**Checklist for Lab07:**

1. **Download Simulink model file ‘Lab07\_ToStudents’ from Canvas system (you may need to save the old model to a newer version of Simulink)**

**The following are similar with Lab04 DC motor open loop control.**

1. **A computer installed with MATLAB compatible with Arduino.**
2. **Uno or others board**
3. **Power sources compatible for DC motor (Battery, 11 V from lab power supply, or \*\*\*5 v from uno board)**
4. **Breadboard and jumper wires**
5. **H-bridge motor drive (L293)**
6. **DC motor with encoder**

In lab 4, we did open-loop control (we wanted specific motor movement patterns, so we sent specific commands to the motor, but we had no way to tell how well/accurately our DC motor actually followed those commands) of the DC motor. In Section 1, we will try to read the encoder and obtain the real-time measurement of the DC motor speed. Then in Section 2, we will review one type of closed-loop control (a.k.a, feedback control). This control compares what you commanded vs. what the encoder says your motor is outputting, then the control will adjust the commands so the motor output matches what you actually wanted.

**Section 1. encoder reading for motor speed.**

In Section 1, we will try to read the encoder and obtain the real-time measurement of the DC motor speed.

* 1. **Connect Arduino with DC motor with encoder**

First, refer to Figure 1 or Figure 2, we connect the Arduino and H-bridge motor driver by jumper wires. **(unplug Arduino !!!)**

**Step 1: place H-bridge with notch facing up.**

**Step 2: connect Arduino 5V and ground to breadboard +/-**

**Step 3: connect driver pin 1 to Arduino pin 9.** (enable and PWM)

When pin 1 receives 5V, then the driver is enabled. If it receives a PWM signal, then that PWM signal will be used to control speed.

**Step 4: connect driver Pin 2 to Arduino pin 8, driver pin 7 to Arduino pin 7.**

If LOW (or 0) is set to Arduino pin 8 (equivalently driver pin 2), and HIGH (or 1) is set to Arduino pin 7 (equivalently driver pin 7), the motor will rotate in one direction. If set oppositely, then the motor will rotate in the opposite direction.

**Step 5: connect breadboard + (5V) to supply voltage driver pin 16, driver pin 4 and pin 5 connect to ground.**

**Step 6: connect driver pin 3 to motor pin 6 (red, motor +), connect driver pin 6 to motor pin 5 (black, motor -).**

**Step 7: connect driver pin 8 to battery +, and ground to battery -.** Alternative: Use the 5V output (positive on breadboard) for motor supply voltage, refer to Figure 2.

**Step 8: connect encoder Vcc pin 3 to breadboard + and encoder ground pin 4 to breadboard -.**

**Step 9: connect motor encoder pin 1 to Arduino pin 2, motor encoder pin 2 to Arduino pin 3.**

Arduino Pin 2 and Pin 3 can be used as the interrupt IO, and the pulses signals from encoder will be counted as interrupts to Arduino.

