ANALOG PID CONTROL FOR A DC MOTOR

COREY COCHRAN-LEPIZ
VIENNA SCHEYER

Date: December 2018.

1. Abstract

This paper examines analog PID control as it relates to a DC motor controller circuit. Our goal was to analyze controls through analog circuitry. In order to do this we derived op-amp circuits that are analogous to PID control blocks in order to create a velocity control scheme. We simulated a motor model in MATLAB with control coefficients for a specific motor. Then we derived transfer functions for op-amp gain, integrator, and differentiator circuits so we could compare the analog control transfer functions to our MATLAB controls simulation. Based on analysis of the steady state error and percentage overshoot, we find good control constants $k_p = 6000, K_i = 3600$, and $K_d = 100$ to get the motor to reach a desired velocity in less than 0.025 seconds.

2. Introduction

Our goals for this project were to implement controls with analog circuitry and show the relationship between controls theory and analog control circuits. In this paper, we will go over the governing equations of DC motors and use a transfer function model for a specific motor, discuss general PID control theory, and then show how general PID control transfer functions relate to analog circuit control op amp transfer functions.

3. Governing Equations of DC Motors

The electric motor provides a bridge between mechanical and electrical energy, also known as a transducer. A motor is essentially an inductor with a permanent magnet arranged in such a way that when we put a voltage across the its terminals the current flowing through it applies a torque on the shaft which we then use to power things like propellers, wheels, and other mechanisms. Although, as stated before, it can also work in the *other* direction. When an external torque is applied to the shaft, current flows.

An important property to note about DC motors is the electromechanical relationships. One such important relationship exists between current (I) and torque ($\mathcal{T}$).

$$\mathcal{T} \propto K_T \cdot I \tag{1}$$

This relationship allows us to design for a desired torque as long as we know K_T, the motor torque-current constant. Another important relationship is the one between the back-emf (V_{emf}) and angular velocity (ω).

$$V_{emf} \propto K_\omega \cdot \omega \tag{2}$$

It turns out that, for all brushed DC motors, K_t and K_ω are the same value, so we can write the general motor constant as K_m.

The reason we care so much about the above equations is that allows us to set a desired torque or angular velocity of the DC motor in our control

loop. In order to do that we just need to find the motor constant K_m which varies from one motor design to another. For this experiment we'll be using a Lego Gear Motor, favored for its low-friction high-speed design. In order to find the motor constant for it we used a table of data from MIT's Random Hall Lego Robotics Seminar [1]. According to this source, the data was taken using a precise high-speed strobe light at MIT for the no-load speed and the voltage source ran at 9V under load. We measured the internal resistance of the motor to be $25\,\Omega$.
In order to find K_T we can setup the equality from equation 1:

$$\mathcal{T} = K_T \cdot I$$

Swapping out I for it's ohmic equivalent

$$0.089Nm = K_T \cdot \frac{9V}{25\Omega}$$

and solving for K_t grants us:

$$K_T = 0.25\frac{Nm}{A}$$

In order to find K_ω we can setup the equality from equation 2:

$$V_{emf} = K_\omega \cdot \omega$$

Since there is no load in this setup that means negligible current which in turn means a V_{emf} of the voltage driving it (9V).

$$9V = K_\omega \cdot 35.6\frac{rad}{s}$$

Which lands us at:

$$K_\omega = 0.25\frac{V}{rad \cdot s^{-1}}$$

Observe that $K_\omega = K_T = K$. With this value for the motor constant, K, we can input a certain emf to set the motor to a desired speed and we can input a certain current to get a desired torque. For our implementation of PID control, which we will discuss later on in this paper, we decided to focus on setting a desired speed by inputting an emf voltage.

