Speed Control of a DC Servo Motor

ECE 476 Advanced Embedded Systems

Jake Glower - Lecture #18

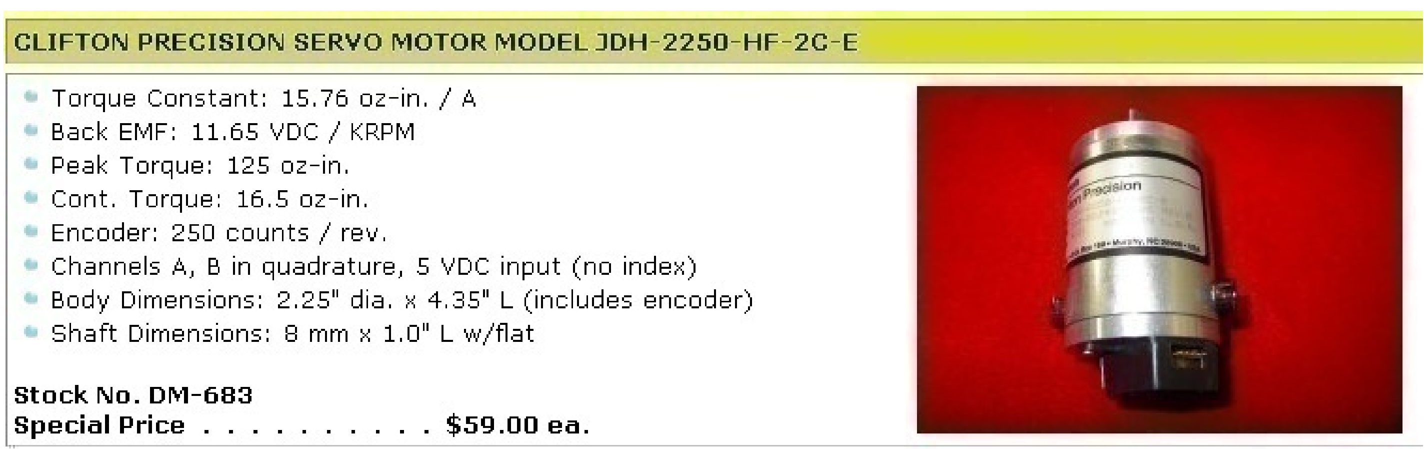
Please visit Bison Academy for corresponding lecture notes, homework sets, and solutions

Introduction:

Once you have edge interrupts and timer interrupts, you are able to control the speed and angle of a DC servo motor. In this lecture, we'll look at

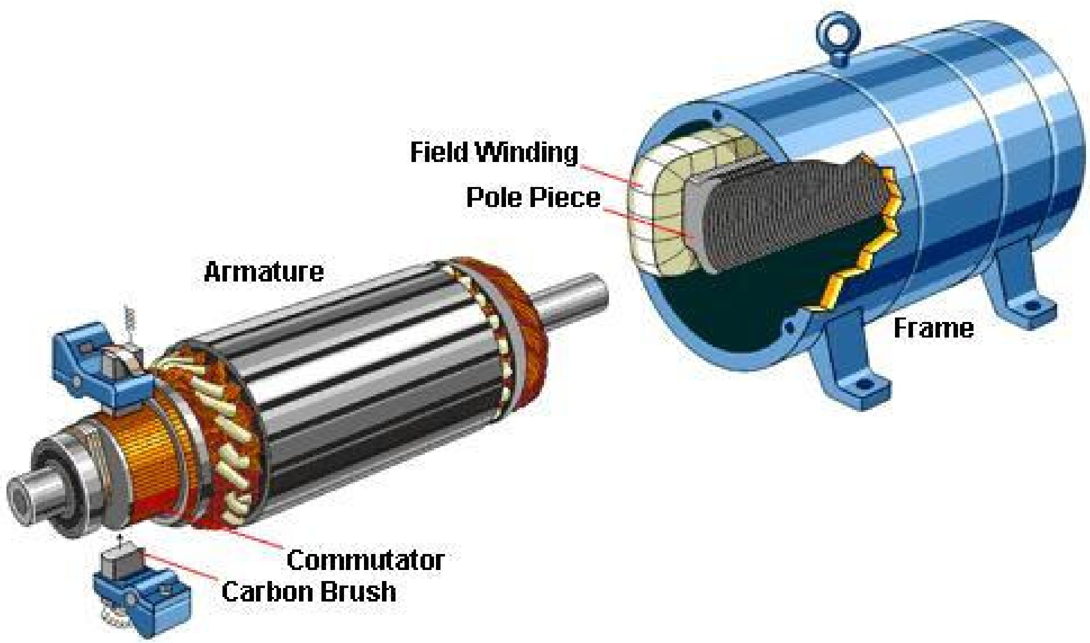
- Modeling a DC servo motor
- Measuring a motor's speed using an optical encoder
- Modeling a DC servo motor, and
- Controlling the speed of a DC servo motor using P and PI compensators (more on this later)

The motor used in this lecture is a Clifton 000-053479-002 DC Servo Motor - but the material herein applies to pretty much any DC motor.



Theoretical Modeling a DC Servo Motor

Typically, a DC servo motor has permanent magnets attached to its frame. The armature then contains several electromagnets connected to external wires through a commutator.



Typical DC Servo motor construction

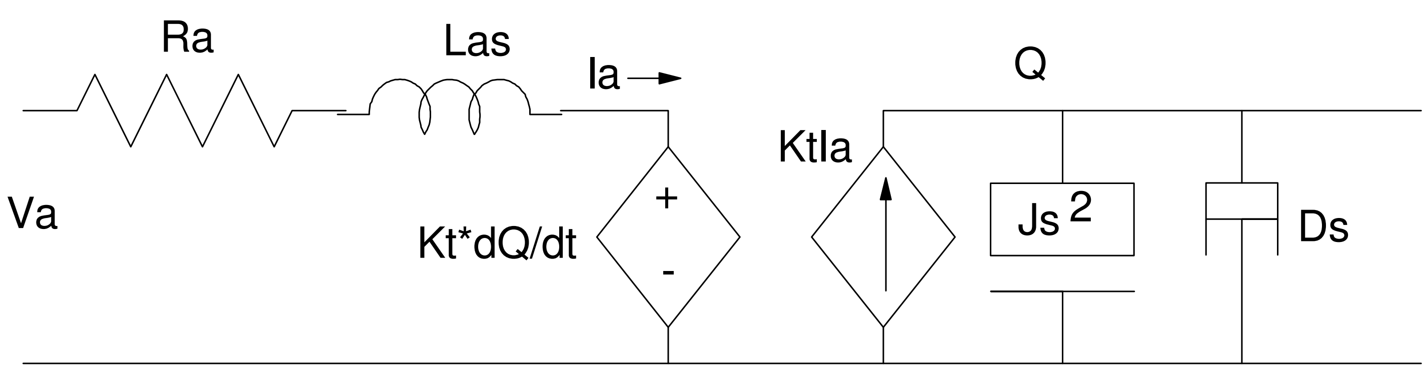
Model for a DC Motor

Motors and geneators are one and the same. model reflects this:

- The left side models the electrical side of the motor
  - The resistance and inductance corresponding to the electromagnetics in the armature
- The right side models the mechanical side of the motor
  - Torque spinning a mechanical system with inertia and friction

Coupling these together are a motor / generator pair

- As you apply current to the motor, it produces torque: $T = k_t I_a$
- As you spin the motor, it produces voltage: $E = k_t \omega$



The resulting equations for a DC servo motor are then

$$V_a = (L_a s + R_a) I_a + k_t \omega$$

$$k_t I_a = (Js + D)\omega$$

where

- (La, Ra) are the inductance and resistance of the armature
- (J, D) are the inertial and friction associated with the armature,
- w is the motor's speed,
- kt is the motor's torque constant, and
- (Va, Ia) are the voltage and current at the armature

Solving for speed as a function of voltage, you get the transfer function

$$\omega = \left(\frac{k_t}{(Js+D)(Ls+R)+k_t^2} \right) V_a$$

If L = 0 (usually a good assumption)

$$\omega \approx \left(\frac{k_t}{(Js+D)R+k_t^2} \right) V_a$$

meaning the motor behaves as a 1st-order dynamic system.

