5

Negative Feedback Control

Introduction

Objectives

Upon completion of this chapter, you will be able to analyze and design:

-
- .

5.1 Negative Feedback

Things do *not* behave as required. That is true for friends, and pets, and power transistors, and op amps, and heaters, and motors, and cars, and aircraft, and … . The solution is shown in Figure 5.1

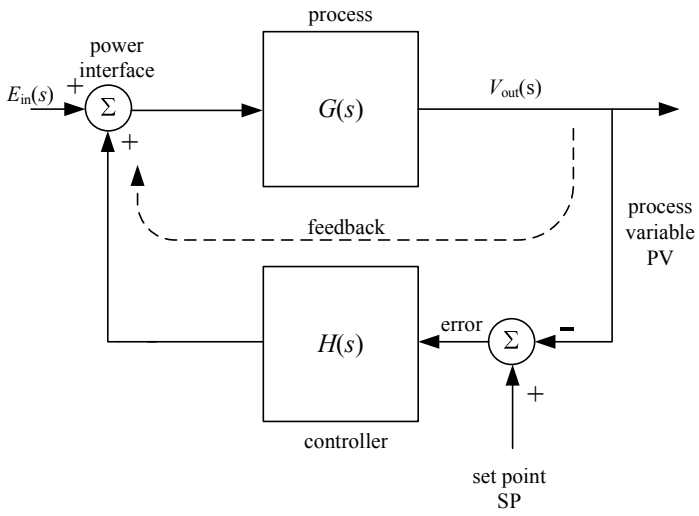


Figure 5-1 Negative feedback system

Energy is provided to the system; money to a friend, voltage to an op amp or motor, fuel to a car or aircraft engine. The power interface adjusts how much of that energy is fed into the system. The system then operates on, responds to, that energy input to produce an output. The relationship between the energy into the system and the system's output is defined by the

$$\text{system's transfer function} = G(s)$$

Unfortunately, that transfer function may not perfectly produce the desired result.

$$V_{out}(s) = \textit{process variable} = PV$$

That process variable is compared to the desired result, called the *set point*, *SP* by subtraction. The result is the *error*.

$$error = SP - PV$$

The goal is to correct the system so that the process variable tracks the set point, driving the error to zero. This is done with *negative feedback*. It is *negative* because the output (*PV*) is subtracted. That error is fed into a *controller* which uses an algorithm or calculation to decide how to reposition the power interface, closing the loop.

When taking a shower, once you have put your hand in the spray (*PV)* and compared it to how hot you want it (*SP),* you determine that the temperature is too low (*error* is negative). Depending on your algorithm, you then turn the hot water valve (power interface) some calculated distance that should produce the desired temperature. But this is a continuous process. As the temperature rises in response to the new hot water valve position, you note the new *PV*, calculate a new *error* and then compute how to readjust the hot water valve.

How well you enjoy the shower, to some degree, depends on just how good you calculate what to do to the hot water valve in order to drive the error to zero. In mathematical terms, that calculation is contained in the

transfer function for the controller = $H(s)$

Understanding the dynamics of your hot water heater and plumbing, *G(s)*, having a good scheme to adjust the valve, $H(s)$, and continuously monitoring *PV* allow you to have a great shower, even as the hot water tank starts to cool, or when someone opens a door letting in cold air.

