Bennett Hermanoff

Denys Bondar

ENGP-3170

27-02-2022

PID Web Simulator: Accessible Control System Learning Demonstration

Motivation and Framework

In High school, I became extremely involved in my school's FIRST Robotics Team. Over three years I served as my team's CAD and Code captain, designing robots, creating drawings, and writing code to control tele-operated and autonomous robots. Today, I am a mentor to the team, helping to teach the next generation of engineering students. One of the hardest concepts to teach to aspiring engineers is control systems. In robotics, control systems are extremely important in creating responsive mechanisms, the most common control loop is the PID controller, which allows for an actuator to be powered based on a sensor reading. These controllers are tuned by a proportional, integral, and derivative term, which influences the response signal.

To help students learn how PID controllers work, and teach them how to tune them, I have created a website that simulates a PID controller connected to a motor. It has an input and output graph, with virtual sliders to influence the 3 PID constants and properties of the simulated motor. There are multiple challenge presets, including virtual sensor readings with less precision, and with random noise added to increase tuning difficulty. As a website, this application allows students and other users to learn the basics of control systems tuning without downloading a language like Julia or others. The website is built in Typescript and Typescript React (TSX) using the React Web Framework with UI and graphing components provided via the Material UI and recharts libraries.

Methods

A Proportional-Integral-Derivative (PID) controller works by calculating a proportional, derivative, and integral response to an Error function. In the web simulator, the aim is to control a motor's rotation angle to be equal to a certain setpoint, and so the error is calculated by subtracting the motor's current angle modulo 2π from the setpoint:

$$\mathrm{e(t)} = \theta_{setpoint}(t) - (\theta_{motor}(t)\ mod\ 2\pi)$$

The standard form of a PID controller can be described as the following:

$$u(t) = K_\mathrm{p}\left(e(t) + \frac{1}{T_\mathrm{i}}\int_0^t e(\tau)\,\mathrm{d}\tau + T_\mathrm{d}\frac{\mathrm{d}e(t)}{\mathrm{d}t}\right)$$

In the Web simulator, the integral and derivatives are calculated using numerical techniques. The PID Controller class simply computes the Riemann sum of each error received by the controller. The simulation runs for a default of 1000 time steps, each equal in length, and so the Riemann sum is computed by adding the current error multiplied by the size of each timestep to the previous sum:

$$\int_0^t e(t)dt\ \approx\ sum(t) = sum_{(t-dt)} + e(t) * dt$$

The derivative of e(t) is computed using the current error and previous error. This is very similar to Newton's backward difference calculation:

$$\frac{de(t)}{dt} \approx \frac{e(t) - e(t-dt)}{dt}$$

For simulating the motor, an input adds a torque onto the motor, with the "noisiness" of the motor, c_n, randomly influencing this torque. The motor also has a friction force, c_f, that can be set by the user.

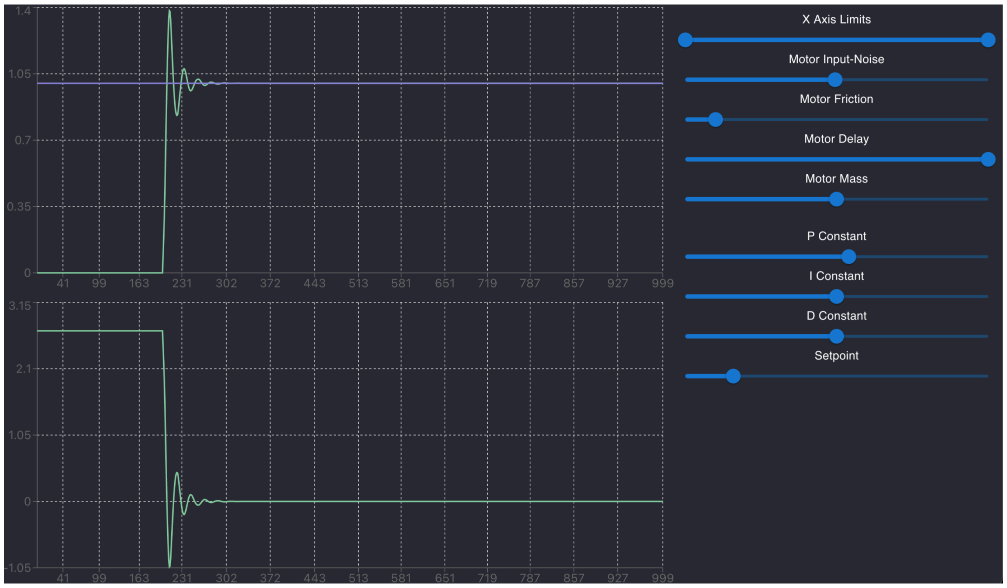
$$\alpha(t) = \frac{input(t) + input(t) * random(t) * c_n}{I} - c_f * \omega(t)$$

The current angular velocity, $\omega(t)$, is calculated by multiplying the current acceleration, $\alpha(t)$, by the size of the timestep, dt. The motor class can also delay inputs by a set amount of timesteps, this simulates a motor that does not respond immediately.

