Angle Control of a DC Servo Motor

ECE 476 Advanced Embedded Systems

Jake Glower - Lecture #19

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

Introduction:

In the previous lecture, the speed of a DC motor was measured and controlled. This is useful if the motor is driving something like a car. If instead you wanted to open and close a valve or control the position of a robotic arm, the angle of the motor is what you want to control.

In this lecture, we look at

- Measuring the angle of a DC motor using optical encoders
- Controlling the angle using a P controller, and
- Controlling the angle using a lead compensator (similar to a PD controller)

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.

Motor Dynamics

From the previous lecture, the dynamics of the motor from voltage to speed is:

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

If you integrate speed, you get angle:

$$\theta = \left(\frac{1}{s}\right) \omega$$

The goal is thus to control the output of a system with dynamics of

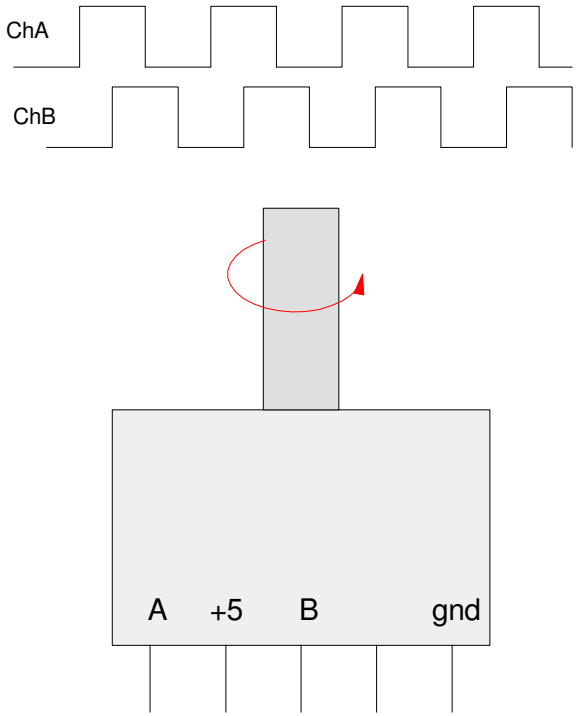
$$\theta \approx \left(\frac{39.3}{s(s+5)}\right) V$$

Measuring Motor Angle

Similar to the last lecture, an optical encoder is used to measure the motor angle.

The output of a optical encoder is a square wave in phase quadrature.

- By counting pulses per second, you get speed.
- By counting edges, you get angle.



There are several options for counting edges.

- The encoder used in this motor has 250 pulses per rotation (for each channel).

Counting Rising Edges of Channel A:

- Gives 250 counts per rotation
- Angle drifts with chater, however

Counting rising and falling edges of channel A

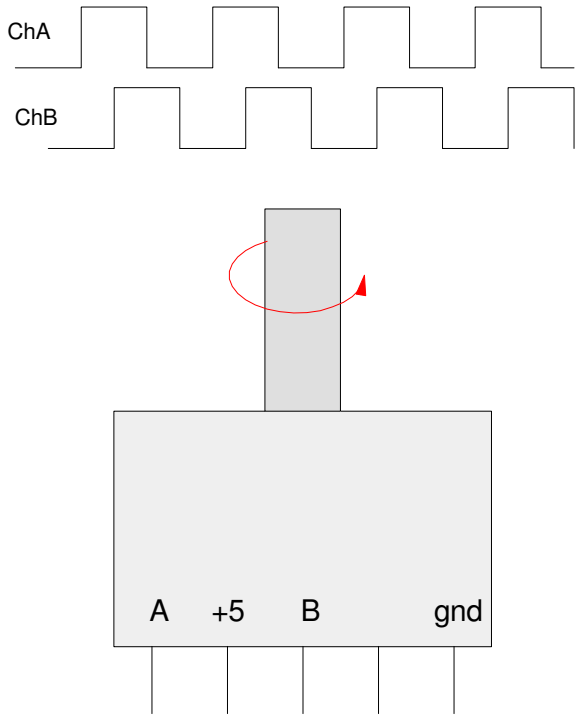
- Gives 500 counts per rotation
- Eliminates drift

Counting both edges of channel A and B

- Gives 1000 counts per rotation

With 1000 counts per rotation, the conversion from counts to radians is

$$\theta = \left(\frac{2\pi}{1000}\right) N$$



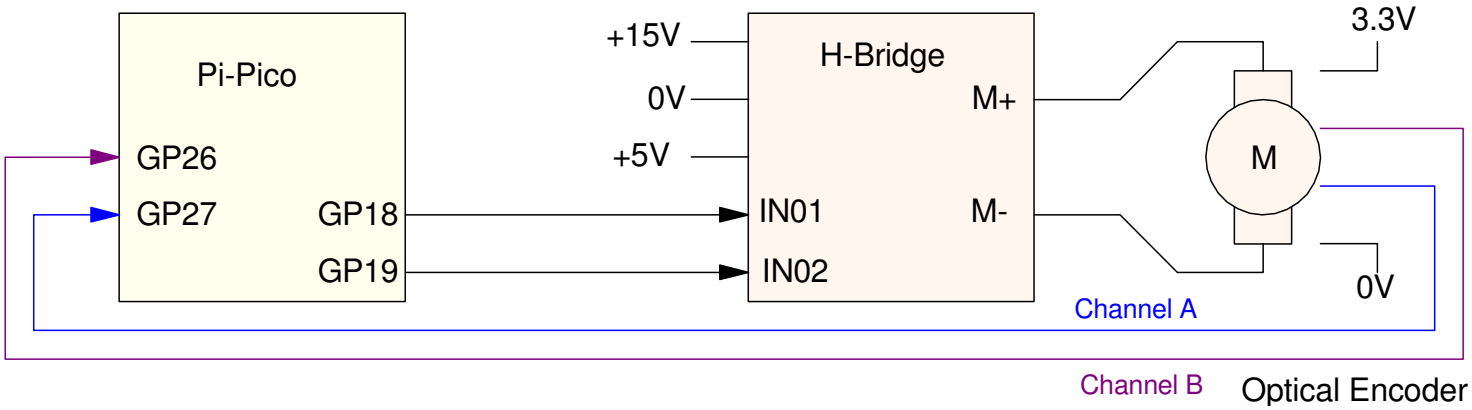
In Code:

```
# Code for measuring motor angle
pin1 = Pin(26,Pin.IN)
pin2 = Pin(27,Pin.IN,Pin.PULL_UP)
N1 = N2 = N12 = 0

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

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

pin1.irq(trigger=Pin.IRQ_FALLING | Pin.IRQ_RISING, handler=ChanA)
pin2.irq(trigger=Pin.IRQ_FALLING | Pin.IRQ_RISING, handler=ChanB)
```

