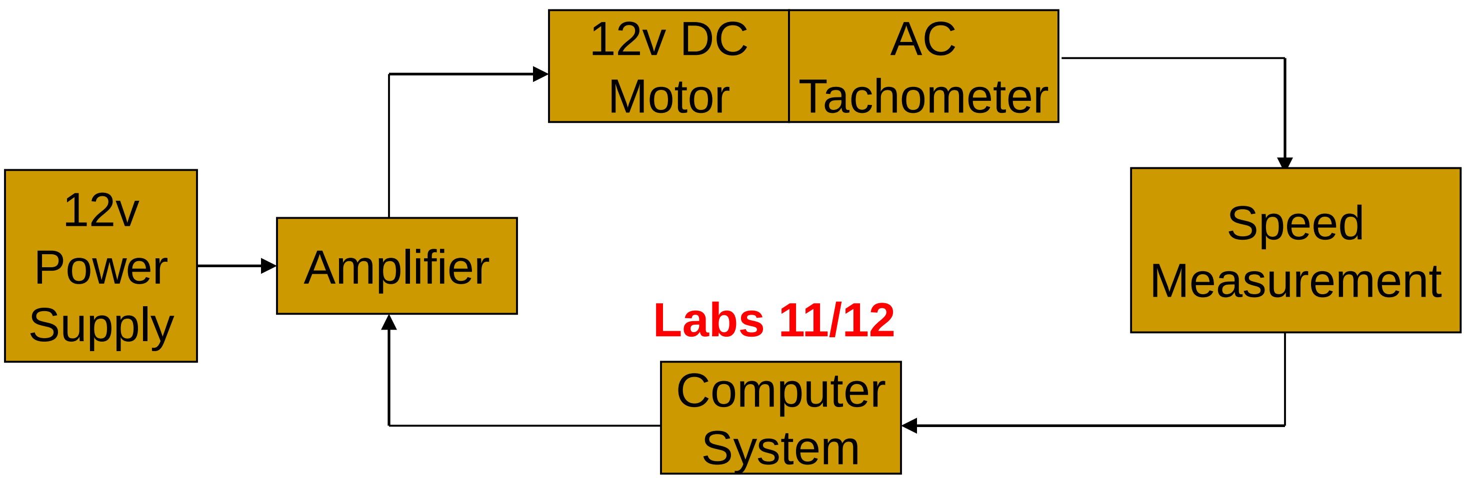
Lab 12. Speed Control of a D.C. motor

Controller Design

Motor Speed Control Project

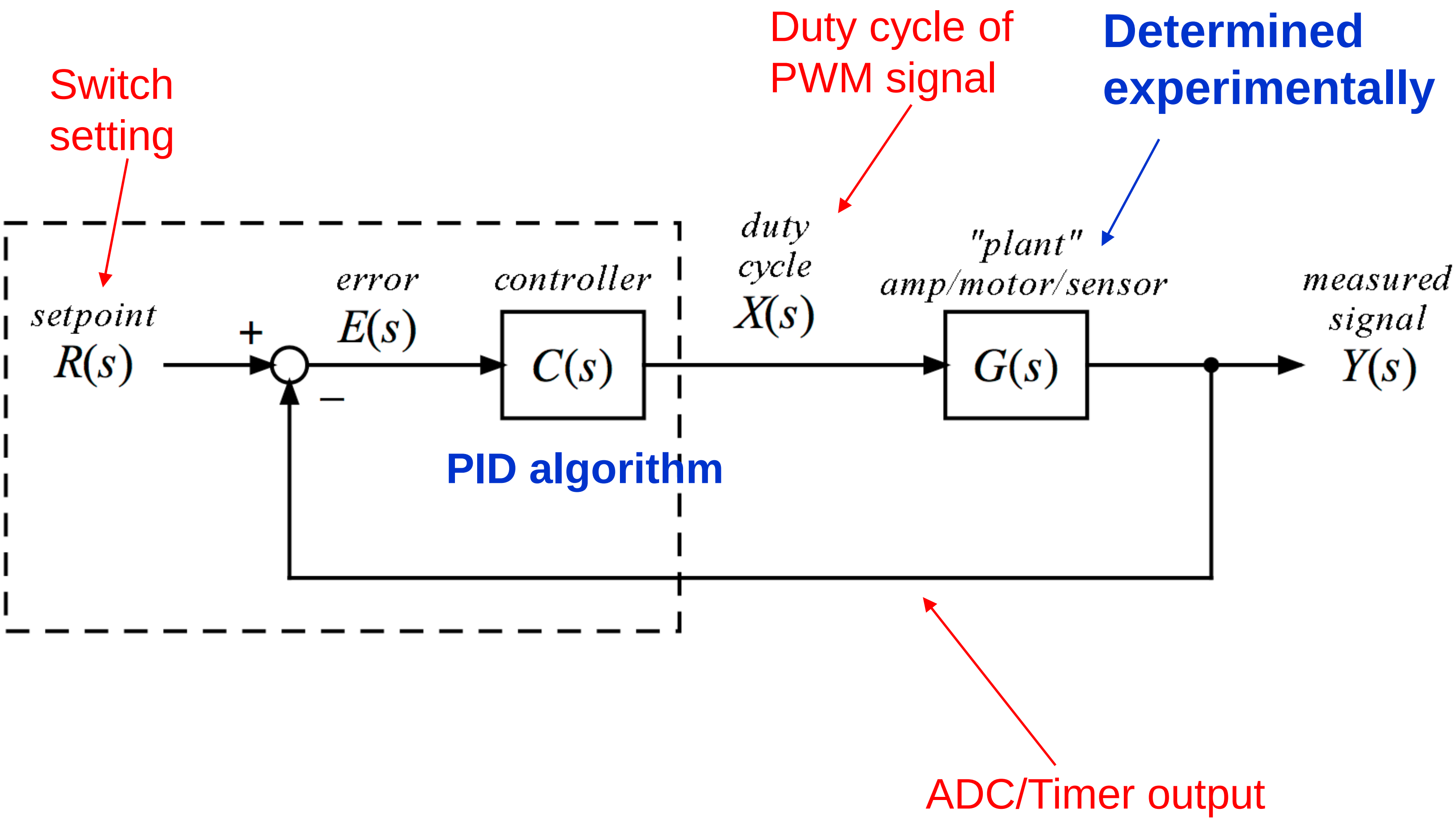
1. Generate PWM waveform
2. Amplify the waveform to drive the motor
3. Measure motor speed
4. Measure motor parameters
5. Control speed with a PID controller



Technical goals for lab

- Design a PID controller to regulate the motor speed
  - Design in Simulink
  - Implement in software
- Demonstrate improvement in motor performance with the PID controller
  - Faster rise time (50% or more)
  - No steady-state error
  - Minimal overshoot of target speed
  - Fast settling time when changing speeds

Simplified system model



PID Controller

- Controller computes "control action" *a(t)*
  - PWM signal duty cycle
- Compensate for error, *e(t),* between set point and measured speed
- PID = *Proportional, Integral, Derivative*
  - Control action *a(t)* based on three "terms":

    P term - proportional to *e(t)*

    I term - proportional to the integral of *e(t)*

    D term - proportional to the derivative of *e(t)*

PID Controller Design

Continuous time domain:

$$a(t) = K_P e(t) + K_I \int_0^t e(t)dt + K_D \frac{de(t)}{dt}$$

e(t) = error detected at time *t*
a(t) = control action computed at time *t*
K_P, K_I, K_D = constants

LaPlace transform/Controller transfer function:

$$C(s) = \frac{A(s)}{E(s)} = K_P + \frac{K_I}{s} + K_D s$$

PID Controller Design

Discrete time domain:

$$a(nT) = K_P e(nT) + K_I \sum_{i=0}^{n} \frac{e(iT) + e(iT - T)}{2} + K_D \frac{e(nT) - e(nT - T)}{T}$$