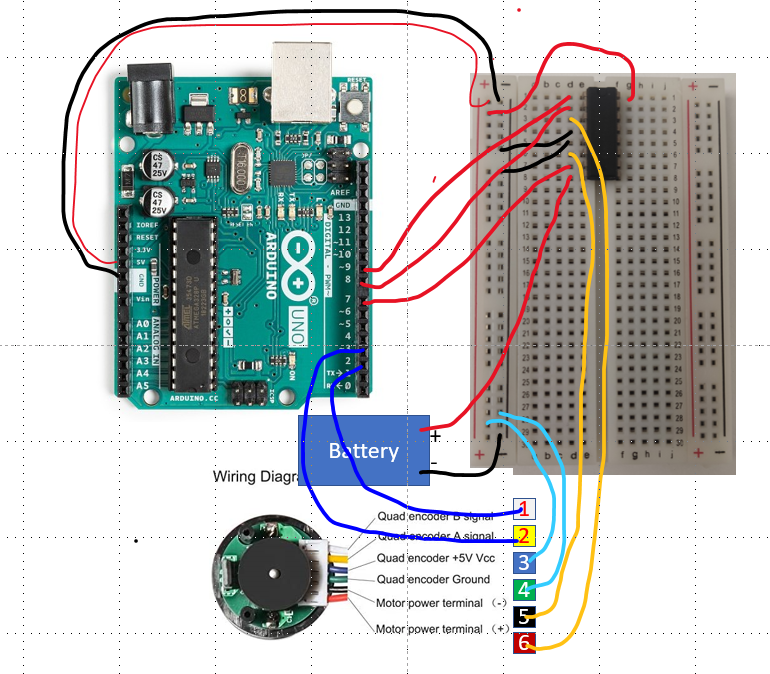


Figure . Wwiring diagram with a battery

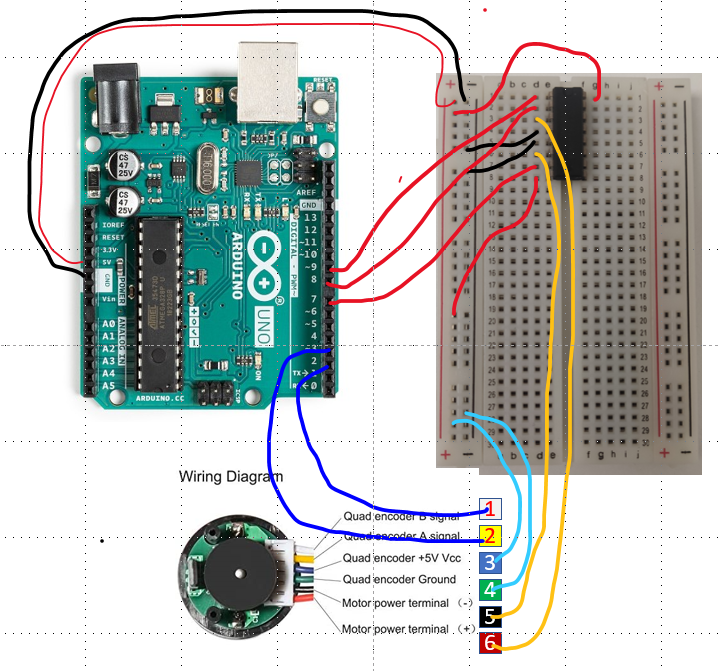


Figure . wiring diagram without a battery.

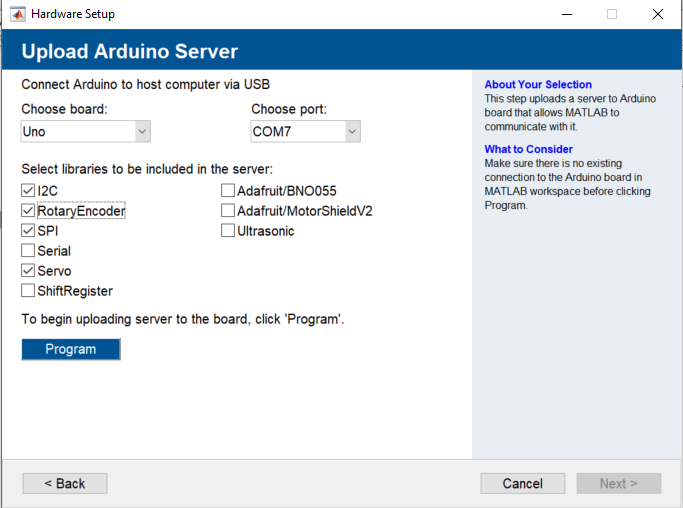
* 1. **Construct Simulink model**

**Step 1:** Download the file ‘*Lab07\_ToStudents*’ from Canvas system and put it under the MATLAB folder: C:\Users\username\OneDrive\Documents\MATLAB\ (or a simple folder path location with no spaces in the file path). Check that MATLAB’s file path points to this folder and that you have added the subfolders to MATLAB’s file path, as shown below:

A screenshot of a computer

AI-generated content may be incorrect.