Measuring the angle of a motor by counting rising and falling edges on both channel A and B.
This results in 1000 counts per rotation

Hardware Connections

To drive the motor, an H-bridge is used - same as was done for speed control



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)

100% PWM results in 13.4V to the motor

- Slight loss in the H-bridge

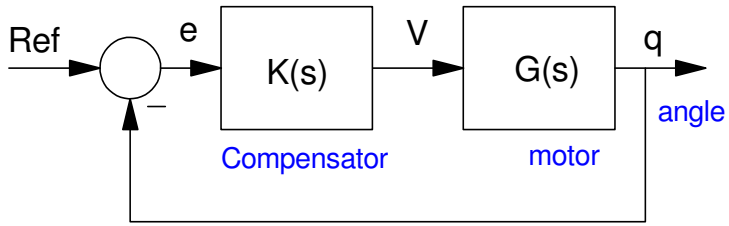
Feedback Control

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 angle (q) is then

$$\theta = GKe$$

$$e = Ref - \theta$$



or after simplifying

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

The response varies based upon what type of controller you select for K(s)

P Control: K(s) = P=k

The simplest type of controller is a simple gain. This results in the open-loop system (GK) being

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

while the closed-loop system is

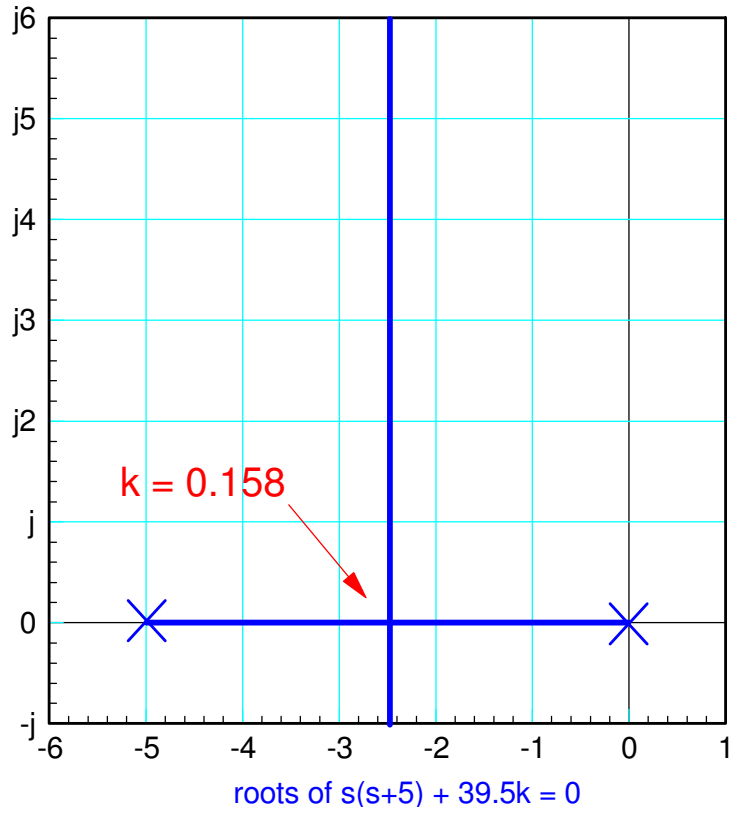
$$\left(\frac{GK}{1+GK}\right) = \left(\frac{39.5k}{s(s+5)+39.5k}\right)$$

The roots of the closed-loop system are the solutions to

To place the closed-loop pole at s = -2.5:

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

$$k = 0.158$$

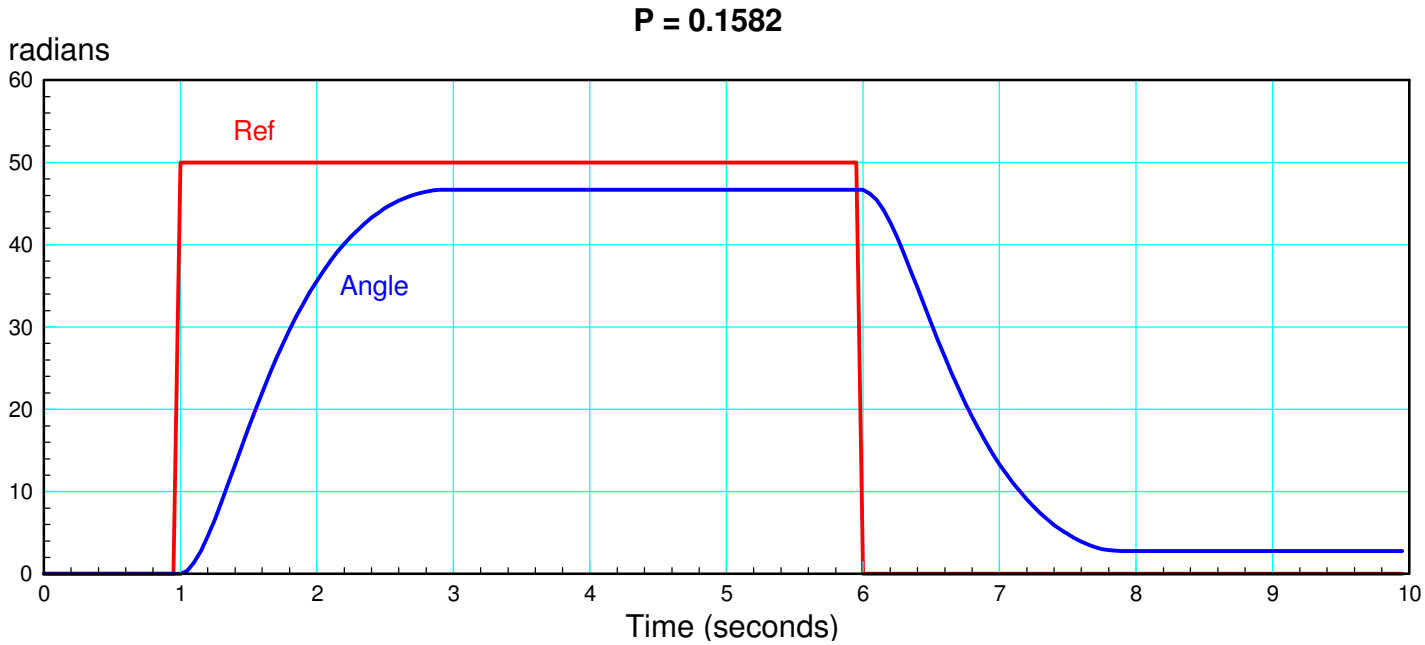


With this controller, the voltage applied to the motor is

$$V = ke$$

$$V = 0.1582(R - \theta)$$

where R is the set point (reference input). The response for a step input of 50 radians is as follows.



Step Response with P Control

PD Control: K(s) = P + Ds

A second type of control which is popular is PD control. With this type of controller

$$K(s) = P + Ds$$

Normally, you don't want to use differentiation with a controller. In this case, it's OK since the output is angle (q) and its derivative is speed (ω). A PD control law just uses the angle (P term) and speed (D term) to determine the voltage:

$$V = (P + Ds)e$$

$$V = P(R - \theta) + Ds(R - \theta)$$

or if the derivative of R is set to zero and replacing $s\theta$ with speed (w)

$$V = P(R - \theta) + D(0 - \omega)$$

Likewise, you don't actually use a differentiator in K(s).

Anyway, with a little algebra, K(s) can be rewritten as

$$K(s) = D\left(s + \frac{P}{D}\right)$$

$$K(s) = k(s + a)$$

With a PD controller, you can place a zero to cancel a pole. In this case, the pole you want to get rid of is at s = -5 (this pole limit how fast the closed-loop system can become)