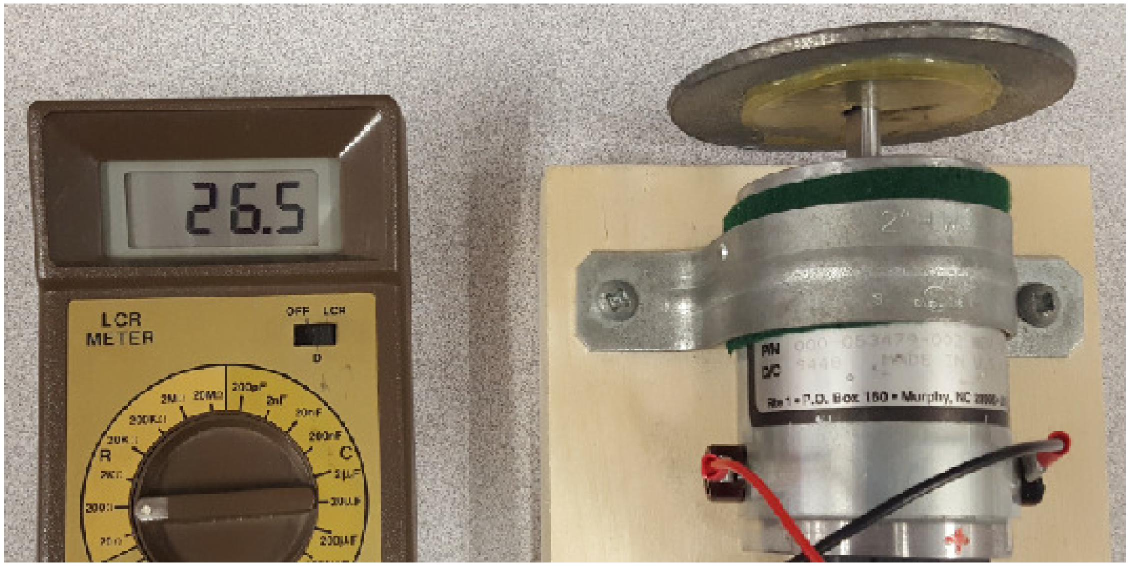
Finding the Transfer Function for a DC Motor

The transfer function for a DC motor has five parameters:

- Ra, La: Armature resistance and inductance
- J, D: Rotor inertia and friction
- Kt: Torque constant

One way to find this transfer function is to plug in the parameters

- Some of these are easy to measure.
  - Ra and La can be measured with an RLC meter
  - Ra = 26.5 Ohms, La = 12.689mH



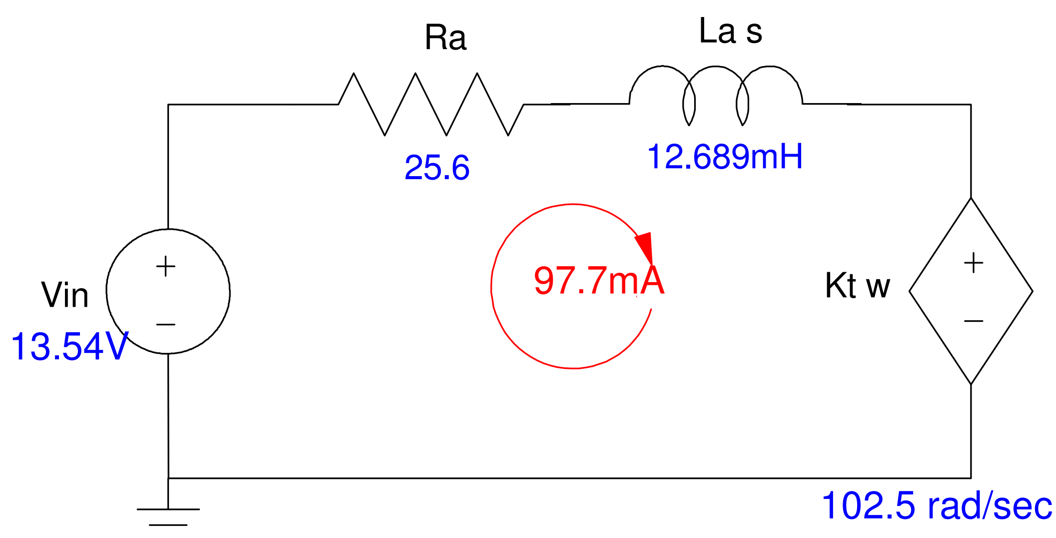
Meauring the torque constant:

- Apply a voltage and let the motor spin freely.
- Measure the voltage, current, and resulting speed.
- Fom this, you can compute the torque constant

$$V_{in} = R_a I_a + k_t \omega$$

$$13.54V = 26.5\Omega \cdot 97.7mA + k_t \cdot 102.5\frac{rad}{\sec}$$

$$k_t = 0.1067\frac{Vs}{rad} = 0.1068\frac{Nm}{A}$$



Measuring Friction

Measure the no-load speed and current draw.

- Do an energy balance.
- On the electrical side, power in is equal to volts times amps.
- On the mechanical side, power out is equal to torque time speed

Conservation of energy requires these be the same

$$P_{in} = P_{out}$$

$$V_a \cdot I_a = I_a^2 R_a + T \cdot \omega$$

The no-load torque in this case corresponds to the friction in the motor

$$V_a \cdot I_a = I_a^2 R_a + (D\omega) \cdot \omega$$

Plugging in numbers:

$$13.54V \cdot 97.7mA = (97.7mA)^2 \cdot 26.5\Omega + D \cdot \left(102.5\frac{rad}{\sec}\right)^2$$

$$D = 0.0001018\frac{Nm}{rad/\sec}$$

Measuring the rotor's inertia

Finally, the motor's rotational inertia (J) can be estimated as a flywheel

- 91mm dia x 2.5mm thick flywheel, solid iron

$$m = \left(7.847\frac{gm}{cc}\right)\left(\pi \cdot (4.55cm)^2\right)(0.25cm) = 127.5gm = 0.1275kg$$

$$J = \frac{1}{2}mr^2 = \frac{1}{2}(0.1275kg)(0.0455m)^2 = 0.000132\,kg\,m^3$$

Putting it all together, a model for the DC motor is:

$$\omega = \left(\frac{k_t}{(Js+D)(Ls+R)+k_t^2}\right)V_a$$

$$\omega = \left(\frac{65{,}658}{(s+4.032)(s+2084)}\right)V_a$$

Note: This model has a fast pole

- (s + 2084)
- The electrical time constant

and a slow pole

- (s + 4.032)
- The mechanical time constant

If you ignore the fast pole you get a simpler model

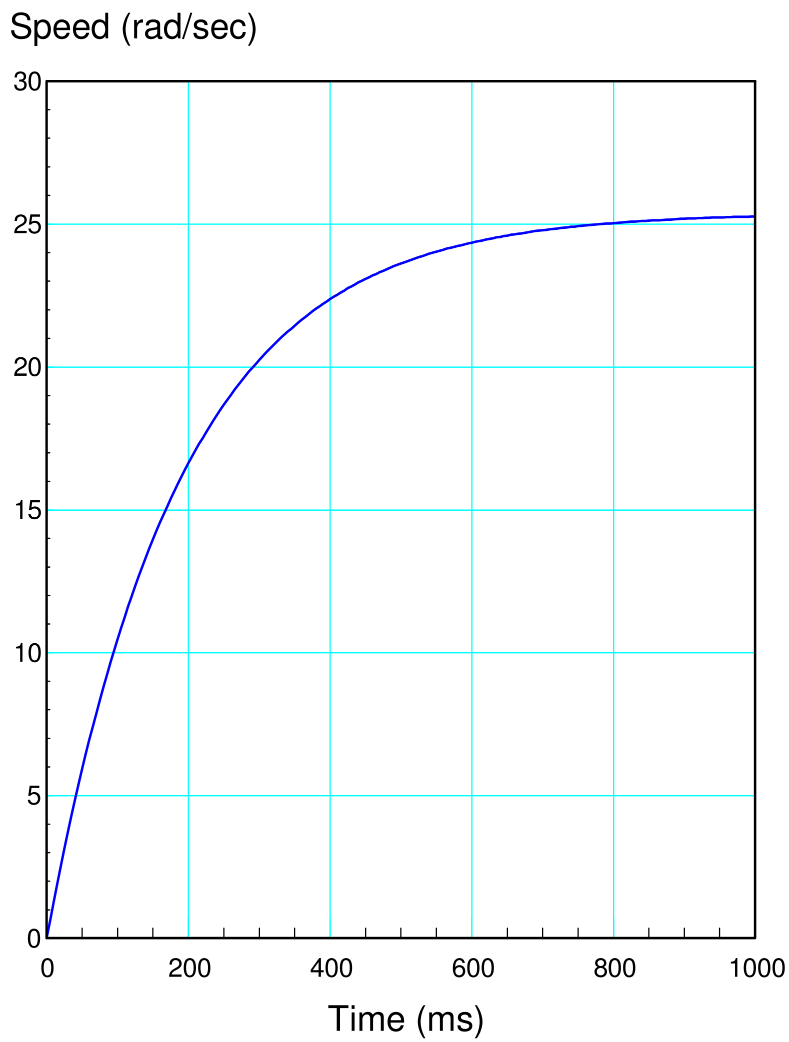
- L = 0 (assume)
- $\omega = \left(\frac{35.51}{s+4.032}\right)\left(\frac{2084}{s+2084}\right) V_a$
- $\omega \approx \left(\frac{35.51}{s+4.032}\right) V_a$

Translation: the step response should be

- A 1st-order system with
- A DC gain of 7.81, and
- A time constant of 248ms (transient decays as exp(-4.032t)

In Matlab, this can be found using

```
>> G = zpk([],-4.032,35.51);
>> t = [0:0.01:1]';
>> w = step(G,t);
>> plot(t,w*10)
```



We can verify this by measuring the step response of the actual DC servo motor.

To do this, however, we first need to be able to measure the motor's speed. For this, optical encoders are useful.

Optical Encoders

Optical encoders output

- Two square waves
- 90 degrees apart
  - phase quadrature

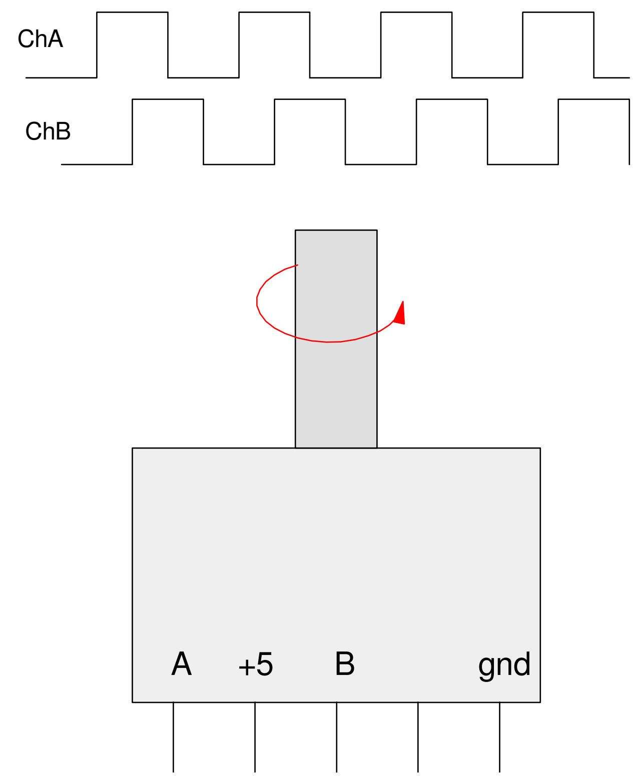
When attached to a motor, they allow you to measure the motor's

- Angle: Count the number of pulses
- Speed: Measure pulses per second

With the motor used in this lecture, the encoder has

- 250 pulses per rotation, or
- 1000 counts per rotation

if you count both rising and falling edges.



Example: Measure the motor's angle

- Count rising edges on channel A (250 pulses per rotation)
- Display the counts (angle) on the console

```
from machine import Pin
from time import sleep

pin1 = Pin(26,Pin.IN,Pin.PULL_UP)
pin2 = Pin(27,Pin.IN,Pin.PULL_UP)

N1 = 0

def ChanA(pin1):
    global pin2
    global N1
    if(pin2.value() == 1)):
        N1 += 1
    else:
        N1 -= 1

pin1.irq(trigger=Pin.Pin.IRQ_RISING, handler=ChanA)

while(1)
    print(N1)
    sleep(0.1)
```

Python code for reading the optical encoders to measure the motor's angle

Example: Measure the motor's speed

- Sample every 50ms (20Hz).
- Speed is then

$$rad = \left(\frac{2\pi}{250}\right) \cdot N_1$$

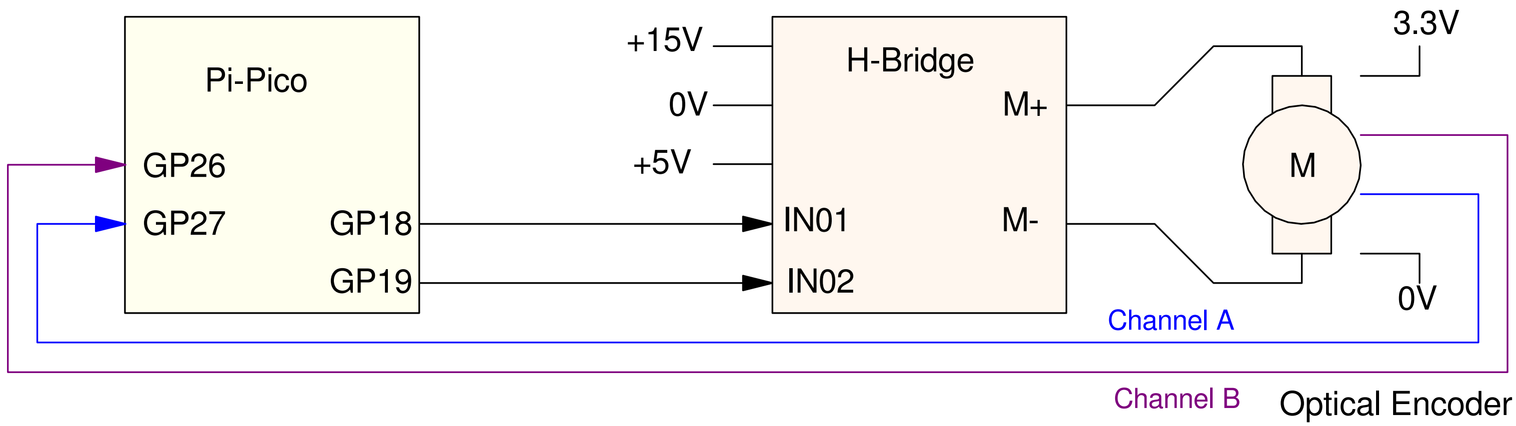
$$\frac{rad}{\sec} = \frac{rad}{0.05} = (0.16\pi) \cdot \delta N_1$$

where δN_1 is the number of counts in the last 50ms (20Hz). In code

```
N1 = N2 = N12 = 0

def tick(timer):
    global N1, N2, N12
    X = N1
    N12 = N1 - N2
    N2 = X

tim = Timer()
tim.init(freq=20, mode=Timer.PERIODIC,
callback=tick)

while(1)
    Speed = 0.16*pi*N12
    print(Speed)
    sleep(0.1)
```