T = sampling time
n = sample number
e(nT) = error computed at nth sampling interval
a(nT) = control action computed at nth sampling interval
K_P, K_I, K_D = constants

z transform:

$$C(z) = \frac{A(z)}{E(z)} = K_P + K_I \frac{T}{2}\left[\frac{z+1}{z-1}\right] + K_D \frac{1}{T}\left[\frac{z-1}{z}\right]$$

Proportional Control

- Controller produces a control action that is proportional to the error at any given time

$$a(t) = K_p e(t)$$

- Advantages:
  - Simple to implement
  - Larger K_p causes greater system response to a given error $e(t)$ (possibly improve system performance)
- Disadvantages:
  - some steady-state error is required to have a control action
  - can overshoot set point and possibly produce unstable behavior
  - controller reacts to high-frequency “noise” in $e(t)$

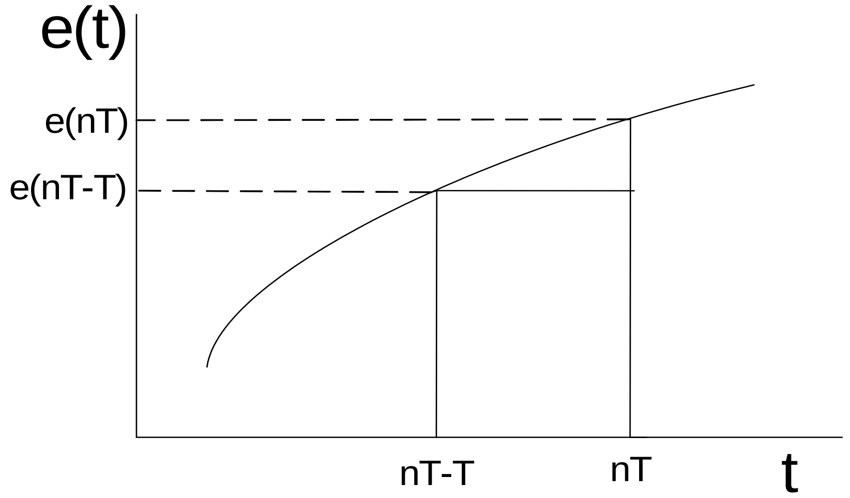
Integral Control

- Controller produces a control action proportional to the integral of the error (area under the error curve)

$$a(t) = K_I \int_0^t e(t)dt$$

- This term determines the steady-state control action when *e(t)=0* (P and D terms are both 0)
- Advantages:
  - eliminates steady state error
  - dampens response to high frequency noise on *e(t)*
- Disadvantages:
  - slows system response
  - can contribute to overshoot

Convert integral term to discrete form



Compute the area under the curve

Consider integral up to previous and current sample times:

$$y(t) = K_I \int_0^t e(t)dt = K_I \int_0^{nT-T} e(t)dt + K_I \int_{nT-T}^{nT} e(t)dt$$

$$y(nT) = y(nT-T) + \Delta y(nT, nT-T)$$

area up to
t = nT-T

incremental area
from *(nT-T)* to *nT*

Convert integral term to discrete form (2)

Area under the curve between *(nT-T)* and *nT:*
(approximate as a trapezoid)

rectangle part triangle part

$$\Delta y(nT-T) = K_I T e(nT-T) + K_I (\tfrac{1}{2}) T [e(nT) - e(nT-T)]$$

$$= \frac{K_I T}{2} [e(nT) + e(nT-T)]$$

Add this to *y(nT-T):*

$$y(nT) = y(nT-T) + \frac{K_I T}{2}\left[e(nT) + e(nT-T)\right]$$

Discrete transfer function of the integral term

Difference equation:

$$y(nT) = y(nT-T) + \frac{K_I T}{2}\left[e(nT) + e(nT-T)\right]$$

z transform:

$$Y(z) = Y(z)z^{-1} + \frac{K_I T}{2}\left[E(z) + E(z)z^{-1}\right]$$

Transfer function:

$$\frac{Y(z)}{E(z)} = K_I \frac{T}{2}\left[\frac{1+z^{-1}}{1-z^{-1}}\right] = K_I \frac{T}{2}\left[\frac{z+1}{z-1}\right]$$