$$K(s) = k(s + 5)$$

resulting in the open-loop system being

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

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

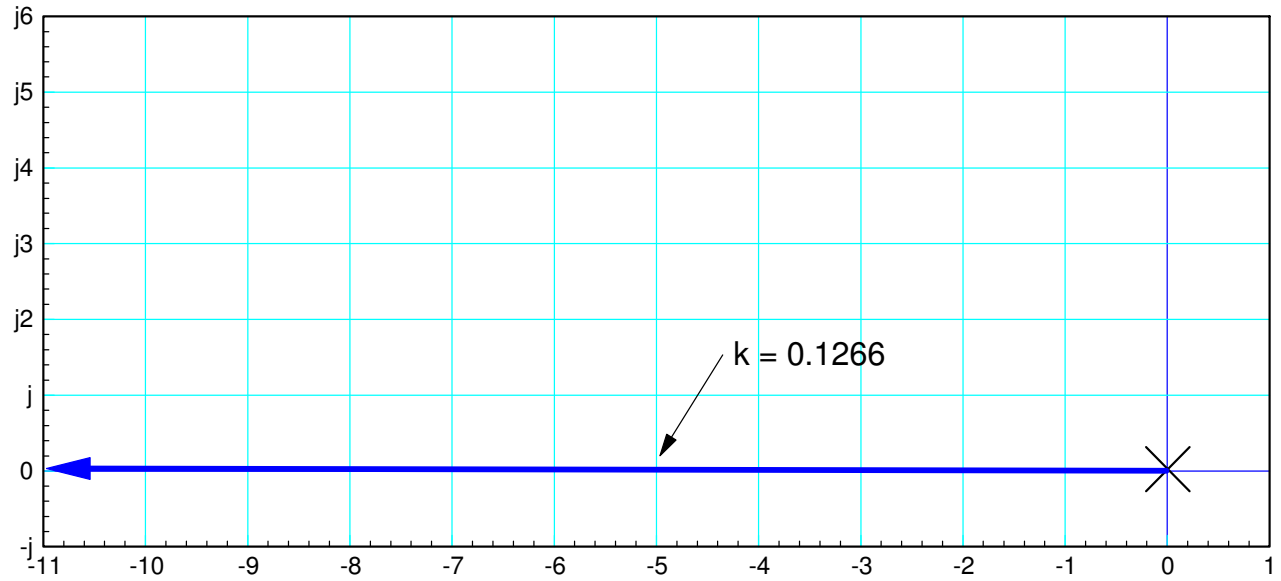
and the closed-loop system being

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

The roots of the closed-loop system are the solutions to

$$s + 39.5k = 0$$

which has the following root-locus plot



Root Locus for s + 39.5k = 0

If you want to place the closed-loop poles at s = -5, then

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

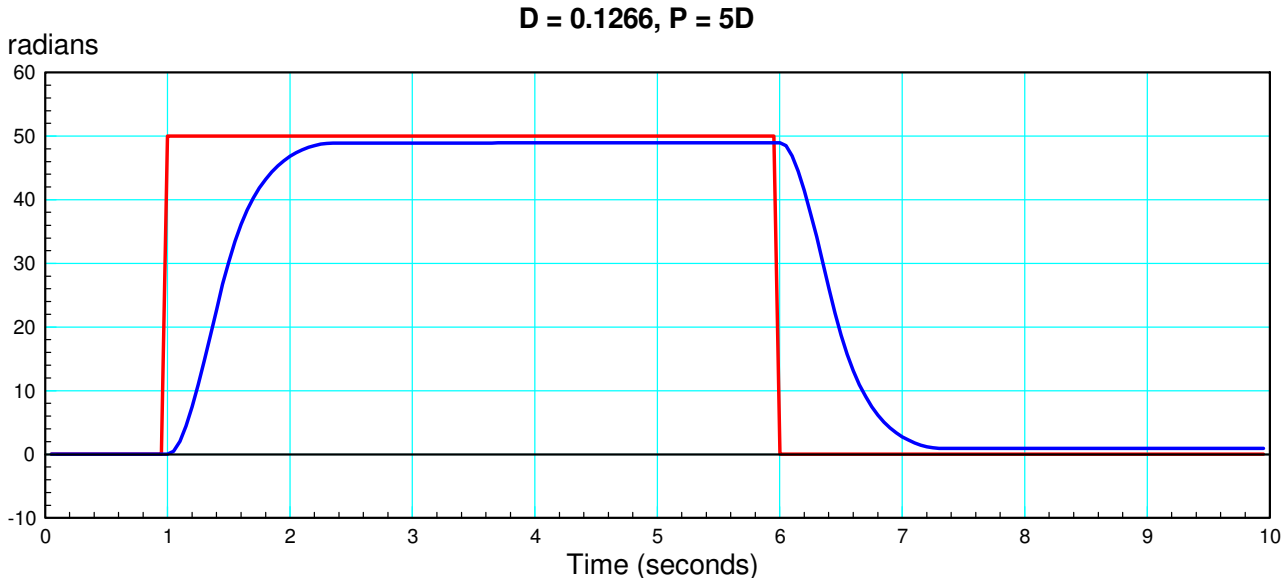
$$k = 0.1266$$

and

$$K(s) = 0.1266(s + 5)$$

$$= 0.1266s + 0.6329 \; = Ds + P$$

The step response with this controller is then



Code for both P and PD control is as follows

```
# Code for PD Control
E = Ref - Angle
sE = 0 - Speed

V = P*E + D*sE
```

Lead Compensator: K(s) = k (s+a)/(s+b)

A third type of controller is called a *lead compensator*.

$$K(s) = k\left(\frac{s+a}{s+b}\right) \qquad b > a$$

The idea behind this type of controller is the system has a slow pole which is causing problems (s + 5 in this case). The zero allows you to cancel this pole and replace it with a faster pole - resulting in a faster overall system. If you want to make the system twice as fast, then