**Step 2:** set up the Arduino board by ‘*arduinosetup’*, and check the libraries as below.



**Step 3:** open and set up the Simulink model *‘enc\_eml.slx’* in the folder *‘EML’*

*(if Simulink version is 2019, use ‘enc\_eml2019.slx’).* The model will look similar to Figure 3.

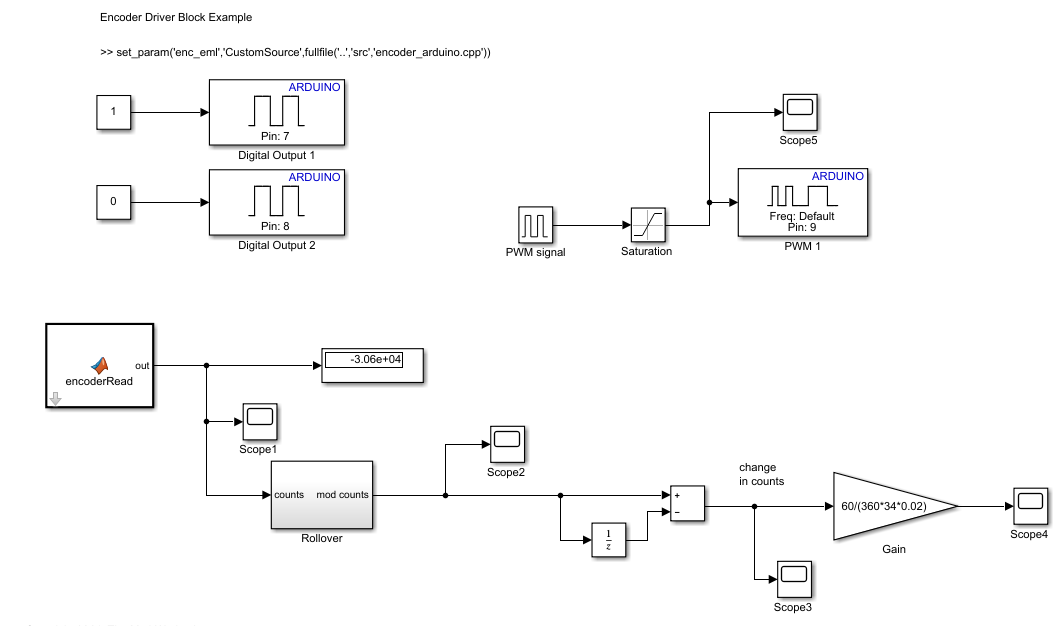
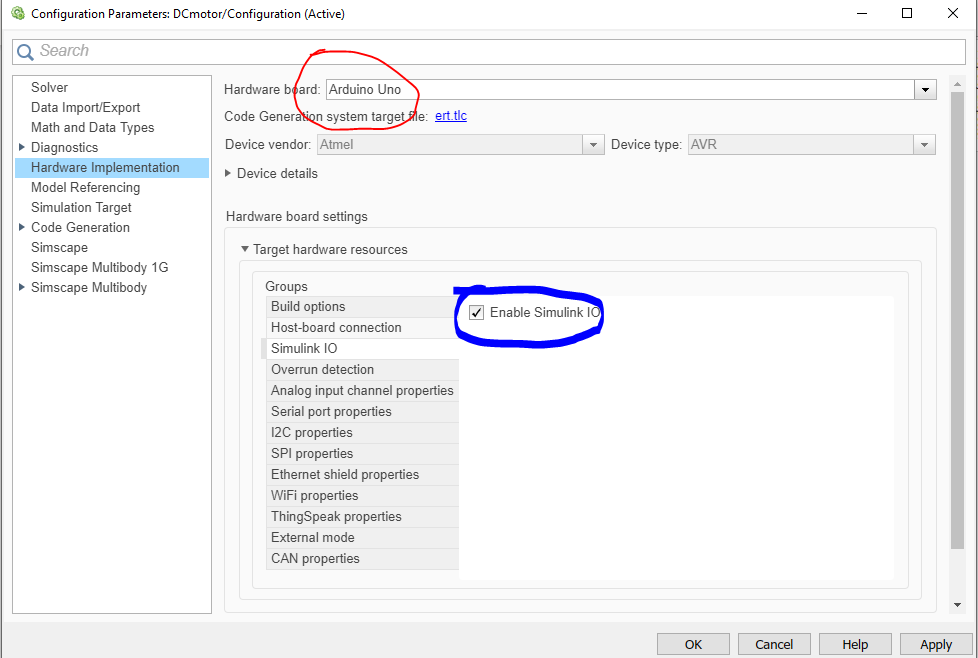
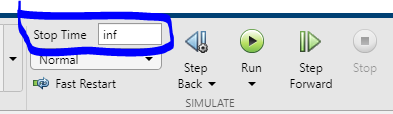