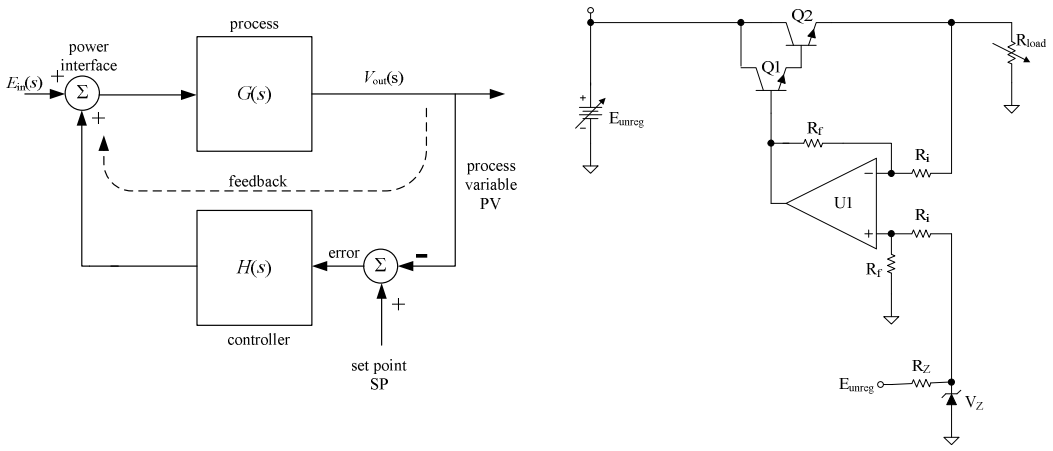


Figure 5-2 Linear power supply is a negative feedback control system

The electronic linear power supply in Figure 5-2 is a negative feedback system. The input energy is provided by E_{unreg}. It is passed to the main power transistor Q2 by Q1. So, Q1 is the power interface and Q2 is the process. Depending on how hard Q2 is turned on, current and voltage are passed to R_{load}. That voltage is the process variable, and is fed to the upper R_i

The zener diode and its supply and resistor produce a fixed, clean voltage, V_Z. That is the set point. The op amp with its two R_i and two R_f form both the error amp (summer in a circle) and the controller, $H(s)$. the output from the op amp is

$$V_{\text{op amp out}} = \frac{R_f}{R_i}(V_Z - V_{\text{load}})$$

If V_{load} is too small, then the output of the op amp increases, driving the transistor harder on, producing move output current and voltage. Conversely, if the input energy increases, V_{load} goes up. This lowers the difference $V_Z - V_{\text{load}}$, lowering the output back to the desired value.

Often it is difficult to alter the characteristics of the process (Q2 in Figure 5-2). But the transfer function of the controller is just it's gain.

$$H(s) = \frac{R_f}{R_i}$$

This allows the closed loop system to be tuned even when there are changes to the process, input energy or load.

