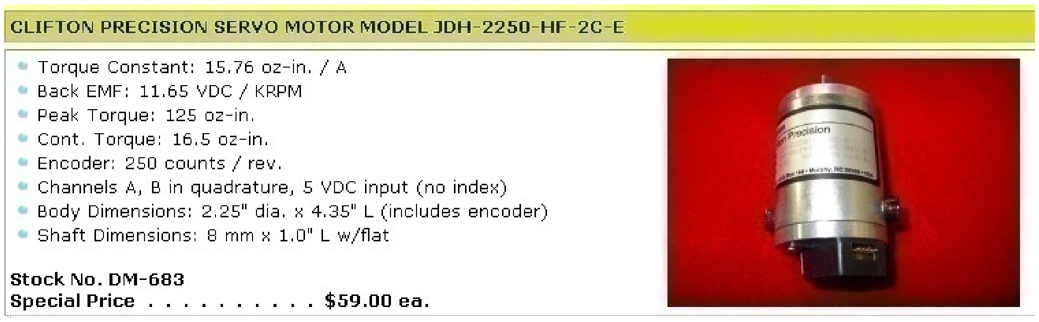
# 18. Speed Control of a DC Servo Motor



CLIFTON PRECISION SERVO MOTOR MODEL JDH-2250-HF-2C-E

- Torque Constant: 15.76 oz-in. / A
- Back EMF: 11.65 VDC / KRPM
- Peak Torque: 125 oz-in.
- Cont. Torque: 16.5 oz-in.
- Encoder: 250 counts / rev.
- Channels A, B in quadrature, 5 VDC input (no index)
- Body Dimensions: 2.25" dia. x 4.35" L (includes encoder)
- Shaft Dimensions: 8 mm x 1.0" L w/flat

**Stock No. DM-683**
**Special Price . . . . . . . . . . $59.00 ea.**

## Introduction:

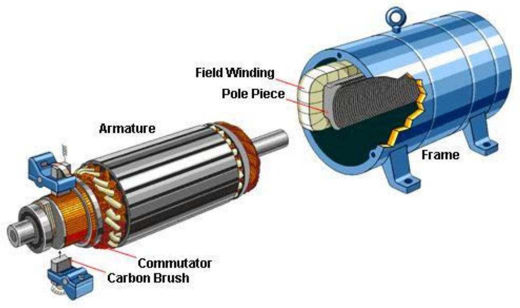
Once you have edge interrupts and timer interrupts, you are able to control the speed and angle of a DC servo motor. In this lecture, we'll look at

- 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

If you remove the commutator and spin the motor, it behaves like an AC generator with a sine wave at its output. With the commutator, the motor only outputs the peaks of the AC waveforms - resulting in a DC output or a DC geneator. From duality, if you reverse the process and apply a DC voltage, it will produce torque and spin. Essentially, DC motors and DC generators are one and the same.

The model for a DC servo motor reflects this:

- The left side models the electrical side of the motor ( a 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)

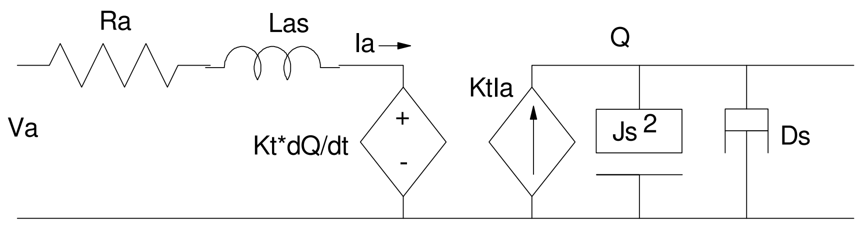
Coupling these together are a motor / generator pair

- As you apply current to the motor, it produces torque

$$T = k_t I_a$$

- As you spin the motor, it produces voltage

$$E = k_t \omega$$



Electromechanical model for a DC servo motor

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

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

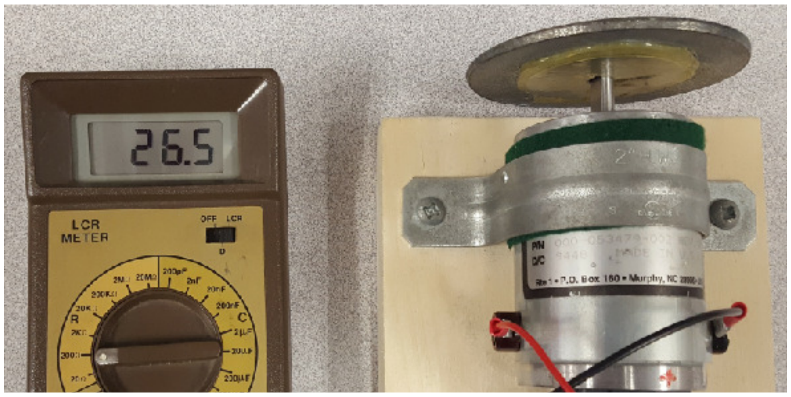
If L = 0 (usually a good assumption)