Experimental Modeling of a DC Motor

Once you can measure the speed of the motor, the step response can be recorded. To drive the motor, an H-bridge is used:



GP18 and GP19 serve as the direction control:

- GP18=1, GP19=0: 100% forward (+13.4V applied to the motor)
- GP18=0, GP19=1: 100% reverse (-13.4V applied to the motor)

Note:

- The actual votlage to the motor is slightly less than +15V
- The H-bridge has a small volage drop across it's elements (1.6V apparently)

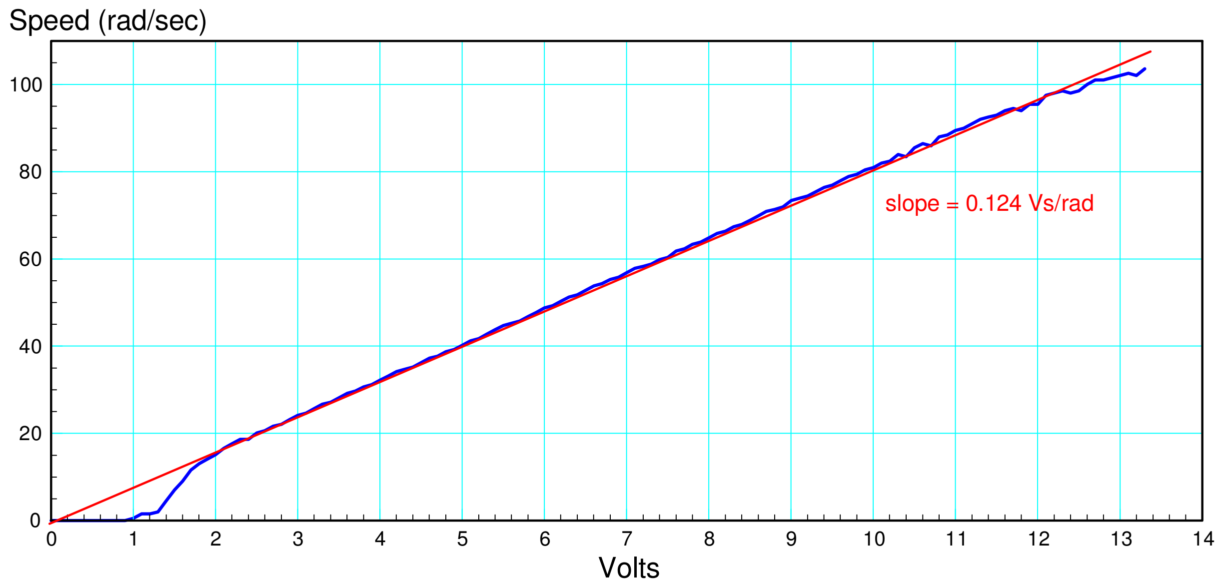
The steady-state relationship between speed and voltage can be found by

- Stepping the input voltage from 0V to 13.4V (0% to 100% PWM),
- Waiting for the speed to settle out (steady state), and
- Recording the resulting speed for each voltage.

```
kv = 65535 / 13.4        # conversion from Volts to PWM
kw = 0.16*pi             # conversion from counts to rad/sec

V = 0

while(V < 13.4):
    fwd.duty_u16(int(V*kv))
    rev.duty_u16(0)
    sleep(0.5)
    Speed = N12*kw
    print('{: 7.4f}'.format(V), '{: 7.4f}'.format(Speed), N12)
    V += 0.1

print('Stop')
fwd.duty_u16(0)
rev.duty_u16(0)
```

Python code for applying a constant voltage to the motor and measuring the resulting speed

The resulting speed vs. voltage is shown below:

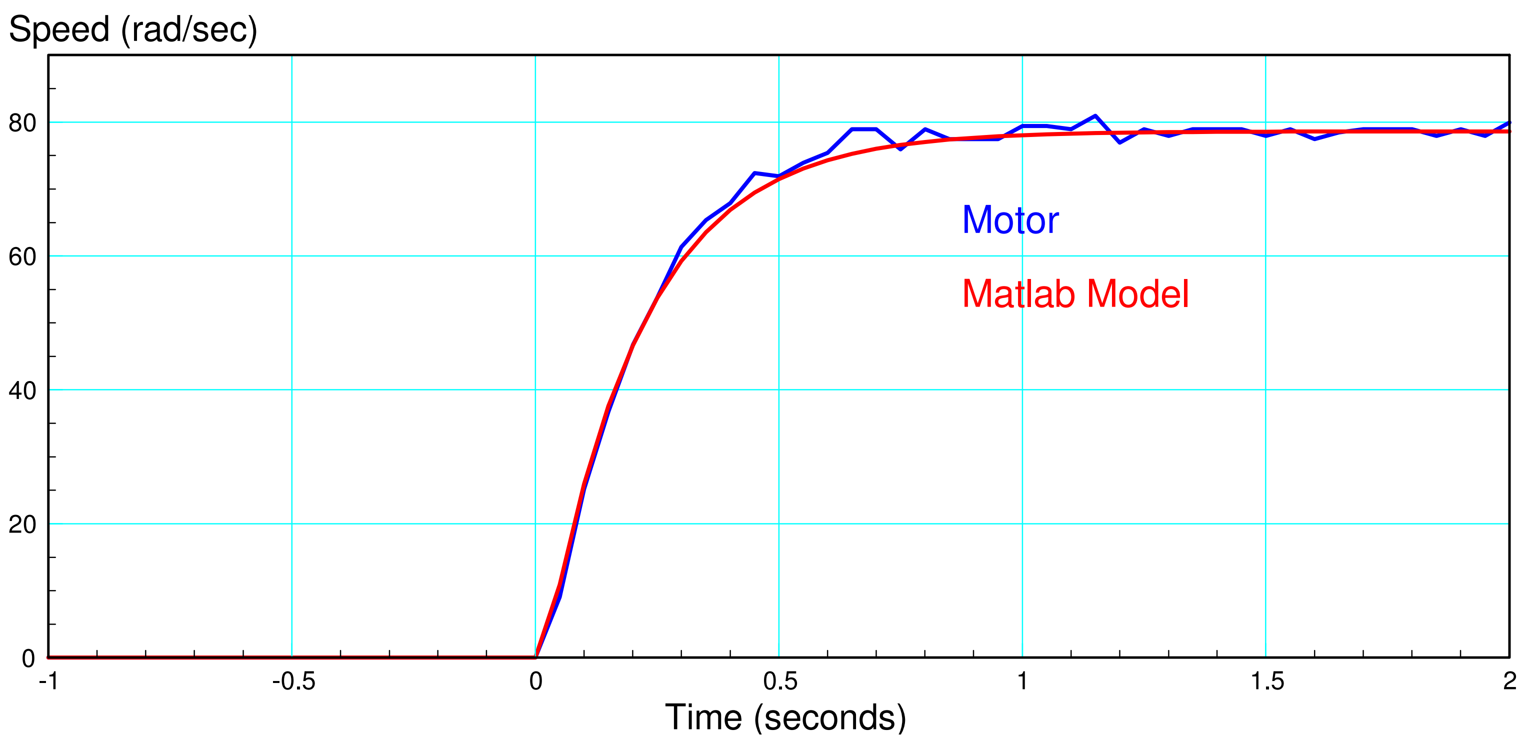


No-Load Speed vs. input voltage

What this shows is

- At low speeds, static friction prevents the motor from spinning. It takes at least 1V (ish) to overcome static friction.
- Past 2V, the motor's speed is pretty much proportional to voltage
- The slope is essentially the torque constant, kt (minus losses