We needed to make a model for our motor in order to simulate PID control before building the physical experimental circuit, so we needed to obtain a transfer function for our motor. Starting with equations 1 and 2, we referred to an article by *Control Tutorials for MATLAB & Simulink* for a transfer function derivation. We will go over the steps here. First, we can derive governing equations from equations 1 and 2 based on Newton's 2nd law and Kirchoff's Voltage Law. First, for the torque and current equation we have

$$J\ddot{\theta} + b\dot{\theta} = K \cdot I \tag{3}$$

[1]http://web.mit.edu/sp.742/www/motor.html

Where J is the moment of inertia or resistance to angular acceleration, b is resistance to angular velocity, i is current through the motor. For the angular velocity and emf voltage equation we get

$$L\frac{di}{dt} + Ri = V - K\dot{\theta} \tag{4}$$

Since $\dot{\theta} = \omega$, we can make the following substitution

$$J\dot{\omega} + b\omega = Ki \tag{5}$$

$$L\frac{di}{dt} + Ri = V - K\omega \tag{6}$$

When we take the Laplace transform of these two governing equations, we get the following two equations

$$\Omega(s)(Js + b) = KI(s) \tag{7}$$

$$(Ls + R)I(s) = V(s) - K\Omega(s) \tag{8}$$

By eliminating $I(s)$, we can solve for this transfer function relating angular velocity, $\Omega(s)$, to input voltage, $V(s)$.

$$\frac{\Omega(s)}{V(s)} = \frac{K}{(Js + b)(Ls + R) + K^2}$$

Where K is the motor constant we discussed earlier in this section, J is moment of inertia, b is the friction coefficient of the motor, L is inductance, and R is the internal resistance of the motor. The Lego motor we used is not well documented so we experimentally found values for these constants. To calculate the moment of inertia: We attached a known weight to the motor axle on a string, wound the string around the motor axle, and timed how long it took for the weight to fall the length of the string. Based on this experiment, we calculated a moment of inertia $J = 0.0124$. We ran out of time and resources to experimentally calculate values for L, and R, so we chose reasonable values. Since out motor has very low friction, we used $b = 0.005$. We measured the internal resistance to be 24, and we used an inductance value of $L = 0.1H$ To summarize the constants we chose for our model:

K	0.25
R	25
L	0.1
J	0.0124
b	0.005

4. PID Control

Proportional-integral-differential (PID) control is ubiquitous in our automated world. From setting a desired room temperature to controlling the pressure in a waterjet tool, PID is one of the most effective control methods. In this section we will look closely at the components that make up PID control: P, PI, and PD. Each of these simpler types of control have certain attributes that, when combined, contribute to the power of PID control. For each of the subsections on P, PI, PD, and PID control, we will begin with general controls theory and then derive transfer functions for op amp circuits that serve the same purposes through analog control.

If we draw a block diagram for a general controlled system, as in Figure 1, we feed an input signal, $d(t)$, into the controller block. Then we pass the resulting control signal, $u(t)$, into the plant block. Our feedback loop compares the output signal, $h(t)$, to the input signal. The difference between the output signal and the input signal gives us our error term, $e(t)$, and we feed the error term back in to the controller.

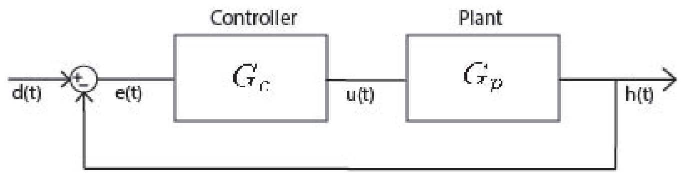

Figure 1. General block diagram with input d(t), error term e(t), control signal u(t), and output h(t)

Mathematically, we can denote the error term as

$$e(t) = h(t) - d(t) \tag{9}$$

Since we haven't added a term for the controller block yet, the control signal $u(t)$ would just be equivalent to the error term.