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

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

## Finding the Parameters for a DC Motor

One way to find this transfer function is to plug in the parameters for the motor. Some of these are easy to measure. The armature's resistance and inductance can be measured using an RLC meter:



Using an RLC meter, Ra = 26.5 Ohms and La = 12.689mH

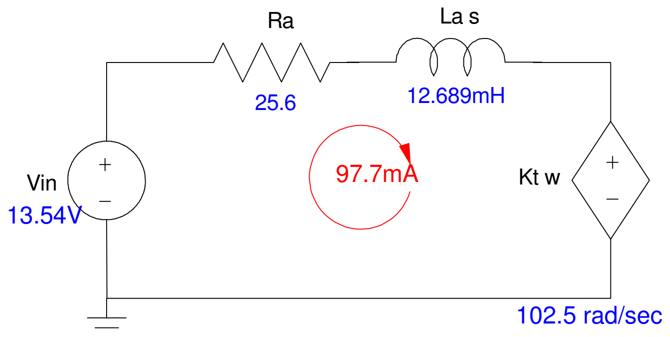
The motor's torque constant can be measured by running the motor without a load and measuring voltage, current, and motor speed. Doing so with this motor, you get

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

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

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

(note: the mechanical and electrical constants are the same if using mks units)



The torque constant can be measured by applying a constant voltage and measuring the resulting speed without a load

The friction of the motor can be measured by noting the no-load speed and doing 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:

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

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

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

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

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

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

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 that this model has a fast pole (s + 2084) and a slow pole (s + 4.032). The fast pole is primarily a result of the electrical time constant (R/L) while the slow pole results from the mechanical time constant (D/J). This is very common. If you ignore the fast pole - meaning you assume L = 0, you get a simpler model which is almost correct

$$\omega = \left(\frac{35.51}{s+4.032}\right)\left(\frac{2084}{s+2084}\right)V_a$$

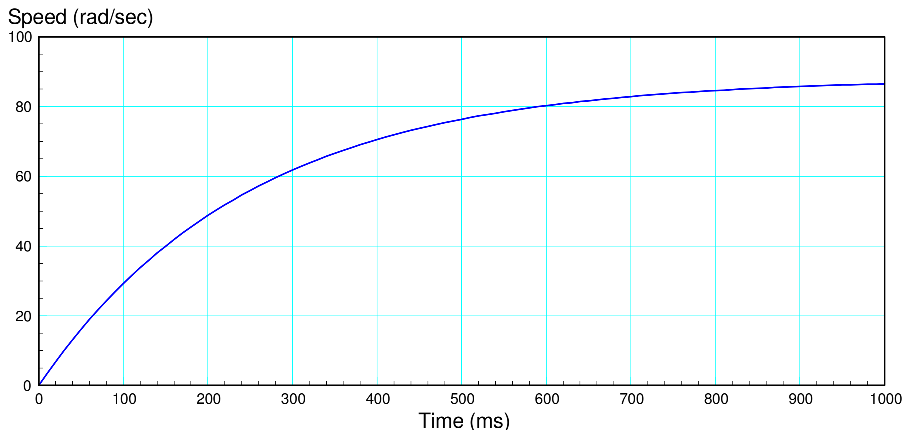
$$\omega \approx \left(\frac{35.51}{s+4.032}\right)V_a$$