A step response of the motor can be found by applying a step input to the voltage. Measuring the response to a 10V step input results in the following graph:



Response to a 10V step input at t=0 (blue) and 1st-order model (red)

Using trial an error, a 1st-order response to match the data was found using Matlab (shown in red) with the final approximate model being

```
>> t = [0:0.01:2]';
>> y = 78.6*(1 - exp(-5*t));
>> plot(t,y)
```

This implies the transfer function of the motor is

$$\omega \approx \left(\frac{39.3}{s+5}\right) V$$

which comes from

- The DC gain of the motor is 78.6 / 10 (10V input), and
- The pole is at s = -5

This is actually fairly close to the theoretical transfer function

```
def tick(timer):
    global N1, N2, N12, flag
    X = N1
    N12 = N1 - N2
    N2 = X
    flag = 1

tim = Timer()
tim.init(freq=20, mode=Timer.PERIODIC, callback=tick)

t = -1
dt = 1/20
kv = 65535 / 13.4   # convert volts to pwm
kw = 0.16*pi        # convert counts to rad/sec

fwd.duty_u16(0)
rev.duty_u16(0)

while(t < 2):
    while(flag == 0):
        pass
    flag = 0
    if(t < 0):
        V = 0
    else:
        V = 10
    fwd.duty_u16(int(V*kv))
    rev.duty_u16(0)
    Speed = N12*kw
    print('{: 7.2f}'.format(t),'{: 7.4f}'.format(Speed))
    t += dt
```

Python code for finding the step response of the motor.

Feedback Control

Typically, in order to control the speed of a motor, feedback is used. This allows you to specify the desired speed (Ref) with the feedback system automatically figuring out what voltage you need to apply to maintain speed (termed automatic control).

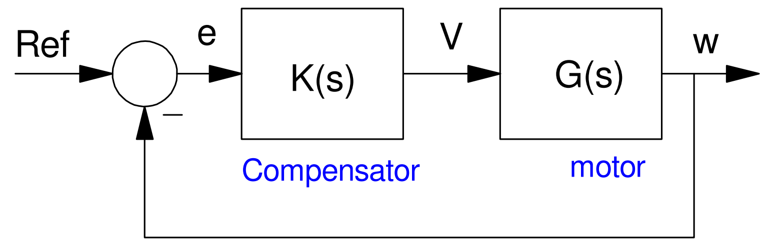
The transfer function from the input (Ref) to the speed (w) is then

$$\omega = GKe$$

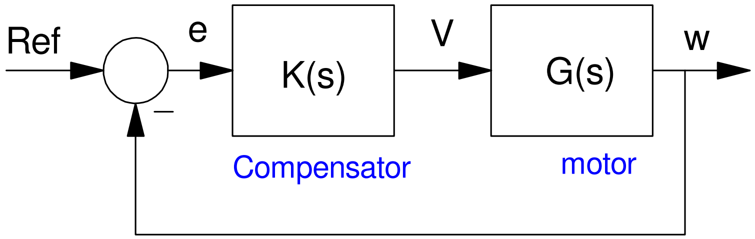
$$e = Ref - \omega$$

or after simplifying

$$\omega = \left(\frac{GK}{1+GK}\right) Ref$$



PID Control



K(s) can take on several forms.

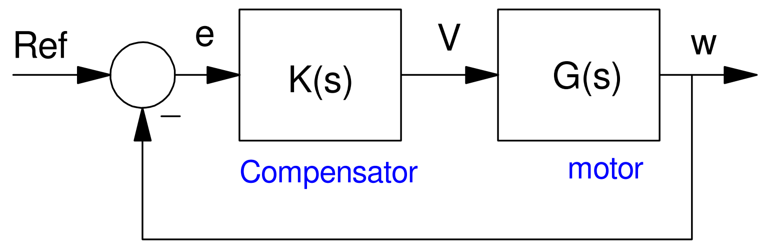
- One common type of controller is called a PID controller.

  $$K(s) = P + \frac{I}{s} + Ds$$

- I: The I term adds integration to the control law. The integrator's job is to search for the constant needed to hold the output at a desired speed..
- P: The P term helps to speed up the motor by canceling a pole which slows the system down.
- D: The D term allows you to cancel a second pole, speeding up the system even more.

Here, we'll look at implementing an I and a PI conroller.

I Control: K(s) = k/s



First, assume K(s) is of the form

$$K(s) = \left(\frac{I}{s}\right) = \left(\frac{k}{s}\right)$$

Substituting:

$$\omega = \left(\frac{GK}{1+GK}\right)R$$

$$\omega = \left(\frac{\left(\frac{39.5}{s+5}\right)\left(\frac{k}{s}\right)}{1+\left(\frac{39.5}{s+5}\right)\left(\frac{k}{s}\right)}\right)R$$

$$\omega = \left(\frac{39.5k}{s(s+5)+39.5k}\right)R$$

Note that the DC gain is always 1.000. This is a property of using in integrator in the control law (the integrator searches to find the constant which forces the error to zero. Once found, the integrator stops searching and outputs a constant V (the integration constant.)

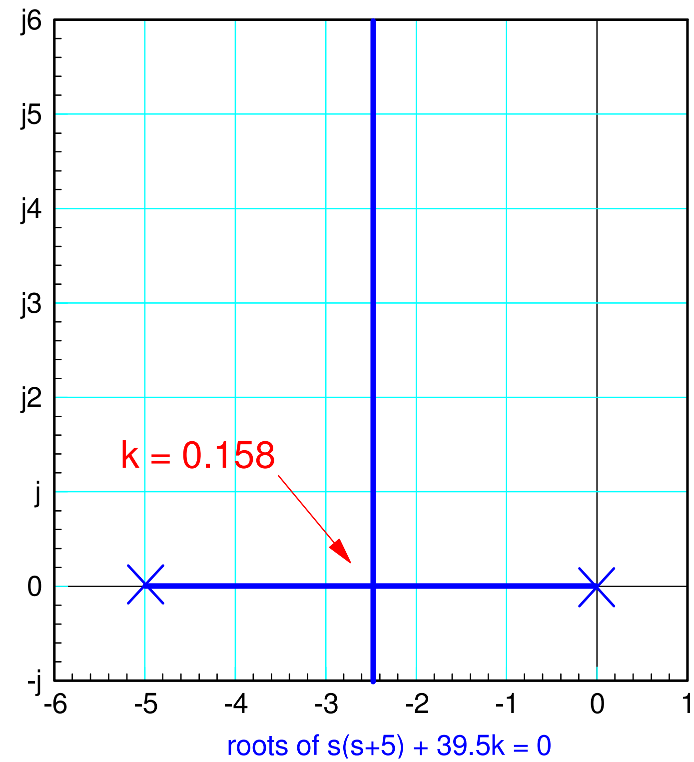
k determines where the roots of the closed-loop system are:

$$s(s+5) + 39.5k = 0$$

If you want repeated poles at s = -2.5, then

$$(s(s+5) + 39.5k)_{s=-2.5} = 0$$

$$k = \left(\frac{2.5^2}{39.5}\right) = 0.158$$



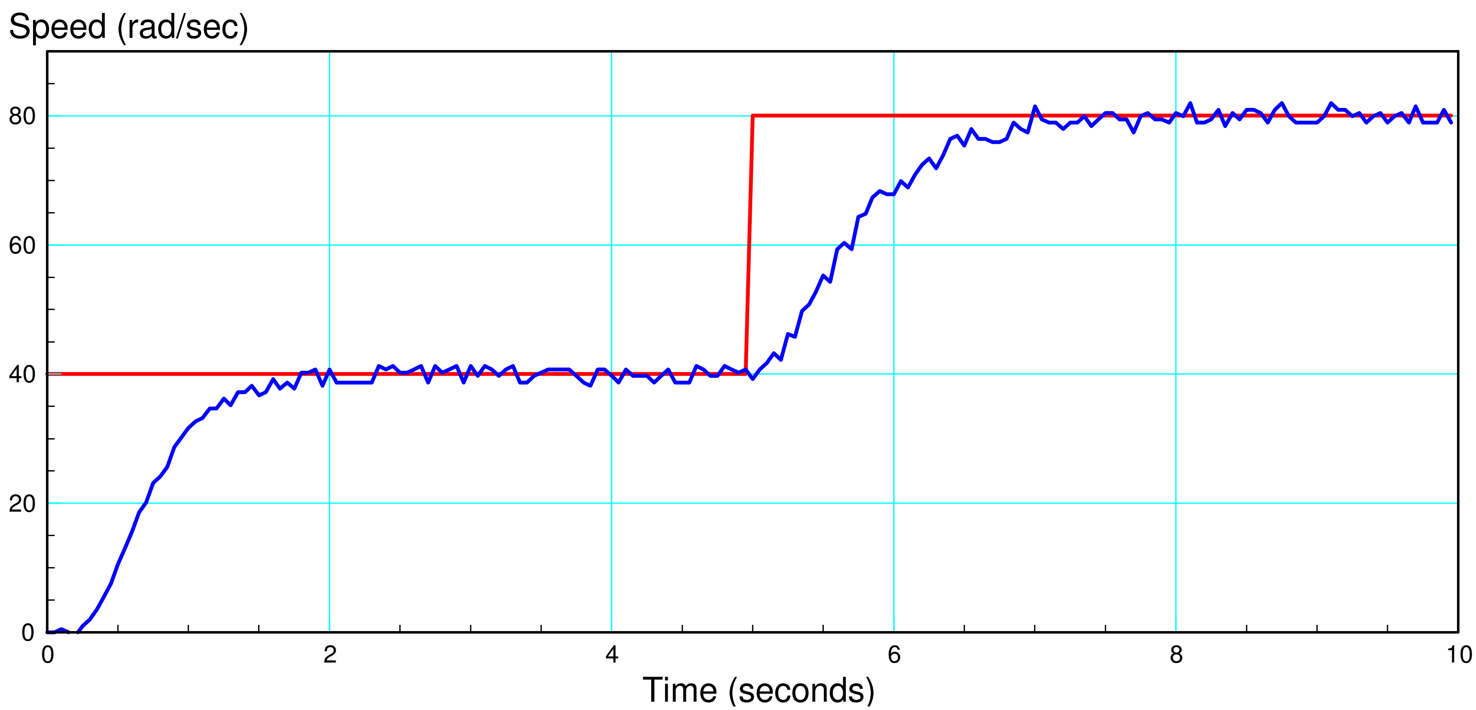
The closed-loop system is then

$$K(s) = \left(\frac{0.158}{s}\right)$$

$$\omega = \left(\frac{6.25}{s^2+5s+6.25}\right)R = \left(\frac{2.5}{s+2.5}\right)^2 R$$

The step response of this motor is shown below. Note:

- The actual speed locks onto the desired speed (due to the integrator)
- It takes about 2.0 seconds to lock onto this speed
- The timer interrupt is used to set the loop time to 50ms (20Hz).



The 2.00 seconds is due to the closed-loop poles being at {-2.5, -2.5}. With repeated poles at s = -2.5, the transient should decay as

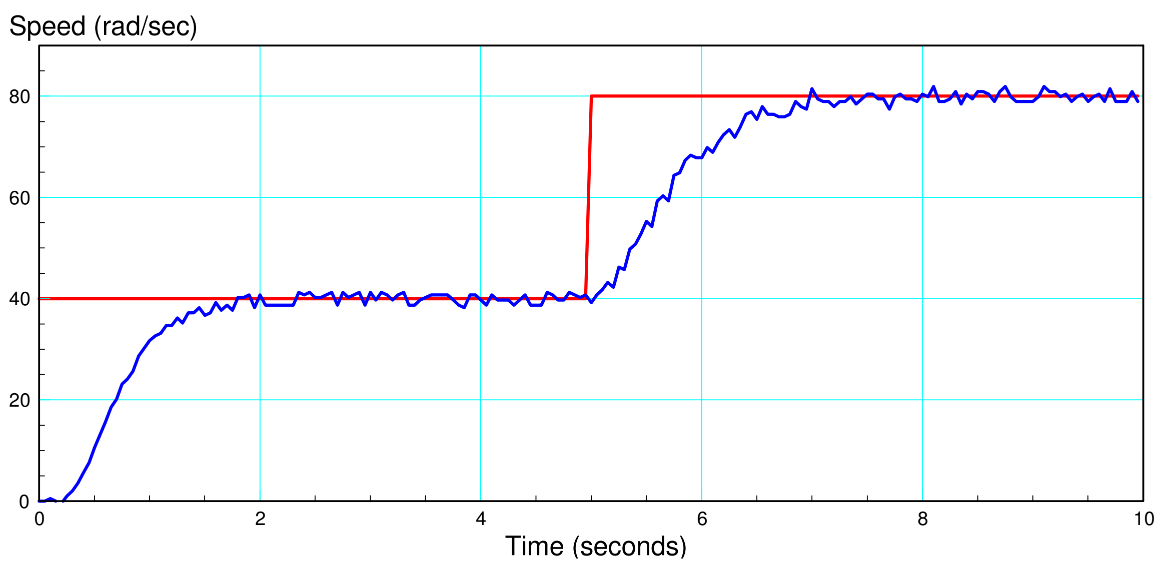
$$e(t) = te^{-2.5t}$$

Picking a small number, such as 2%, to find the theoretical settling time and you get

$$0.02 = te^{-2.5t}$$

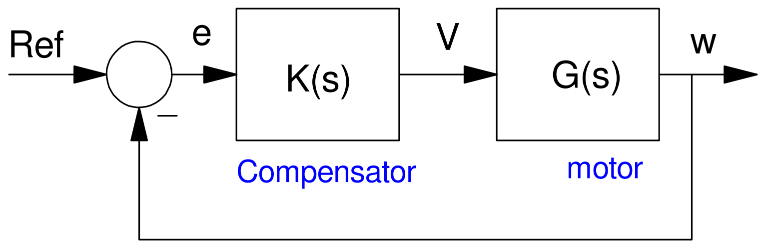
$$t = 1.798$$

or about two seconds as found experimentally.



```
I t = V = 0
dt = 1/20
kv = 65535 / 13.4    # convert volts to pwm
kw = 0.16*pi         # convert counts to rad/sec

fwd.duty_u16(0)
rev.duty_u16(0)

while(t < 10):
    while(flag == 0):
        pass
    flag = 0
    Ref = floor(t/5) * 40 + 40

    Speed = kw * N12
    dV = 0.159*(Ref - Speed)
    V += dV * dt

    if(V > 0):
        fwd.duty_u16(int(V*kv))
        rev.duty_u16(0)
    else:
        fwd.duty_u16(0)
        rev.duty_u16(int(-V*kv))

    print(t,Ref,Speed)
    t += dt

print('Stop')
fwd.duty_u16(0)
rev.duty_u16(0)
```

PI Control



With I control, if the gain, k, is increases, the poles shift as:

$$s(s+5)+39.5k=0$$

- The pole at s=0 is good:
  - This is an integrator which searches to find the voltage needed
- The pole at s=-5 is bad
  - It limits the speed of the system to s = -2.5 + jX

What PI compensators, you can cancel the pole at s = -5

$$K(s)=P+\frac{I}{s}$$

Doing some algebra

$$K(s)=\left(\frac{Ps+I}{s}\right)$$

$$K(s)=P\left(\frac{s+I/P}{s}\right)=\left(\frac{k(s+a)}{s}\right)$$

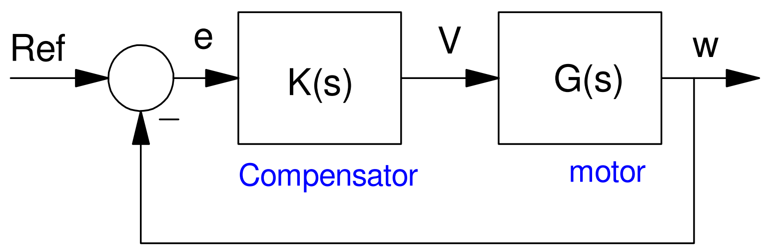
With a PI compensator, you can add a zero to cancel a pole.