We can use MATLAB to simulate the behaviour of this system. We can use one of MATLAB's built in functions [2] to look at the step response. The step response shows us the output of a system over time when the input changes from zero to one very rapidly. The following code plots the step response for the DC motor alone with no controller using the DC motor transfer function for our model

```
P = K/((J*s+b)*(L*s+R)+K^2)
step(P)
```

The resulting plot in Figure 2 can serve as our baseline for the system so that we can see how P, PI, PD, and PID control compares to the uncontrolled

[2]https://www.mathworks.com/help/control/examples/plotting-system-responses.html

system. While the uncontrolled system does not overshoot, it takes the motor at least 2.5 seconds to come close to reaching steady state, which is sub-optimal for a motor when set to a certain speed.

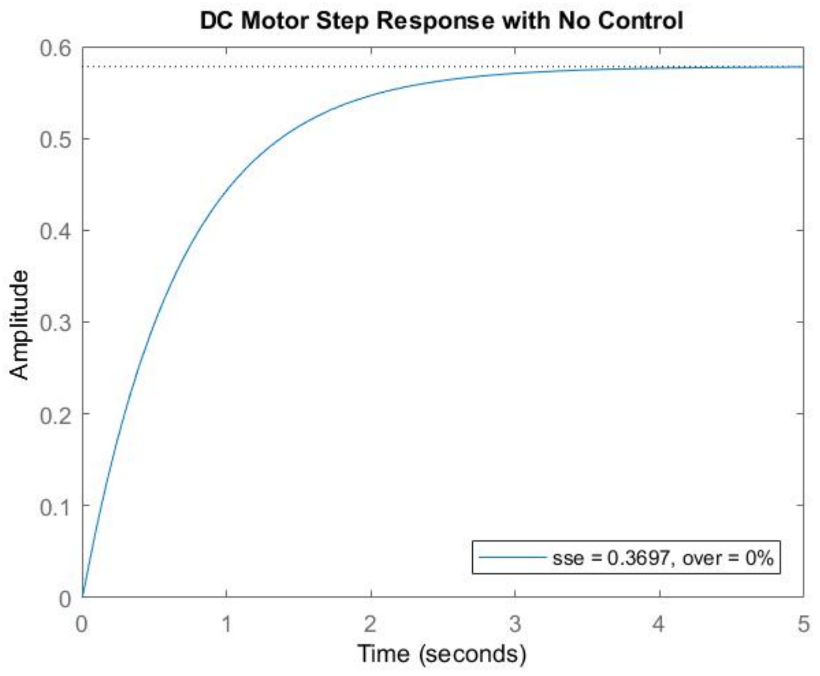

FIGURE 2. DC motor step response with no control

We calculate the steady state error by taking the difference between the expected steady state, 1, and the actual steady state. We can use the following code to calculate the steady state error

```
y = step(P);
sserror=abs(1-y(end))
```

The steady state error for the uncontrolled system is 0.3697.

4.1. **Proportional Control.** In proportional control, we multiply the error term by a proportionality constant, usually called K_p. This serves the purpose of causing the system to react more quickly to the control because the error term is magnified. Now we can write the control signal as

$$u(t) = K_p e(t) \tag{10}$$

To get the transfer function for the controller block, first we take the Laplace transform of this equation to get the following equation

$$U(s) = K_p E(s) \tag{11}$$

Then we rearrange to get the ratio of the output over the input, which is the transfer function

$$\frac{U(s)}{E(s)} = K_p \tag{12}$$

Once we have the transfer function for the controller, we can combine it with the transfer function for our DC motor model, which in this case is the plant

$$\frac{\Omega(s)}{V(s)} = \frac{K}{(Js+b)(Ls+R)+K^2}$$

With the transfer functions for the controller and plant blocks, we can use the following formula to calculate the overall transfer function for the system

$$\frac{V(s)}{D(s)} = \frac{G_c G_p}{1 + G_c G_p} \tag{13}$$

For our purposes it is not entirely necessary to algebraically find the overall transfer function for each kind of controller since we can use MATLAB to simulate the behaviour of this system, but equation 13 shows how one would calculate the overall transfer function. We can use the MATLAB step function as before, but this time we add a proportionality constant. Based on the MATLAB function documentation, we can use the following code to plot the step response, where P is the DC motor transfer function and C is the proportional part of MATLAB's PID control function

```
P = K/((J*s+b)*(L*s+R)+K^2)
C = pid(kp)
T = feedback(C*P,1)
t = 0:0.01:2;
step(T,t)
```

