ECE 557: *Control, Signals, and Systems Laboratory*

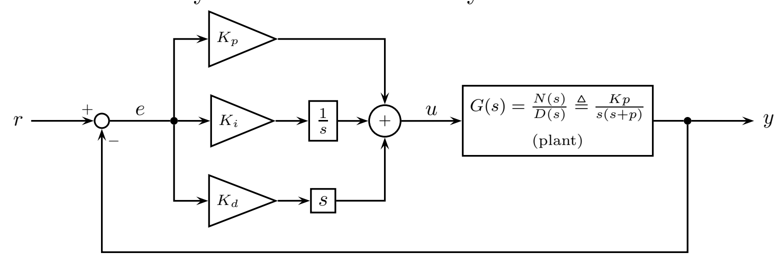
Notes for Lab 7 (Tuning a PID Controller)

1. Return lead compensation pre-lab and give some notes.
   - Optimization (calculus) is key to 1(a).
   - Sometimes wrong form of compensator was recorded on paper (e.g., $(s+a)/(s+b)$ instead of $(1+s/a)/(1+s/b)$), but apparently correct form was used in `rltool`.
2. Return lag compensation lab reports and give some notes.
   - Try to plot expected (i.e., theoretical) data on top of measured data (for comparison).
   - Even if capture time is long, **zoom in** on interesting data (e.g., step edge).
   - Post-lab questions ask about Bode steady-state error and lag compensator speed (relative to gain) in general.
3. In simulations, force a time vector (`help step`) or use **fixed-step** *Simulink* methods with small steps.
4. **REMINDER:** *Lab 8* **AND** *lab 9* next week.
   - Complete **both** prelabs. Both are PID labs, but they use different plants.
5. Some notes on proportional–integral–derivative (PID) control.
   - Assume plant can be well modeled by 2^{nd}-order system.
   - Gain compensation alone:
     - Decreases rise time (i.e., increases bandwidth (ω_d)).
     - Decreases error (i.e., it amplifies error input, which increases feedback response).
     - Gives little control over damping (i.e., settling (σ) largely determined by plant).
   - Lag compensation shifts root locus toward DC (i.e., toward $s = 0 + j0$):
     - Relatively high DC gain gives low error even with low gain (i.e., damping *ratio* improves).
     - Relatively low AC gain slows down system (i.e., low rise time (ω_d) and settling time (σ)).
   - Lead compensation shifts root locus toward leftward (i.e., toward $s = -\infty + j0$):
     - Relatively high AC gain increases speed (i.e., high damping (σ) means quick transients).
     - Increased phase improves stability margins (relates to high damping ratio ζ).
     - Higher speed at lower gain greatly improves damping ratio (high σ for low ω_d means high ζ).
   - Old methods have three degrees of freedom (i.e., gain, pole–zero center, pole–zero width).
   - PID uses tunable gains to give three degrees of easily implementable freedom.
     - Adds an integrator and two zeros.
     - Zeros act as "targets" for root locus.
     - Diagonal movement provides control flexibility (i.e., gain, rise time/error, damping).
     - Gains are easy to build and tune **in the field.**
     - **Adaptive PID controllers** tune their own gains.
   - Derivative term causes some problems.
     (i) True differentiator is impossible to build. Zeros outnumber poles, and so it's *non-causal.*
     (ii) Error signal is not differentiable at instant of step input.
     (iii) Gain that increases with frequency amplifies high-frequency noise.
       - High frequency oscillations can damage plant.

     So we apply *low-pass filter* $a/(s+a)$ to **output** derivative. Make corner a relatively large.
     - System is causal, but LPF impulse response rings control at every input quantization step.
       * Each impulse peaks at $\sim(2\pi/1024) \times K_d\, a$.
         · So differentiator can cause clipping.
         · If K_d is too large, can cause *dangerous* chatter or oscillations. So keep $K_d < 1$.

