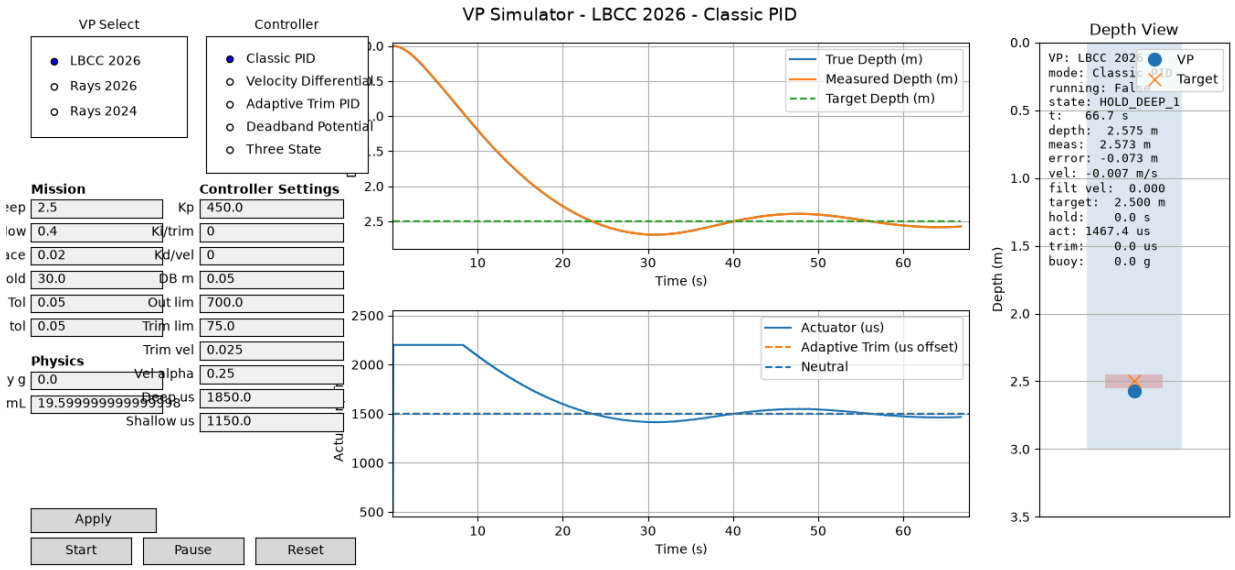
$$a = F_{net} / m$$

Now that we can model the forces involved with the vertical profiler and its position over time. We can take that position and feed it back into a control algorithm.

