Lab 1: Steady State Error and Step Response
MAE 433, Spring 2012

Instructors: Prof. Rowley, Prof. Littman
AIs: Brandt Belson, Jonathan Tu
Technical staff: Jonathan Prévost
Princeton University

Feb. 14-17, 2012

1 Overview

In the first half of the course, we will perform a series of labs that address the control of a simple motor. In the first lab, we identify input-output relations for this system. First, the applied voltage to the motor (input) is related to the angular velocity (output). Second, the input voltage to the motor is related to the voltage measured by a tachometer. A tachometer is a device that transforms the rotation of the motor into a voltage (think about it as an inverse motor). Once these relationships are known, we use them to build models for open- and closed-loop control of the system.

2 Goals

Our hands-on goals for today are:

- Familiarize yourself with the lab equipment.
- Determine the "no dynamics" constants for the motor and the tachometer.
- Implement a proportional controller for motor speed and assess the steady state error.
- Capture open- and closed-loop transients.

There is a one-page lab report that you will hand in before leaving the lab. **Save your simulink models and data for future use.**

3 Laboratory Setup

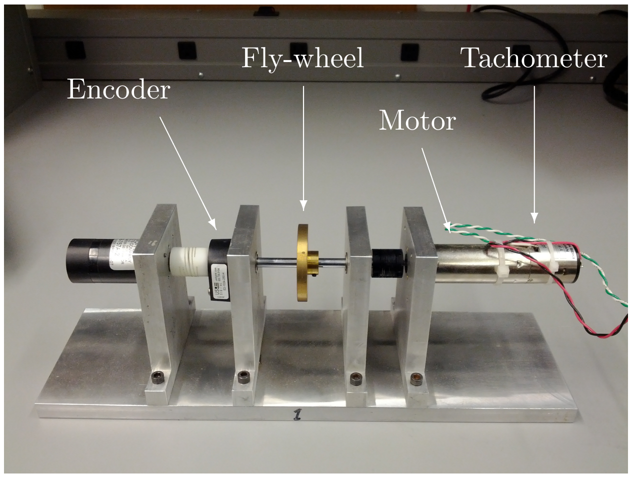


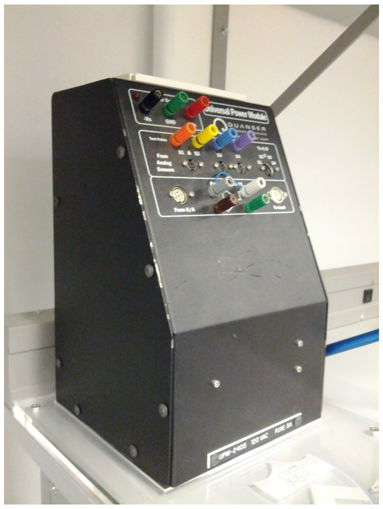
Figure 1: Motor test stand.

Motor test stand

The motor test set contains a small permanent magnet DC motor that drives a fly wheel. A voltage is applied to the motor through the green-white twisted pair. Two devices are used to monitor the output of the motor:

- An optical encoder, which is coaxial with respect to the motor and flywheel. The encoder is a low-friction angle-measuring sensor that produces 4096 counts per revolution, as measured by the Quanser hardware. We will use the encoder to determine the angular speed of the motor by differentiating the angular position signal.
- A tachometer (a permanent magnet DC generator), which converts the rotation of the motor shaft into a voltage. The tachometer is mounted inline and directly behind the motor. (On the new motorstands there are two tachometers mounted. We will use the one directly behind the motor.) According to its datasheet, the tachometer outputs roughly 2.4 V for every 1000 RPM (revolutions per minute). The tachometer output is read using the red-black twisted pair.

The motor, flywheel, and encoder are linked to each other with *flexible couplings*. The flexible couplings are needed to avoid binding as the three shafts (motor, flywheel, and encoder) turn. Binding would be occur if there were any relative misalignment of the turning shafts.


(a)

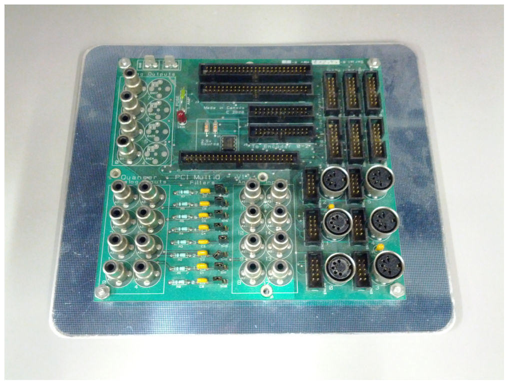

(b)

Figure 2: (a) Universal power module, (b) Quanser control board.

Control hardware

The Quanser control board is used to generate a voltage to drive the motor. While the hardware can deliver a sufficient voltage, it is not designed to produce very much current. Motors, on the other hand, require lots of current. To account for this we will connect the Quanser control board to the Quanser Universal Power Module (UPM). The UPM is a power amplifier wired to work as a voltage follower (i. e., $V_{\text{out}} = V_{\text{in}}$). It is very important that you **DO NOT** connect the Quanser board directly to the motor. Doing so will damage the Quanser hardware. The motor receives its current from the UPM via the green-white twisted pair.