6. PID systems theory with derivative approximation: Making theory match reality.

   - The prototypical PID control system for our laboratory looks like:



     (i) The r-to-y and r-to-u transfer functions are:

$$\frac{Y(s)}{R(s)} = \frac{\left(K_d s^2 + K_p s + K_i\right) N(s)}{\left(K_d s^2 + K_p s + K_i\right) N(s) + sD(s)} \quad \text{and} \quad \frac{U(s)}{R(s)} = \frac{\left(K_d s^2 + K_p s + K_i\right) D(s)}{\left(K_d s^2 + K_p s + K_i\right) N(s) + sD(s)}.$$

     As expected, these two transfer functions have the same (three) poles and different zeros (i.e., the internal dynamics are the same, but outputs have changed). The output transfer function Y/R has three poles and two zeros, and so it is causal and can be analyzed in `rltool`. However, the control transfer function U/R has three poles and *four* zeros, which makes it *non-causal*. So we can predict what a step response **would** look like **if** such a system could exist, but we cannot implement the system.

     (ii) The derivative of e does not exist at the instant of a step, and so *Simulink*'s du/dt block (when using a **fixed-step** `ode45`/`ode5` solver) will ignore that point. Consequently, the *Simulink* and `rltool` r-to-y step responses will differ.

     (iii) Even without the causality and differentiability issues, a differentiator amplifies the normally benign high frequency oscillations from measurement noise.

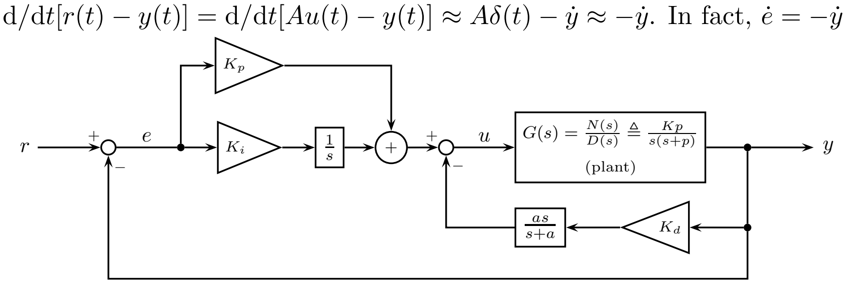
   - A first attempt to solve both problems is to replace the differentiator s with the "filtered differentiator" $as/(s+a)$ that has $a \gg 0$.

     (+) The control transfer function U/R is now *causal* and thus *realizable*.

     (−) When r is a unit step and $y(0-) = 0$, the differentiator places a K_d-*impulse* into the $a/(s+a)$ filter, which initially peaks at $K_d a \gg 0$ (i.e., impulse response is $K_d \, a \, e^{-at}$).

       * So $u(0) = K_p + K_d a$, which is very large and can damage plant (or cause saturation).
       * To keep $u(0)$ small, both a and K_d must be very small. Making a small makes the differentiator approximation poor, and making K_d small reduces control flexibility.
       * Intuitively, K_p should determine the available control effort and not K_d.

   - Second attempt: use filtered $a\,s/(s+a)$, but connect to $-y$ instead of e. For *regulation* (i.e., step input), $\dot{e} = \mathrm{d}/\mathrm{d}t[r(t) - y(t)] = \mathrm{d}/\mathrm{d}t[Au(t) - y(t)] \approx A\delta(t) - \dot{y} \approx -\dot{y}$. In fact, $\dot{e} = -\dot{y}$ for $t > 0$.



     (+) For step (i.e., "often constant") inputs, this control "acts" like PID because $e' = r' - y' \approx -y'$.

     (+) For $a \gg 0$, the r-to-y (and r-to-u) step responses roughly match trajectories from *Simulink*.

     (+) A step input r does not excite the $K_d \, a/(s+a)$ impulse response (i.e., $K_d \, a \, e^{-at}$).

     (+) For $a \gg 0$, $\max\{u(t)\} \approx u(0) = K_p$, which matches intuition.