Control Algorithms

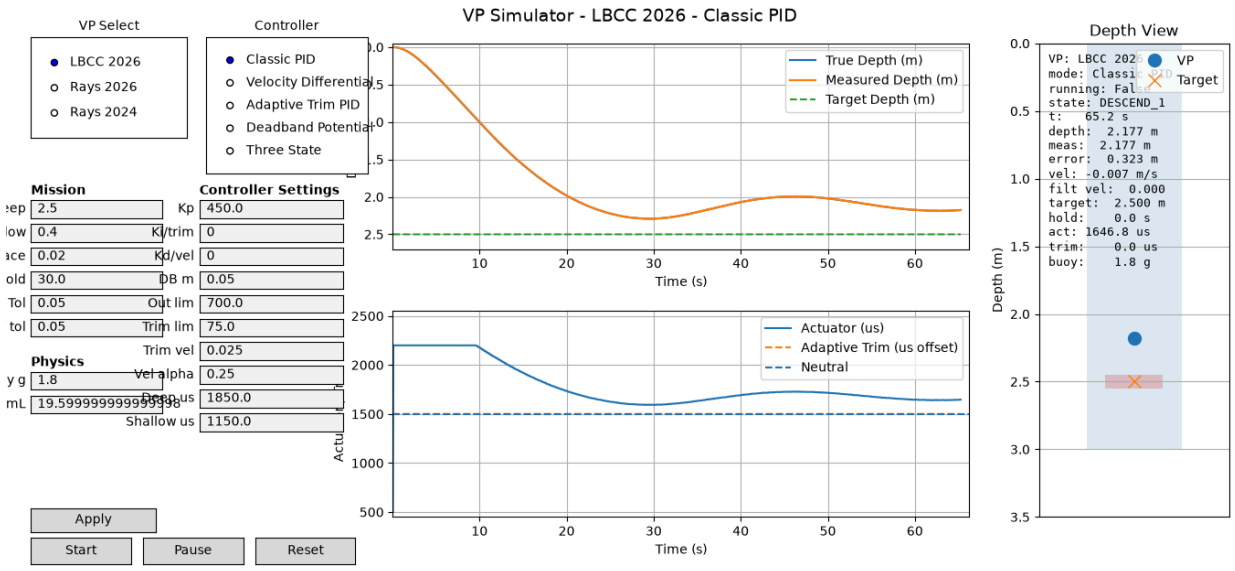
Just Potential

This is simply the potential part of a standard PID loop, with the Integral and the derivative constants zeroed out. For the simulations that we ran, we used a potential constant of 450 us/m. We did not use normalized values to better understand what exactly is going on in the simulations. Below is the control simulation:



There is a constant overshooting with almost all of the control algorithms. We then tried some different derivative techniques to try and limit the overshoot, but nothing stuck out as a great solution.

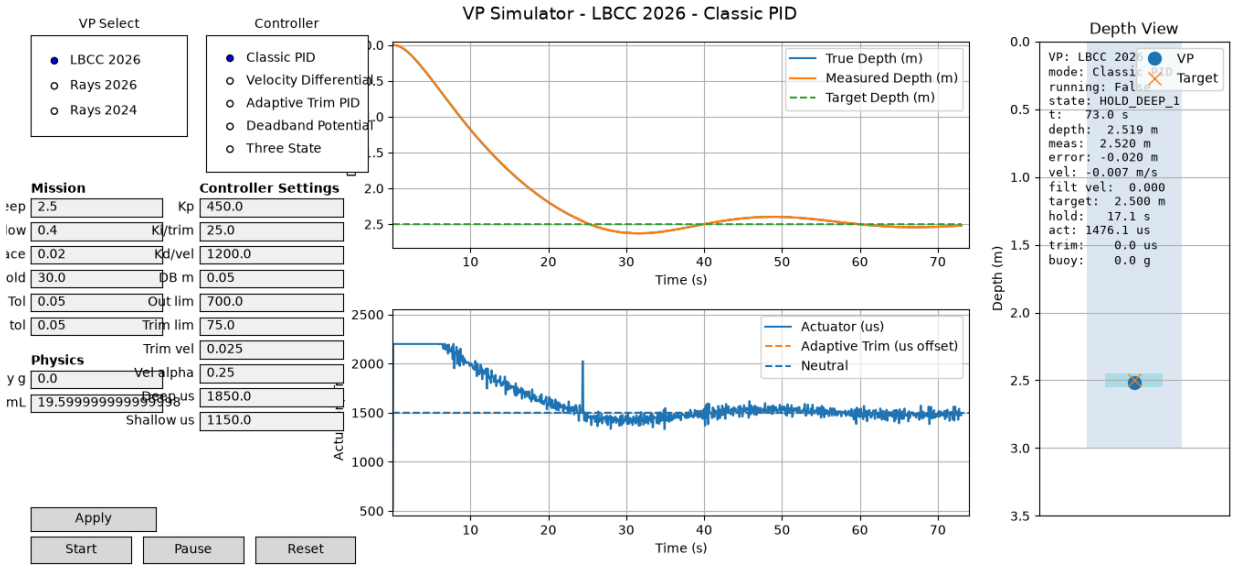
In a later pool test with hardware, we found that the ballast of the vertical profiler had a large effect on where the vertical profiler hovered. An error as small as 0.15% on the overall mass can throw off where the vertical profiler hovers as much as 30cm. This can pose a large issue as it can be difficult to have a constant density within 0.1% of the vertical profiler's mass. Therefore, we also tested to verify our theory and to see how different control algorithms react to an error in ballast mass. Below is the potential algorithm with an error of +0.18%.

