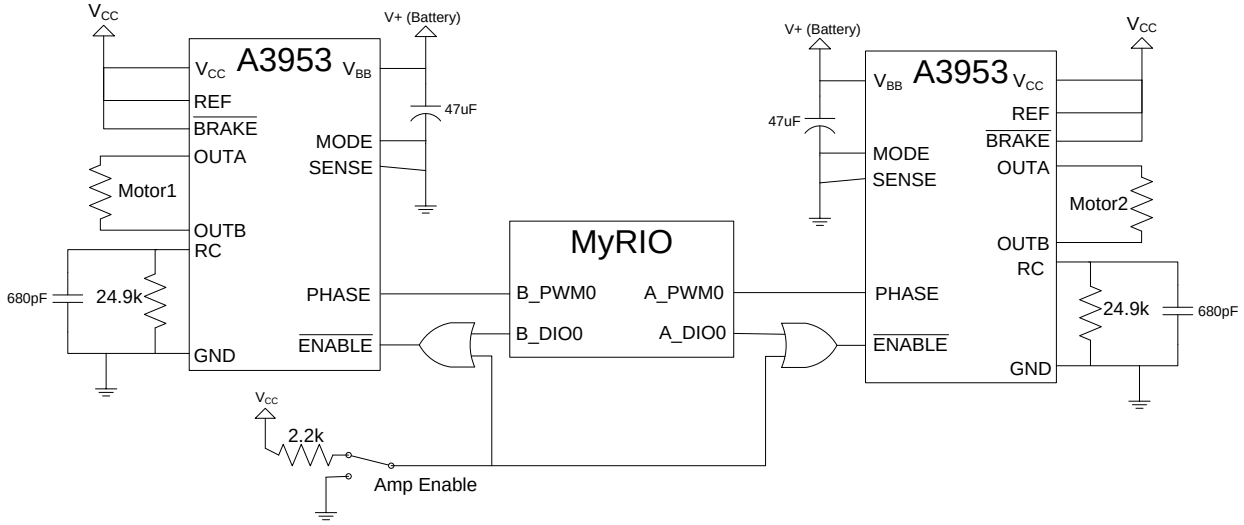
ME 461 Laboratory #7
Open Loop Motor Control and Friction Compensation

Goals:

1. Develop Labview code for the MyRIO that drives two permanent magnet DC motors using PWM signals to command H-bridge amplifier chips.
2. Calculate in Labview the motor's rotational velocity using encoder signals.
3. Identify a nonlinear feed-forward equation to compensate for friction in the motors.

Exercise 1:

In this exercise you will create Labview code to send (torque) commands to the motors. As you probably have guessed, you will be using a PWM signal and an H-bridge to control each of the motors. The configuration hardware related to the Allegro A3953 motor bridges is shown below.



Motor Controller Schematic

Start out in Labview by creating a new project. Select the "myRIO" template and then "myRIO Project." Select Next. Now give your project a name and select a directory to store you project. What I suggest to do is create a directory in your C:\drive directory called "myrio". Then inside "myrio" create a new directory for each Labview project you create for this class. Find your benches myRIO in the list of WIFI connected myRIOs. Make sure to ask which myRIO is yours. Then select Finish to create your project.

In the Block Diagram view of your new project's Main.vi, delete the while loop and all items inside of it. Leave the Flat Sequence and the error items. You will place most of your blocks in the middle frame of the flat sequence. Most of the code (*block diagrams*) you will create for labs 7, 8 and 9 will be run inside a Control & Simulation Loop. This Loop must be synchronized with a 1KHz timer allowing us to

achieve a minimum real-time sample period of 1 millisecond. When we deal with the robot car in labs 7, 8, and 9 we will use a sample period of 50ms. So place a Control & Simulation Loop in the middle frame of your flat sequence. Bring up its properties and set the following: Initial Time 0, Final Time 200, ODE Solver Runge-Kutta 1 (Euler), Step Size 0.05, Uncheck Auto Discrete Time and set Discrete Step Size to 0.05, Decimation 0. In the Timing Parameters tab check Synchronize Loop to Timing Source, Source 1kHz, Auto Period checked, Offset/Phase 0, Priority 100, Deadline -1, Timeout -1, Processor Assignment Manual and Processor 0. Now first inside your simulation loop place a "Halt Simulation" block found in Control Design and Simulation->Simulation->Utilities. Then create a control for this block and change it to a Stop button. Finally change the Stop button's mechanical action to "Latched when Released."

