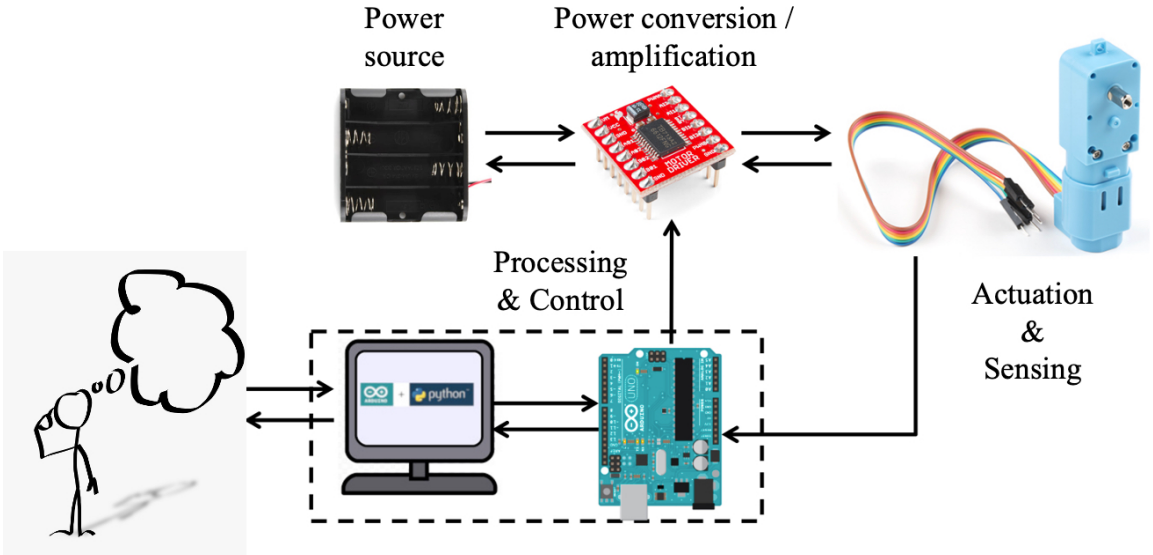
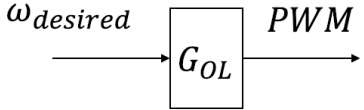

Power source
Power conversion / amplification
Processing & Control
Actuation & Sensing
python

Recap on OL control

You experimented with open-loop control of your PMCD motor shaft speed and position.



You should have studied two different ways to arrive at an open loop algorithm G_{OL} for specifying the PWM command given a desired motor shaft speed.

Recap on OL control (cont.)

⇀ $G_{OL} = K$, a constant gain defined using measured speed and PWM values from testing

⇀ An algorithm based on your motor torque-speed curve model parameters

You also studied how well you could position your output shaft by running at controlled speed over specified time periods.

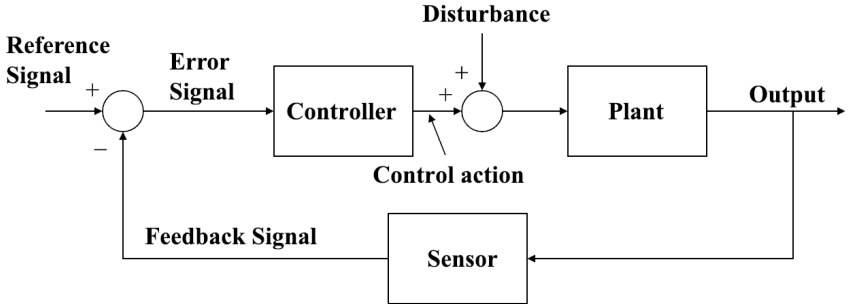
In the next week, you will study how feedback control can be implemented and how effective it is for speed control of this PMDC motor.