Figure . Simulink Model.

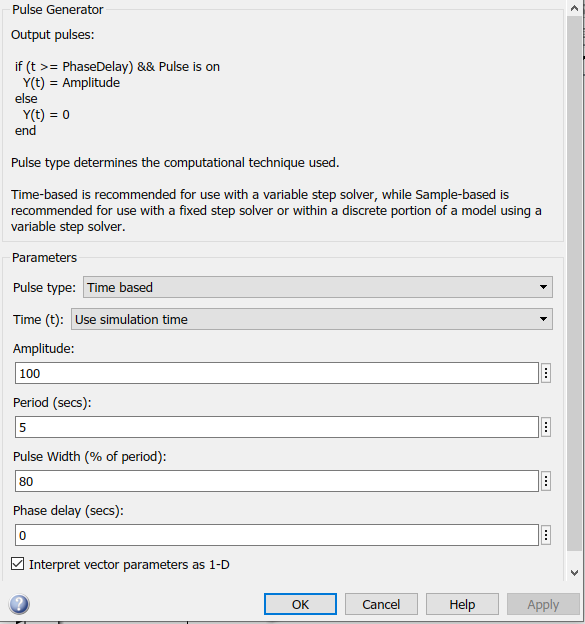
Go to ***‘Model setting’***, verify the fixed-step size to **0.02**. Go to ***‘Hardware implementation’***, and change Hardware board to ***‘Arduino Uno’.*** Check the external mode is serial. If you have an older version of MATLAB, go to ***Hardware board setting***, change check the ***enable Simulink IO***.



Change stop time to inf,



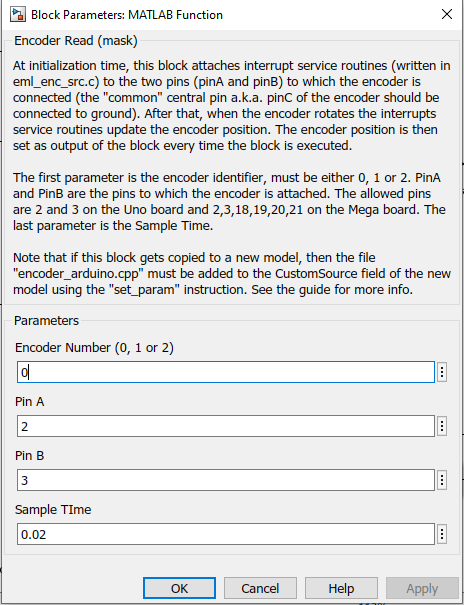
**Step 4**: Set up the PWM pulse signal block for a feedforward control DC motor as shown below.