     (−) The role of K_d is to shape the plant rather than shape the control response.

     (−) Quantization steps from digital measurement of y (i.e., encoder count) puts impulses of size $2\pi/1024 \approx 0.006$ into $K_d \, a/(s+a)$, and so K_d and a should still be picked with care.

- The SISO transfer functions for the r-to-e, r-to-u, and r-to-y systems can be found using the formula
$$\frac{\text{OUT(s)}}{\text{IN(s)}} = \frac{\text{sum of forward paths from } \textbf{in} \text{ to } \textbf{out}}{1 + \text{sum of negative feedback paths from } \textbf{out} \text{ back to } \textbf{out}}.$$
So
$$\begin{aligned}
\frac{E(s)}{R(s)} &= \frac{1}{1+\left(K_p+\frac{K_i}{s}\right)\frac{G(s)}{1+K_d\frac{as}{s+a}G(s)}}\\
&= \frac{1+K_d\frac{as}{s+a}G(s)}{1+K_d\frac{as}{s+a}G(s)+\left(K_p+\frac{K_i}{s}\right)G(s)}\\
&= \frac{s\,(s+a)+K_d\,a\,s^2G(s)}{s\,(s+a)+K_d\,a\,s^2G(s)+(s+a)\,(K_ps+K_i)\,G(s)}\\
&= \frac{s\,(s+a)\,D(s)+K_d\,a\,s^2N(s)}{s\,(s+a)\,D(s)+K_d\,a\,s^2N(s)+(s+a)\,(K_ps+K_i)\,N(s)},\\
\frac{U(s)}{R(s)} &= \frac{\left(K_p+\frac{K_i}{s}\right)}{1+G(s)\left(\left(K_p+\frac{K_i}{s}\right)+K_d\frac{as}{s+a}\right)}\\
&= \frac{(s+a)\,(sK_p+K_i)}{s\,(s+a)+G(s)\,((s+a)\,(sK_p+K_i)+K_d\,a\,s^2)}\\
&= \frac{(s+a)\,(sK_p+K_i)}{s\,(s+a)+K_d\,a\,s^2G(s)+(s+a)\,(sK_p+K_i)\,G(s)}\\
&= \frac{(s+a)\,(sK_p+K_i)\,D(s)}{s\,(s+a)\,D(s)+K_d\,a\,s^2N(s)+(s+a)\,(sK_p+K_i)\,N(s)},
\end{aligned}$$

control signal $u(t)$

r-to-u has same poles as filt. PID

and
$$\begin{aligned}
\frac{Y(s)}{R(s)} &= \frac{\left(K_p+\frac{K_i}{s}\right)\frac{G(s)}{1+K_d\frac{as}{s+a}G(s)}}{1+\left(K_p+\frac{K_i}{s}\right)\frac{G(s)}{1+K_d\frac{as}{s+a}G(s)}}\\
&= \frac{\left(K_p+\frac{K_i}{s}\right)G(s)}{1+K_d\frac{as}{s+a}G(s)+\left(K_p+\frac{K_i}{s}\right)G(s)}\\
&= \frac{(s+a)\,(sK_p+K_i)\,G(s)}{s\,(s+a)+K_d\,a\,s^2G(s)+(s+a)\,(K_ps+K_i)\,G(s)}\\
&= \frac{(s+a)\,(sK_p+K_i)\,N(s)}{s\,(s+a)\,D(s)+K_d\,a\,s^2N(s)+(s+a)\,(K_ps+K_i)\,N(s)}.
\end{aligned}$$
As expected, these three transfer functions have the same poles because they come from the same system. However, they have different zeros because they reflect different outputs.
  - The resulting system cannot be tuned easily in `rltool`.
  - Instead, use the transfer functions directly or use *Simulink*.
  - As long as $K_p \leq 5$ (i.e., the saturation threshold), these transfer functions will closely model laboratory behavior.
  - Other effects that are not modeled include:
    - Random measurement and actuator noise (usually negligible).
    - DAC output quantization error (usually negligible).
    - Mechanical static friction thresholds (usually negligible due to type-1 controller).
    - Shaft encoder quantization error (noticeable control spikes — suppress with small a).
  - Nonlinear effects are easy to model within *Simulink*, but they are difficult to handle analytically. Most of their negative effects are magnified by large choices of a, but small choices of a reduce effectiveness of "differentiator" (i.e., reduce system damping).