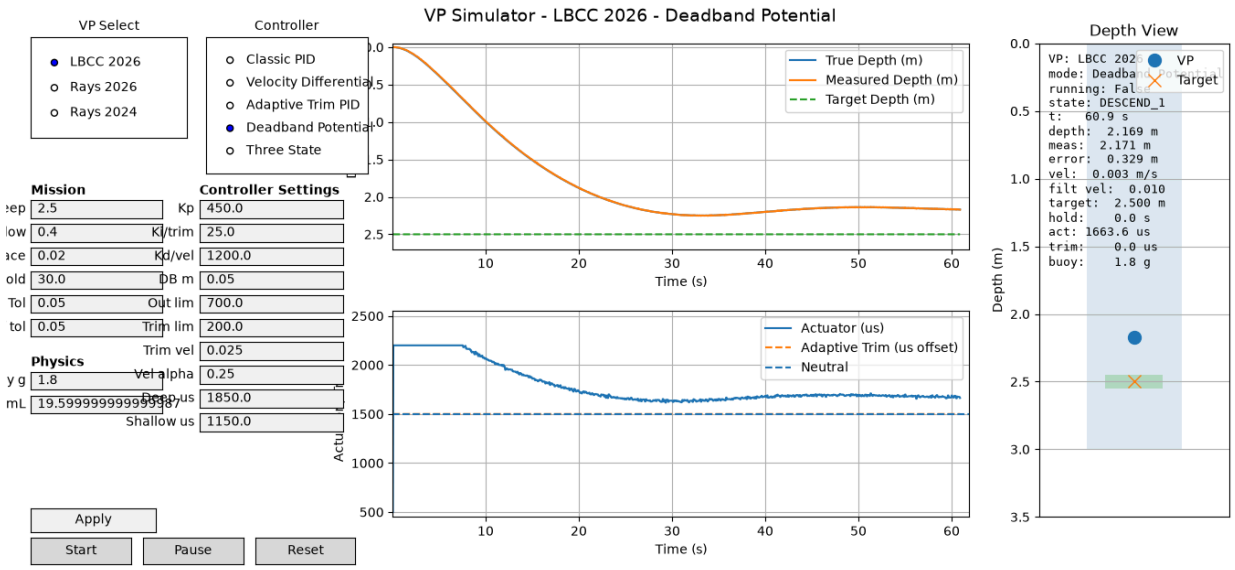


You can clearly see that the vertical profiler is hovering above the target.

Standard PID

This is a standard PID loop that you will find anywhere there is a control system involved. We used 450 for our potential, 25 for our integral, and 1200 for our derivative. In the end, we have a very similar performance to the algorithm that only used the potential. Below is the control and the 1.8g error.