**Supplemental (not used):** Instead of using the Arduino Encoder block, A black and white sign with a gear and a cross

AI-generated content may be incorrect., one can set up their own encoder reading with various fundamental blocks, an example is commented out in the provided model (we are having version discrepancies with the fundamental blocks this semester, so we will use the Arduino Encoder block this time). Setting up an encoder programs into the MATLAB function block-‘encoderRead’ can be done as follows:

Double left clicks on the **encoderRead block** and verify the parameter settings



Copy and paste the command below in the MATLAB window. This command will embed the Arduino .cpp program (in the Lab07\_ToStudens src subfolder) to the MATLAB function-encoderRead block.

**>> set\_param('enc\_eml','CustomSource',fullfile('..','src','encoder\_arduino.cpp'))**

Now, when you click the down arrow on the lower left corner of the encoderRead function block, A close-up of a computer screen

AI-generated content may be incorrect., the encoder\_arduino.cpp file is now linked to function block and will open up in a new Simulink tab for viewing and editing.

* 1. **Encoder reading.**

For teaching purposes, We will explain the coder reading with the old model.

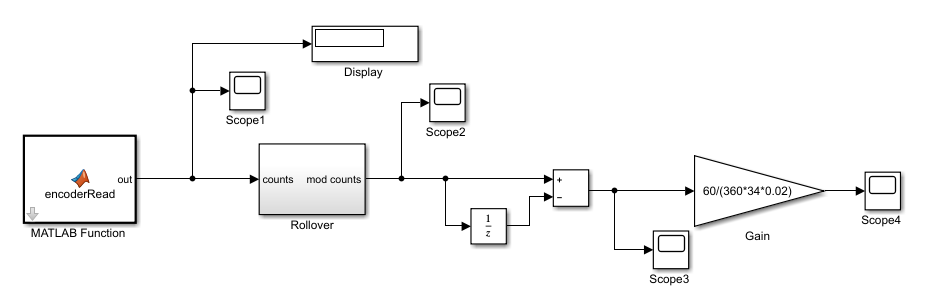


Figure 4. Encoder reading conversion

Summary of scopes 1-4:

Scope 1: pulse count from encoder reading at each sample time

Scope 2: pulse count after corrected Rollover

Scope 3: difference of counts at two sampled time points

Scope 4: motor speed in unit RPM

The MATLAB function *encoderRead* will count the pulses to the Arduino pin 2 and 3 (equivalently encoder signal A (pin 2) and B (pin 1)). The buffer of Arduino keeps track of the number of counts, and the buffer can only represent numbers between -32768 and 32767 (i.e. it uses 16-bits, 15 bits for the number and 1 bit for the sign). Then the count will reset and start from the initial value (i.e., essentially after the encoder disk rotated more than +/-32767 pulse counts, the counter resets back to “0”). An example of this rollover using Figure 4’s Scope 1 is shown below.

A screen shot of a graph

AI-generated content may be incorrect.

The rollover block will alter the resetting and make sure the count keeps going in one direction and ‘rollover’ (i.e. the rollover block keeps a cumulative sum of the counts instead of resetting).

The red box in Figure 4 will calculate the count difference between two sampling points. Implicitly, the difference output of the red box can be used as an indicator of motor speed. We will then use the Gain block to make the conversion from that sampling difference into units of RPM.

The equation in the gain is that .

CPR: count per resolution;

GearRatio: Gear ratio of the motor

SampleTime: sampling time (fixed step in model setting)

Specifically for the motor in lab (See the sticker on Motor), the gain is 60/(360\*34\*0.02).

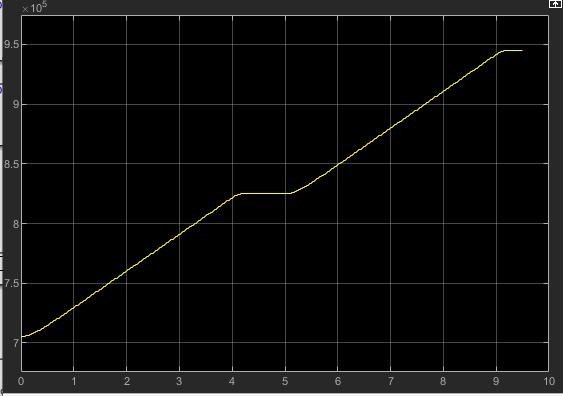
In our model, the pre-built Arduino block accomplishes the tasks of both the encoderRead and Rollover blocks.