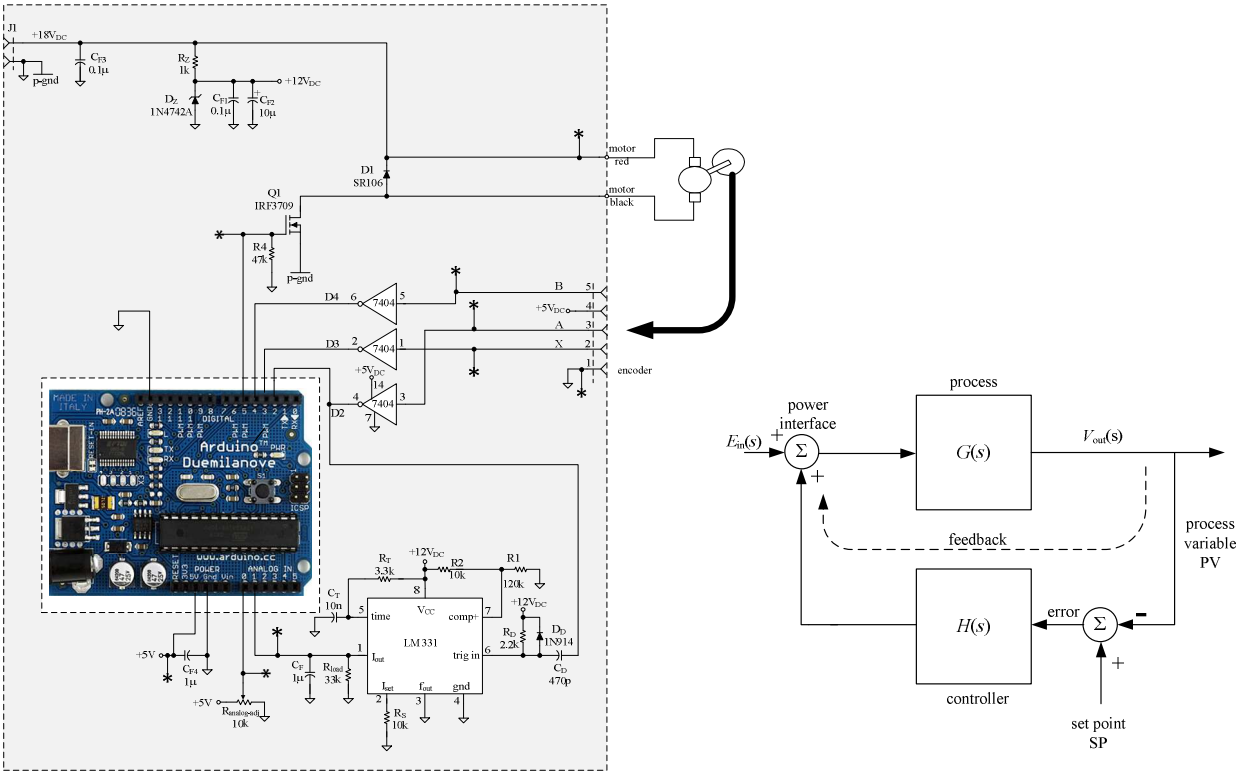


Figure 5-3 Microprocessor motor speed control

Figure 5-3 is a closed loop motor speed control system. Energy passes from an external power supply, through J1 and out to the red jack and the motor [the process. $H(s)$]. The MOS transistor, Q1, turns on to connect the motor to ground, making it spin. The motor is the process, and its first order transfer function is in Chapter 4.

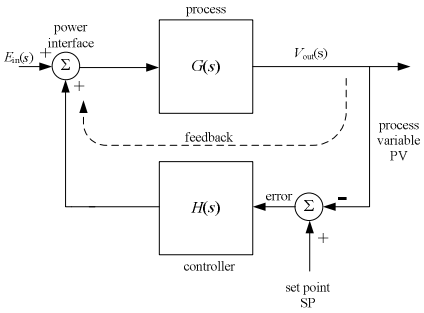
The motor's shaft spins a quadrature encoder that outputs pulses whose frequency is a measure of the shaft speed, the process variable. That frequency is buffered and then converted into a proportional voltage by the f-v IC LM311. Its output is an analog voltage representation of the process variable, and is read by the ADC in the microprocessor. Another ADC input reads the voltage at the wiper of $R_{analog\text{-}adj}$, the set point.

An algorithm within the microprocessor calculates $H(s)$, and may contain proportional, integral, and derivative terms. This variable then sets the pulse width driving the MOS.

If the motor is running too slowly, the frequency from the encoder is low, producing a low analog voltage into the microprocessor. Comparing this *PV* to a higher analog voltage *SP* from the potentiometer, the

microprocessor calculates a new, wider pulse width. The output turns the transistor on longer, providing more energy, causing the motor to speed up.

This all take time. Balancing the motor's time constant, with the filter in the f-v IC, the conversion and calculation delays and constants, and the frequency of the output pulse width are all critical. This is precisely why the Laplace functions of the motor, $G(s)$, and the program within the microprocessor, $H(s)$ are so important.



5.2 Process Regulation

The negative feedback control block considered so far has *two* inputs, $E_{input}(s)$ and $SP(s)$. Both may change' altering the output, $PV(s)$. In fact they interact. This gets far too complicated, as a starting point. So, the system's performance and design are considered in two separate cases.

Process regulation allows the input energy E_{in} to change, but the set point, *SP*, is held constant. In a well regulated system, a fixed set point produces a fixed process variable, *PV*, even when the input energy E_{in} changes. That is, the system alters its behavior in response to changing inputs to produce a constant output. That is precisely the purpose of voltage regulators, and HVAC systems, and even car cruise controllers.

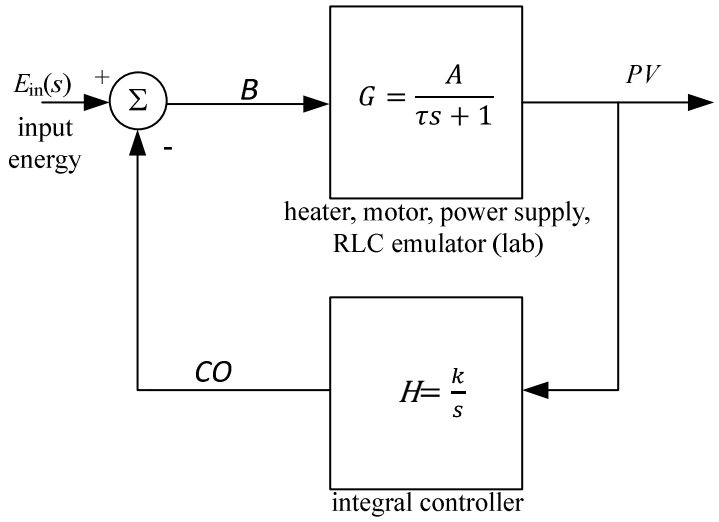


Figure 5-4 Process regulation block diagram

In Figur e5-4, the *SP* has been removed. It is a constant that is included in the calculation within the controller block, $H(s)$. To determine the overall system performance and transfer function requires a few steps of *algebra*.