7. Complete the *Tuning a PID Controller* lab
    - Implement PID control for position regulation of DC servo.
        - In *Simulink*, choose two Summers from the Math section of the library.
            - On one, change |++ to |+- to make it the error summer.
            - Change the other's |++ to ++- and shape to *Rectangular* to make it the PID output.
        - Do **not** use PID block. Use components from **Math** and **Continuous**.
            - Implement K_p with gain.
            - Implement K_i with gain and **integrator**.
            - Implement K_d with gain and **transfer function**.
                - Use **transfer function** to implement $as/(s+a)$ "derivative+filter." Set $a = 200$.
                - Wire from **output** and *not* error. Control will start far too high otherwise.
                - Make sure you **subtract** eventual result (because we're wiring from *output*).
                - These modifications slow response, but they make derivative *safe* and *realizable*.
        - *If you wish,* wire up a simulated system for comparison. Capture its output as well.
            - You might relate this to using an *observer* (a subject of ECE 650 and ECE 750).
    - Tune your PID gains for $< 2\%$ overshoot and < 0.5 s settling time.
        - Initial output magnitude is K_p. If $K_p > 5$, initial output will be clipped.
        - Quantization noise from encoder makes derivative very noisy. Keep K_d very low ($K_d < 1$).
        - Use numeric inputs in *ControlDesk* for tuning.
            - ⋆ K_p provides control effort and much of rise time/shape (**p** for **p**roportional? **p**otential/**p**ower!).
            - ⋆ K_i reduces error but **i**ntroduces overshoot and lag (**i** for integrator? **i**ntroduce?).
            - ⋆ K_d damps overshoot but introduces error (**d** for **d**erivative? **d**amping!).
        - **Save THREE of your iterations.**
            - Only **one** must fit specifications.
            - The other two should show your grasp of **tuning rules.**
        - While tuning, recall the similar process in the gain compensation lab. Is PID more flexible?
    - You do not need separate controllers for the slow version of system, but keep slow system in mind when analyzing data in report! (e.g., compare expected *slow* response to data)
    - ⋆ AT ANY TIME, IF MOTOR STARTS CLICKING VERY QUICKLY, STOP THE EXPERIMENT — DISCONNECT THE MOTOR IF NECESSARY!! High-frequency switching can cause **permanent damage**! It can be caused by unstable systems (e.g., high gains or positive poles).
    - ⋆ Because the system contributes one integrator and your controller contributes **another integrator**, you should expect steady-state error to decay.
        - Due to controller integrator, the static friction dead zone in motor won't be as much of a problem. Error should *eventually* decay away (until error under ADC LSB threshold).

- Tips:
    - Do **work** out of directory on **local** hard drive — use as MATLAB working directory.
    - In *Simulink*, the hotkey for building a model is Ctrl - B .
    - Start *dSPACE ControlDesk* before doing *Simulink* builds.
    - In MATLAB, change *Termination* settings for DAC block — check box to set 0 V stop value.
    - In *dSPACE* add a simState control.
        (i) Wire simState to 2-option radio button — Setup options "Run" (2) and "Stop" (0).
        (ii) Set Capture Settings to *automatically* restart and set *capture time* to simulation time.

        Restart simulation as needed by using simState control (i.e., no need to change modes).

        (i) To stop early, change simState to Stop.
        (ii) Before restarting, re-initialize Capture Settings by clicking Stop and then Start.
        (iii) When you're ready to start (e.g., after changing gains), set simState to Run.