Summary of these slides

1. Review closed-loop control diagram
2. Show you how to model feedback control of your PMDC motor
3. Show you how to simulate your feedback control in Python
4. Discuss what you will do in lab

Closed-Loop Feedback Control Diagram

Standard diagram and terminology is conveyed in this diagram:



Plant - any physical system to be controlled
Controller - can generate inputs to the plant to achieve a desired objective
Sensor - means by which plant output is transformed to feedback information

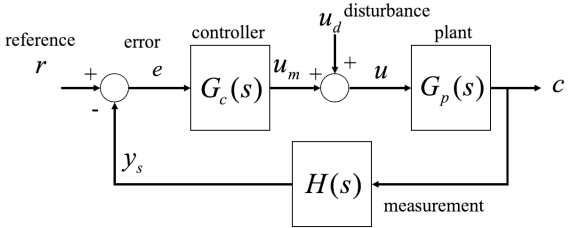
Effect of closing the loop

Advantages:
Provides for disturbance rejection
Reduces sensitivity to parameter variations
Use error signal for dynamic tracking
Enhance accuracy, extend bandwidth, etc.

Disadvantages:
Can lead to excess oscillation or instability
May require tuning and/or use of a model to properly select and design control

For linear cases, you can derive a closed-loop transfer function

A closed-loop transfer function (CLTF) is derived using block diagram algebra and relates the output (controlled variable) to the inputs (reference, disturbance).



$$c = \frac{G_c G_p}{1 + G_c G_p H} \cdot r + \frac{G_p}{1 + G_c G_p H} \cdot u_d$$

This relation allows you to study how the CL system responds to a reference input (r) as well as to disturbances (u_d).

Basic control actions based on error

Proportional (P) control generates a control action proportional to error.

$$u_p = K_p e; \quad \text{and in s} - \text{domain} \Rightarrow G_c = K_p$$

Integral control reduces or eliminates steady-state error, but has reduced stability.

$$u_i = K_i \int e dt; \quad \text{and in s} - \text{domain} \Rightarrow G_c = K_i/s$$

Derivative control yields an increase in effective damping, improving stability.

$$u_d = K_d \cdot \frac{de}{dt}; \quad \text{and in s} - \text{domain} \Rightarrow G_c = K_d s$$

PID control is most common in practical application.

$$u = K_p e + K_i \int e dt + K_d \frac{de}{dt}$$

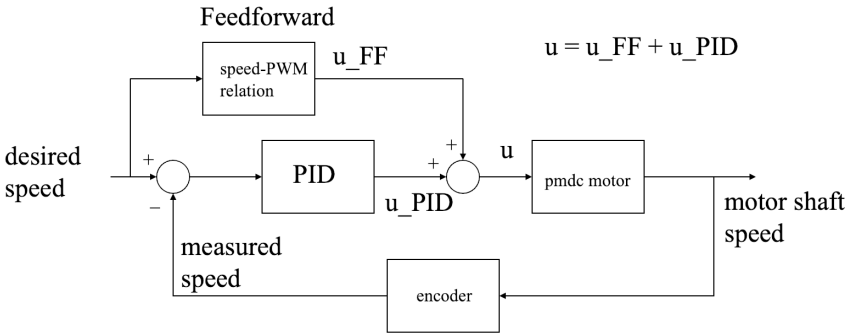
Expect to see these control actions implemented in various ways.

A typical digital implementation

Let k be the iteration in a loop, then for x the variable to be controlled and r the reference (or set point), and k the loop iteration:

Error calculation	$e(k) = r(k) - x(k)$
Proportional control	$u_p(k) = K_p e(k)$
Integral control	$u_i(k) = K_i \sum_{i=1}^{k} (e(k) - e(k-1))/2$
Derivative control	$u_d(k) = -K_d [x(k) - x(k-1)]$
PID control	$u(k) = u_p(k) + u_i(k) + u_d(k)$

Combining Open-Loop (as Feedforward) and PID Feedback



⋆ Because of the high stiction torque in the small hobby PMDC motors we are using, it is helpful to include feedforward with feedback. In this way, feedforward does the 'heavy lifting' and we allow the PID control to compensate for errors due to model and disturbance effects.

PMDC motor model and simulation

See the posted python code on Canvas lecture page. All programs require the RK4 module rk.py also found on the lecture page (and used in previous labs).

