Massachusetts Institute of Technology

Department of Mechanical Engineering

2.004 *Dynamics and Control II*

Laboratory Session 2: Characterization of The Flywheel Plant

Laboratory Objective:

In Lab 2 we introduce a microcontroller as an embedded system[1] that can be programmed to perform various tasks, including feedback control applications. The micorcontroller we use is the Arduino Uno board[2] that employs an ATmega328P microcontroller chip[3] as its computational engine. You will write code in Arduino C/C++ programming language, and use the program to process/acquire data, and to drive a DC motor. Specifically in this lab session you will learn how to

- setup a servomotor/flywheel plant,
- verify encoder data and calibration factor,
- write code to derive angular velocity from the encoder data,
- write code to smooth the derived velocity data,
- create plant open loop transfer function, and
- verify transfer function response.

In the subsequent lab sessions we will learn how to program the microcontroller to perform feedback control.

Arduino Basics

Run Arduino Integrated Development Environment (IDE), and if you are not familiar with Arduino coding or need to refresh your memory, open a built-in example code such as "AnalogInOutSerial" or "AnalogInput" under "03.Analog." Read and understand the operation of this simple program. Note the two standard Arduino `setup()` and `loop()` functions, and understand what they do in this case. Use the Arduino help system (**Help** → **Reference**) to examine `pinMode(), analogRead(), analogWrite()`, and `delay()` functions.

[1] `https://www.electronics-notes.com/articles/digital-embedded-processing/embedded-systems/what-is-embedded-microcontroller-mcu.php`

[2] `https://store.arduino.cc/usa/arduino-uno-rev3`

[3] `https://www.microchip.com/wwwproducts/en/atmega328p`

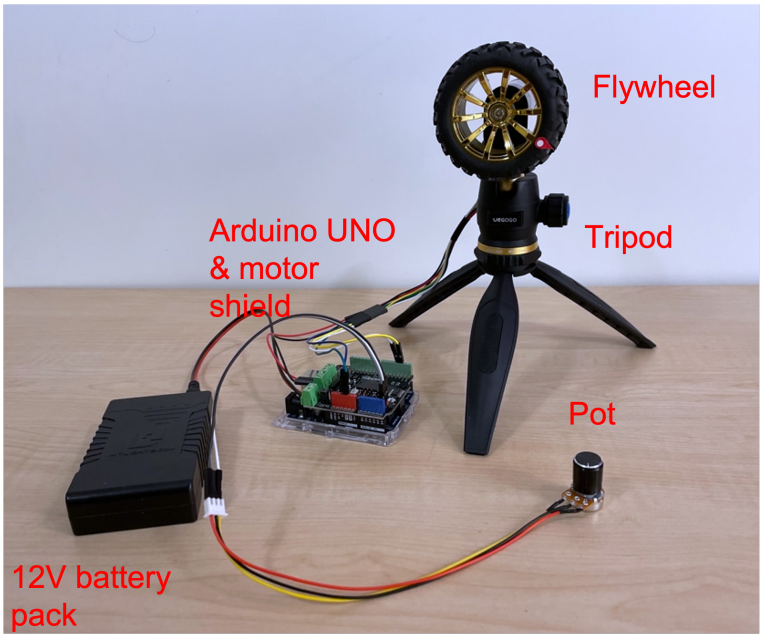

Figure 1: 2.004 DC Motor / Flywheel Plant.

Experiment #1: Encoder Calibration

1. Setup your DC motor/flywheel plant as shown in Fig. 1.
2. Download all Lab 2 files from Canvas: `https://canvas.mit.edu/courses/10470/assignments/129628`.
3. Open the Arduino template file "motor_encoder_template.ino" and read through it. Ask a lab staff to explain the code if you have questions about it.
4. Connect the Arduino Uno board to your computer using the USB cable. On the top menu bar, go to **Tools** → **Board** and select the appropriate board, then **Tools** → **Port** to select the appropriate serial port.
5. Make sure `#define desired_pwm` is set to 0 (Line 13), and upload the code to the Arduino board.
6. Open the "Serial Monitor" window from the Arduino IDE and set the baud rate[4] to 115200 (baud rate selection is located at the bottom of the serial monitor window).
7. The serial monitor window should print out the encoder counts calculated based on quadrature decoding[5] using Arduino's interrupt routines.