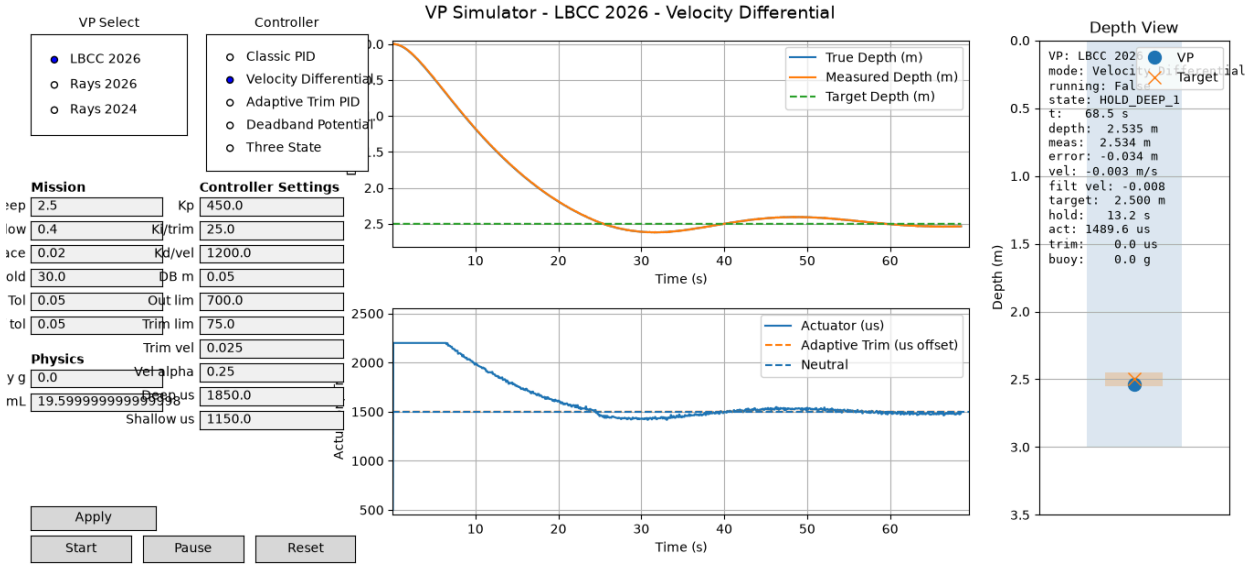


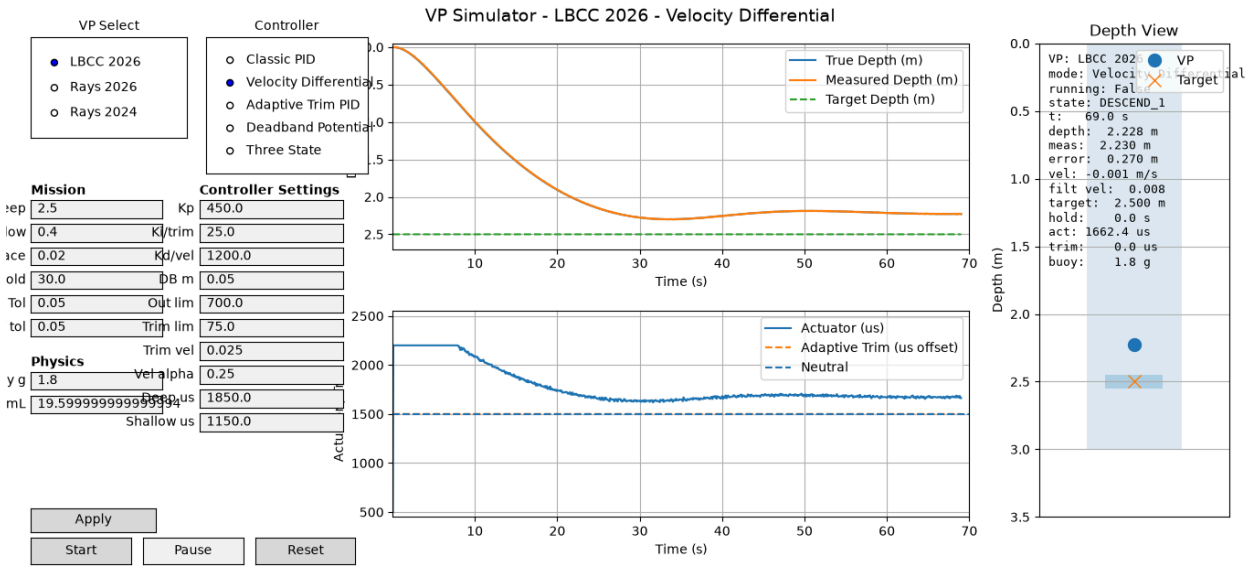


The simulation also models noise so hee how the different algorithms react. It does this by running the calculated velocity through a filter before feeding it into the algorithm. We have noticed that this algorithm performs pretty poorly in the presence of noise. We believe this is due to the way that the derivative is calculated.

Velocity Dampened Differential

This algorithm's goal was to try to prevent some of that overshoot that we are seeing. It does this by taking the second derivative instead of the first, so that the derivative scales sooner. This seems to solve the noise problem, but it still doesn't fix the error in the ballast. It seems to have helped the overshooting, but not to the point that we are happy with.