With P control the system approaches steady state significantly faster than without control, however, there are some trade-offs with P control. Figure 3 shows the step response of the system with 3 different values of K_p. The legend shows the steady state error, *sse*, and overshoot percentage, *over*, for each value of K_p. This data indicates that increasing K_p has the effect of reducing the steady state error but it increases the overshoot. The reason the system overshoots so drastically with P control is because there is nothing to keep the control signal in check when we multiply the proportionality constant with the error term. The system just gradually oscillates closer to equilibrium as the error, and therefore the control signal, decreases.

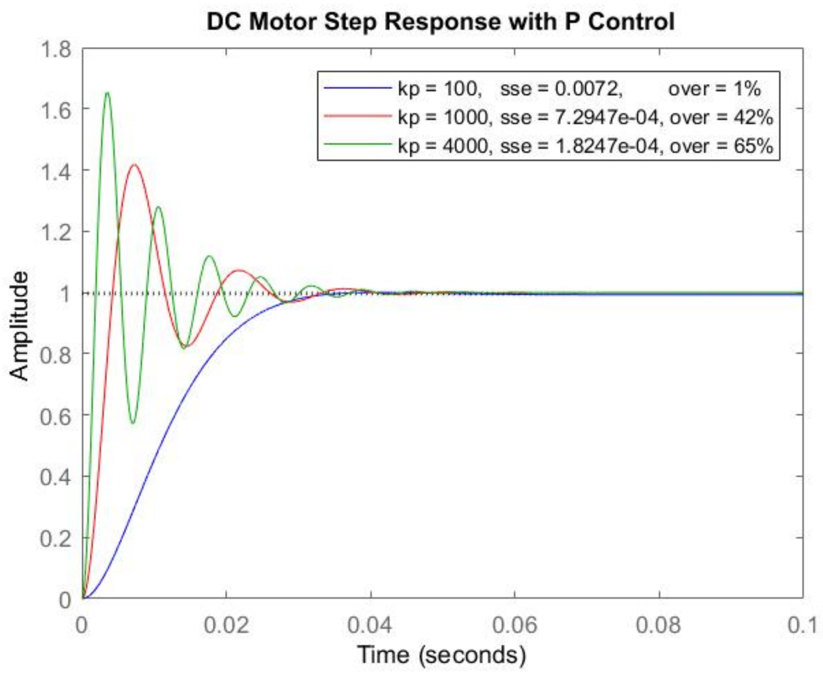

Figure 3. DC motor step response with proportional control

Now that we have looked at the general transfer function for a proportional controller, we will look at how this relates to analog circuitry. We can make a proportional controller using an operational-amplifier (op-amp). The op-amp follows a couple of simple rules: the inputs of the op-amp draw (virtually) no current, and the output is VDD if $V_+ > V_-$ or VSS if $V+ < V_-$. It also exhibits quite a useful behaviour when used in a feedback loop within itself, that is to say that if one of the inputs is tied to its output then the op-amp outputs the necessary voltage to keep the two inputs the same. Let's see how that's helpful to us in terms of finding an equivalent circuit.

In figure 4 we have an op-amp building block. An input voltage is tied to the negative input of the op-amp, and since the positive input is wired to ground that means that the negative input is also at ground. Keeping in mind that the op-amp does *not* allow for current to flow through the inputs that results in

$$I_{R_1} = I_{R_2}$$

and since the op-amp will output whatever voltage necessary to allow for the current of the two voltages to flow through R_2 the voltage drops can be

