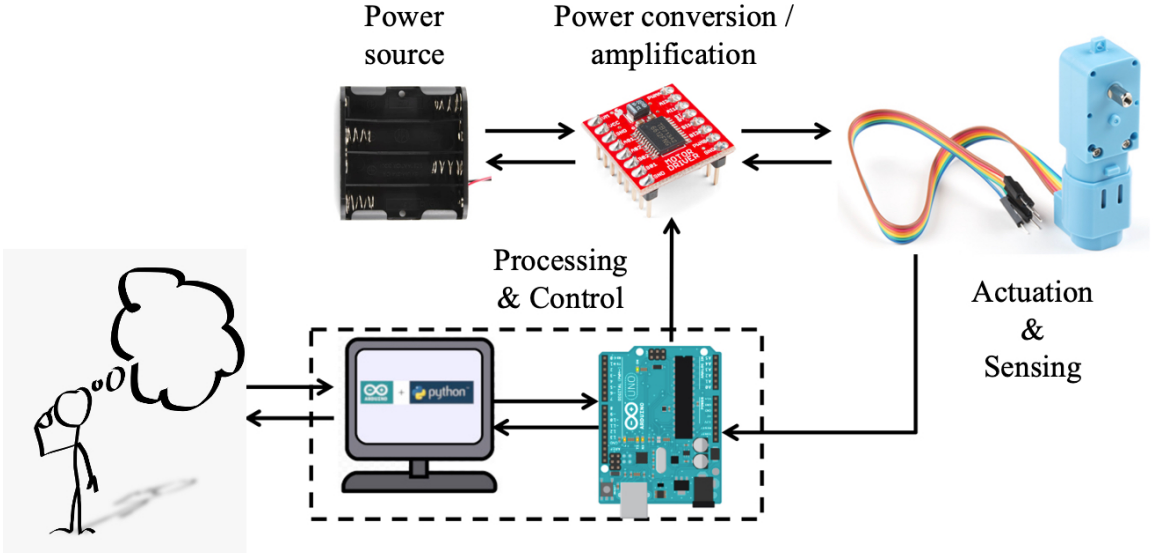

Power source
Power conversion / amplification
Processing & Control
Actuation & Sensing
python

Summary of these slides

1. Review torque-speed curve as steady-state model for PMDC motor
2. Briefly introduce concepts of closed- and open-loop control
3. Derive open-loop control approaches for PMDC
4. Review model simulation of open-loop control of PMDC motor
5. Sample results of open-loop control of PMDC motor

Recap on PMDC motor modeling and testing

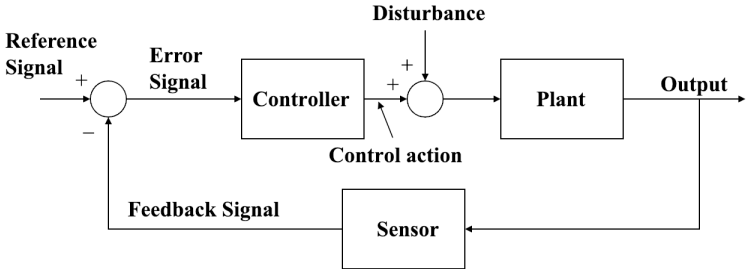
This past week, you conducted experiments with a PMDC motor, and used the measured data to estimate model parameters. These model parameters can be used in the torque-speed curve,

$$\therefore T_o = \frac{r_m}{R_m} v_{in} - \left[\frac{r_m^2}{R_m} + B_m \right] \omega_m$$

In this next week, we begin study of control methods, starting with open-loop control based on this model of the PMDC motor.

Closed-Loop Feedback Control Diagram

Feedback control relies on a measured feedback signal to form an error signal, as shown below. The error signal is used by the controller to generate a control action. The reference signal is used to issue a command.



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

Open loop control relies on a model without feedback

Say you had a simple system with model, $y = f(x)$, then open loop says:
to achieve y_r, just invert the model to find the x needed to achieve that value

$$x = f^{-1}(y_r)$$

This can be done with any model that you can invert, even a dynamic system.

The drawback, of course, is that if the model is not good, ignores effects that can affect the output, or if model parameters change, then the results and error between what you want to achieve and what you actually get can get large.

These are disadvantages that closed-loop control can eliminate. Nevertheless, we can get often get good results so will study open loop control before closing the loop.

Testing open-loop control with the PMDC motor

In lab, you will implement open-loop *speed* control of the PMDC motor.

The open-loop control can be evaluated two ways.

First, simply compare the mean speed achieved (e.g., over an observed time period) with the speed command and find the error values.

A second way is to drive the motor at constant speed over a known time interval, T, which should result in an angle, $\Delta\theta = \omega_s T$, where $\omega_s = \omega_m / GR$ is the output shaft speed. A quick check is to compare $\Delta\theta$ expected with a rough measure of what is achieved. It is easy to assess 180 or 360 degree turns.