$$transfer\ function = \frac{PV}{E_{in}}$$

Starting at the output, $PV = G \times B$

$B = E_{\text{in}} - CO$ where *CO* is the controller output

$$PV = G(E_{\text{in}} - CO)$$

$$CO = PV \times H$$

$$PV = G(E_{\text{in}} - PV \times H)$$

$$PV = GE_{\text{in}} - PV \times GH$$

$$PV + PV \times GH = GE_{\text{in}}$$

$$PV(1 + GH) = GE_{\text{in}}$$

$$\frac{PV}{E_{\text{in}}} = \frac{G}{(1 + GH)}$$

Process regulation

transfer function

Example 5-1

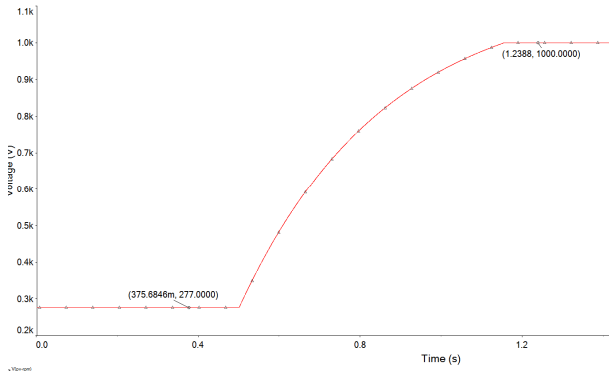
Determine the response of a motor to a step in input energy that would cause the motor running open loop to go from 277 RPM to 1000 RPM using $H = 10$.

a. Plot the results using Multisim.
b. What is the impact of increasing $H = 100$