Inside the simulation loop place a myRIO PWM block (Don't use the PWM block under "Advanced"). Label it PWM_0 (for now) and use channel "A/PWM0 (Pin 27)", Frequency 20KHz, and Duty cycle set using an input. Place another PWM block with all the same settings except using channel "B/PWM0 (Pin 27)" and label it PWM_1. In below steps you will figure out which motor (left, right) is driven by PWM_0 and PWM_1 and re-label them PWM_left and PWM_right.

The PWM block takes as input a number between 0 and 1 (0% and 100% duty cycle). For the next lab it will be more appropriate for your controller to have an output command that is in the range of -10 to 10. Since we are driving with a PWM signal the DIR pin of the H-bridge we want to make -10 equate to 0 (0% duty cycle), 0 equal to 0.5 (50% duty cycle) and 10 equal to 1 (100% duty cycle). Develop the Labview code that implements this offset and scaling for both PWM channels that converts a number between -10 and 10 to a number from 0 to 1. Make sure to saturate the -10 to 10 input so that a value greater than 10 is capped at 10 and a value less than -10 is capped at -10. For the Enable pin of the H-bridge you will need to use two MYRIO Digital IO blocks to enable or disable each H-bridge. Use pins A/DIO0 and B/DIO0. Add a Labview control for each of these GPIO pins that allows you to set and clear the pins. Now try out your program. Run your code and note the state of the A/DIO0 and B/DIO0 pins that enable the motors. Send a value of 6 to your PWM outputs (This 6 will be scaled by your code to be a value between 0 and 100.) Note which direction the motors are spinning. If they are not spinning flip the toggle switch and/or change the state of A/DIO0 and B/DIO0. Note if these digital I/Os need to be set or cleared to enable the H-bridge.

For Labs 7, 8 and 9, we will define the forward direction of the robot such that when the car is driving forward the caster is behind the wheels (in back). Knowing this set the polarity correctly for the control effort command to the motors. In other words, when you apply a value between 0 and 10 to the motor the wheel should turn in the positive direction and when the value is between 0 and -10 the wheel should turn in the negative direction. Check that this is working for both the left and right wheels.

Once you have tested your Labview code and found that you can command your motors to spin in different directions and at different speeds, turn the code you just created into to a Simulation Subsystem (not a Sub-VI). Have the Simulation Subsystem take four inputs: Command Motor1 (double value between -10 and 10), Enable Motor1 (Boolean True or False), Command Motor2, Enable Motor2.

Once the Subsystem is finished place it in your main VI's Simulation Loop and test that it works the same as the previous code just did.

Demonstrate to your TA.

Exercise 2:

This exercise introduces using the Labview MYRIO Optical Encoder block for reading the optical encoder angle sensor attached to each of the robot car's Faulhaber motors. With this angle measurement you will be able to calculate the velocity of the robot car.

The Labview Optical Encoder block counts the quadrature A (ENCA) and B (ENCB) signals from a standard incremental optical encoder. When spinning forward the count increments, when spinning backwards the count decrements. Find the MYRIO Optical Encoder Block. Add two of these blocks inside your simulation loop. Setup one for Channel A/ENC and name it "Left Encoder" and select "Quadrature phase signal" as its output signal type. Do the same for the second Optical Encoder block but name it "Right Encoder" and select Channel B/ENC. The encoder's count is returned from this block. Convert the count to radians by using the fact that the motor's encoder has 141 periods per revolution of the wheel. Because this is a Quadrature optical encoder the number of counts per revolution multiples this number by four making 564 counts per revolution.

Add Labview code to calculate the velocity of the wheels. Read the value of the two optical encoder sensors every 50ms. Using the backward difference rule, calculate the velocity of the wheels in rad/s. The backward difference rule is shown below for reference. T_S is the sample time.

$$v_k = \frac{x_k - x_{k-1}}{T_S} \qquad \text{(Backward difference rule)}$$

We will find it convenient in the future to work with linear velocity instead of angular velocity. Convert the angular velocity to linear velocity in ft/sec using a wheel radius of 2 inches. Display the velocities of the two wheels and **demonstrate to your TA** for various control efforts.