[4] `https://learn.sparkfun.com/tutorials/serial-communication/rules-of-serial`
[5] `https://www.dynapar.com/technology/encoder_basics/quadrature_encoder/`

8. Make sure there is a once-per-revolution marker on the wheel and turn the wheel clockwise by hand for exactly one revolution. Verify the encoder reading is around 1632, which corresponds to 6.28 radians.

Experiment #2: Derive Angular Velocity

1. Based on the angular displacement and timing information we can write code to compute angular velocity, ω, using finite difference approximations of derivatives.

$$\omega(t) = \frac{d}{dt}\theta(t) \approx \frac{\theta(t) - \theta(t - \Delta T)}{\Delta T}$$

   where t denotes the current time step, and ΔT is the sampling period. Using the three variables: `wheel_pos, pre_wheel_pos`, and `loop_time`, implement the above equation to compute `wheel_vel` (Line 103).

2. Set `#define desired_pwm` to 150 (Line 13).

3. Uncomment the line `Serial.print(wheel_vel); Serial.print(''\t'');` (Line 136), comment out Lines 134 & 135, and upload the code. Make sure to switch on the 12V battery pack.

4. Use Serial Monitor (or Serial Plotter) to monitor the output. For your reference, setting PWM to 150 produces roughly 28 rad/s angular velocity.

Experiment #3: Implement a Smoothing Filter

1. The derived angular velocity signal contains unwanted noise since it is not a direct velocity measurement and the encoder signal is not continuous. You will write code to implement a simple first-order, low-pass filter to reduce the effect of high frequency noise in the signal. Here we will implement the *exponential moving average algorithm*[6] that has the form:

$$\hat{\omega}(t) = \alpha\omega(t) + (1 - \alpha)\hat{\omega}(t - \Delta T),$$

   which is the weighted sum of the current velocity $\omega(t)$ and the previous filtered velocity $\hat{\omega}(t - \Delta T)$. The weighting factor α is defined as

$$\alpha = \frac{\Delta T}{(\Delta T + \tau_f)}$$

   where τ_f is the desired low-pass filter time constant. Use the variables `filt_vel, wheel_vel`, and `alpha` for this purpose (Line 104).

2. Uncomment Line 15 `#define POT` in the code to enable the potentiometer.

[6] `https://tttapa.github.io/Pages/Mathematics/Systems-and-Control-Theory/Digital-filters/ExponentialMovingAverage/Exponential-Moving-Average.html`

3. Experiment with different weighting factor `alpha`, between 0 and 1, to see the effect of smoothing (Line 65). Typically `alpha` is set to a value between 0.1 and 0.5 for effective filtering of high frequency noise. Note that significant time delay may occur with aggressive filtering (*i.e.,* smaller `alpha`). Monitor both `wheel_vel` and `filt_vel` with Serial Plotter and change the wheel velocity by turning the potentiometer to see the effect.

4. Determine the value of α if the sampling period is 0.01s and the desired filter cutoff frequency is 15Hz. Now put the α value in the code.

Experiment #4: Estimate Transfer Function

1. A first order transfer function can be constructed based on its step response:

$$G_p(s) = \frac{\Omega(s)}{V_c(s)} = \frac{K_{dc}}{\tau s + 1}.$$

   All we need to do is to estimate the values for the two model parameters: time constant τ and steady-state gain (or DC gain) K_{dc}.

2. Set the desired PWM to 150 and comment out `#define POT`.

3. Comment out `#define PRINT_DATA` (Line 28) and uncomment `#define MATLAB_SERIAL_READ` (Line 29), and upload the sketch to the Arduino Uno board.

4. Open the Matlab script `SerialRead.m` and change the serial port name (Line 25 or 26) to the Arduino port on your computer.

5. Close Arduino's Serial Monitor or Serial Plotter to free up the serial port.

6. Run the Matlab script to capture a step response. The script will open a Matlab figure window containing real-time data plots. **Press any key on the keyboard (e.g., the space bar) when the figure window is the active window to stop the data streaming.** Roughly 2 second worth of data capture should be sufficient.

7. Estimate the transfer function that relates the PWM pin voltage, `vc`, to the wheel velocity, `wheel_vel`, from the response curve.

8. Generate an equivalent step response from the transfer function and overlay the simulated response on top of the actual response. Refer to the following Matlab code for this task:

```
>> G = tf(10, [0.1 1]);  % create transfer function for the plant 9use your
   own values)
>> [Y, T] = step(2.94*G, 2);  % generate a step response with 2.94V input
   (PWM = 150)
>> hold on    % hold the current plot
>> plot(T, Y, 'm')    % plot the simulated response in magenta on top of the
   current plot
```

Extra Credit Task: Simscpe Simulation

Simscape is a software package built on Simulink to directly model physical systems without the need for underlying equations[7]. Here you will need to determine the appropriate physical parameter values for the given Simscape model, run simulation and generate a step response, and fine tune some parameter values to match the actual response.

1. Open the provided Simscape model: "flywheel_simscape_model.slx" that contains details of the physical flywheel plant. Fill in the provided parameter values to the appropriate blocks.

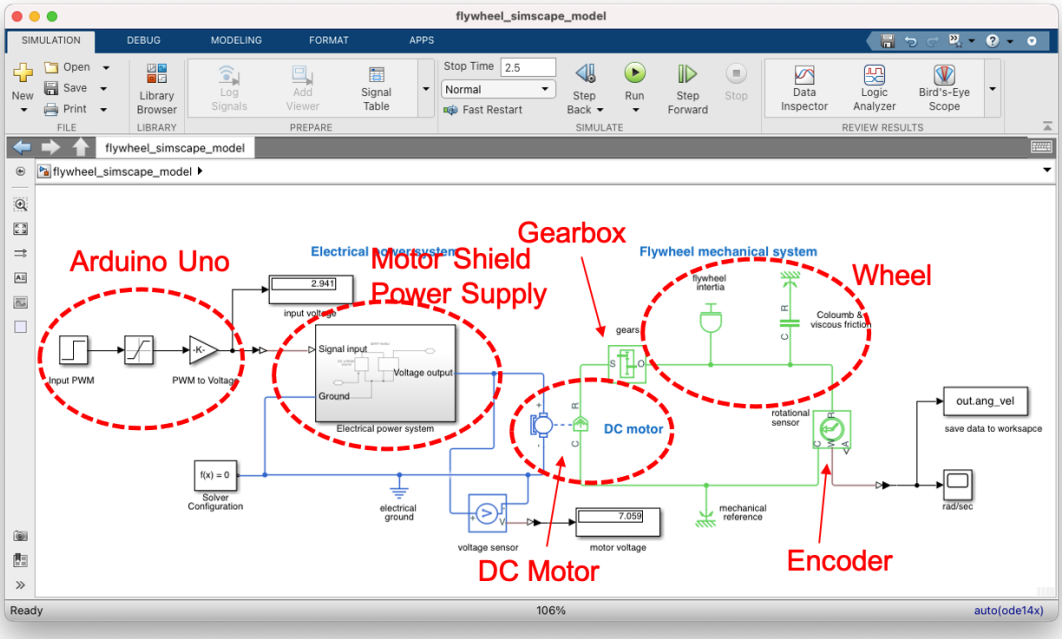

Figure 2: Flywheel Simscape Model.

- Motor back-emf constant $K_v \approx 0.008\ V/(rad/s)$ (determined experimentally).
- No load current $I \approx 140\ mA$ at $12\ V$ (determined experimentally)).
- Armature resistance $R \approx 3.2\ ohms$ (measured with a digital multi-meter).
- Gear ratio $N = 34$ (from data sheet).
- Inertia of the wheel $J = 1.47 \times 10^{-4}\ kg \cdot m^2$ (calculated based on its mass and dimensions).
- Determine the viscous friction coefficient b from J, R, K_v and measured τ from Experiment #4 according to the following relationship:

[7] `https://www.mathworks.com/products/simscape.html`

$$\frac{\Omega(s)}{V_s(s)} = \frac{K_v/R}{Js + (b + K_v^2/R)} = \frac{K_{dc}}{\tau s + 1}.$$

and put the value into the Coulomb & viscous friction block.

2. Run the model and generate a step response.

3. Estimate the time constant and steady-state value, and compare them to the values obtained from the actual response. You may need to adjust some of the parameters by up to $\pm 15\%$ to match your experimental data. Try to adjust R and K_v in the DC Motor block first.

4. Generate a plot that contains both actual and simulated responses. You can use the `hold on;` command to hold the current plot and then plot additional data on the same figure. For your information the simulated velocity data from the Simscape model is saved to a workspace variable called `out.ang_vel` as a timeseries data set. You can plot it by using the command: `plot(out.ang_vel);`.

5. Record your final K_v, R, b values.