EENG498BC: Systems Exploration, Engineering, and Design Laboratory
Spring 2016
Exercise 2b: Motor Control

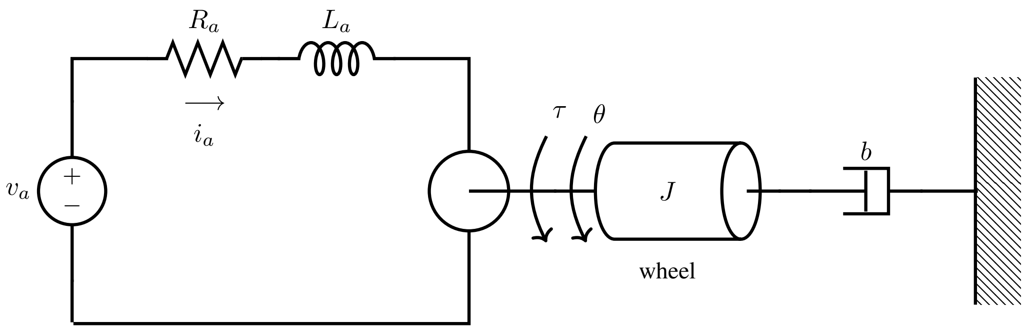
Vibhuti Dave and Tyrone Vincent[*]

1. Introduction

The purpose of this exercise is to design a control system that regulates the speed of a motor, and implement it using an Arduino.

2. The System

A DC motor driving a wheel with bearing friction has the following ideal component representation.



3. Exercises

- Mount the motor on a frame to make it stable, and add a wheel. Find the internal resistance and motor constants, using the specifications listed at `https://www.pololu.com/product/1444/specs`. You can assume $L_a = 0$. For a specific value of v_a, use the stall torque and current and the relationship $\tau_m = K_t i_a$ to find K_t. With that same value of v_a, use the steady state circuit relationship $\frac{v_a - K_e\dot{\theta}}{R_a} = i_a$ with the free-run and stall values of i_a and $\dot{\theta}$ (note that stall means $\dot{\theta} = 0$) to give you two equations to solve for K_e and R_a. Don't forget to convert to SI units.
  - If necessary, review Lecture 11, Motors and Hydraulic Actuators
- Add a motor driver to the Arduino, hook up the motor driver to the motor with wheel attached, and use the Arduino to spin the motor, using the `analogWrite` command to command the motor driver to supply a pulse width modulated signal of a desired pulse width.
  - `https://www.pololu.com/product/1444`
  - `https://www.pololu.com/product/2503`
- Use the Arduino to read the rotary encoder.
  - `http://en.wikipedia.org/wiki/Rotary_encoder#Incremental_rotary_encoder`
  - `http://playground.arduino.cc/Main/RotaryEncoders` (Use the high performance encoder library)

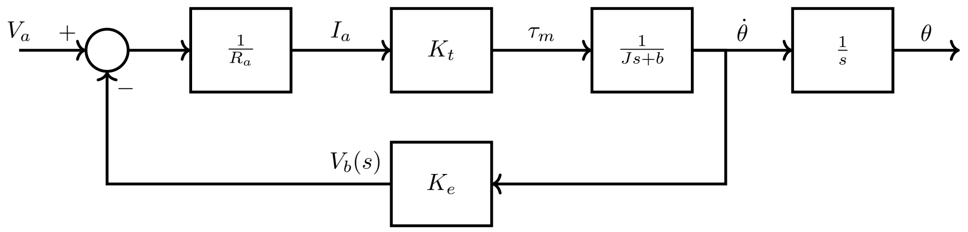
[*]Developed and edited by Tyrone Vincent and Vibhuti Dave

- Set up an Arduino program that will perform a step response experiment. This program should have a main() function that runs at a fixed cycle time, also called the sampling time. Use the function micros or millis to control the sample rate, by waiting at the end of your main function until the correct time has passed since the last time through. Have the program read the angular position, and from this data and the known sampling rate, calculate the angular velocity. For the step experiment, the program will initially output a motor voltage command of 0, changing to a desired positive value at 1 second. (By motor voltage command, we mean the value used in the `analogWrite` function; this should be a number between 0 and 255.) Use the serial link to output the results. Use a sampling rate of about 1ms.

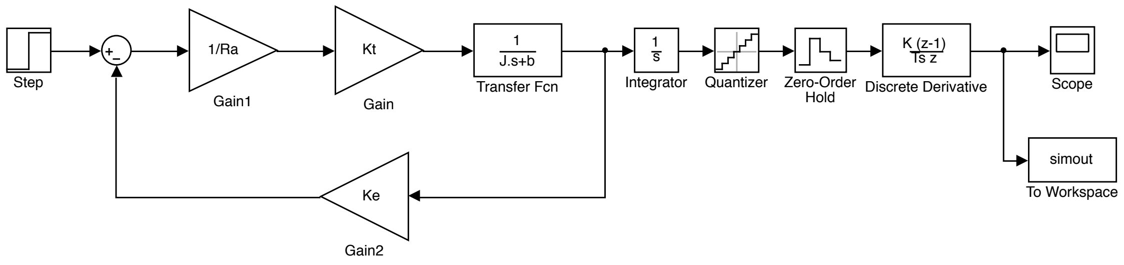
- Perform a step response experiment, and estimate the transfer function from motor voltage command to angular velocity. Although the motor is theoretically second order, when the motor inductance is small, the response can be well approximated as first order. You will probably use a voltage command greater than 1, so don't forget to scale the results appropriately.