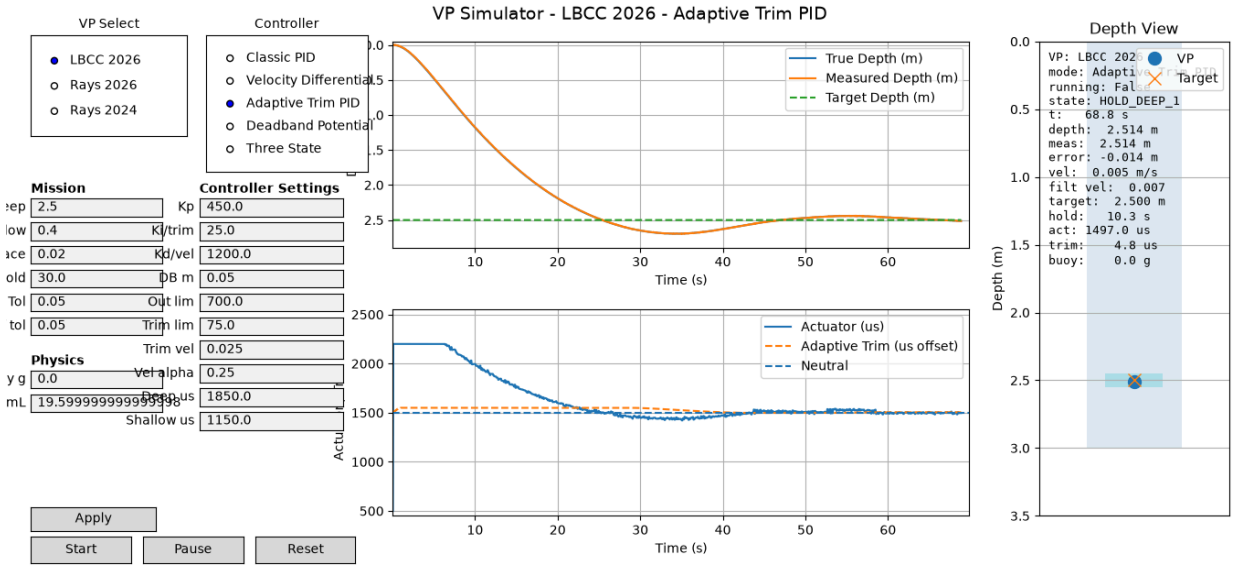


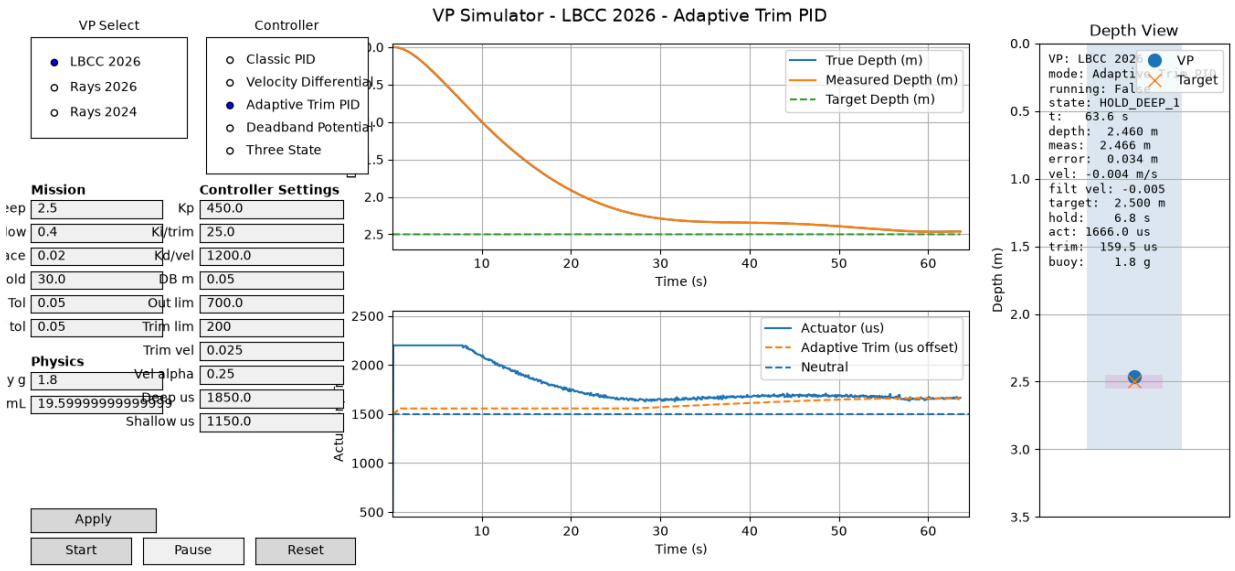


Adaptive Trim

The direct goal of this algorithm was to solve the error in the ballast. It does this by having a higher gain integral component, but it is case sensitive. The integral only affects the center point of the buoyancy engine. By default, it is set to 1500 us with the goal of the vertical profiler to be perfectly neutral at that position, but we know that can be hard to accomplish.

The integral only turns on when the vertical profiler is far from the target and has a low velocity. Once the vertical profiler meets those points, the integral starts trimming the center point. The gain of the integral is affected by how far it is from the target, allowing the vertical profiler to inch towards the target without overshooting and getting the wrong trim value.