Translation: the step response of this DC servo motor should be that of 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)
```

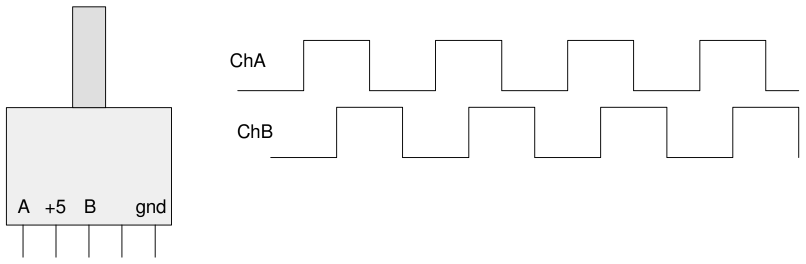


Theoretical response of the DC servo motor to a 10V step input

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

One common way to measure a motor's position and speed is to add an optical encoder to the motor. An optical encoder outputs a square wave in phase quadrature.



Optical encoders attached to the shaft of a motor output two square waves, each 90 degrees apart

This allows you to measure

- The position of the motor by counting edges, or
- The speed of the motor by counting edges per second.

With the motor used in this lecture, the encoder has 250 pulses per rotation, meaning you get 1000 counts per rotation if you count both rising and falling edges.

If you want to measure the speed of the motor, you can edges per second. For example, the following code will count rising edges on Channel A as you rotate the motor:

```
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

Speed can then be found by counting edges every 50ms (20Hz) with the conversion to rad/sec being

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

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

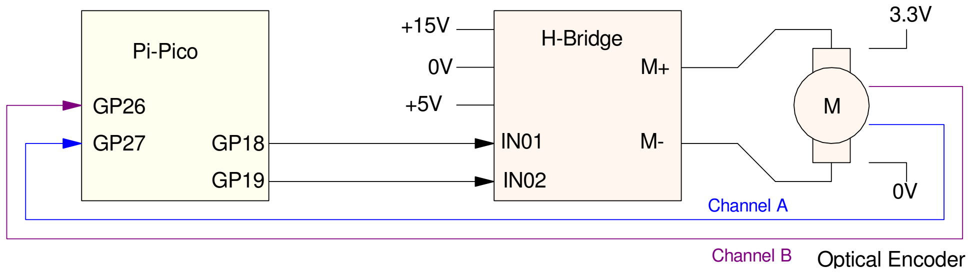
where $\delta N_1$ is the number of counts in the last 50ms (20Hz). In code

```
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)
```

Python code for measure the motor's speed every 50ms (20Hz)

## 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:



Hardware set-up for connecting a Pico to an H-bridge and a DC servo motor
A 15V, 2A power supply drives the output of the H-bridge (6V to 24V @ 1A)

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)

The power supply is somewhat arbitrary: whatever I could find lying around. With this motor, the maximum current it should draw is determined by Ra when the motor is stationary (no back EMF):

$$I_{\max} = \left(\frac{13.4V}{26.4\Omega}\right) = 507mA$$

As long as the power supply can output 507mA, it should work (it was actually rated at 2A).

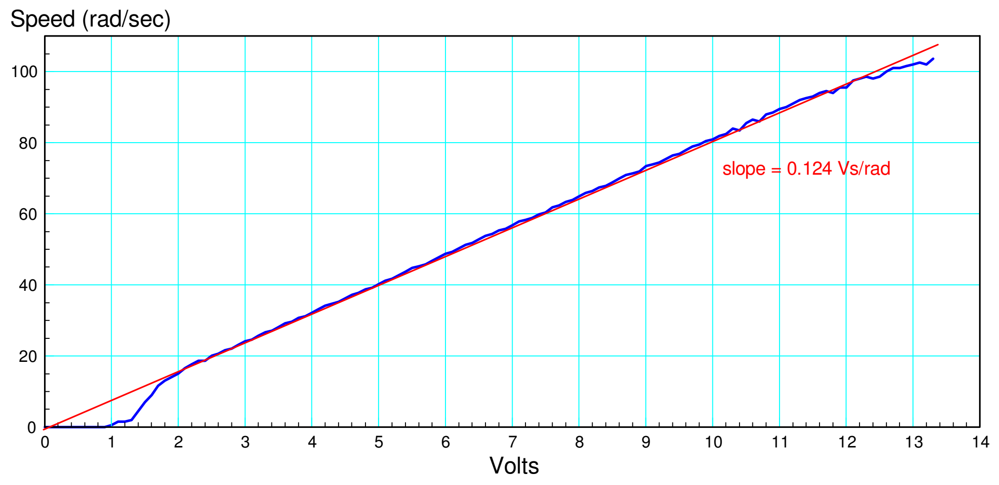
The steady-state relationship between speed and voltage can be found by

- 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:

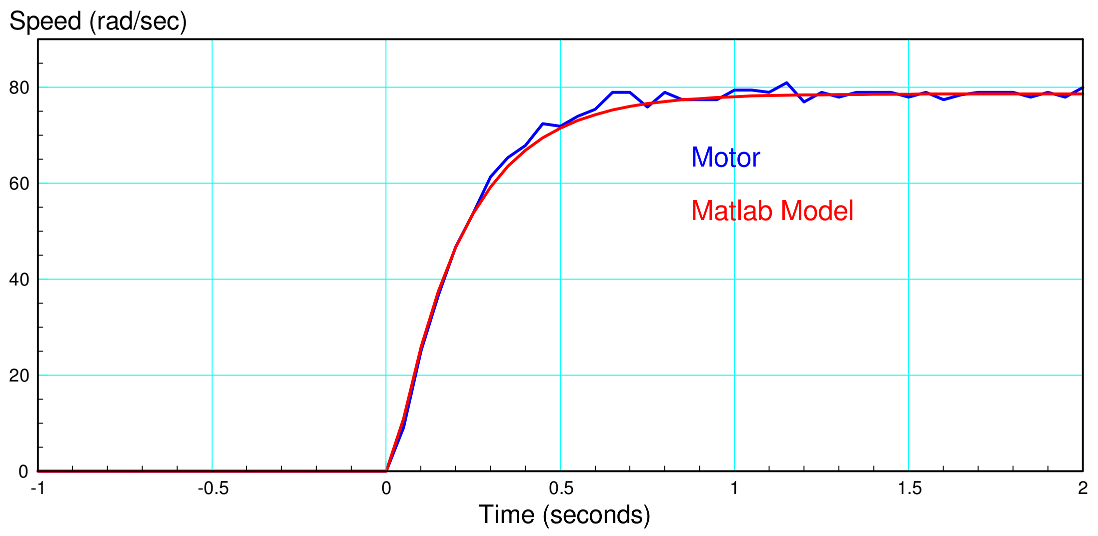


No-Load Speed vs. input voltage

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

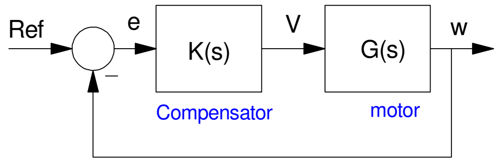
```
def tick(timer):
    global N1, N2, N12, flag
    X = N1
    N12 = N1 - N2
    N2 = X
    flag = 1

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

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

Python code for finding the step response of the motor.

Now that we have a model, we can control the speed 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).



Feedback Configuration

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

$$\omega = GKe$$

$$e = Ref - \omega$$

or after simplifying

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

K(s) can take on several forms. For this case, the input to the motor (V) should be a constant when driving the motor at a constant speed (you need to apply a non-zero voltage to a DC motor to make it spin.) Exactly what that constant is depends upon the motor and the loading on the motor.

One common type of controller is called a PID controller. Here, K(s) contains three terms:

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

- 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

$$(s+2.5)^2 = s(s+5)+39.5k$$

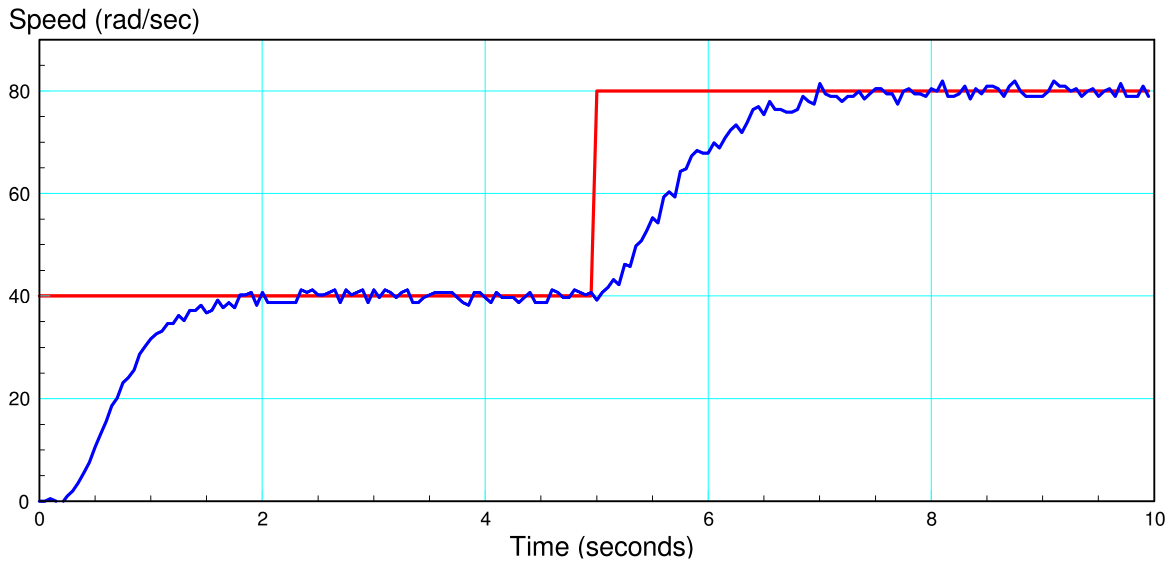
$$k = \left( \frac{2.5^2}{39.5} \right) = 0.158$$

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 figure below shows the response of this motor when R is a constant (red line).



Speed vs. Time with I control. The set point R (red) and actual speed (blue) shown

Note that

- 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). *(You need to know the sampling rate when implementing an integrator. If the sampling rate is off, your integrator's gain is off. Hence, the use of the timer interrupt and a flag to set the sampling rate,)*