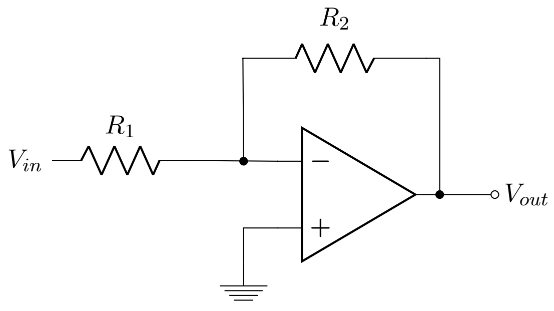

FIGURE 4. Op-amp gain block where $V_{out} = (V_{in})\frac{-R_2}{R_1}$

written as

$$\frac{0 - V_{out}}{R_2} = \frac{V_{in} - 0}{R_1}$$

with a little bit of rearranging.

$$V_{out} = (V_{in})\frac{-R_2}{R_1}$$

In the previous section we highlighted the relationship between velocity and emf (equation 2), so in order to do velocity PID we'll be tuning the controller to aim for the ideal emf for a desired velocity. In order to calculate V_{emf} we sum the motor input and (negative) output voltage to get a single voltage reading for the emf. The above circuit looks similar to the transfer function that we use for proportional control where $R_2/R_1 = K_p$ we can then use the above op-amp block for proportional control within the circuit with V_{emf}.

We will simulate the proportional controller op amp in LTspice later in this paper to see how the op amp P controller serves the same functionality as a digital controller like the one we simulated in MATLAB.

4.2. **Proportional Integral Control.** In PI control, we expand on P control by adding another term to the transfer function. We add an integrator constant, K_i, which we multiply by the integral of the error term. With the addition of the integral term, our new control signal is

$$u(t) = K_p e(t) + K_i \int e(t)dt \tag{14}$$

Taking the Laplace transform, we get the following transfer function

$$\frac{U(s)}{E(s)} = K_p + \frac{K_i}{s} \tag{15}$$

Using the same MATLAB function, we can plot the step response of this system. We can add integral control to the PID function in MATLAB by editing the code as follows. We can keep K_p at a constant value, $K_p = 100$, to just see the effect of changing K_i.

```
C = pid(Kp,Ki)
```

And then we can plot the step response in the same way as before. Figure 5 shows the resulting plot. We can see that the increasing K_i causes a decrease in the steady state error but it also causes the overshoot to increase. In other words, the steady state error and overshoot percentage have an inverse relationship. It would be great to have a steady state error as small as $1.1102e - 16$ when $K_i = 3000$, but it would not be ideal to have the 25% overshoot. The reason adding integral control reduces the steady state error is because if there is a persistent steady state error, the integral term gets bigger and bigger over time and eventually the control signal increases enough to virtually eliminate the steady state error.