**Now, we are ready to run the model with external mode (latest MATLAB version names it ‘monitor & tune’).**

**Excise 1: Encoder reading in scopes using ARDUINO encoder block**

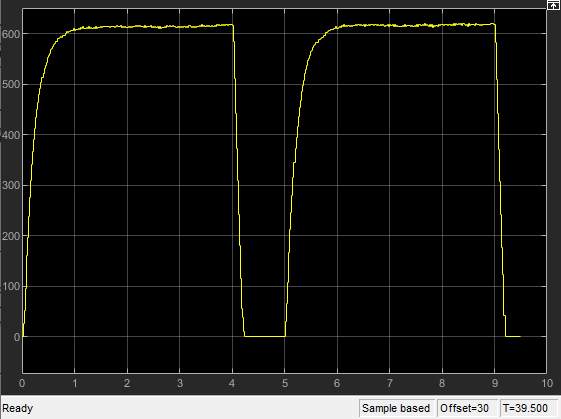
Attach the response from scope 2 or pass off with TA. (encoder count corrected ‘rollover’)

Response screenshot:

Example result: 

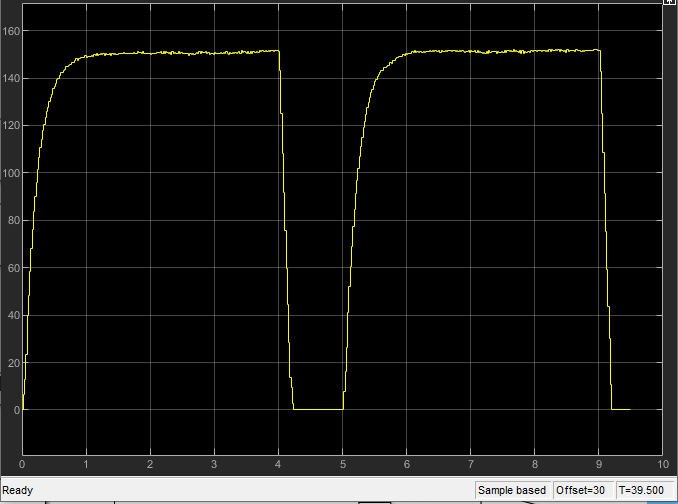
Attach the responses from scope 3 in or pass off with TA. (motor speed)

Response screenshot:

Example result: 

Attach the responses from the scope 4 in Figure 4. (motor speed in RPM units)

Response screenshot:

Example result: 

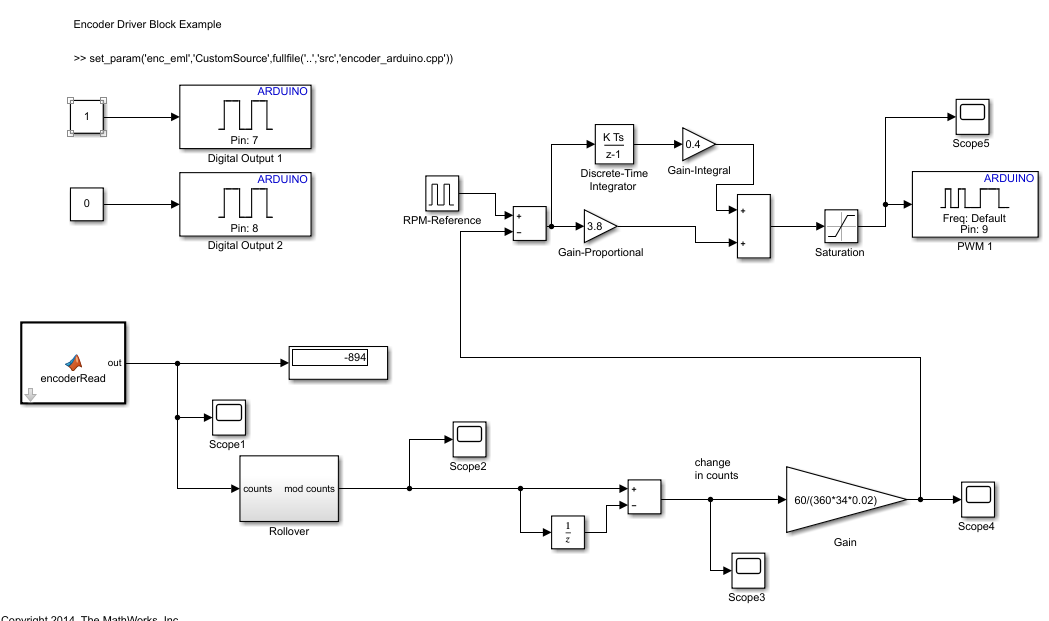
**Section 2. Feedback control of DC motor**