Derivative control

- Controller produces a control action proportional to the derivative of the error
  - anticipates direction of error changes
  - can decrease overshoot
  - can dampen oscillatory behavior
  - BUT: increases sensitivity to high frequency noise in $e(t)$

- Normally, derivative term used only in conjunction with P and/or I terms

The discrete form of the D term:

Compute the slope of *e(t)* at the current sample time:

$$y(nT) = K_D \left[\frac{e(nT) - e(nT - T)}{T} \right] = \frac{K_D}{T} \left[e(nT) - e(nT - T) \right]$$

Z transform: $$Y(z) = \frac{K_D}{T} \left[E(z) - E(z) z^{-1} \right]$$

Transfer function: $$\frac{Y(z)}{E(z)} = \frac{K_D}{T} \left[1 - z^{-1} \right] = \frac{K_D}{T} \left[\frac{z - 1}{z} \right]$$

Implementing the PID controller

Combine the discrete P, I and D terms:

$$a(nT) = K_P e(nT) + y(nT - T) + \frac{K_I T}{2}\left[e(nT) + e(nT - T)\right] + \frac{K_D}{2}\left[e(nT) - e(nT - T)\right]$$

Consider the previous sample time:

$$a(nT - T) = K_P e(nT - T) + y(nT - T) + \frac{K_D}{2}\left[e(nT - T) - e(nT - 2T)\right]$$

Solve 2nd equation for *y(nT-T)* and substitute into 1st equation:

$$a(nt) = K_P e(nT) + a(nT - T) - K_P e(nT - T) - \frac{K_D}{T}[e(nT - T) - e(nT - 2T)]$$

$$+ \frac{K_I T}{2}[e(nT) + e(nT - T)] + \frac{K_D}{T}[e(nT) - e(nT - T)]$$

Implementing the PID controller (2)

Simplify by combining terms involving *e(nT), e(nT-T), e(nT-2T):*

$$a(nT) = a(nT - T) + A_0 e(nT) - A_1 e(nT - T) + A_2 e(nT - 2T)$$

A_0, A_1, A_2 are constants
e(nT), e(nT-T), e(nT-2T) are the 3 most recent error values
a(nT-T) is the previous control action

Software procedure at each sample time:

- Sample speed and compute error *e(nT)*
- Calculate new control action: 3 multiply, 2 add, 1 subtract
- Update duty cycle with new control action
- "Delay" error values to get ready for next sample

Other conversion approaches

- Can use Matlab to perform the conversion
- See the lab write up for detailed explanation

Some practical issues

- Interrupt service routine with PID calculation time cannot exceed interrupt period
  - Pre-compute all constants (*A0, A1, A2*)
  - Avoid floating-point (real #) operations
  - Represent fractions as ratio of integers
  - Denominator of the form 2^k allows shift instead of divide

<u>Example:</u> $A_0e(nT)$, where A_0=0.312

0.312 = 312/1000 = 80/256 (approximately)

$A_0e(nT) = 80 * e(nT)/2^8$ = (80*enT)>>8 (in C)