Open-loop control approaches for the PMDC motor

A quick approach for open-loop control in this case is to use ω_m vs PWM data, and to simply build a relation, $PWM = K_m\omega_m + \text{constant}$. The K_m has units of $1/(\text{speed units})$. Recall there is a lower limit on PWM for these motors.

Another way to derive an open-loop control is to use the torque-speed curve (from slide 3), as follows.

1. Recall, $v_{in} = R_m i_m + r_m\omega_m$, and $v_{in} = mcc \cdot v_b$.
2. At steady-state, $T_m = r_m i_m = B_m\omega_m + T_{mo}$, so we can replace i_m.
3., Set ω_m as the 'desired' or reference value, ω_{md}.
4. Solve for mcc (see on next slide)

Open-loop control approaches for the PMDC motor (cont.)

The result is,

$$mcc = \left[R_m \left(T_{mo}\text{sgn}(\omega_{md}) + B_m\omega_{md}\right)/r_m + r_m\omega_{md}\right]/v_b$$

By including T_{mo} to estimate the Coulomb torque, this relation can model how a minimum mcc is needed.

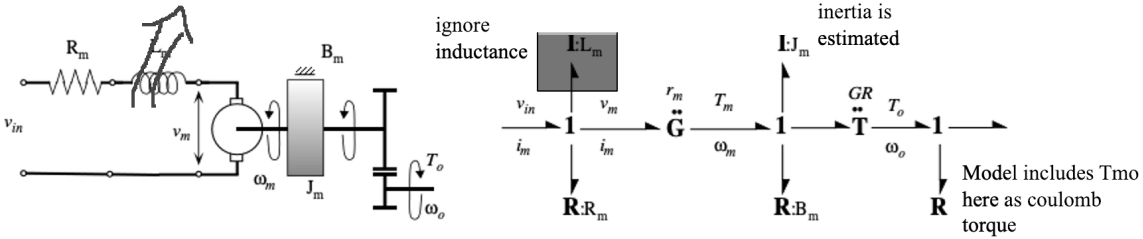
The value of T_{mo} does need to be estimated during testing. If it is not known, its value can be adjusted so it matches the motor behavior.

Lastly, for model simulations, mcc can be used directly but when used in a real implementation for the H-bridge used in this lab, $PWM = mcc \cdot 255$.

First, let's test this in a simulation of the PMDC motor model.

PMDC motor model and simulation

The diagram below shows the dynamic model of the PMDC, with inductance ignored.



The dynamic state equations for the PMDC can be shown to be:

$J_m\dot{\omega}_m = -B_m\omega_m - T_{mo}\text{sgn}(\omega_m) + r_m i_m$

where,

$i_m = (v_{in} - r_m\omega_m)/R_m$ We also include an equation, $\dot{\theta}_m = \omega_m$, to solve for the angle of rotation. Then the output shaft angle is, $\theta_s = \dot{\theta}_m/GR$.

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).

1. **pmdc_motor_sim.py** - this program models just the PMDC motor. The input voltage is turned on and off at known times.

⇒ See next slide 12 for results generated by this simulation.

2. **pmdc_motor_OL_sim.py** - this program builds on the program in (1) but includes the open-loop control concept described in the previous slides. In this case, the PWM is turned on for a time interval.

The time interval is determined based on the desired rotation of the output shaft, θ_d, and the motor speed setting, ω_d; i.e., $T = \theta_d/\omega_d$.

⇒ See slide 14 for results generated by this simulation.

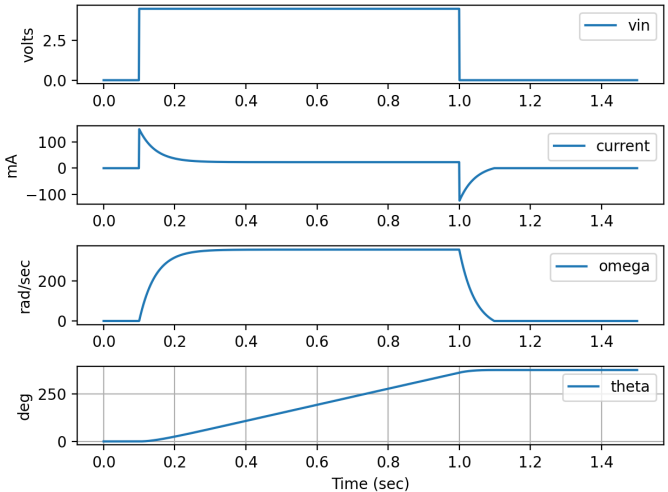
Simulation of PMDC motor - simple on/off

Notes:

1. Simple on/off of input voltage; no H-bridge model
2. The model allows computing the current, im
3. the states are motor shaft speed and angle
4. the 'y' lets you output computed values like current (im) and the current voltage (vinc)

```
# define the system ODEs
def pmdc(x, t, rm, Rm, Bm, Jm, To, ton, toff, vb):
    # the two states are shaft speed and position
    omegam, thetam = x[0], x[1]

    # this model has a voltage that turns on and off
    if (t>ton) and (t<toff):
        vinc = vb
    else:
        vinc = 0

    im = (vinc - rm*omegam)/Rm
    omegamdot = (rm*im - Bm*omegam - To*np.sign(omegam))/Jm
    thetadot = omegam

    # specify outputs
    y = [im, vinc]

    return np.array([omegamdot, thetadot]), y
```

Simulation of PMDC motor - simple on/off



Simulation of PMDC motor - OL control

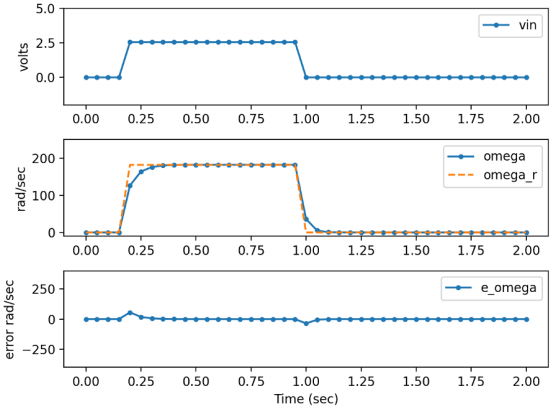
Notes:

1. A while loop is used just like the Arduino control code.
2. The while loop time step is 'dt', which should be set equal to the time step in the actual Arduino loop.
3. H-bridge output is modeled by
$\Rightarrow$ `vinc = mcc*vb`,
using the `mcc` value set by the OL algorithm previously discussed.
In Arduino code, remember
`PWM = mcc*255`

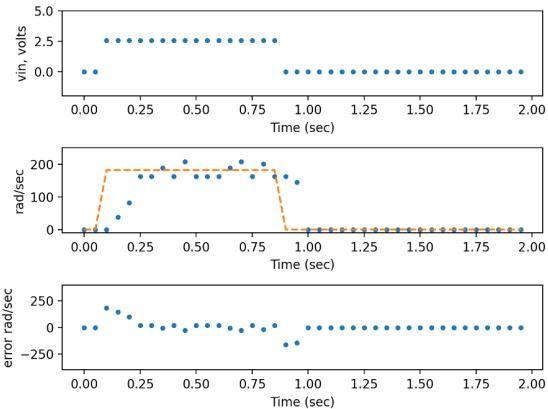
```
81  while tc < 2:
82      # determine next step
83      if (tc>ton) and (tc<toff):
84          # look at slides to get explanation for this next line:
85          omega_r = omega_d
86          mcc = (Rm*(Tmo*math.tanh(omega_d)+Bm*omega_d)/rm + rm*omega_d)/vb
87      else:
88          mcc = 0
89          omega_r = 0
90
91      # Simple model of H-bridge voltage conversion
92      omegar.append(omega_r) # store the reference value of omega for plotting
93      vinc = mcc*vb # this is voltage input to armature of motor
```

Simulation vs Experiment Results with OL control

Simulation – open-loop control

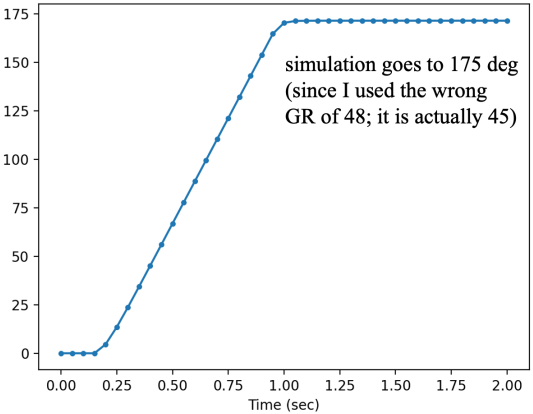


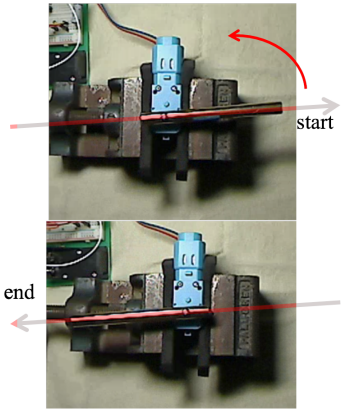
Experiment – open-loop control



Open-loop control of the angular position of motor shaft

Results on left show the output motor shaft rotation from open-loop simulation, and on the right are shown images of start and end positions in an experiment.





Summary

To prepare for lab, you will study the open-loop control simulation described in these slides.

The open loop algorithm derived from the torque-speed curve (slide 8) gives mcc for a desired motor speed.

In lab, you can use this same algorith on Arduino, with $PWM = mcc \cdot 255$ (for our H-bridge).

Python code will also be provided to help you control experiments and collect data from the Arduino.