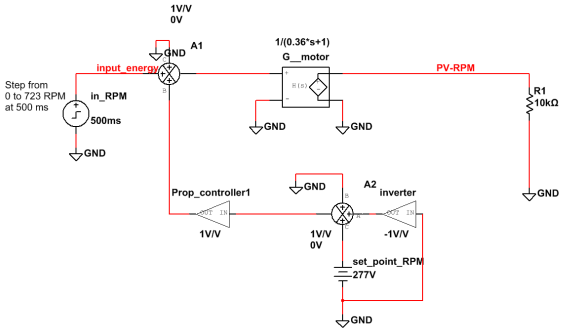
From Example 4-2

$m = 1$
$\tau = 0.362$ sec
$\omega_0 = 277$ RPM

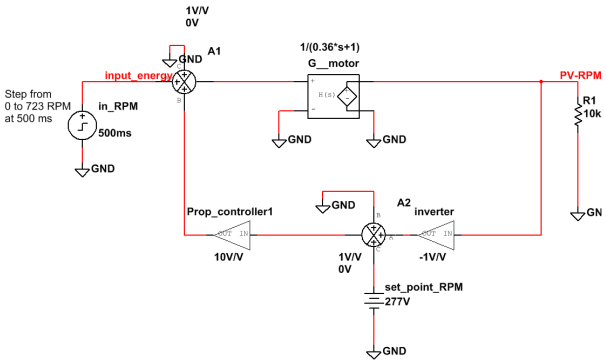
$$G = \frac{m}{(\tau s + 1)}$$



(a) Motor response without negative feedback



System without negative feedback



(b) Process control negative feedback with $H = 10$

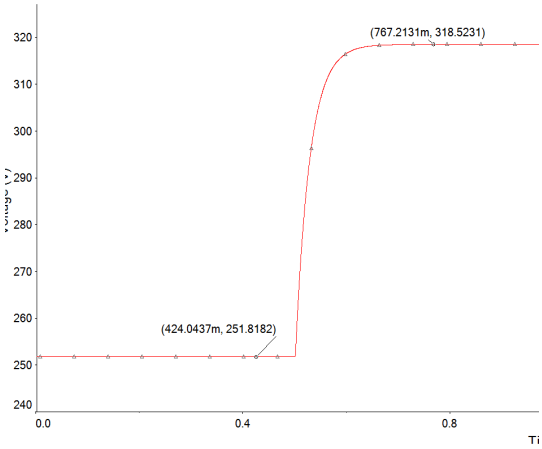


Figure 5-5 Motor regulation response to a step in input voltage

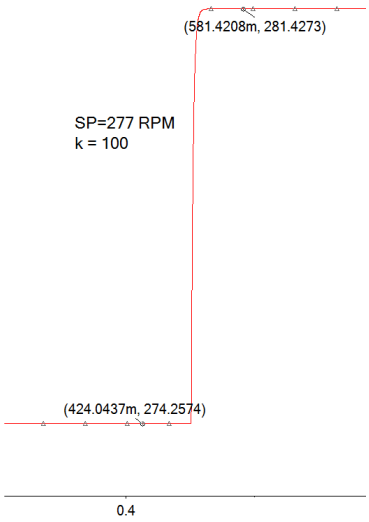
The objective of the controller is for the motor's speed to ignore changes in the input energy, and to continue to run at 277 RPM. The response of the motor alone is shown in Figure 5-5(a) The motor is given more energy and it spins faster, an expected but undesirable result.

The controller is added in Figures 5-5(b). The process variable, **PV-RPM**, is inverted and added to the 277 RPM **set_point_RPM**. This difference is the error and is sent to the proportional controller, $H = 10$. The controller's output is combined with the **input_energy** step and sent to drive the motor.

Before the step in input energy, the controlled motor is running a little slow, since PV subtracts a little from SP. Following the step in input energy (that sent the open loop system up 723 RPM to 1000 RPM),

the closed loop system only steps up to 318 RPM, and increase of only 1.5%. Increasing the controller, $H = 100$, improves performance before and after the step, and the step speed.

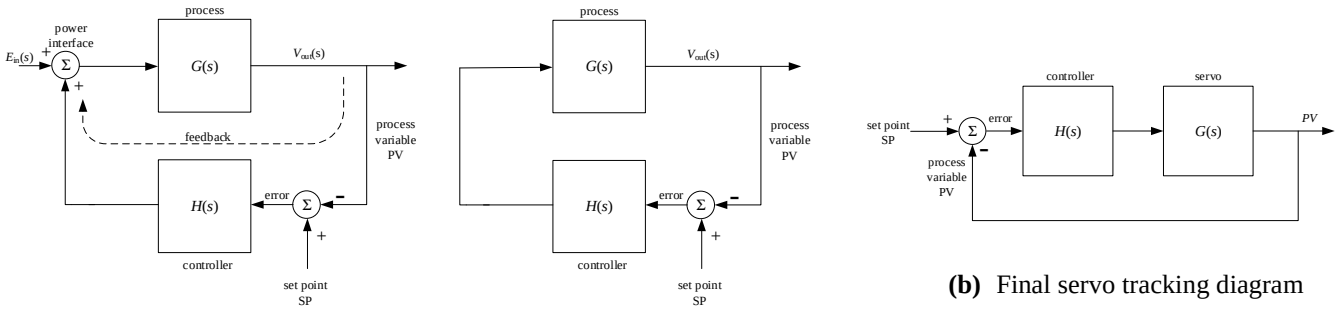
The motor speed *process* is indeed *regulated.*



5.3 Servo Tracking

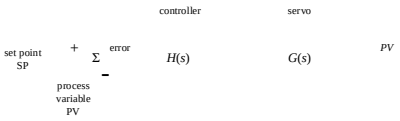
Process regulation assumes a fixed set point and tries to keep the output, *PV*, constant when the input energy changes. The alternative is *servo tracking*. The input energy is assumed to remain constant, and the desired output, as dictated by the *SP*, is driven to follow changes in that set point. This is how robotic manipulators and CNC tools are placed, how autonomous vehicles travel, and even how a rocket arrives at the international space station.

As with process regulation, the standard feedback diagram, Figure 5-6(a) may be simplified. Since the input is constant, changes into the system all come from the controller, Figure 5-6(b). It is traditional to place the driving force (the set point in servo tracking) to the left, and the output to the right. That final diagram is shown in Figure 5-6(c).



(a) Standard diagram **(c)** E_{in} is a constant **(b)** Final servo tracking diagram

Figure 5-6 Simplification of negative feedback to servo tracking control



This minor rearrangement makes a *huge* difference in the system's transfer function, and therefore its performance.

$$PV = GH \times error$$

$$error = SP - PV$$

$$PV = GH(SP - PV)$$

$$PV = GH \times SP - GH \times PV$$

$$PV + GH \times PV = GH \times SP$$

$$PV(1 + GH) = GH \times SP$$

Servo tracking transfer function

$$PV = \frac{GH}{(1 + GH)} SP$$

$$\frac{PV}{SP} = \frac{GH}{(1 + GH)}$$

To appreciate the impact of servo tracking control, below are the results of running the example motor open loop, without negative feedback (servo tracking) control.