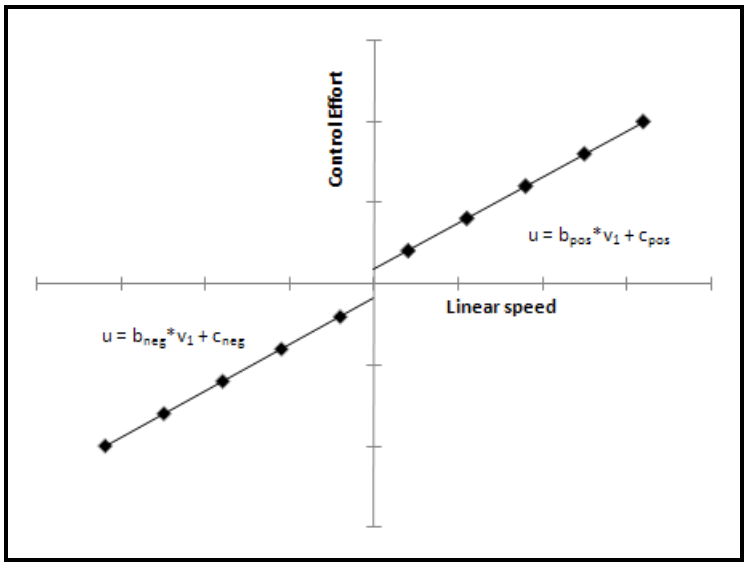
Exercise 3:

You are now going to identify coulomb and viscous friction coefficients and implement a friction compensation algorithm to make your robot behave more like an ideal linear plant. This exercise should be performed with the robot on the floor near your bench. For various control inputs, record the average wheel velocity in the table below. If there is a large difference in friction between the two motors (to the point that the robot drastically deviates from a straight line), you will have to adjust the control input to one of the motors so that the wheels spin at the same rate. In this case, record the adjusted control values in the second column.

Motor 1/Motor 2 Input (PWM units)		Average linear velocity (ft/s)
-8		
-6		
-5		
-3		
-2		
0.0		
2		
3		
5		
6		
8		

If you notice a deviation from the linear relationship in the data you collect, you should only use the linear portion of the control effort – velocity data.

Generate a plot of control effort vs. average wheel speed (you will have two plots if you had to adjust control effort values for one of the motors). Fit two linear regressions to the data, one for $u < 0$ and one for $u > 0$. Your plot should look like the example below. The y-intercepts of the regressions are the coulomb friction constants and the slopes are the viscous coefficients. Record the values in the table below.



Friction Nonlinearity in Motors

Motor 1

Coulomb Friction	Viscous Friction
c_{pos} =	b_{pos} =
c_{neg} =	b_{neg} =

Motor 2 *(if needed)*

Coulomb Friction	Viscous Friction
$c_{pos,2}$ =	$b_{pos,2}$ =
$c_{neg,2}$ =	$b_{neg,2}$ =

In your program, implement a controller that supplements the control effort based on the current velocity of the robot. If the robot is moving forward, use the positive coefficients, and if it is moving backward, use the negative ones. Your compensation algorithm should resemble the following.

if $v_1 > 0$

$u_1 = c_{pos} + b_{pos} \times v_1$

else if $v_1 < 0$

$u_1 = c_{neg} + b_{neg} \times v_1$

Test your program with the robot on the floor. Give the robot a little shove forward. With a little tuning, it should coast as if no friction is present. If the robot speeds forward, scale the coefficients down by a constant, and if it hardly moves, scale them up. It is possible that you will have to scale one set of coefficients (negative or positive, motor 1 or motor 2) differently than the other(s). The goal is that the robot would move at the same (constant) initial velocity you impart, within the limits of saturation. Once you have found good values for your friction coefficients, make sure to tell Labview to save these values as default but right clicking on each text box and selecting Data Operation->Make Current Value Default." **Demonstrate to your TA.**