in Section 2, we will review one type of closed-loop control (a.k.a, feedback control). This control compares what you commanded vs. what the encoder says your motor is outputting, then the control will adjust the commands so the motor output matches what you actually wanted.

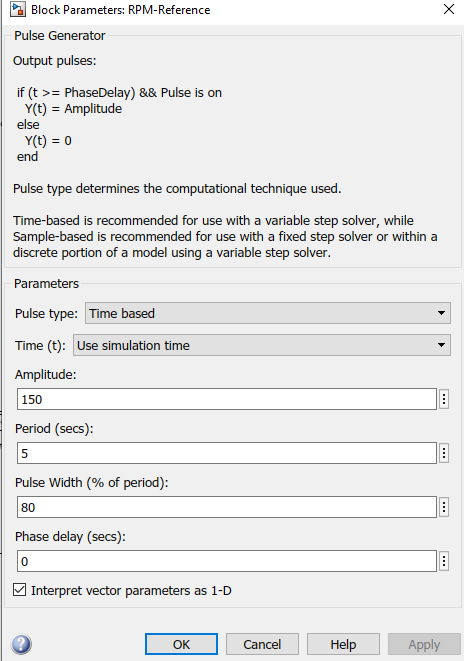
Refresher on PID: <https://www.mathworks.com/videos/understanding-pid-control-part-1-what-is-pid-control--1527089264373.html>

**2.1 Proportional-Integral (PI) control**

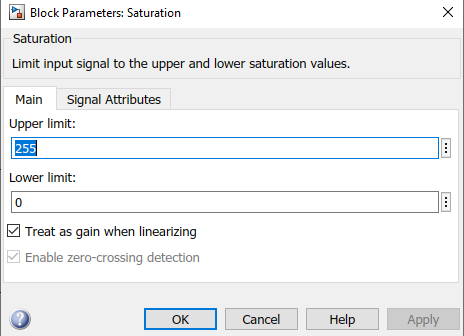
Construct the Simulink model as below (except use the Arduino Encoder block instead of the “encoderRead” and “Rollover” blocks) to create a PI controller.



The “RPM-Reference” Pulse Generator block defines the reference speed for the motor to track. We will want the controller to track an RPM **RPM = 150**. The RPM reference block is setup as



The saturation block is set as



Create the variable “Ts” in the model Workspace (in Model Explorer) and set it equal to the Fixed-step size defined earlier.

