2.12: Introduction to Robotics
Lab 3:
Motor Control*

Spring 2023

Assigned on: 23rd Feb 2023 Due by: 1st March 2023

Instructions:

1. **We will be present to answer questions, and help you to debug any issues. Please make sure to attend.**
2. **You will need to submit your answers to the below questions to Canvas. Be sure to include screen shots, and use the transfer function that you derive to explain what you see from your experiments.**

1 Introduction

This week, you will learn how to control a brushed DC motor using Arduino. The compensator will be a PID type for position and velocity control. Download the lab3 files from the github.

```
cd ~                  # note: make sure we are at home folder
git clone https://github.com/mit212/lab3_2023.git
```

2 Velocity Control Using P and PI Control Actions

In the continuous time domain, the PID control law is defined as:

$$PID_{Output} = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{d}{dt} e(t), \tag{1}$$

where $e(t)$ is the error signal at each timestep. It is computed at:

$$e(t) = SetPoint(t) - SensorOutput(t). \tag{2}$$

In the main loop of your Arduino code, the above PID equation is implemented in discrete time as:

*

1. Version 1 - 2020: Dr. Harrison Chin
2. Version 2 - 2021: Phillip Daniel
3. Version 3 - 2023: Ravi Tejwani and Kentaro Barhydt

```
error = set_point - filt_vel; //error signal
d_error = (error - error_pre) / loop_time; // derivative of error
filt_d_error = alpha * d_error + (1 - alpha) * filt_d_error; // filtered, derivative of error
error_pre = error; // previous error
sum_error += error * loop_time; // integral of error

Pcontrol = error * kp; // P control action
Icontrol = sum_error * ki; // I control action
Dcontrol = filt_d_error * kd; // D control action

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

Error is the difference between the set point and the sensor value. Since we want to control the wheel velocity, we will use the variable *filt_vel* as the sensor output. We also have a variable called *set_point* that you will use as the desired velocity. *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).

The block-diagram of your system with this velocity controller is given in Fig. 1. The closed loop transfer function for the system in Fig 1 is provided in the introduction slides and you need to explain what you observe in your experiments.

2.1 Experiment 1: Proportional Control of Velocity

In this experiment you will look at 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 1.5.
2. Uncomment **#define SQUARE_WAVE** so the wheel velocity will follow a square wave signal switching between 0 and desired_vel. You may want to look at the square wave code to understand its operation.
3. Set Kp = 60 (keep Ki = Kd = 0) and upload the code to Arduino. Open "Serial Plotter" in the Arduino IDE to monitor the three waveforms: **set_point, filt_vel**, and **vc**. **vc** is the voltage command sent to the motor. A color legend will be in the upper right of the serial plotter. The colors in the legend are read from left to right, and correspond to the wave-forms in the aforementioned order (see Fig 2 for a sample plot).
   (a) Set the baudrate of the serial plotter to 115200 baud
   (b) If you see a "permission denied" error when uploading, run the below command in the terminal window and then re-upload the script to your Arduino Uno:
   ```
   sudo chmod a+rw /dev/ttyACM0
   ```
4. Repeat the above procedure with Kp = 100, 300, 600 and describe the effect of Kp on the voltage command and the steady state error.
5. Use an extremely high proportional gain: Kp = 1200. Upload the code and record the response. Comment on your observation. Capture a screen shot of your system's response to the square wave at each of the five values of Kp.

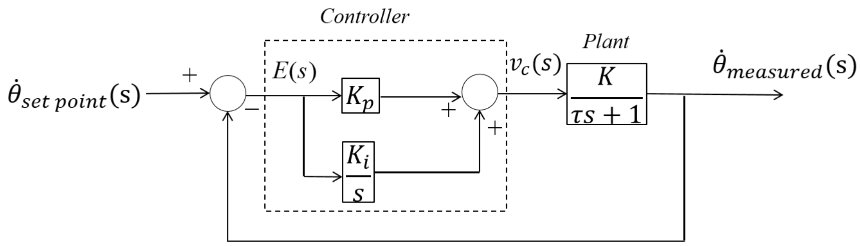

Figure 1: Block diagram of controller and system. E(s) is the error signal, τ is the plant's time-constant, and K is the plant's steady state gain.

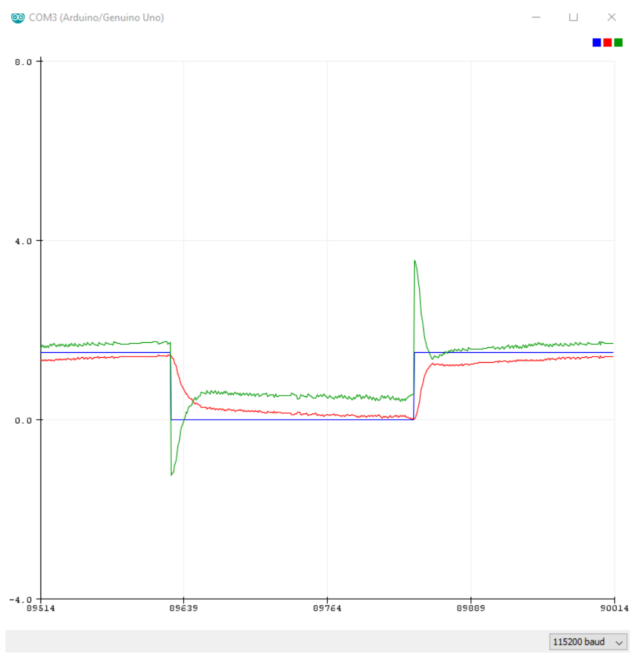

Figure 2: The colors in the legend are read from left to right, and correspond to the wave-forms in the order, **set_point (blue), filt_vel (red)**, and **vc (green)**.

2.2 Experiment 2: PI Control

In the previous experiment you have noted that there was a steady-state error to a constant angular velocity command. 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 and proportional plus integral control. The transfer function of a PI controller can be expressed as,

$$G_c(s) = K_p + K_i \frac{1}{s}. \tag{3}$$

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,

$$s_n = s_{n-1} + e_n \Delta T \tag{4}$$

where e_n is the error at the n^{th} iteration, and ΔT is the time-step. For a trapezoidal integration one gets,

$$s_n = s_{n-1} + (e_{n-1} + e_n)\Delta T/2 \tag{5}$$