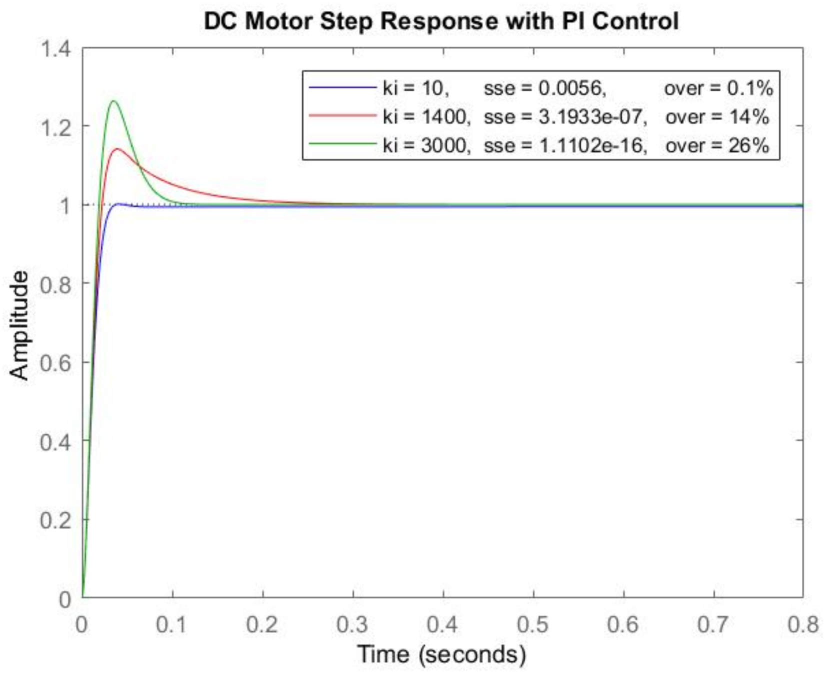

FIGURE 5. DC motor step response with proportional integral control

The way we implement integral control is by using another op-amp building block, the integrator in figure 6. Let's break it down knowing that the current though R is equal to the current through C:

$$\frac{V_{in}}{R} = -C\frac{dV_{out}}{dt}$$

And isolate V_{out}

$$dV_{out} = \frac{-1}{RC}V_{in}dt$$

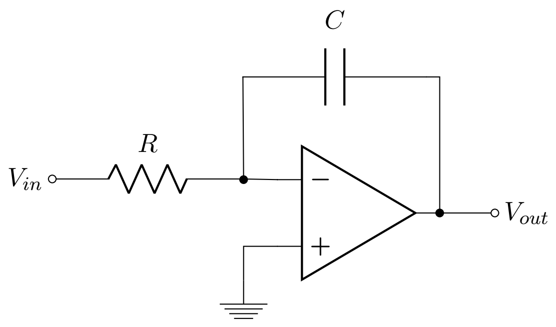

FIGURE 6. Op-amp integrator where $V_{out}(t) = \frac{-1}{RC}\int V_{in}(t)dt$

and finally take the laplace transformation

$$\frac{V_{out}(s)}{V_{in}(s)} = \frac{-1}{RCs}$$

Which is the same as the transfer function in equation 15 if you treat K_i as $\frac{-1}{RC}$. The circuit equivalent allows us to sum the voltage from P and I to get PI control.

4.3. **Proportional Derivative Control.**

Next we can simulate PD control by adding a derivative term to the proportional control instead of an integrator term. In the time domain, this looks like

$$u(t) = K_p e(t) + K_d \frac{de(t)}{dt} \tag{16}$$

Taking the Laplace transform and rearranging for the output over the input we get the following transfer function

$$\frac{U(s)}{E(s)} = K_p + K_d s \tag{17}$$

With PD control, the derivative term means that the control signal u won't change unless the error term changes (since derivative is change over time). This means that, unlike with PI control, PD control does not decrease the steady state error. In fact, the high the K_d value, the bigger the steady state error. This is because once the error term stops changing, even if the steady state has a large error term, the derivative of the error is no longer significant to affect the control signal. The addition of the derivative term can cause a very overdamped system (as shown in Figure 6) because, if the error starts sloping upwards, the control signal increases with it. This is the main advantage of PD control because higher K_d values have the effect of causing the system to reach steady state faster.

The final op-amp block in the ensemble is the differentiator circuit in figure 8. This may look very similar to the integrator, and that's because it