- Choosing I/P = 5 to cancel the pole at s = -5
- The open-loop system becomes

$$GK = \left(\frac{k(s+a)}{s}\right)\left(\frac{39.5}{s+5}\right)$$

$$GK = \left(\frac{k(s+5)}{s}\right)\left(\frac{39.5}{s+5}\right)$$

$$GK = \left(\frac{39.5k}{s}\right)$$



and the closed-loop system being

$$\omega = \left(\frac{GK}{1+GK}\right)R = \left(\frac{\left(\frac{39.5k}{s}\right)}{1+\left(\frac{39.5k}{s}\right)}\right)R$$

$$\omega = \left(\frac{39.5k}{s+39.5k}\right)R$$

Note here that

- The DC gain is always 1.000. This results from using an integrator in K(s)
- The closed-loop pole is at s = -39.5k

If you want to place the closed-loop pole at s = -10

$$39.5k = 10$$

$$k = 0.253$$

or

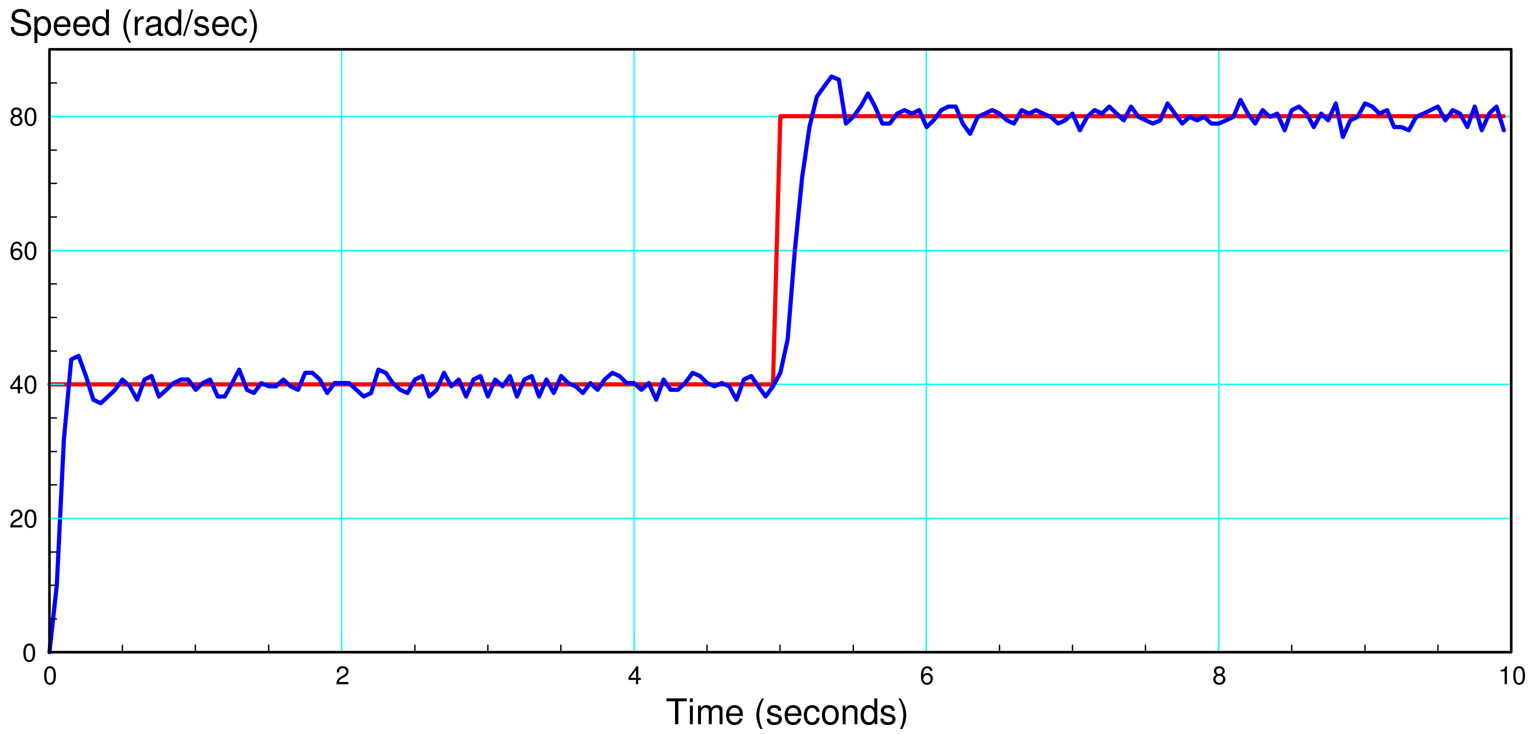
$$K(s) = 0.253\left(\frac{s+5}{s}\right)$$

resulting in the closed-loop system being

$$\omega = \left(\frac{GK}{1+GK}\right)R = \left(\frac{10}{s+10}\right)R$$

The step response for a PI controller is shown below. Note

- The actual speed (blue) tracks the desired speed (red)
- Tracking happens after about 0.5 seconds (about)
- This is what you expect with a closed-loop pole at s = -10



```python
t = V = I = 0
dt = 1/20
kv = 65535 / 13.4   # convert volts to pwm
kw = 0.16*pi        # convert counts to rad/sec

fwd.duty_u16(0)
rev.duty_u16(0)

while(t < 10):
    while(flag == 0):
        pass
    flag = 0
    Ref = floor(t/5) * 40 + 40

    Speed = kw * N12
    E = Ref - Speed
    I += E*dt
    V = 0.254*E + 1.272*I

    if(V > 0):
        fwd.duty_u16(int(V*kv))
        rev.duty_u16(0)
    else:
        fwd.duty_u16(0)
        rev.duty_u16(int(-V*kv))

    print(t,Ref,Speed)
    t += dt

print('Stop')
fwd.duty_u16(0)
rev.duty_u16(0)
```

PI Control via Interrupts

In the above code,

- A timer interrupt is used to set the sampling rate to 50ms (20Hz).
- The interrupt then uses a flag to tell the main routine when 50ms has elapsed
- Then the next iteration begins

This isn't completely necessary.

- The timer interrupt is already being executed every 50ms
- It would not take much coding to add the PI compensator to the timer interrupt routine

By doing so, the main routine is completely free to do whatever you want: the calculations for the motor controller are then done in the background inside the timer interrupt routine.

