ME 461 Laboratory #8
System Identification and Model-Based Control Design

Goals:

1. Perform data-based system identification on the robot using least-squares estimation.
2. Use a discrete transfer function model of the robot to design and simulate a controller.
3. Implement decoupled controllers on both motors on the robot.
4. Implement a coupled controller to steer the robot.

NOTE: From this point forward, you will be using 60% of the friction compensation you developed in Lab 7. This means you will need to identify the system and design and implement the controller while supplementing the control effort with 60% of the nominal value necessary to eliminate friction.

Exercise 1:

In this exercise, you are going to perform system identification of the robot car. The three-wheeled robot with friction compensation can be modeled as (linear) first-order system. In the continuous time domain, the transfer function that relates wheel velocity to control input is

$$\frac{V(s)}{U(s)} = \frac{K}{\tau s + 1}$$

where τ is the time constant of the robot and K is the DC gain. We can discretize this system by assuming a piecewise constant input signal formed through *zero-order-hold* of a discrete-time signal. This approach amounts to taking the z-transform of the sampled *unit pulse response* of the continuous system. The formula for converting from a continuous transfer function to a discrete transfer function through ideal sampling and *zero-order-hold* A/D is

$$G(z) = (1 - z^{-1})\mathbf{Z}\left\{\Lambda^{-1}\left\{\frac{1}{s}G(s)\right\}_{t=nT}\right\}$$

If we apply this result to the first-order system above, we obtain the discrete transfer function of the robot.

$$\frac{V(z)}{U(z)} = \frac{K\left[1 - e^{-T/\tau}\right]}{z - e^{-T/\tau}}$$

The corresponding difference equation is

$$v[n] = c_1 v[n-1] + c_2 u[n-1]$$

where

$$c_1 = e^{-T/\tau} \text{ and } c_2 = K[1 - e^{-T/\tau}].$$

We will now identify the two parameters c_1 and c_2 using least-squares estimation. We write the difference equation as

$$V = \Phi\hat{\theta}$$

where in Matlab

$$V := \begin{bmatrix} v[2] \\ v[3] \\ \vdots \\ v[N] \end{bmatrix} \qquad \Phi := \begin{bmatrix} v[1] & u[1] \\ v[2] & u[2] \\ \vdots & \vdots \\ v[N-1] & u[N-1] \end{bmatrix} \qquad \hat{\theta} := \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

and *N* is the number of measurements. Then, the least-squares estimate can be found in Matlab using the "\" operator

$$\hat{\theta} = \Phi \backslash V \, .$$

We are going to excite the robot with a step input and record the resulting velocity response in order to compute the unknown parameters.

Starting with your LabView program for the myRIO from Lab 7, add a SimTime Waveform block inside your simulation loop to collect a step response of the robot's speed given a step input. Continue to use a sample period of 5ms and compute the average velocity of the robot's wheels at each step. Plot this average wheel speed in your SimTime Waveform block. In the first frame of your flat sequence structure add a 3 sec "Wait" so that you have time to flip on the robot's amp enable switch before the robot is commanded to move. Then inside your simulation loop simply apply an open-loop input of 6 to the motors. Collect about 4 seconds worth of data and then stop your application. (If you did not add a "Stop Simulation" block and a Stop push button to your simulation loop in Lab 7, do that now.)

Place your robot on the floor and run your LabView program. The robot should move for a few seconds and then stop when you press your stop button.

Once you stop your application the data should still be shown in your SimTime Waveform plot in the front panel view. Save this data so that it can be pulled into Matlab. There is a number of ways you can get this data into Matlab. Right click the on the plot in the front panel view and select "Export" and then either export the data to the clipboard or to Excel. If you export the data to the clipboard you can open "Notepad++" and paste the data there. Then you can load this data file into Matlab. I am sure you can do the same with an Excel file. Plot the data in a Matlab plot window and **Show your TA.**