1. Investigate pure integral control by setting Kp = 0, and Ki = 80. Keep the desired velocity at 1.5 rad/s and disable square wave. Upload the code and record the waveforms using a screen capture. What can you say about the steady-state of the response? Is the response acceptable?

2. Use PI Control. Start with Kp = 100 and Ki = 60, enable the square wave, upload the code and capture the step response using a screen capture.

3. Keep the Kp gain but change Ki to 80, 120 and 200, and repeat the experiment. Capture the step response. Comment on the effect of Ki on the transient behavior.

4. Keep the Kp gain but use an extremely high integral gain: Ki = 2000. Upload the code and record the response. Comment on your observation.

5. Disable the square wave, change Ki back to 200, and upload the code. Qualitatively examine the effect of the above controller by pressing a finger on the disk to add a constant disturbance torque. Observe the controller output and make a note of what happens, no need to capture a screen shot of this behavior. You can also gradually increase the pressure from your finger and then release it to observe the effect of integrator windup.

6. Comment on your observations and results. (What do the P and I control actions look and feel like? You may want to set the desired velocity to 0 and manually disturb the disk to see the effect.)

7. Design and implement a PI controller to do closed-loop velocity control, since it guarantees zero steady state velocity tracking error. Find gains Kp and Ki such that you achieve zero steady state error and zero overshoot before the square wave changes amplitude. Capture a screen shot of this behavior.

3 Closed-Loop Position Control, and the Effect of Derivative Control Action

In section 2, we investigated closed-loop control of the angular velocity of the rotational plant. In this lab, we switch to position control with the goal of commanding the disk to move to a given angular position. We will see that, for this particular plant, proportional control does not generate satisfactory transient behavior and that the use of PD (proportional + derivative) control allows us to achieve much improved response characteristics.

A PD controller has a transfer function,

$$C(s) = K_p + K_d s. \tag{6}$$

The block diagram of your position controlled system is shown in Fig 3. Compute the closed loop transfer function for the system in Fig 3, and explain what you observe in your experiments below based on this closed loop transfer function.

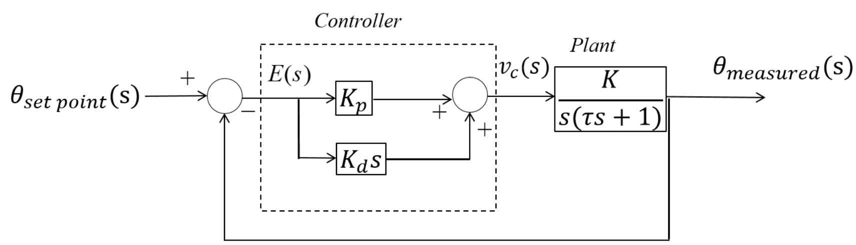

Figure 3: Block diagram of controller and system. E(s) is the error signal, τ is the plant's time-constant, and K is the plant's gain.

3.1 Experiment 1: Proportional Control of Position

Most shaft encoders, such as the one used in this DC motor, are incremental. This means they do not have an inherent absolute zero position. You can set the current position of the disk as zero position whenever you reset the Arduino by re-uploading the code or pressing the "RESET" button on the Arduino board.

Set up your controller with proportional control (Ki = 0, Kd = 0) for Kp = 100; 150; 350, all with a desired position of 1 radian. Enable square wave setpoint so as to command the wheel to rotate between 0 and 1 radian periodically. Make a plot of each of the responses. Would you classify the response as "satisfactory" based on the speed of the response?

3.2 Experiment 2: PD Control

1. Start with Kp = 150 and Kd = 5. Use the same square wave with an amplitude of 1 rad and save the step response. Select a complete positive step section of the response and generate a plot of it. What is the peak voltage command "**vc**" to the motor? Is the controller "saturating"? Is the response with PD control more satisfactory than responses with only proportional control?

2. Keep Kp = 150, repeat (1) with Kd = 10 and then Kd = 20. In each case, make a plot of the step response.

3. Compare your three plots. Briefly describe how the value of Kd has affected 1) any "overshoot" in the step response, 2) the time to the peak response, and the time to reach the steady-state response.

3.3 Experiment 3: PD Controller Design

1. Design and implement a PD controller that yields approximately zero steady state tracking error (within ±10%), no overshoot, and reaches steady state before the square wave changes amplitude. Record these parameter values, and capture a screen shot of this behavior.

    (a) You should not need an integral controller to meet this design requirement, however feel free to also play around with a non-zero value for Ki if you would like.

2. Apply this PD controller to a sine wave to see how well it tracks. Enable the sine wave by commenting out the line "**#define SQUARE_WAVE**," and un-commenting the line "**#define SINE_WAVE**" in your Arduino code. You do not need to submit plots of this response.