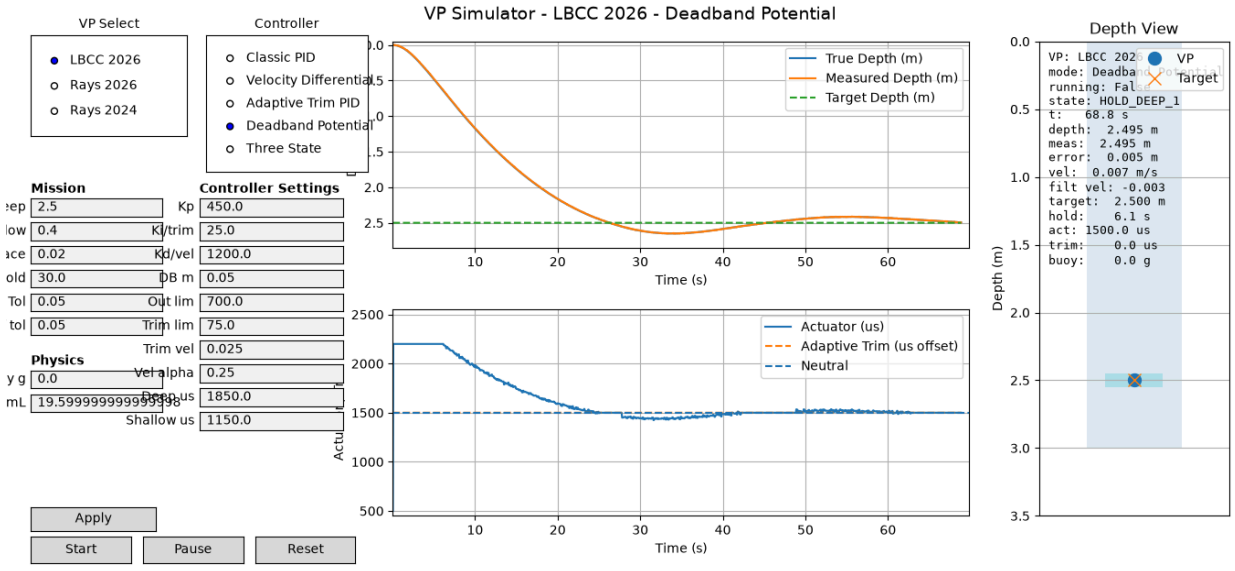




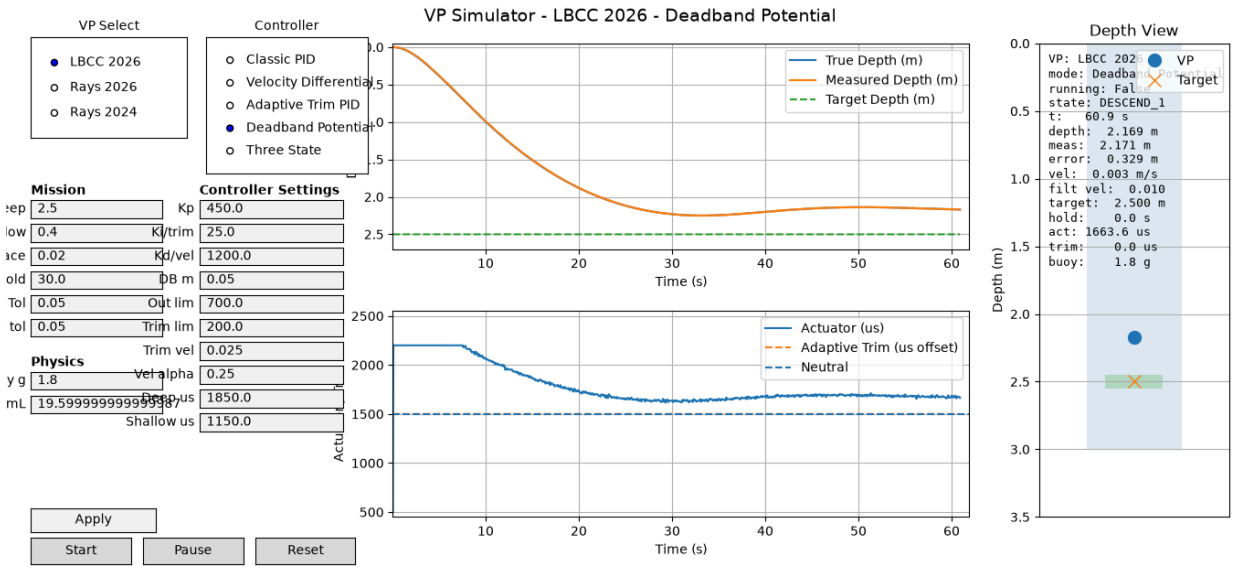
This proves to be a very capable control algorithm; it solves the error issue, and it looks like it has the potential to be tuned further to get closer to a perfectly damped-like model.