In Matlab you should have an array of time values and an array of average wheel velocity. Another array you will need is the input which has each of its elements set to 6 and the length of this array will be the same length as your time and velocity arrays. Using the velocity array and the input array, create the Φ matrix and the V vector. Then, use Matlab's backslash (\) operator to compute the least-squares estimate. Type `help slash` at the Matlab command line for help. When you have computed the unknown parameters, record them below.

c_1 = _____ c_2 = _____

Use the 'tf' function to generate your discrete transfer function using c1, c2 and sample period 0.005s. Create a step response plot that has both the data you collected and the identified transfer function's step response. Use 'hold on' to plot this data on the same plot. **Show to your TA.**

Exercise 2:

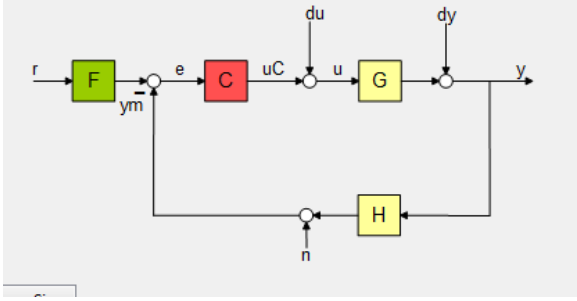
Use MATLAB's root locus tool to design a discrete PI controller for the robot. The controller will be of the form

$$\frac{U(z)}{E(z)} = K_i \frac{T_s}{2}\frac{z+1}{z-1} + K_p$$

where *e* is the error signal and *u* is the control effort. Use the parameters that were determined in exercise 1 to write the discrete transfer function of the robot. Create the transfer function in MATLAB and call `rltool` with this transfer function as the argument. You should now see the root locus view of the SISO design tool. Complete the following to customize the tool for our task.

1. Click the "Preferences" gear button and select "Options" tab, choose the "zero-pole-gain" parameterization for the compensator. This will display transfer function in the format we are familiar with.

2. Click the "Edit Architecture" icon and notice that the default block diagram configuration is being used and your controller "C" is in the error path. Click Cancel because we do not want to change anything.



3. In the "Controllers and Fixed Blocks" section, double click on your controller "C". A "Compensator Editor" window will pop up. Here you will add your discrete PI controller by adding an integrator (real pole at z = 1) and real zero to the compensator by right-clicking in the "Dynamics" pane. Start with the default zero location given. In the next step you will adjust the zero location for a good system response.

4. In the "Responses" window, right click on "IOTransfer_r2u" and choose "Plot->Step". You should now have three plots. One may be behind another so arrange the plot windows so all of them can be seen. We will use the step response characteristics to choose our gains K_p and K_i.

You should know from prelab 8 that the K_p and K_I determine the compensator zero and loop gain. To design your controller move the compensator zero along the real axis in addition to varying the loop gain. Iteratively move the compensator zero and the loop gain to achieve the following design objectives.

- Rise time (10%-90%) of no more than 0.5 seconds
- Overshoot of no more than 5%

- Settling time (5%) not more than 1 second
- Magnitude of control effort does not exceed 10.0

When you are satisfied with the design of your controller, make note of the zero location and the loop gain. Use these values to determine K_p and K_i. Record them here:

K_p = _____ K_i = _____

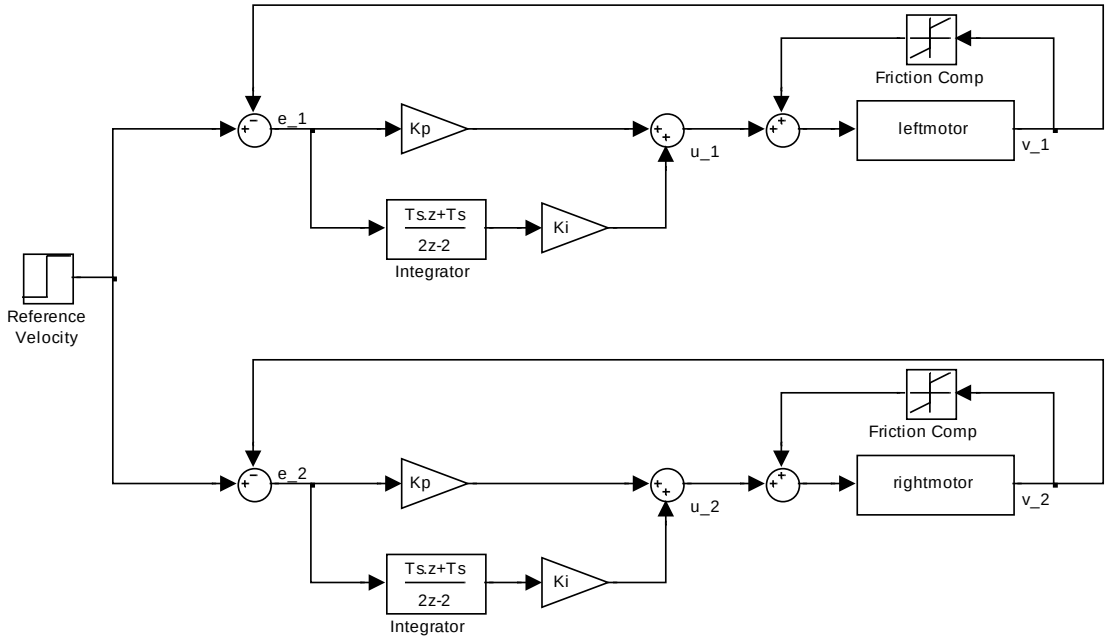
Exercise 3:

Implement the controller you designed on both motors of the robot. Use the following difference equations to form your control algorithm.

$$u[k] = K_p e[k] + K_i i[k]$$

$$i[k] = i[k-1] + T_s \frac{e[k]+e[k-1]}{2} \quad \text{(Tustin Rule)}$$

where $e[k] = r - v[k]$ for some reference signal *r* and wheel speed *v*. The block diagram for your system should resemble the following.



Decoupled PI controller diagram

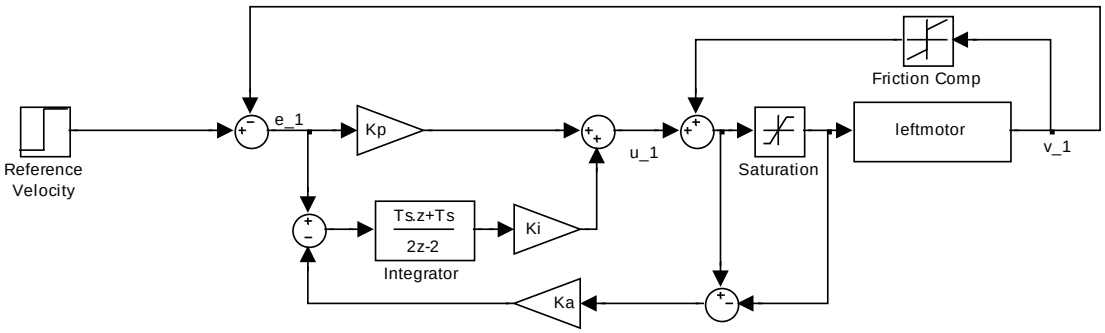
The general algorithm will be:

Set reference signal r
Measure wheel speed v
Compute error e = r – v
Increment integral by $T_s(e + e_{old})/2$
Compute control effort $u = K_p e + K_i i$
Store $e_{old} = e$

Implement the controller and check to see that the speed matches various set-points in the range of -1.7 to 1.7 ft/sec. **Demonstrate to your TA.**

It is necessary to note that integrators have "memory", which means they are affected by past behavior. Give the robot a set-point of 1 ft/s, then hold one of the wheels for a few seconds and release. Note the behavior. The wheel spins faster than the set-point to "burn off" the extra integrated error accumulated while the wheel was being held. Such *saturation* of the control input causes what is called *integral wind-up*. To prevent this behavior we must implement an *anti-windup controller*. One approach is to multiply the integral by a constant less than unity (e.g. 0.95) when the control effort is saturated to force a decay of the integral.

As an alternative, we may implement the anti-windup scheme depicted in the block diagram below. This scheme reduces the integral term based on the difference between the desired control input and the actual control input.