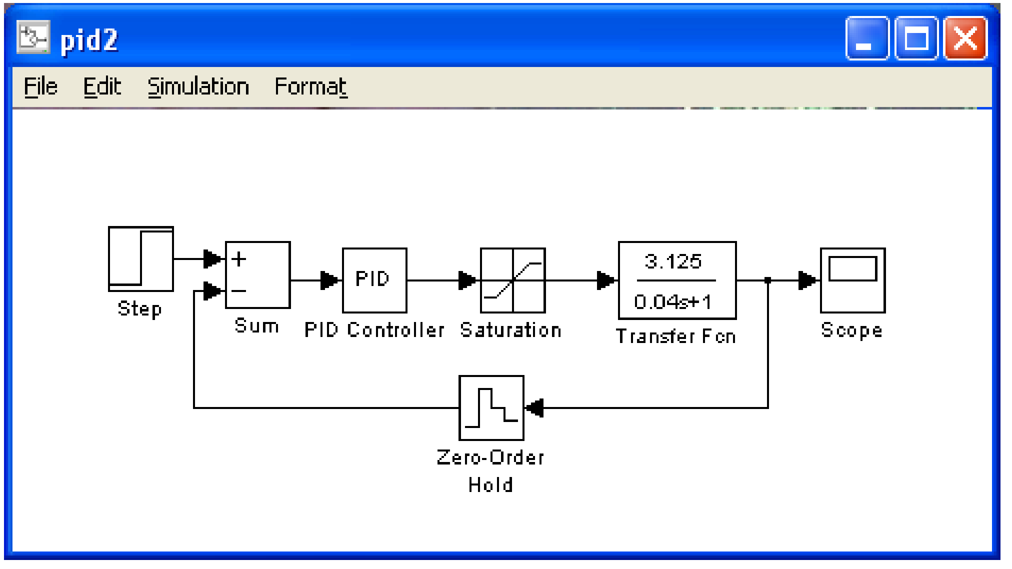
More practical issues

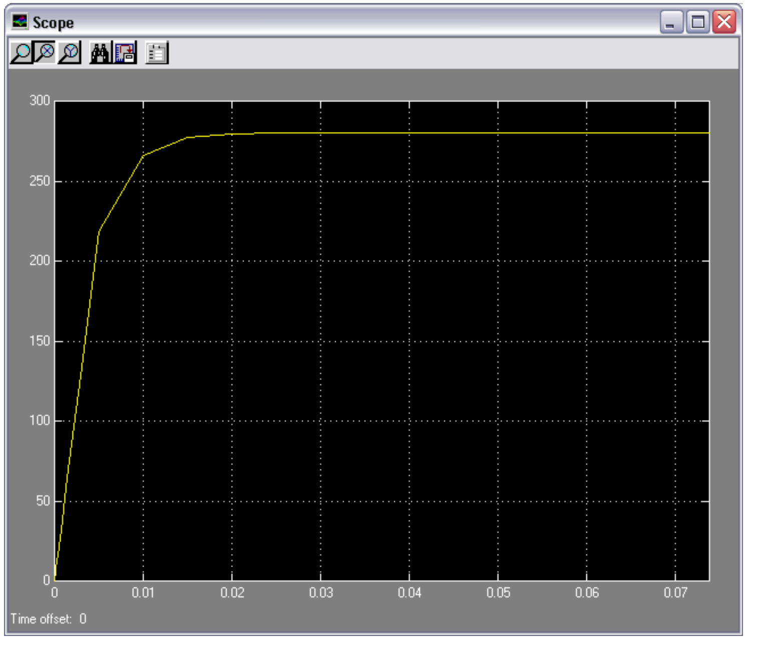
- Duty cycle cannot exceed 100%, nor go below 0%
  - Saturate values in Simulink model
- System operation is discrete, not continuous
  - Use zero-order hold for speed in Simulink model
- Simulink simulation gives OK starting values for constants, but real system usually varies from the model

Test program requirements

- Eight switch-selectable settings – stopped and seven increasing speed values.
- Respond to speed setting changes at any time while the motor is running (without stopping the motor)
- Respond to changes in motor load to return speed to the selected setting (we won't test this)
- Rise/fall times at least 50% faster than the uncompensated motor
- No steady state error
- Minimal overshoot of the desired speed while responding to a change
- Fast settling time after responding to a change

Design the controller in Matlab/Simulink





Select P-I-D constants to produce the desired response.

- Start with P value to improve response time
- Use I term to eliminate steady-state error
- Use D term to further improve response

Lab Procedure

- Re-verify hardware from previous labs

  Note that circuits can still be damaged with incorrect connections/operation!
- Design your PID controller in Matlab/Simulink (determine the P-I-D constants)
- Modify the software to implement the PID controller
- Test the controller by measuring responses to step inputs
- Compare the compensated and uncompensated step input responses