In-Rush Current and Start-Up Sequence

One problem with an I and PI controller is called *in-rush current*. The current to a DC motor is:

$$V_a = I_a R_a + L_a s I_a + k_t \omega$$

$$I_a = \left(\frac{V_a - k_t \omega}{R_a + L_a s} \right)$$

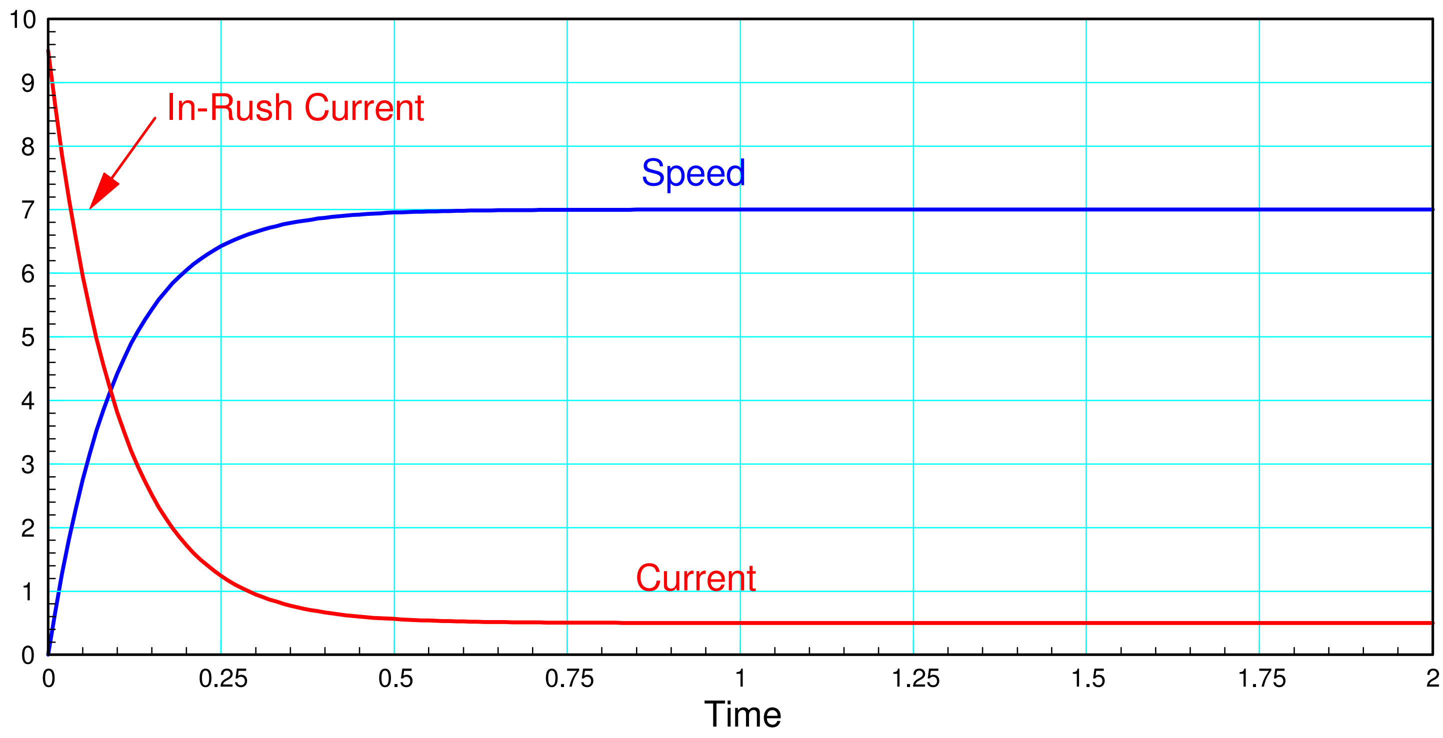
If you stall a DC motor (w = 0),

- The motor no longer generates any back-emf
- The current is only limited by the armature resistance
- This can burn out the motor

In-rush current is the large current you see on startup

- The motor hasn't started spinning yet
- The back-emf is small or zero
- The current is large

This large current on startup can burn out the motor



For small motors like the ones used in this lecture, the in-rush current isn't that large:

$$\max(I_a) = \frac{13.4V}{26.5\Omega} = 505mA$$

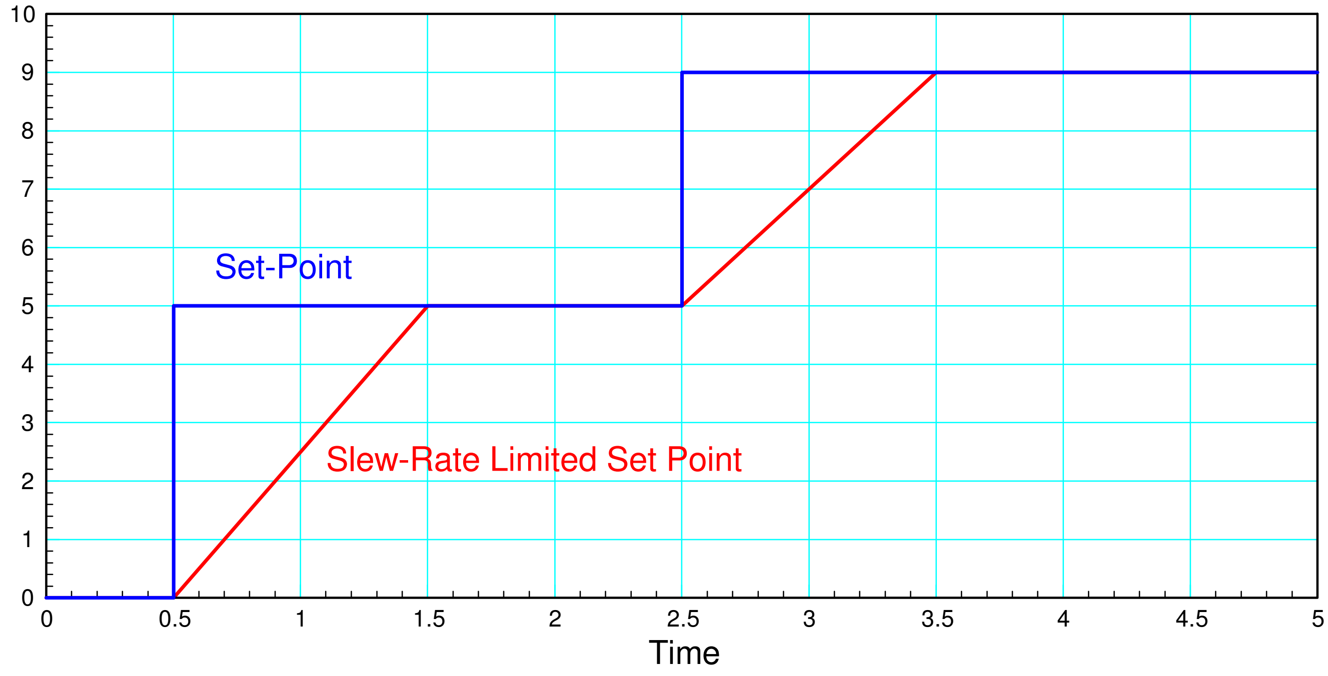
For larger motor, such as an Allen Bradley CDP3353 1/2 hp DC servo motor, the in-rush current is

$$\max(I_a) = \left(\frac{90V}{0.664\Omega}\right) = 135.5A$$

This is well above the maximum rated armature current of 5A. If you apply 90V to this motor right away, you'll probably burn out the armature windings.

To limit the current on start-up, several options are available:

- Add a slew-rate limit to the set point (turn a step input into a ramp input, allow the motor a chance to get up to speed before you hit it with 90V)
- Remove the load and add a resistor in series with the armature on startup. Once the motor gets up to speed, then remove the resistor and then add a load.



To reduce the in-rush current, a slew-rate limit can be added to the set-point (R)

Summary

Once you have edge interrupts and timer interrupts, measuring and controlling the speed of a DC servo motor isn't that hard:

- PWM signals along with an H-bridge allow you to drive the motor from 0% to 100% in both directions.
- Edge interrupts along with an optical encoder allow you to measure the motor's speed as pulses per second.
- Timer interrupts let you set the sampling rate - needed for numerical integration, and
- I and PI compensators can then be implemented by using just a couple multiplication and additions.

This results in a motor which can track a constant set point dead on, or a time varying signal fairly well (as long as the set point doesn't change faster than the closed-loop system's bandwidth).