Feedback anti-windup scheme

Implement one of these schemes on your robot. In your code, check to see whether the control input is saturated (has a magnitude greater than 10). If it does, trim the integral using one of the

methods described above. Test the anti-windup scheme by holding and releasing one of the wheels. The wheel speed should now match the set-point after it is released without as much overshoot. **Demonstrate to your TA.**

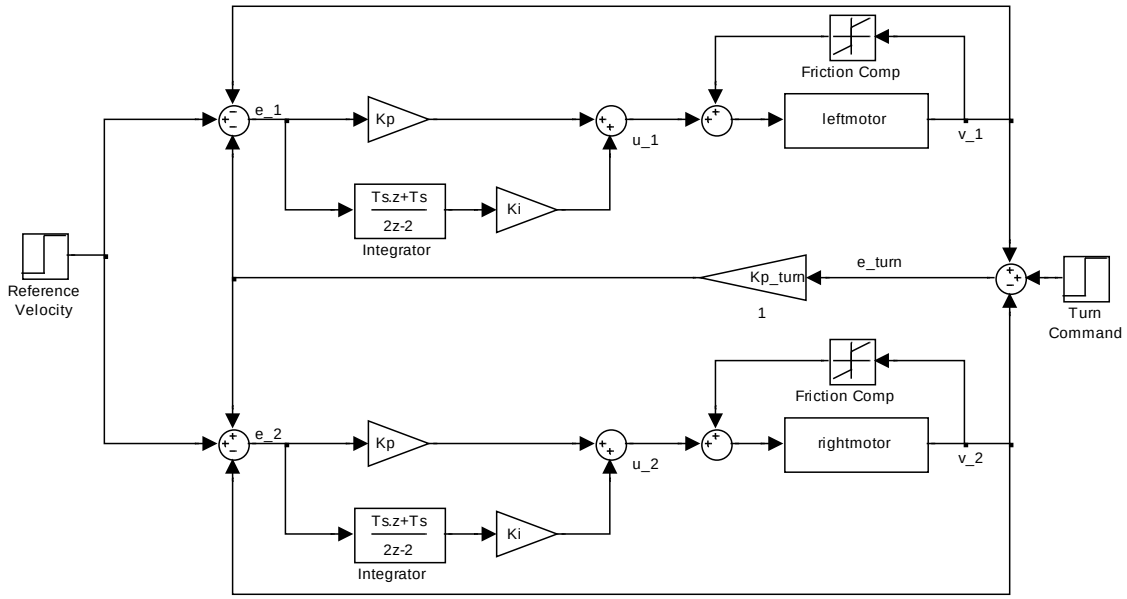
Exercise 4:

Implement a steering controller by coupling the motor control loops. Introduce a turn setpoint t_{ref} and form the turn error signal $e_{turn} = t_{ref} - (v_2 - v_1)$. You can see from this equation that the turn setpoint controls the amount by which one motor's speed exceeds the other motor's speed. Multiply the turn error by a gain K_{turn} and adjust the overall error signals as follows:

$$e_1 = ref - v_1 - K_{turn} e_{turn}$$

$$e_2 = ref - v_2 + K_{turn} e_{turn}$$

The approach is depicted in the block diagram below.



Coupled PI control structure

If we assume the steady-state error goes to zero (due to the integrators), we arrive at the following conclusions:

$$v_1 \to ref - \frac{K_{turn}}{1-2K_{turn}} t_{ref} \qquad v_2 \to ref + \frac{K_{turn}}{1-2K_{turn}} t_{ref} \qquad \frac{v_1+v_2}{2} \to ref \qquad v_2 - v_1 \to \frac{2K_{turn}}{1+2K_{turn}} t_{ref}$$

So the steering controller ensures that the average velocity tracks the reference and the turn command injects a difference in wheel speeds that is symmetric about the average velocity. Test your steering controller with several velocities and turn commands. Then using a slider control or a similar LabView control drive the robot car around the room. **Demonstrate to your TA.**