  - 307 Notes: Lecture 15, System Identification

- Create a simulation that replicates your step response experiment. By attaching the wheel, you have added a load which can be model by an inertia and damper, which has transfer function $\frac{\dot{\theta}(s)}{\tau(s)} = \frac{1}{Js+b}$, where J and b are parameters that you will have to find. The motor block diagram is of the form



From above, find the transfer function $\frac{\dot{\theta}(s)}{V_a(s)}$, and choose J and b so that the step response of this transfer function matches the one you found in the experiment. In your simulation, add blocks that account for non-ideal effects such as the discrete nature of the encoder. For example, since the motor position is read using the encoder, the position is quantized to one of 3200 counts per revolution. We can simulate this using quantization block (in Discontinuities folder, or search for quantizer) with a quantization interval of `2*pi/3200`. The position is only read periodically, so we need to simulate the sampling process. This is done by using a Zero-Order-Hold block (found in the Discrete folder), with the sampling time set to the sampling rate of your Arduino program. Finally, you can implement the code that you use to calculate the velocity from the measured position using either a Matlab function block or a Discrete-Derivative block. The resulting simulink block diagram will look something like the following:



- Using the transfer function $\frac{\dot{\theta}(s)}{V_a(s)}$ as the system to be controlled, design a PI controller to regulate the motor speed. That is, you should be able to specify a desired rotational velocity that the wheel will turn, even if there are disturbances. Aim for specifications of a rise time of 150 ms and an overshoot of less than 5%. Document your design procedure. (You can check your design against the autotune function of the Simulink PID block, but you should not use this as your primary design mechanism.) Simulate the closed loop system in Simulink.

  – 307 Notes: Lecture 19, PID Control

- Implement the controller using the Arduino. Don't forget about units!
  – 307 Notes: Lecture 19, PID Control, Appendix.

  Find the closed loop step response (that is, going from a setpoint of 0 to a setpoint of 1 rad/s), and compare to what you predicted from your design.

4. Reporting

- Document results, such as
  – your control design and design method
  – the simulated motor response, and
  – the actual motor response.
- Document and store simulink and arduino code you can use to verify the motor controller is working correctly. That is, you should be able to quickly simulate a step response, and perform an experimental step response, and quickly compare the results in Matlab.

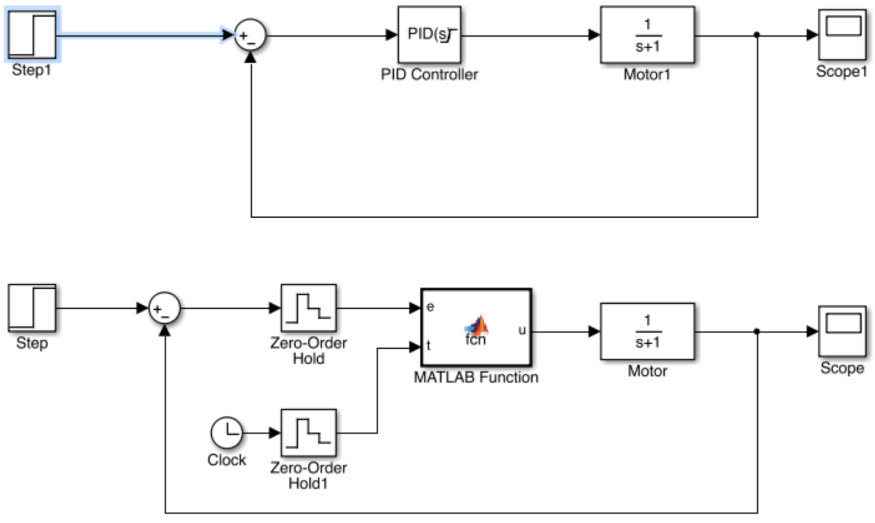
5. Appendix 1: Verifying a Discrete Time Controller Using Simulink

An easy way to simulate your embedded processor in Simulink, is to use a Matlab function block. The Matlab function block can be found in the User-Defined Function library. If you drag it into your Simulink block diagram, and double click on the function, it will open the Matlab editor. The following code can be used to implement a PI controller using the Matlab function block. Variables that need to be remembered between function called are defined as `persistent`.

```
function u = fcn(e,t)
%% Define Variables
% Control Gains
Kp=1;
Ki=1;
umax = 10;
% Memory variables
persistent I_past;
persistent t_past;
if isempty(I_past), I_past=0; end;
if isempty(t_past), t_past=0; end;

%% Calculate Controller
Ts = t - t_past;
% Integrator
I = I_past+e*Ts;
% PI control calculation
u = Kp*e+Ki*I;
if abs(u)>umax,
 u = sign(u)*umax;
 I = (u-Kp*e)/Ki;
end;
I_past=I;
t_past=t;
```

The following block diagram shows a Simulink block diagram with one loop implemented using a PI controller, and another implemented using the Simulink block. The zero order hold block is used to choose the sample rate. This can be entered by double clicking on the element. We have also added a clock so the Matlab function knows the elapsed time.


Step1
+
−
PID(s)
PID Controller
1
s+1
Motor1
Scope1
Step
+
−
Zero-Order Hold
e
t
fcn
u
MATLAB Function
1
s+1
Motor
Scope
Clock
Zero-Order Hold1