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('{: 7.2f}'.format(t), '{: 7.2f}'.format(Ref), '{:
7.4f}'.format(Speed))
    t += dt


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

## PI Control

In the previous case, if the gain, k, is increases, the poles of the closed-loop system determined by:

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

will not get any faster. All that happens is the poles become complex with the real part being at s = -2.5. The pole at s=0 is good: that is the integrator that causes the speed to match the set point. The pole at s = -5 is bad: it is limiting the speed of the closed loop system.

What PI compensators do is they allow you to cancel one pole. With a PI control, K(s) is of the form

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

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 results in the open-loop system being:

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

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

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

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

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

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)

In theory, the transient should decay as

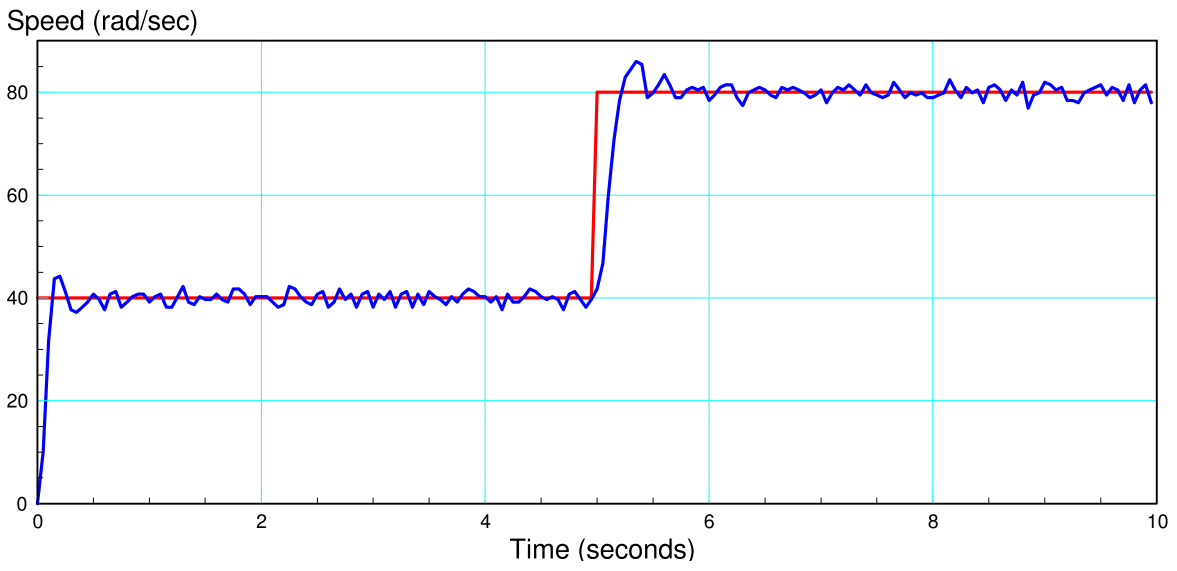
$$e(t) = e^{-10t}$$

Calling 2% about zero (termed the 2% settling time)

$$0.02 = e^{-10t}$$

$$t = \frac{-4}{-10} = 0.4 \text{ seconds}$$

About what's measured.



PI Code

```
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('{: 7.2f}'.format(t), '{: 7.2f}'.format(Ref), '{:
7.4f}'.format(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 and the next iteration should start.