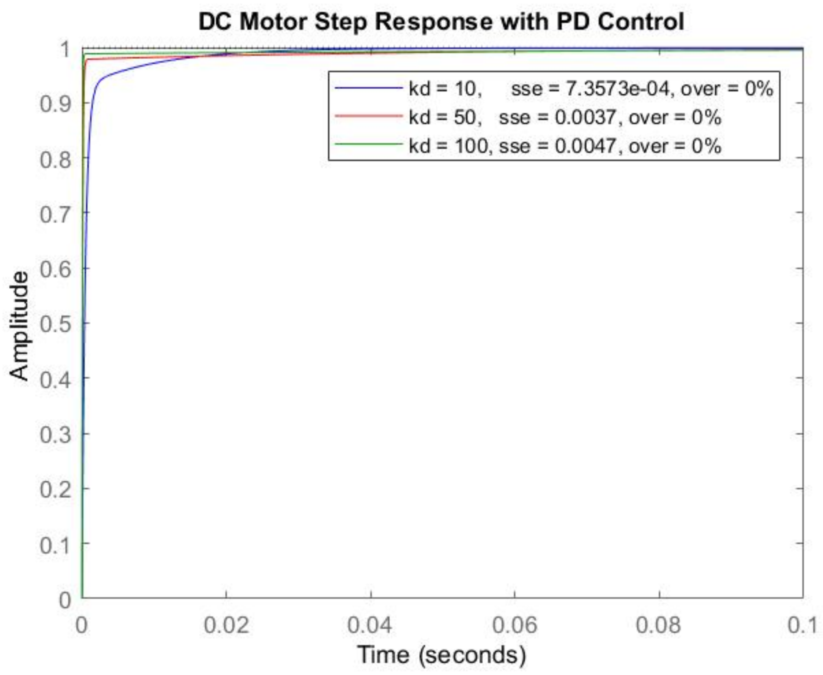

FIGURE 7. DC motor step response with proportional derivative control

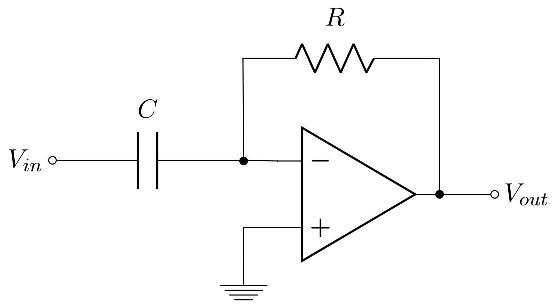

FIGURE 8. Op-amp differentiator where $V_{out}(t) = -RC\dfrac{dV_{in}(t)}{dt}$

is! This the time the derivative is of the input voltage instead of the output. Looking at the circuit we can deduce that:

$$\frac{dV_{in}}{dt} = \frac{-V_{out}}{RC}$$

and applying a laplace transformation grants us

$$\frac{V_{out}(s)}{V_{in}(s)} = -RCs$$

Subbing $-RCs$ for K_d lands us with the same transfer function in equation 17.

4.4. **Proportional Integral Derivative Control.** Finally, when we combine these concepts and use PID control, we get a combination of the quick reaction of the proportional controller, the reduction in steady state error from the PI controller, and the damping effect of the PD controller. The result, as shown in Figure 7, is a step response that approaches steady state within 0.025 seconds, has a steady state error of 1.4375e-06, and does not overshoot at all. In this idealized simulation in MATLAB, we settled on values of $K_p = 6000$, $K_i = 3600$, and $K_d = 100$, to achieve such a low steady state error and quick response time, however, when we apply this in analog circuitry we will find that we have to choose K constants that are within the voltage range of our circuit and within the range of available resistor and capacitor values.