Deadband Potential

The goal of this model was to help solve the overshoot while having a simpler model. It just has a deadband around the target so that the vertical profiler can use the high drag to slow itself down. A dummy way to start braking sooner.

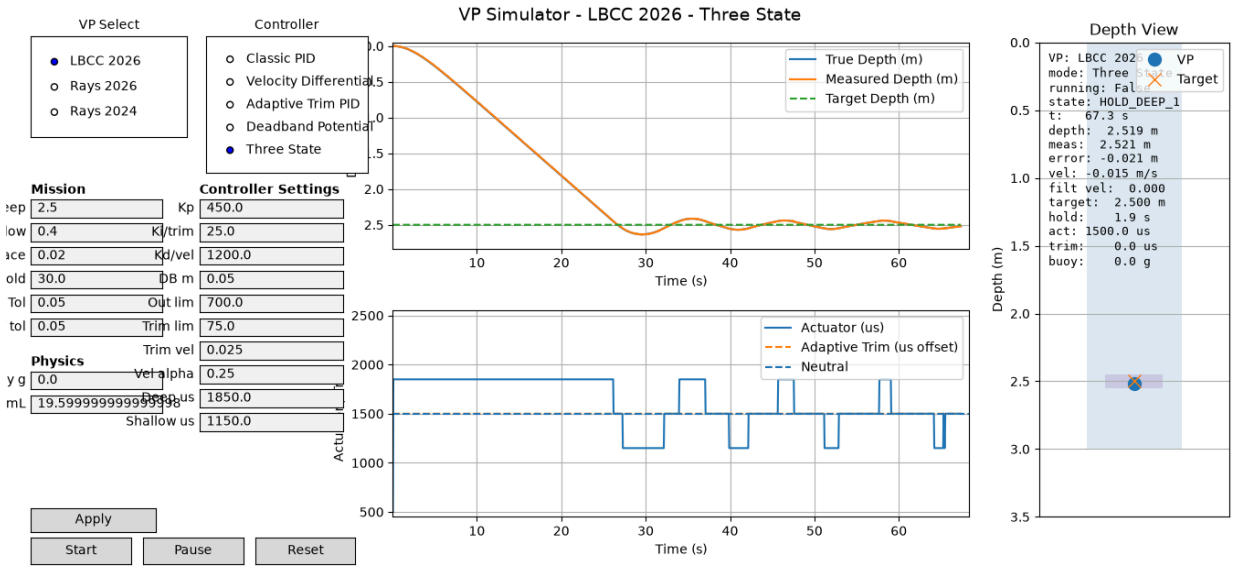


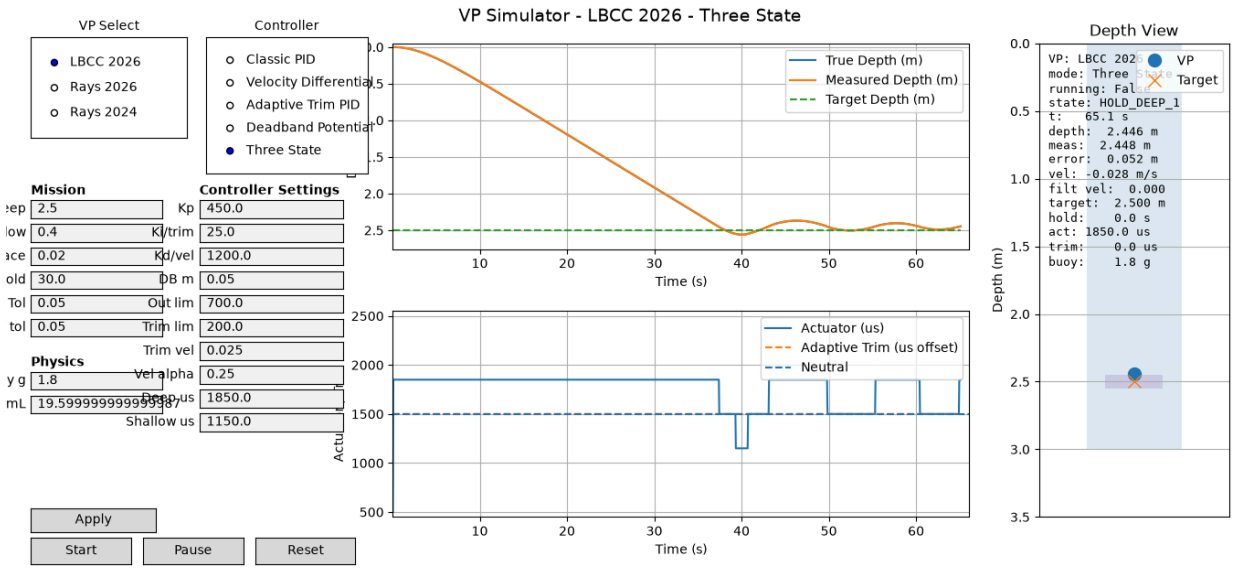
This seems not to have solved the overshoot problem, and it does nothing to address the issue of it not hitting the target when the ballast is off.