⇒ **pmdc_motor_PID_sim.py** - this program builds on the programs we used for motor model and open-loop simulation, but now a PID algorithm is included.

⇒ See next slide 12 for image of key Python code and slide 14 for results generated by this simulation, and compared to experiments.

Note that now the `mcc` value is set by adding the OL (FF) and PID controls.

PMDC motor - FF + PID control simulation - 1

The simulation allows you to 'turn on' a reference speed command during a given time period. Here the torque-speed algorithm is shown to specify the FF control as well.

```
66      if (tc>ton) and (tc<toff):
67          omega_r = omega_d
68          rps_r = omega_d/(2*math.pi)
69          uff = (Rm*(Tmo*math.tanh(omega_d/500)+Bm*omega_d)/rm + rm*omega_d)/vb
70      else:
71          omega_r = 0
72          rps_r = 0
73          uff = 0
```

You can imagine using a simple structure like this to provide a schedule for how different speed commands are given, possibly based on different conditions or inputs.

PMDC motor - FF + PID control simulation - 2

The PID algorithm used here follows the algorithm given on slide 9:

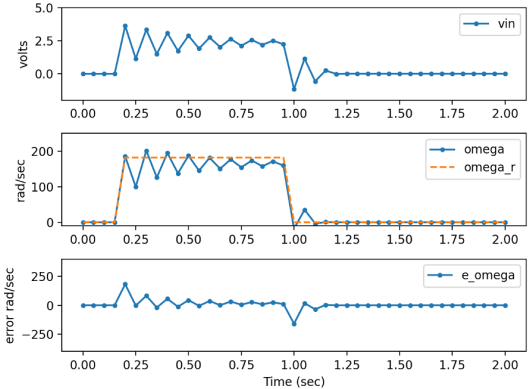
```
75      omegar.append(omega_r) # save the reference / command
76      rpsc = omegac/(2*math.pi) # find current omega in rev/sec
77      e_omegac = omega_r - omegac # error in rad/sec
78      e_rpsc = e_omegac/(2*math.pi) # error in rev/sec - as in Arduino code
79
80      sum_e = sum_e + (e_omegac + e_omega[i-1])/2
81      sum_erps = sum_e/(2*math.pi) # scale sum_e to rev/sec units
82      # select control either with rad/sec or rps (rps is used in Arduino)
83      # going with rps is meant to see if I can get gains to match with
84      # the experimental setup
85      u = Kp*e_omegac + Ki*sum_e - Kd*(omegac - omegac_p) # PWM based on rad/sec
86      u = Kp*e_rpsc + Ki*sum_erps - Kd*(rpsc - omegac_p/(2*math.pi)) # PWM based on rev/s
```

Note that the values are computed here in SI units (rad/sec) as well as rev/sec (rps), since the latter may be used in Arduino implementation. This allows this model simulation to be used for tuning the gains as closely as possible to those finally implemented.

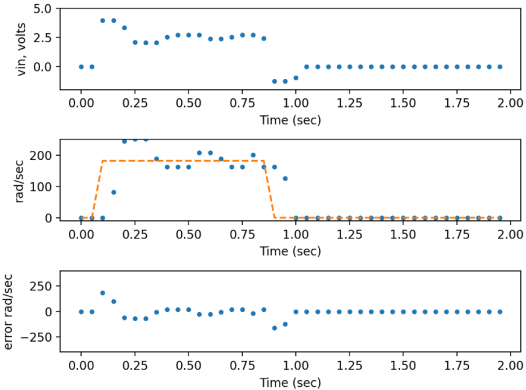
Simulation (left) vs Experiment (right) with FF + PID control

The results compared below used *identical* PID control gains.

Simulation – FF + closed-loop control



Experiment – FF + closed-loop control



Summary

As we did for OL control, we can simulate now feedback control of the PMDC motor. Here we looked at combining the OL (as feedforward, FF) and PID controls.

Combining FF + PID allows you to control a system using a relatively good model with OL, reducing the burden of the PID so it just compensates for model and disturbance errors.

In some systems you may not need the FF with PID as shown here. In simulation as well as in lab you should experiment to see how well PID performs without the help of FF.