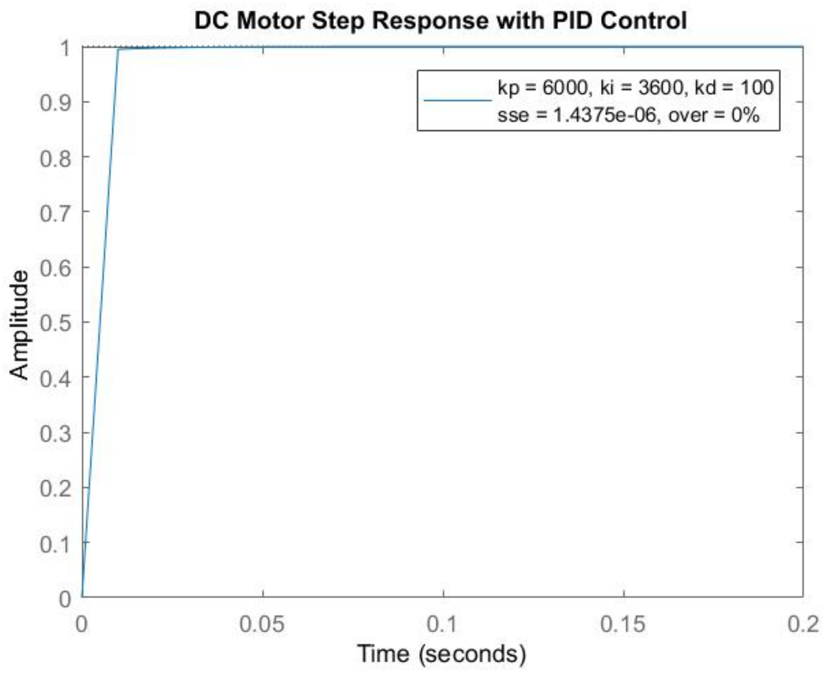

FIGURE 9. DC motor step response with proportional integral derivative control

The overall transfer function for PID control with the DC motor transfer function is as follows:

$$\frac{k\,(ki + s\,(kp + kd\,s))}{\left(K^2 + (b + Js)\,(Ls + R)\right)\,s + k\,(ki + s\,(kp + kd\,s))}$$

FIGURE 10. DC motor step response with proportional integral derivative control

The equivalent LTspice circuit for this would be:

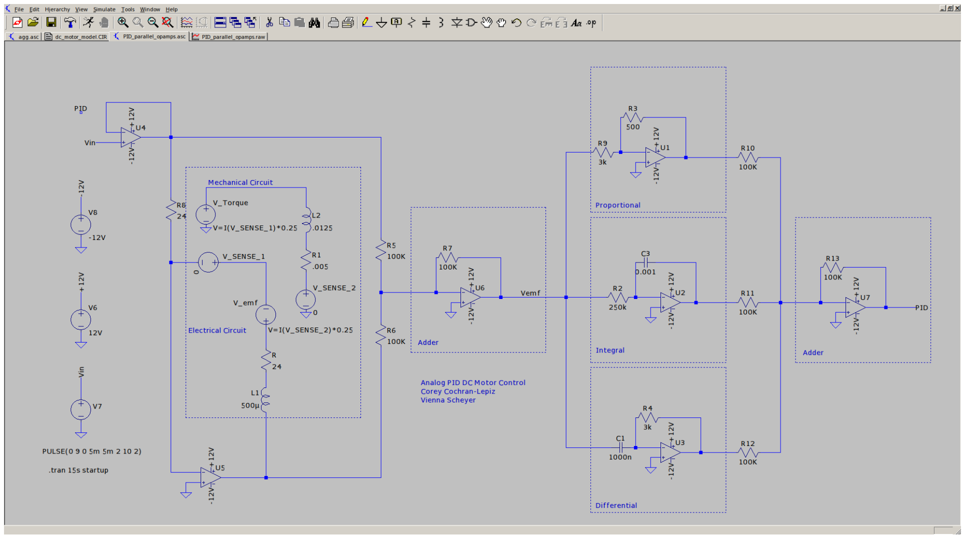

FIGURE 11. Analog PID LTspice circuit

5. CONCLUSION

We successfully used a motor model with reasonable coefficients and simulated PID control for that motor in MATLAB. We also showed how the individual parts of PID control (P, I, and D) directly correlate to op amp transfer functions in a controller circuit. we show that with the a set of K values we found our model reaches desired velocity in 0.025 seconds.

6. References

"AB-025: Using SPICE to Model DC Motors - Precision Microdrives." Accessed December 9, 2018. https://www.precisionmicrodrives.com/content/ab-025-using-spice-to-model-dc-motors/.

"Control Tutorials for MATLAB and Simulink - Motor Speed: System Modeling." Accessed December 9, 2018. http://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeedsection=SystemModeling.

"Emf Equation of a DC Generator." Circuit Globe, November 16, 2015. https://circuitglobe.com/emf-equation-of-dc-generator.html.

"LTSpice Circuit Simulation Tutorials for Beginners." ElectronicsBeliever (blog), November 19, 2014. http://electronicsbeliever.com/ltspice-circuit-simulation-tutorials-for-beginners/.

https://github.com/vscheyer/Analog_PID_MC