3-state Controller

This one is trying to be as simple as you can possibly get. It works by simply seeing if you are above the target or below. If you are above, move down. If you are below, move up. This algorithm seemed to handle the error in the ballast, and there was an acceptable amount of overshoot.

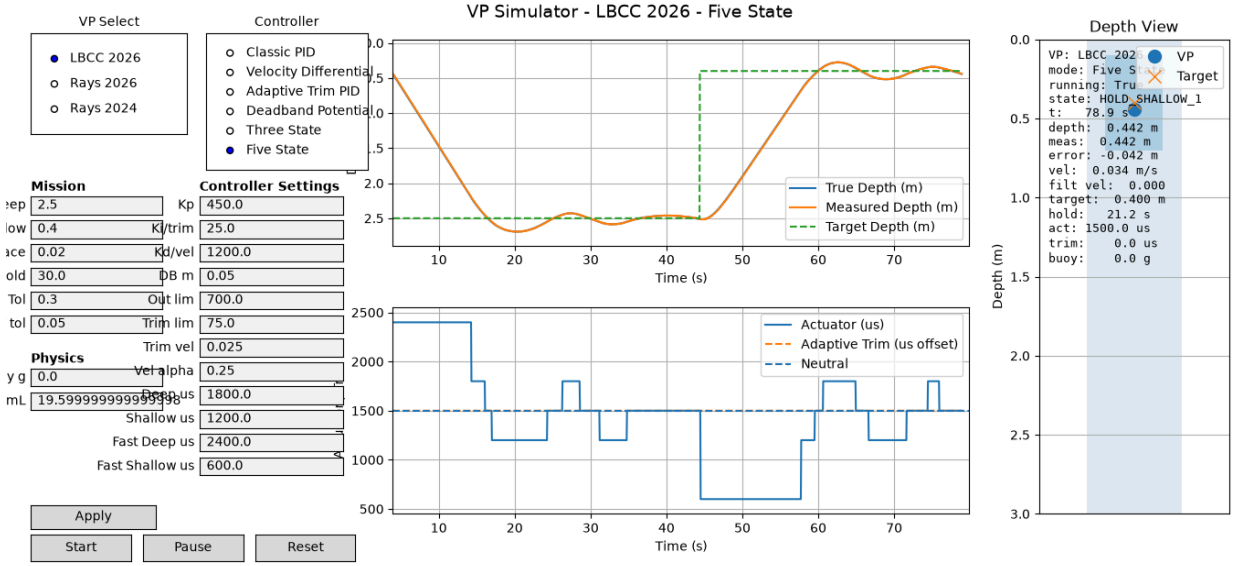




This looks like a very promising solution. It solves most of our problems while staying extremely simple, avoiding case scenario bugs, and creating a more reliable product.

5-Sate Controller

This one is very similar to the 3-State controller but adds another step to the algorithm. One of the factors that we did not like about the 3-state controller is how slow it was getting to the point. Take over 35 seconds to travel the 2.5 meters. So with the 5-state, we add 2 states with a higher microsecond difference to help with that, but then use the other values closer to the target so we get the same performance that we were liking so much. Below is just the control run out a little longer than our 60-second simulations.



One factor that we really liked about the algorithm is it was the only one that started to move and almost got to the second hold at 40cm in 60 seconds.

Conclusion

We have decided to use the 5-State controller. Its simplicity makes it reliable, where one can trust it to not fail in the field, but it does not compromise performance when you need it. It fits all of our requirements of: handling a large ballast error, performing high enough to complete MATE's task, and being reliable enough that we are more than willing to trust it in the field.