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

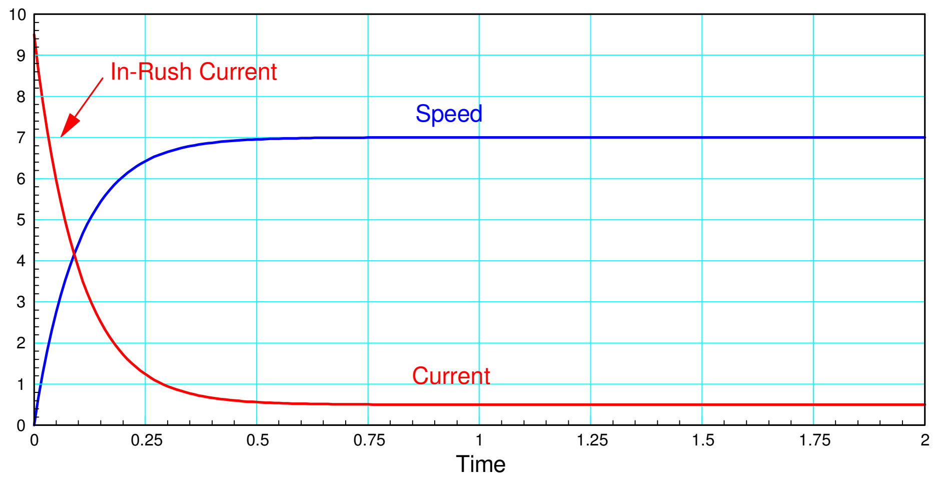
One problem with an I and PI controller is called *in-rush current*. The current to a DC motor is:

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

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

If you stall a DC motor (w = 0), the motor no longer generates any back-emf. This results in the armature current being limited by only the armature resistance. Larger motors are not designed to withstand currents this large: the armature windings are sized with the assumption that the motor will be spinning - meaning the back-emf ($k_t\omega$) helps to limit the armature current.

A similar problem happens at startup. When you turn on a motor, the motor presumably is not spinning (w=0). This means that the only thing limiting the armature current is the armature resistance. Once the motor gets up to speed, the armature current drops. This large initial current is called *in-rush current.*



When a motor is turned on, current can spike until the motor gets up to speed. This is called *in-rush current*

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

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

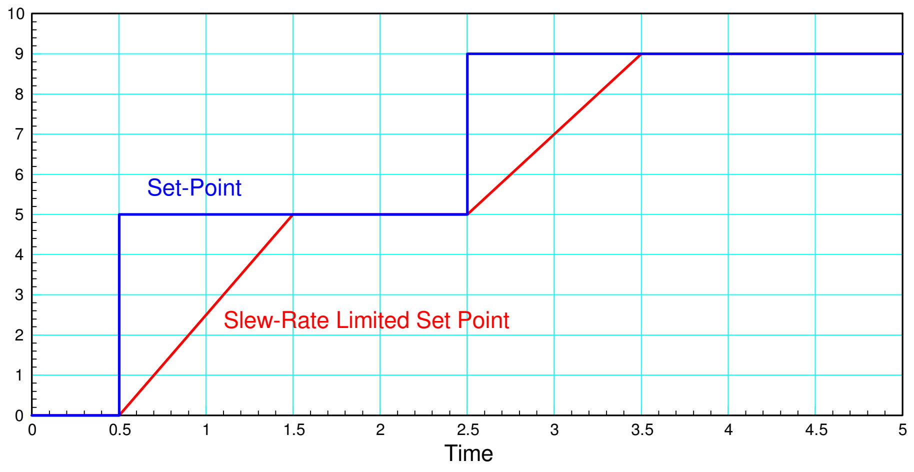
For larger motor, such as an Allen Bradley CDP3353 1/2 hp DC servo motor, the in-rush current is

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

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).

## References

Pi-Pico and MicroPython

- https://github.com/geeekpi/pico_breakboard_kit
- https://micropython.org/download/RPI_PICO/
- https://learn.pimoroni.com/article/getting-started-with-pico
- https://www.w3schools.com/python/default.asp
- https://docs.micropython.org/en/latest/pyboard/tutorial/index.html
- https://docs.micropython.org/en/latest/library/index.html
- https://www.fredscave.com/02-about.html

Pi-Pico Breadboard Kit

- https://wiki.52pi.com/index.php?title=EP-0172

Other

- https://docs.sunfounder.com/projects/sensorkit-v2-pi/en/latest/
- https://electrocredible.com/raspberry-pi-pico-external-interrupts-button-micropython/
- https://peppe8o.com/adding-external-modules-to-micropython-with-raspberry-pi-pico/
- https://randomnerdtutorials.com/projects-raspberry-pi-pico/
- https://randomnerdtutorials.com/projects-esp32-esp8266-micropython/