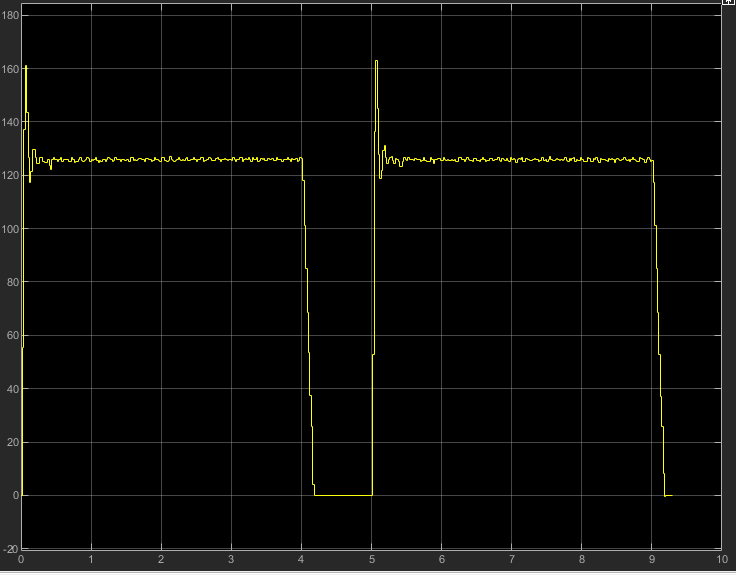
**Excise 2: Feedback control by Proportional control and Proportional-Integral control**

We want to use our controller to regulate motor speed to track 150 RPM. Proportional control will lead to steady-state close to 150 RPM, but the steady-state error cannot be removed only by proportional control. Adding the integral term will regulate motor speed to accurately 150 RPM in steady state. We will implement and compare P control and PI control on the DC motor in this section.

**2.1 P-Control: Set integral control gain as 0, and tune the Proportional gain such that the steady state is close to 150. Attach scope 4’s response. (the steady-state RPM will not be exactly 150.)**

Scope 5:

Example result: (steady-state is around 125RPM only by proportional control) The P control improves the settling time relative to un-controlled motor responses.



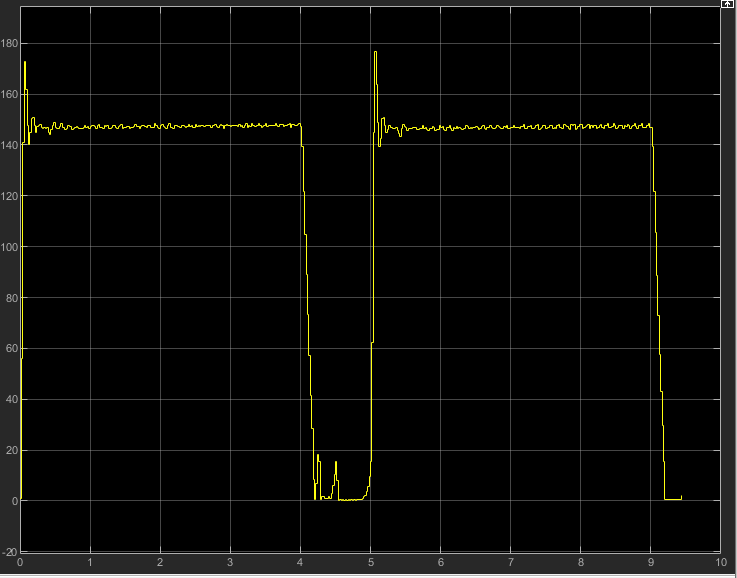
**2.2 PI-Control: Tune both proportional gain and integral gain of PI controller such that the steady-state in scope 4 is around 150. List your gains and attach scope 4 response.**

Proportional Gain:

Integral Gain:

Scope 4:

Example result: (With integral gain, the response is regulated to steady state of 150, and overshoot is possibly observed)



**2.2 Proportional-Integral-Derivative (PID) control**

**Extra Excise 3: Feedback control by Proportional-Integral-Derivative control**

This Excise is not required to complete, but 2 bonus points will be awarded if completed.

The additional derivative term is supposed to reduce the overshoot.

3.1 PID-Control: Tune proportional gain, integral gain, and derivative gain of a PID controller to reduce and smooth out overshoot. List your gains and attach scope 4 response.

Proportional Gain:

Integral Gain:

Derivative Gain:

Scope 4:

example PID controller model:

