Massachusetts Institute of Technology

Department of Mechanical Engineering

2.004 *Dynamics and Control II*

Laboratory Session 3: Velocity Control Using P and PI Control Actions

Laboratory Objectives:

(i) Study of the effect of controller gain on transient and steady-state behavior.

(ii) Investigate the elimination of steady-state error through the use of *Integral* (I), and *Proportional* plus *Integral* (PI) control.

Experiment #1: Download all Lab 3 files from Canvas: `https://canvas.mit.edu/courses/10470/assignments/129629`. Open the Arduino code "`velocity_control_template.ino`." Implement PID code that will be used to perform feedback control. In the continuous time domain, the PID control law is defined as:

$$\texttt{PID\_Output} = K_p \cdot e(t) + K_i \int_0^t e(t)dt + K_d \frac{d}{dt} e(t)$$

where the error signal $e(t)$ at each time step is computed as:

$$e(t) = SetPoint(t) - SensorOutput(t).$$

In the main loop you will write code to implement the above PID equation (Lines 162 – 169).

```
    error = ;            // error signal
    d_error = ;          // derivative of error
    error_pre = error;   // previous error
    sum_error = ;        // integral of error

    Pcontrol = ;         // P control action
    Icontrol = ;         // I control action
    Dcontrol = ;         // D control action

    Icontrol = constrain(Icontrol, -255, 255);  // limits for I control
    pwm = Pcontrol + Icontrol + Dcontrol;        // controller output
```

where `error` is the difference between the set point and the sensor value, Since we want to control the wheel velocity, we will use the smoothed velocity signal `filt_vel` as the sensor output. We also have a variable called `set_point` that you can use as the desired velocity. The variable `sum_error` is the cumulative sum of errors, and `d_error` is the difference between the current and the previous errors (approximation of the derivative of the error signal), respectively.

Experiment #2: In this experiment you will examine the closed-loop velocity response when given a step input. The goal is to investigate the effect of K_p on 1) the closed-loop time-constant and 2) the fractional steady-state error. Proceed as follows:

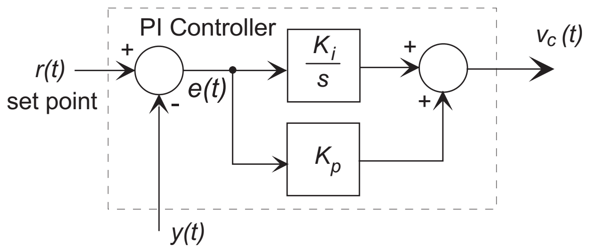
1. Set `#define desired_vel` to 15.0 (Line 31).
2. Uncomment `#define SQUARE_WAVE` (Line 41) so the wheel velocity will follow a square wave signal switching between 0 and `desired_vel`. Look at the square wave code to understand its operation.
3. Set $K_p = 5$ (keep $K_i = K_d = 0$) and upload the code to Arduino. Open "Serial Plotter" to monitor the three waveforms: `set_point, filt_vel,` and `vc.` In the template file the low-pass filter coefficient α is set to 0.5, which provides adequate high frequency noise filtering without significant time delay.
4. Enable `#define MATLAB_SERIAL_READ`, disable `#define PRINT_DATA`, and upload the code. Capture at least one full cycle of data with "SerialRead.m." *Note: In case the serial object created by the Matlab script is not cleared properly due to an error, run the two lines of code:* `delete(s1);` *and* `clear s1;` *(Lines 82 and 83 on the Matlab script) to clear the serial object.*
5. Zoom in to a relevant section of the trace and estimate (1) the time-constant, τ, and (2) the fractional steady-state error, $\Delta = \frac{\Omega_r - \Omega ss}{\Omega_r}$.
6. Repeat the above procedure with $K_p = 10, 15$ and describe the effect of K_p on τ and Δ.
7. Use an extremely high proportional gain: $K_p = 100$. Upload the code and record the response. Comment on your observation.

Experiment #3: In the previous experiment you have noted that there was a steady-state error to a constant angular velocity command if there was viscous damping and/or other frictional loss. In many control problems it is desirable to eliminate the steady-state error, and the most common way of doing this is through the use of *integral* control action or *proportional* plus *integral* control.

The transfer function of a PI controller can be expressed as

$$G_c(s) = K_p + K_i \frac{1}{s}$$

with a block-diagram



and a time domain response

$$v_c(t) = K_p e(t) + K_i \int_0^t e(t)dt$$

where $v_c(t)$ is the controller output.

In digital control systems such as this, real-time integration is done through an approximate numerical algorithm, such as rectangular integration, where the integral is represented as a sum s_n

$$s_n = s_{n-1} + e_n \Delta T$$

where e_n is the error at the n^{th} iteration, and ΔT is the time step, or trapezoidal integration

$$s_n = s_{n-1} + (e_{n-1} + e_n)\,\Delta T/2.$$

1. Investigate pure integral control by setting $K_p = K_d = 0$, and $K_i = 100$ so that

$$G_c(s) = \frac{100}{s}.$$

   Keep the desired velocity at 15 rad/s and disable square wave. Upload the code and record the waveforms. What can you say about steady-state of the response? Is the response acceptable in terms of speed of the response?

2. Use PI control, that is with

$$G_c(s) = K_p + K_i\frac{1}{s} = \frac{K_p s + K_i}{s} = K_p\left(\frac{s + K_i/K_p}{s}\right).$$

   Start with $K_p = 10$ and $K_i = 25$, enable the square wave, upload the code and capture the step response.

3. Keep the K_p gain but change K_i to 50 and 100, and repeat the experiment. Describe the effect of K_i on the transient behavior by looking at the settling time, peak time and percent overshoot.

4. Keep the K_p gain and use an extremely high integral gain: $K_i = 500$. Upload the code and record the response. Comment on your observation.

5. Keep the K_p gain and set $K_i = 100$, disable the square wave and upload the code. Qualitatively examine the effect of the above controller by pressing a finger on the flywheel to add a constant disturbance torque. Observe the controller output and make a note of what happens. You can also gradually increase the pressure from your finger and then release it to observe the effect of integrator windup. You can also observed this effect by turning off the battery power for a second or two, and then turn it back on.

Experiment #4: In this exercise we will design and implement a PI controller to do closed-loop velocity control since it guarantees zero steady state velocity tracking error. For this design we ask you to find gains K_p and K_i such that $\omega_n = 10$ rad/s and the closed loop response is critically damped. Recall that a pure second order system is in this form:

$$G(s) = \frac{K_{dc}\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}.$$

1. Determine K_p and K_i values according to the above closed-loop specifications. Implement this PI controller and test it with the square wave input with a desired velocity of 15 rad/s. Optionally you can enable `#define POT` and `#define PRINT_DATA`, upload the code and control the desired velocity by turning the potentiometer while viewing the data with the Serial Plotter.

2. Determine the desired angular velocity of the motor that matches the frequency of the spokes on the wheel with a given strobe frequency[1]. The required angular frequency of the motor shaft for the wheel spokes (in Hz) is given as $f_{motor} = f_{strobe}/m$ where f_{strobe} is the frequency of the strobe (in Hz) and m is the number of repeated patterns per rotation. From there we can compute the desired velocity for the wheel as $\Omega_{motor} = 2\pi f_{motor}$.

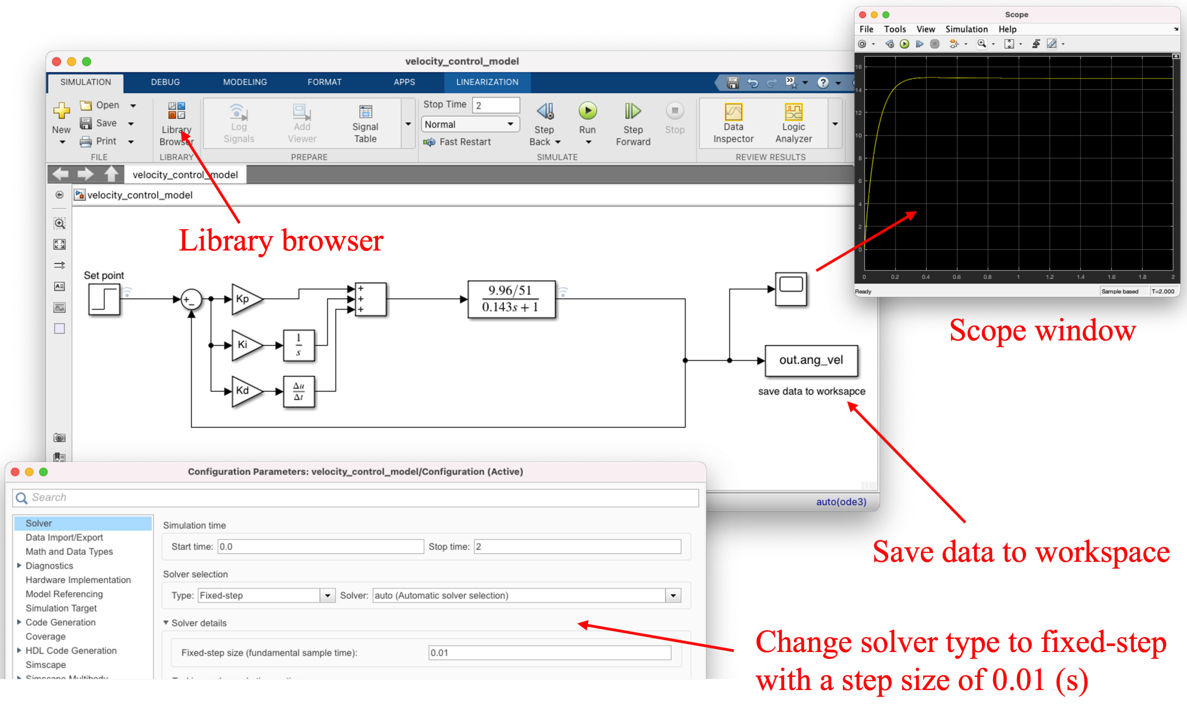
   We will use your phone's camera as the strobe source since a typical video capture rate for a camera is 30 frames per second (fps). Use your phone's camera app to view the motion of the wheel. Make sure the camera is set to video mode with a sampling rate of 30 fps. You my need to fine tune the desired velocity to sync up the video image.

Extra Credit Task: Create a Simulink model based on your Lab 2 transfer function and perform feedback control simulation.

1. Add a "Transfer Function" block from the Library Browser to the model and put in your flywheel transfer function ($\frac{K_{dc}/51}{\tau s+1}$).

2. Create P, I, and D blocks and connections according to the figure below.

[1]`https://demos.smu.ca/demos/optics/14-strobe`



Configuration Parameters: velocity_control_model/Configuration (Active)

Simulation time

Start time: 0.0 Stop time: 2

Solver selection

Type: Fixed-step Solver: auto (Automatic solver selection)

Solver details

Fixed-step size (fundamental sample time): 0.01

3. We can use variables such as `Kp`, `Ki`, `Kd` for the PID gain blocks so we can change the gain values from the Matlab Command Window (*e.g.*, `>> Kp = 10;`), or we can type in a value directly into the PID gain blocks.

4. Open the Step input block and set the final value to 15. This will produce a desired velocity of 15 rad/s.

5. Set `Kp` and `Ki` to the gain values you calculated in Experiment 4.1. Make sure to set `Kd` to 0.

6. Set the "Stop Time" to 2 on the top menu bar and run the model.

7. Compare the simulation output, `out.ang_vel`, to the actual response by plotting them together in one plot.

Appendix A: Integral Control Action

There is a steady-state error in the angular velocity of the plant when there is a viscous disturbance torque present. Integral control action is very commonly used to eliminate the steady state error.

Pure Integral Control: Assume that we replace our proportional controller with an integrator with gain K_i so that the controller output is

$$\begin{aligned} v_c(t) &= K_i \int_0^t e(t)dt + v_c(0) \\ &= K_i \int_0^t \left(r(t) - y(t)\right) dt + v_c(0) \end{aligned}$$

where $e(t) = r(t) - y(t)$ is the error. For simplicity also assume that $v_c(0) = 0$. Then the transfer function $G_c(s)$ of the controller is

$$G_c(s) = \frac{K_i}{s}$$

The integrator will function as follows:

- If the error $e(t)$ is positive, that is $r(t) > y(t)$, the controller output (and hence the torque produced by the motor) will *increase* at a rate proportional to the error.
- Similarly, if $e(t) < 0$, the controller output will *decrease* at a rate proportional to the magnitude of the error.
- If the error is zero, the integrator output will be *maintained at a constant value.*

The result is that the integrator will *continually adjust the motor torque so as to drive the error to zero*, at which point the supplied torque remains constant.

Assume that our system has an open-loop transfer function

$$G_p(s) = \frac{K}{\tau s + 1}$$

Then the closed-loop differential equation relating the angular velocity $\Omega(t)$ to the set-point $\Omega_r(t)$ will be

$$\tau \frac{d\Omega}{dt} + \Omega = KK_i \int_0^t \left(\Omega_r(t) - \Omega(t)\right) dt + T_d$$

where T_d is an external disturbance torque. If we differentiate this equation and rearrange, we end up with the closed-loop differential equation

$$\tau \frac{d^2\Omega}{dt^2} + \frac{d\Omega}{dt} + KK_i\Omega = KK_i\Omega_r + \frac{dT_d}{dt}$$

and we now have a second-order system.

Now consider the steady-state behavior. If $\Omega_r(t)$ is constant, and all derivatives are set to zero (steady-state), clearly

$$\Omega_{ss} = \Omega_r$$

and if T_d is constant, it has no effect on the steady-state error. The result is that integral control action has *eliminated the steady steady-state errors* due to the viscous friction and any constant external disturbance torque. You can show for yourself that the closed-loop transfer function gives the same result.

Proportional plus Integral (PI) Control: Pure integral control is rarely used in practice, and you will see why in the course of this lab. PI control, on the other hand, is used very often. In PI control, the controller uses a linear combination of proportional and integral control actions:

$$\begin{aligned} G_c(s) &= K_p + K_i\frac{1}{s} \\ &= \frac{K_p s + K_i}{s} \\ &= K_p\left(\frac{s + K_i/K_p}{s}\right) \end{aligned}$$

With the plant transfer function $G_p(s)$ as before, the closed-loop transfer function is

$$\begin{aligned} G_{cl}(s) = \frac{\Omega(s)}{R(s)} &= \tfrac{G_c(s)G_p(s)}{1+G_c(s)G_p(s)} \\ &= \tfrac{K(K_p s+K_i)}{\tau s^2+(1+KK_p)s+KK_i} \end{aligned}$$

For a constant input $\Omega_r(t) = A$, the final-value theorem states

$$\begin{aligned} \lim_{t\to\infty} \Omega(t) &= \lim_{s\to 0}(s\Omega(s)) = \lim_{s\to 0} \frac{A}{s} s \frac{K\left(K_p s + K_i\right)}{\tau s^2 + \left(1 + KK_p\right)s + KK_i} \\ &= A \end{aligned}$$

so that again, the steady-state error is zero. PI control eliminates steady-state error, just as does pure I control.

We note in passing that I control has introduced an open-loop pole at the origin ($s = 0$), and that PI control has introduced a pole at the origin, and an open-loop zero at $s = -K_i/K_p$.