From Example 4-2

m= 20577 RPM/V
τ = 0.362 sec
ω_0 = 277 RPM

E_{in} = 5/s a step

$$G = \frac{mV}{(\tau s + 1)} + \frac{\tau \omega_0}{(\tau s + 1)}$$

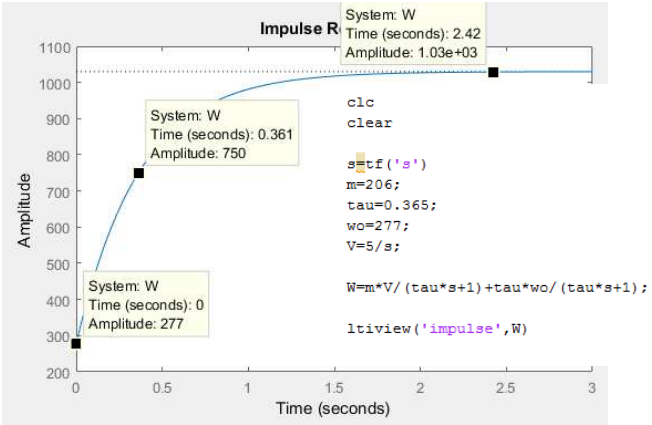
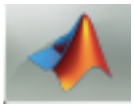
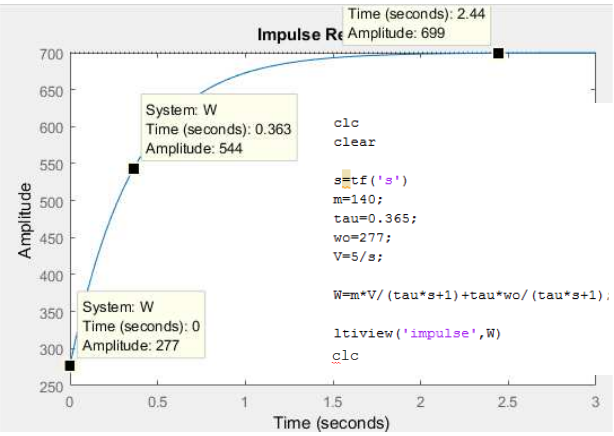


Figure 4-10 Motor speed response calculated and plotted with Matlab's ltiview

System element parameters change, because of manufacturing variations, from abuse, or fatigue. Figure 5-7 (a) shows the effect on the motor's performance if it's gain changes from 206 RPM/V to 140 RPM/V. Figure 5-7 (b) shows the effect of a longer time constant.





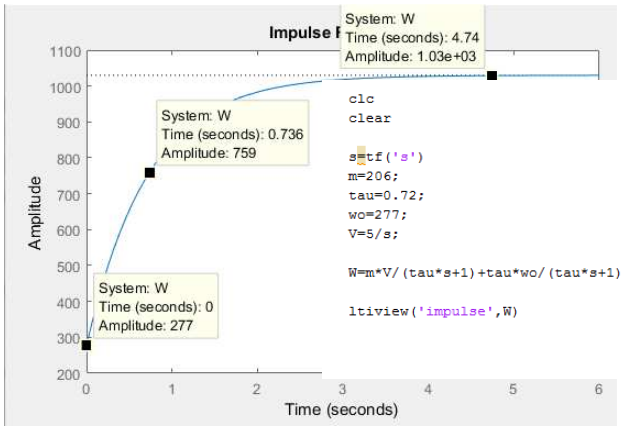


Figure 5-7 Effects of changes in motor parameters

As the motor ages, is abused, or is replaced, the system performance changes *drastically*, producing an entirely different speed for the same applied voltage, or taking considerably longer to settle.

Changing to servo tracking produces several changes. The Matlab script is shown in Figure 5-8. The motor's gain and time constant have been retuned to their original values. So, it is reasonable to expect performance similar to the original, shown in Figure 4-10. Since that original speed, *PV* was 1030 RPM with an initial ω_0 of 277 RPM, the set point, *SP* has been set to

$$SP = 1030 \text{ RPM} - 277 \text{ RPM} = 753 \text{ RPM}$$

In the Laplace domain a constant standing alone is divided by *s*. That also is treated as a step of 753 RPM at $t = 0$.

The motor's transfer function, *G* now, inistead of *W*, is driven by the output of the controller, and that is accounted for in the overall transfer function GH/(1+GH). So, *V* has been removed from the equation for *G*.

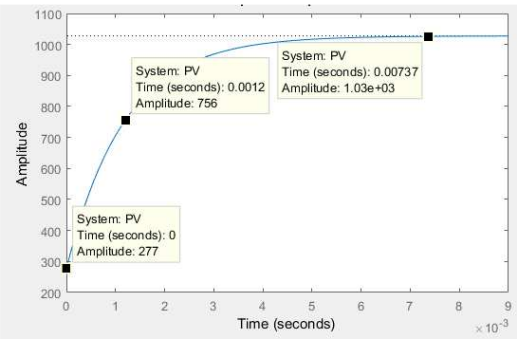
There are a host of controller calculations, even using integrals and derivatives. But to start with the control calculation is as simple as possible. $H = 1$.

$$PV = \frac{GH}{(1+GH)}SP$$

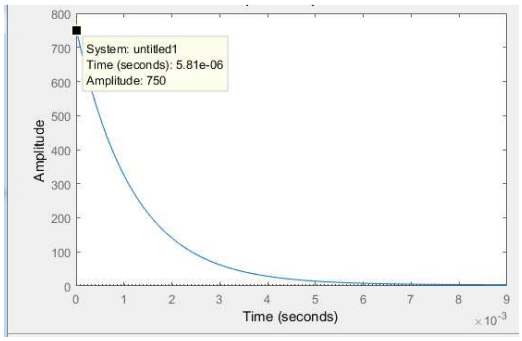
```
clc
clear

s=tf('s')
m=206;
tau=0.362;
SP=753/s;
wo=277;

%motor transfer function with initial speed
G=(m/(tau*s+1))+wo*tau/(tau*s+1)

H=1;        %controller

%output with initial speed
PV=wo/s+SP*G*H/(1+G*H)

ltiview('impulse',PV)
```

Figure 5-8 Motor servo tracking script

The initial



(a) Servo tracking – initial performance



(b) Servo tracking – controller output

speed 277/s has
been added to the *PV* equation.

Figure 5-9 Servo tracking – Matlab calculations

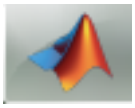


Figure 5-9(a) shows the initial performance, with th
speed, and producing the same final value. But, look at th
to 63% of the step height. For the open loop system in Figure 4-10, that took one time constant, $\tau = 360$ msec. The servo tracking plot in Figure 5-9(a) shows that the motor's speed rises to that value in only 1 msec, 360 times faster. Since the goal of the system is to move the motor speed to 1030 RPM, the faster the system rises the better – 360 times better!

How is that happening? It seems way too magical. Figure 5-9(b) plots the controller output.

```
CO=PV/G;
```

That plot shows that the controller output must jump to 750 V. That's not practical. Normal controllers will not go nearly that high, and normal motors will not tolerate that voltage.

This is a weakness of Matlab's `s=tf('s')`. It requires an unlimited, linear system and does not take into account the performance of system elements that max out. Instead, needed is a simulator that can handle Laplace functions *and* can limit component values to realistic levels.

With a little thought and trickery, Multisim can be made to handle both these tasks. Look at Figure 5-10.

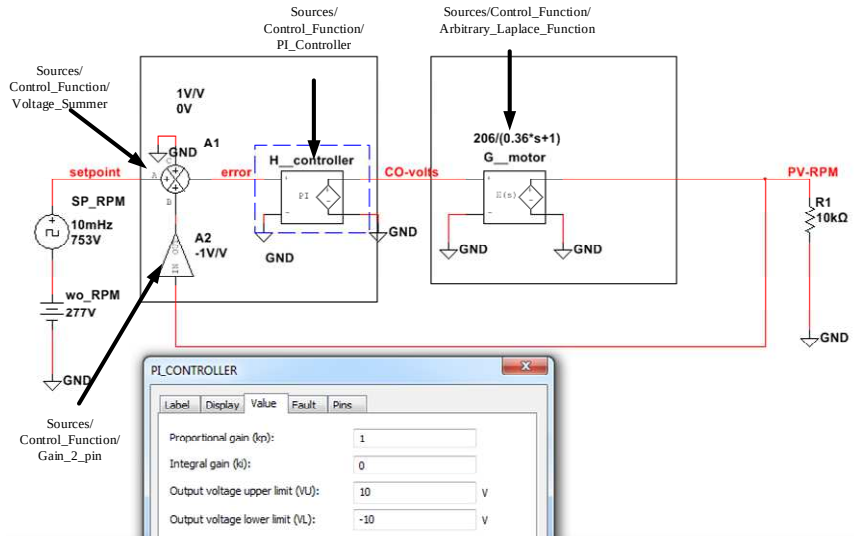


Figure 5-10 Multisim simulation schematic of motor servo tracking

The initial speed of 277 RPM is created as a DC source in series with the step of 753 RPM from the pulse generator. The subtraction of *SP-PV* to produce *error* is done with the two math blocks from the Source/Control_Function menu.

Although the controller gain, for now, is very simple, it is implemented with the PI_Controller block. Its dialog box is left open to show that this is where both the controller value = 1 (gain kp) and the output limits are set; ± 10 V is a reasonable value. As in Figure 4-5, the Laplace transfer function for the motor is created with an Arbitrary_Laplace_Function from the Sources/Control_Function menu as well.

The results from Multisim are not as spectacular as those offered by Matlab. At $t = 0$, the PV = 275 RPM (expected 277 RPM). SP has just jumped up 753 RPM to 1030 RPM. That means that the error is

$$error = SP - PV = 1030 \text{ RPM} - 277 \text{ RPM} = 753 \text{ RPM}$$

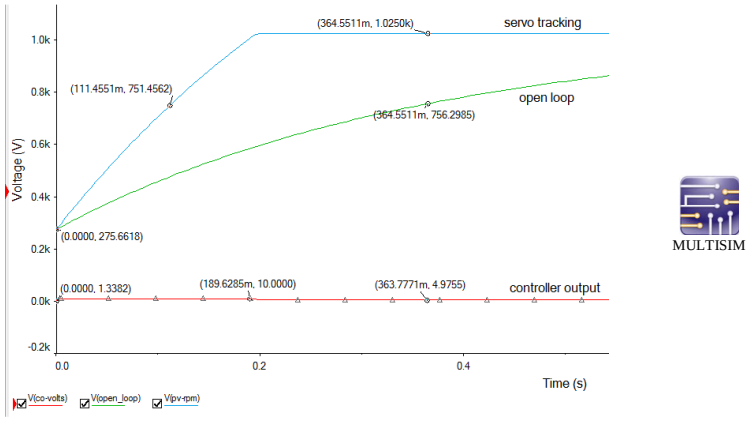


Figure 5-11 Multisim controller output and motor speed with servo tracking

With H = 1, the controller output calculates to 753 V. But the controller block limits it to 10 V, just as real controller hardware would be sent to its maximum. This continues for the first several hundred msec. The final speed reachs the 1025 RPM predicted, with the controller's output at 4.98 V. Since the motor is driven by a constant 10 V (rather than 5 V), the time to 63% is 111 msec, much less than the open loop motor's τ = 360 msec.

Servo tracking also drastically narrows performance variation as the motor's parameter degrade, Figure 5-12.

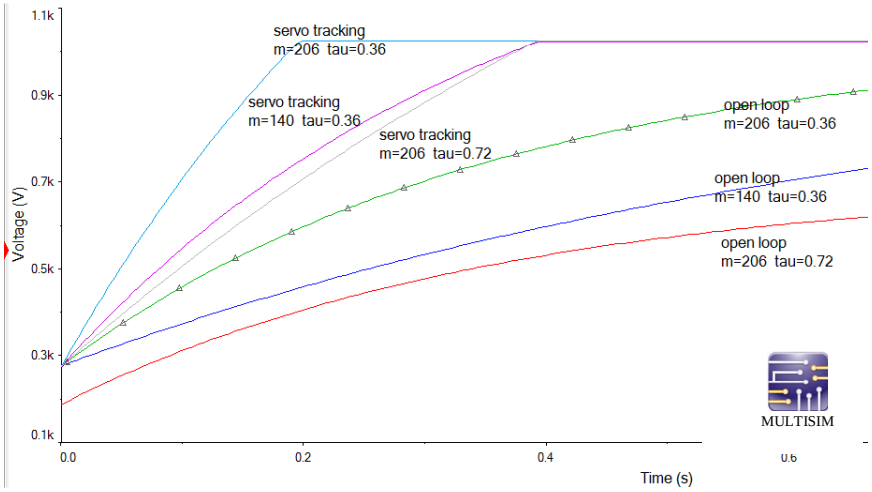


Figure 5-12 Effects of servo tracking on motor parameter variation