$$K(s) = k\left(\frac{s+5}{s+10}\right)$$

The open-loop system then becomes

$$GK = \left(\frac{39.5}{s(s+5)}\right)\left(\frac{k(s+5)}{s+10}\right)$$

$$GK = \left(\frac{39.5k}{s(s+10)}\right)$$

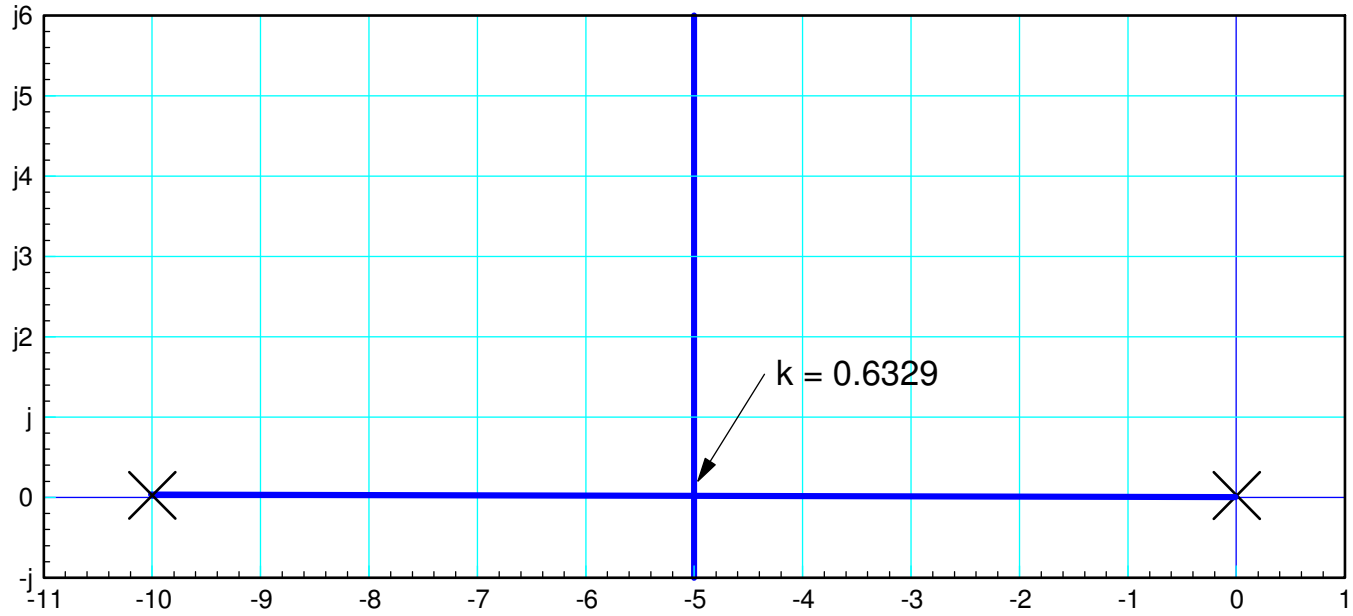
and the closed-loop system becomes

$$\left(\frac{GK}{1+GK}\right) = \left(\frac{39.5k}{s(s+10)+39.5k}\right)$$

The roots of the closed-loop system are the solution to

$$s(s+10)+39.5k=0$$

which has the following root-locus plot



root locus plot for s(s+10) + 39.5k = 0

If you want to place the closed-loop poles at s = -5, then k is the solution to

$$(s(s+10)+39.5k)_{s=-5}=0$$

$$k=0.6329$$

or

$$K(s)=0.6329\left(\frac{s+5}{s+10}\right)$$

To implement K(s), there are two main options

- Implement in the s-plane
- Implement in the z-plane

s-Plane Implementation: In the s-plane, rewrite this as a proper fraction

$$K(s) = 0.6329\left(\frac{s+10-5}{s+10}\right)$$

$$K(s) = 0.6329\left(1 - \frac{5}{s+10}\right)$$

and the voltage is

$$V(s) = K(s) \cdot E(s)$$

$$V(s) = 0.6329\left(1 - \left(\frac{5}{s+10}\right)\right)E(s)$$

Define a dummy-variable, X, to be the filtered error signal

$$X(s) = \left(\frac{5}{s+10}\right)E(s)$$

This can be implemented in code by cross-multiplying and solving for sX

$$(s+10)X = 5E$$

$$sX = -10X + 5E$$

In code:

```
# Code for lead compensator
E = Ref - Angle
sX = -10*X + 5*E

# integrate to find W
X = X + sW*dt

# determine the voltage
V = 0.6329*(E - W)
```

s-plane implementation of a lead compensator

z-Plane Implementation: A second (and better) way to implement K(s) is to convert to the z-domain. The conversion from the s-plane to the z-plane is

$$z = e^{sT}$$

where T is the sampling rate (50ms here). The zero and pole for the lead compensator then convert as

$$s = -5 \quad z = e^{sT} = 0.7788$$

$$s = -10 \quad z = e^{sT} = 0.6065$$

meaning

$$K(s) = 0.6329\left(\frac{s+5}{s+10}\right)$$

is equivalent to

$$K(z) = k\left(\frac{z-0.7788}{z-0.6065}\right)$$

To find k, match the DC gain

$$K(s=0)=0.3164$$

$$K(z=1)=0.3164=k\left(\frac{z-0.7788}{z-0.6065}\right)$$

$$k=0.5629$$

or

$$K(z)=0.5629\left(\frac{z-0.7788}{z-0.6065}\right)$$

In code, this translates to

$$V=0.5629\left(\frac{z-0.7788}{z-0.6065}\right)E$$

Cross multiply

$$(z-0.6065)V=0.5629(z-0.7788)E$$

Solve for the highest power of z

$$zV=0.6065V+0.5629(z-0.7788)E$$

Convert back to time

$$V(k+1) = 0.6065V(k) - 0.5629(E(k+1) - 0.7788E(k))$$

Time shift by one (change of variable)

$$V(k) = 0.6065V(k-1) - 0.5629(E(k) - 0.7788E(k-1))$$

In code:

```
# Code for lead compensator
E1 = E0
E0 = Ref - Angle

# determine the voltage
V = 0.6065*V - 0.5629*(E0 - 0.7788*E1)
```

z-plane implementation of a lead compensator

The step response with a lead compensator is then as follows:

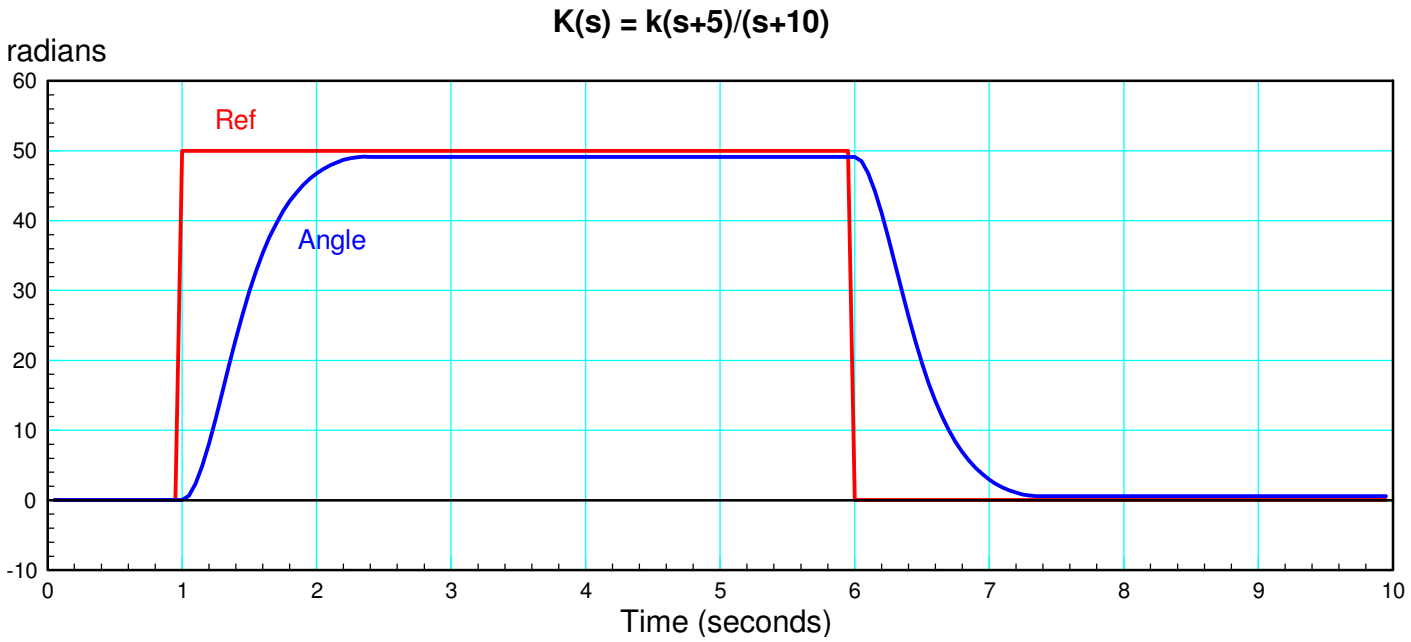
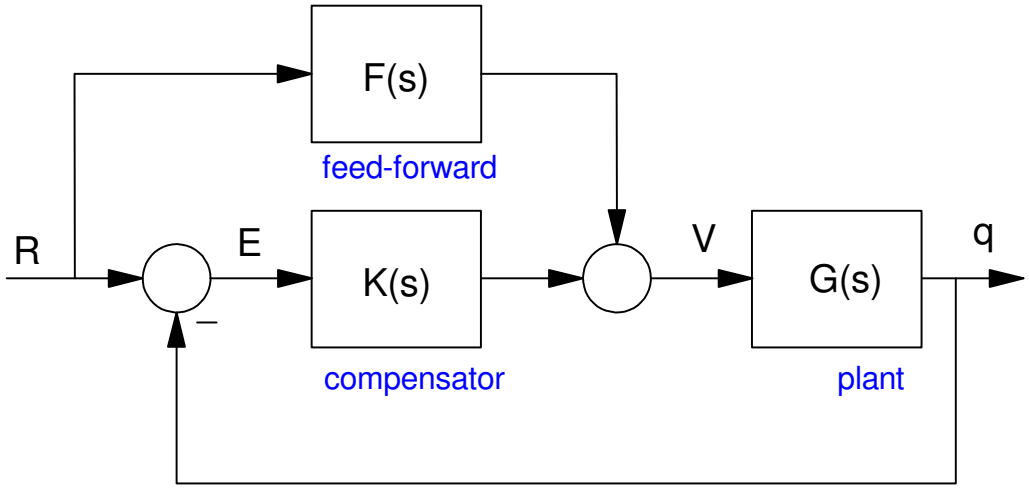