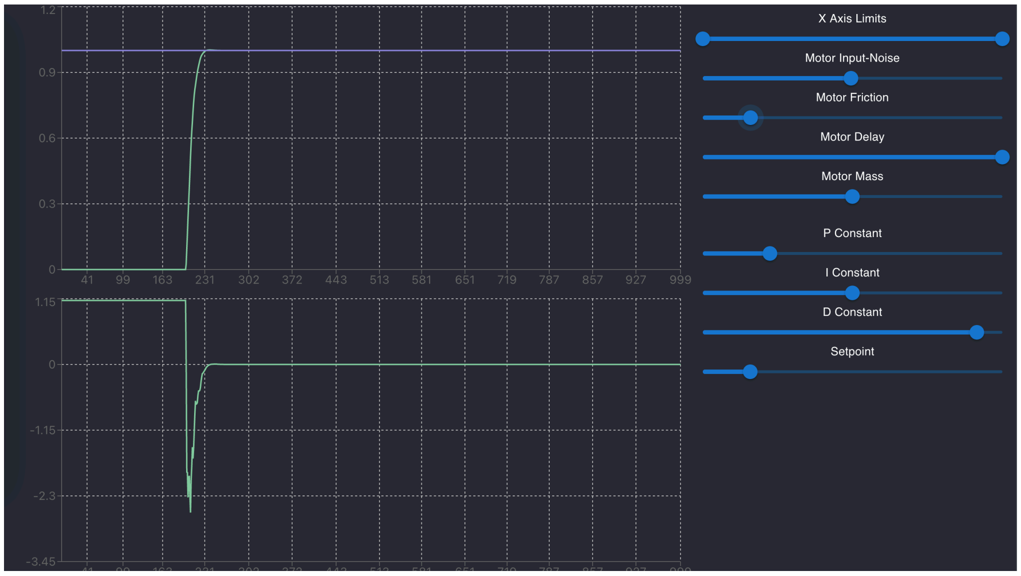
On the user display side, whenever an input is changed, an Effect hook is called, and the simulation runs and calculates the values for each timestep. The graph is then updated and displayed to the user.

Results and Examples of Use

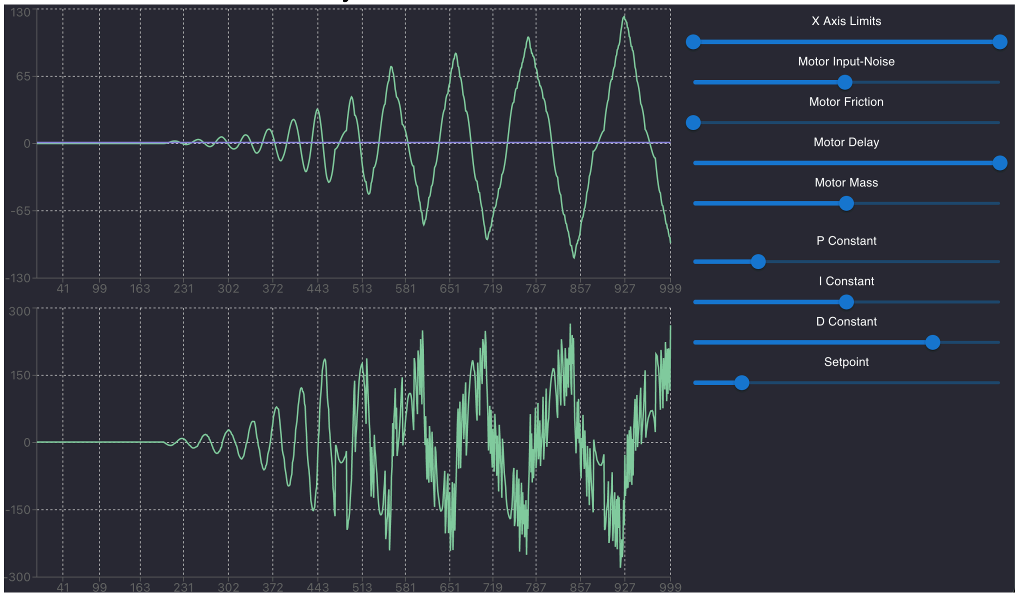
The Default state represents a poorly tuned, but functional control loop, with a lot of overshoot and settle time.



This default state can be tuned up by the user by lowering the P constant, and adding a small amount of D constant can also decrease the overshoot.



Another good example is to see what happens when the motor friction is completely removed. A PID controller relies on the motor being imperfect and having any amount of friction to limit the motor’s rotational velocity.



This extreme state is also a good way to show the limits of the numerical integration and derivation used in the simulator. The frequency and amplitude of the PID’s output is too large for the relatively large timesteps used in the simulation.

Conclusion

I am really proud of this little application I have built here. The simulator is simple to use, is very adaptable to different scenarios, and responds almost instantly to user input. The simulator is availible on my personal website (hermanoff.dev/pid) and I hope that it can be utilized in the future to show others the basics of a PID controller and how to tune one.