Figure - step response with a lead compensator

Note that the step response of the motor with a lead compensator is similar to what you get with a PD compensator. An advantage of the lead compensator is you don't need to measure motor velocity. This simplifies the code and removes some noise in the system.

Path Planning and Feed-Forward Control

Note that in the previous step responses, there is a lag between the input and the output. This doesn't have to be there. One way to reduce or eliminate this lag is to add a feed-forward term in the controller (F(s)):



Plant with feedback (K) and feedforward (F) inputs

The idea is this. The output of the plant is related to the input voltage as

$$\theta = \left(\frac{39.5}{s(s+5)}\right) V$$

If you want the output to match the reference input (R), then the voltage should be

$$V = \left(\frac{s(s+5)}{39.5}\right) \theta$$

or

$$V = \left(\frac{s(s+5)}{39.5}\right) R = F(s) \cdot R$$

Essentially, F(s) is an inverse of the plant: it specifies the voltage required for a given input, R. Ideally, the feedforward term (the voltage you computed using F(s)) forces the output to track the set point exactly. If there is some error or noise in the system, the feedback, K(s), will drive this error to zero.

One problem with feedforward control is the voltage is proportional to the first and second derivatives of the set point, R

$$V = \left(\frac{1}{39.5}\right) s^2 R + \left(\frac{5}{39.5}\right) sR$$

This lead into the area termed *path planning*: how you define the path that the motor is to follow.

Previously, r(t) was a step input. This path doesn't work well with feed-forward control since the first and second derivatives of a step function are infinite. Instead, we need a path from 0 radians to 50 radians which has finite first and second derivatives.

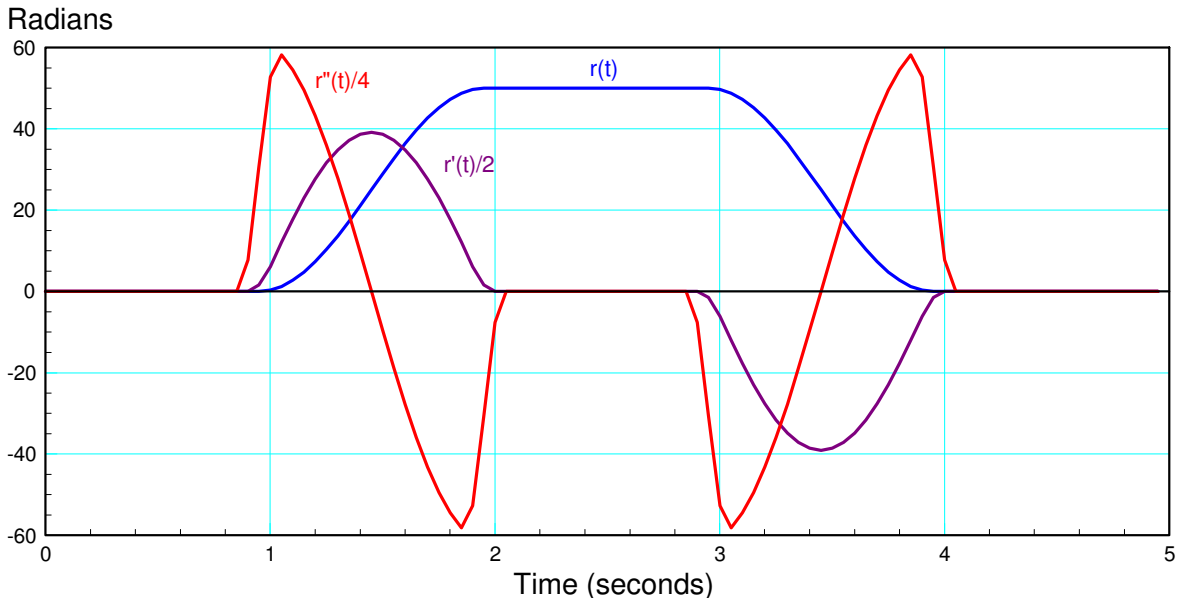
One option (there are others) is to define r(t) as a cosine-function. Let r(t) go from 0 to 50 radians in one second as:

$$r(t) = 50\left(\frac{1-\cos(\pi t)}{2}\right) \qquad 0 < t < 1$$

If r(t) is to go from 50 radians to 0 radians in two seconds

$$r(t) = 50 - 50\left(\frac{1-\cos(\pi t)}{2}\right)$$

With this definition of the path from 0 to 50 radians and back, the first and second derivatives are finite:

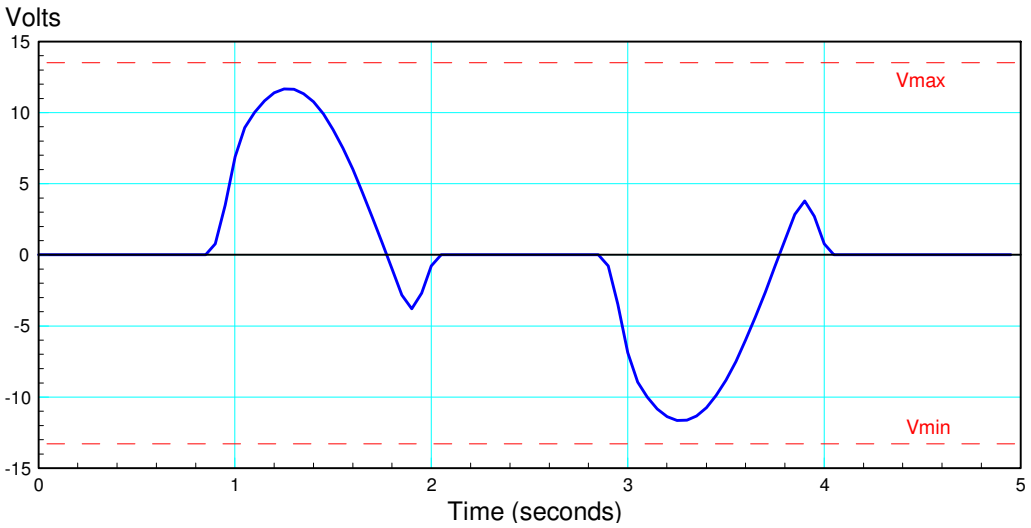


r(t) along with its first and second derivatives

Once you know r(t) and its derivatives, you can compute the voltage required for this output

$$V_{ff} = \left(\frac{s(s+5)}{39.5}\right) R$$

$$V_{ff} = \left(\frac{1}{39.5}\right) s^2 R + \left(\frac{5}{39.5}\right) sR$$



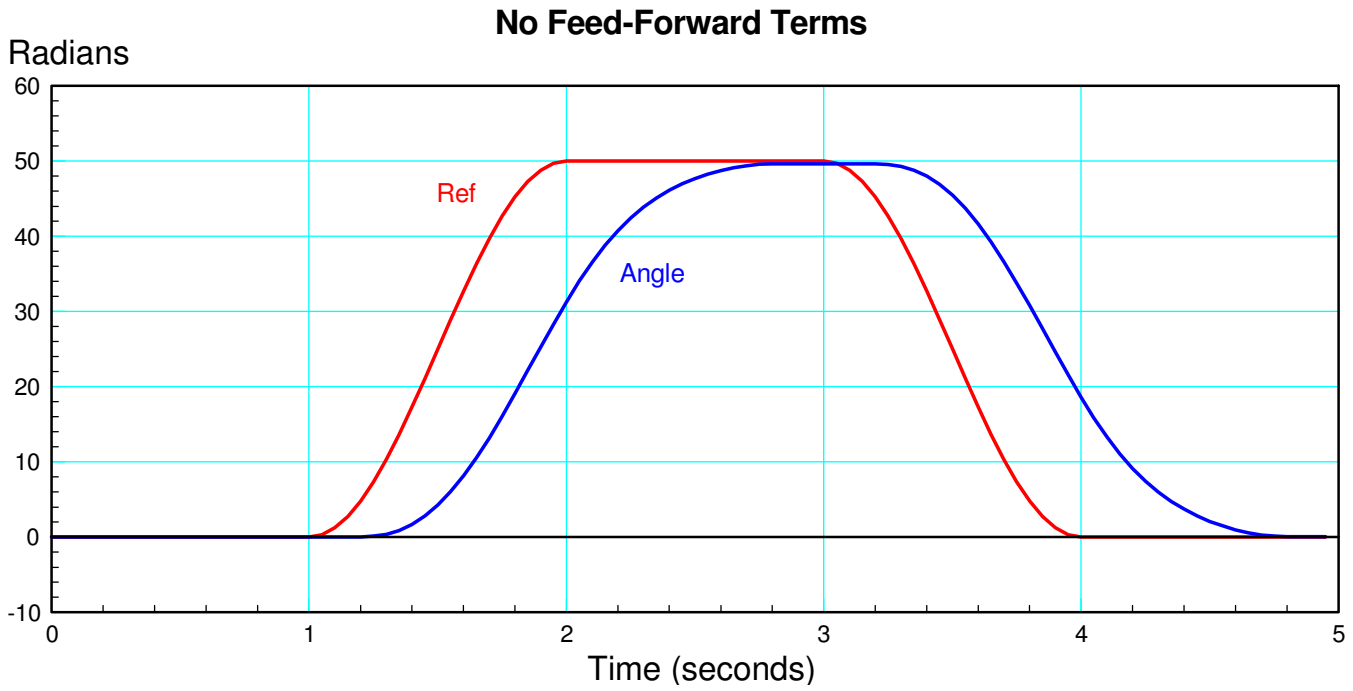
Voltage required to produce path r(t)

The feed-forward voltage tells you a couple of things:

- The max and min voltage is less than the power supply limits (+/- 13.4V). This path should be achievable.
- You *could* make the system slightly faster - there is still some room between the maximum voltage and the power supply limit

Finally, results on the actual motor. In the figure below, a lead compensator is used to supply the voltage to the motor without any feedforward terms:

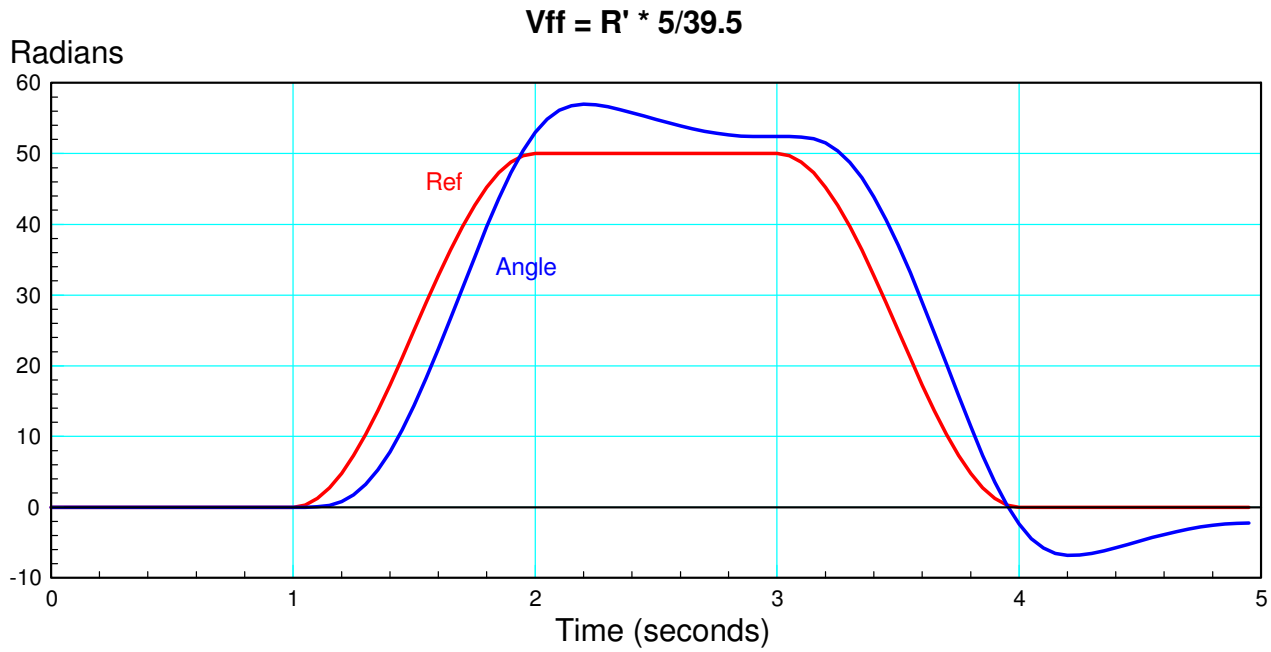
$$V = \left(\frac{k(s+5)}{s+10} \right) E$$



Tracking with Lead Compensation and No Feed-Forward Terms

If you add a term proportional to the derivative of the set point, r(t), tracking improves:

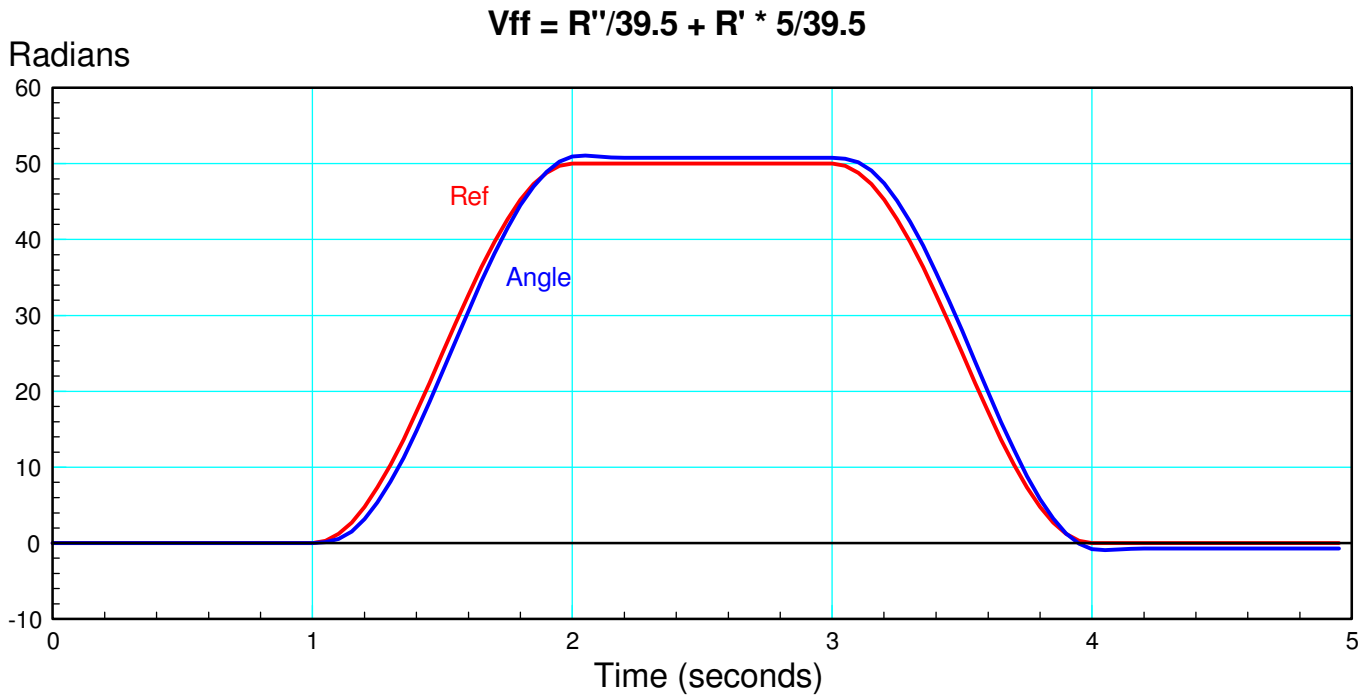
$$v(t) = \left(\frac{k(s+5)}{s+10}\right)E + \left(\frac{5s}{39.5}\right)R$$



Tracking improves if you add a feed-forward term proportional to the derivative of r(t)

Finally, adding a term proportional to the second derivative results in almost perfect tracking:

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



Tracking is almost perfect if you add terms proportional to r'' and r'

Summary

Once you have

- An optical encoder to measure a motor's angle and
- An H-bridge allowing you to adjust the motor's input using PWM

you can control the angle of a motor using output feedback.

- PD and lead compensators work better than P controllers
- PD or lead compensation along with feed-forward control works even better.

In order to use feed-forward control, however, you need to define a reference signal which has finite fist and second derivatives (path planning).

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/