Testing the hardware

Now it's time to familiarize yourself with the equipment:

1. Verify that the Quanser board is connected to the UPM input via a DIN cable.
   Use DA, channel 0.

2. Verify that the UPM output is connected to the motor via banana plugs.
   Power to the motor is supplied through the green/white twisted pair. Match the negative lead (black or green, with a protruding tab on the side) to the ground on the UPM. If you later find that the motor spins the wrong way, reverse the order of the leads.

3. Verify that the tachometer is connected to the Quanser board.
   Use AD, channel 0. The tachometer output is read through the red/black twisted pair. To connect it to the Quanser board you will need a binding post adapter (taking the signal from the two banana plugs and sending it through a single RCA plug).

4. Verify that the encoder is connected to the Quanser board.
   Use ENCD, channel 0. The encoder cable is cream-colored and should be in your toolbox. Orient the encoder plug such that the disconnected wire matches up to the "index" pin on the encoder. (Depending on your encoder, this pin might be labelled "no connection" or

something similar.) If you need help identifying the correct orientation, please see an AI or the lab technician.

5. Connect one channel of the Tektronix oscilloscope to the Quanser board via a BNC cable.
   Use DA, channel 0. This channel will be used to verify the output voltage of the Quanser board. You may need to create a "T-shaped" adapter in order to connect both the oscilloscope and the UPM to this channel. See an AI or the lab technician if you need help building this adapter.

6. Connect a second channel of the oscilloscope to the Quanser board via a BNC cable.
   Use AD, channel 0. This channel will be used to measure the output voltage of the tachometer. You will need an adapter to connect the BNC cable to the tachometer binding posts. The adapter should fit into the tachometer binding posts in a perpendicular manner. See an AI or the lab technician if you need help making this connection.

7. Make sure that the UPM is powered.
   Check that the UPM is both plugged in and turned on. The power switch is located on back of the UPM. A red light on the front of the UPM will indicate that it is powered.

8. **Before proceeding, check with an AI to make sure your setup is correct.**

4 Laboratory Procedure

Initial setup

The first step is to set up the software interface that makes it possible to interact with the Quanser hardware and provide the motor with a desired voltage.

1. Log onto the computer.
   Use the username `controls` *and the password* `G101Quanser`*.*

2. Open MATLAB and start Simulink.
   To start Simulink either type in `simulink` *in the command line or click on the Simulink icon. Then open a new model and drag in blocks from the Simulink library and connect them as needed.*

3. Create the Simulink model shown in Fig. 3.
   This will allow you to generate a voltage on the analog output of the Quanser system. Make sure that you use the Quanser MultiQ-PCI library for any Quanser blocks. You can find it under `Quanser Toolbox` → `Quanser Consulting MultiQ-PCI Series`*. Do not use any of the other Quanser libraries, as they are for different hardware.*

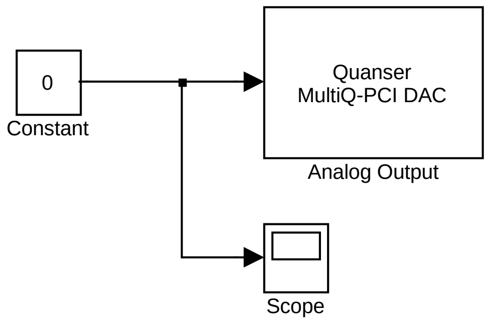


Figure 3: Model for measuring motor input voltage.

4. Set the simulation parameters for this model.
   You will need to do this every time you open/create a new Simulink model!
   (a) Click on `Configuration Parameters` in the `Simulation` drop-down menu.
   (b) Make sure that you are in the `Solver` tab (left-hand menu).
   (c) Set the `Solver` to `ODE5 (Dormond-Prince)`.
   (d) Set the `Tasking mode` to `SingleTasking`.
   (e) Click `OK` to apply the settings.

5. Build the model by clicking `Build` under the WinCon drop-down menu.
   This generates C code that will be used to control the motor in real time. If you change the structure of your model, for example by adding a block, you will have to rebuild it. However, you are allowed to change things like the value of the constant without rebuilding. You can even do this while the model is running.

6. Monitor the motor input voltage by creating a `Scope` and `Digital Meter`, which are located under the `Plot` drop-down menu of the WinCon server (not Simulink).
   For each, you will need to specify the variables to display. The variable names correspond to the blocks in your Simulink model. In the `Scope`*, it is possible to display multiple variables simultaneously, which will be important later. To change the displayed variable, go to* `Variables` *under the* `File` *drop-down menu of the* `Scope/Digital Meter`*.*

7. Set the value of the `Constant` block to 3. Click `Start` on the WinCon server to run the model. Click `Stop` to stop it.
   Sometimes there is a lag that causes the model to keep running after you click `Stop`*. If this happens, start the model again and stop it again. To avoid this, rather than using the* `Stop` *button, use the* `Pause/Break` *button on the keyboard to stop the model.*

8. Run the model for various values of the constant block. **DO NOT exceed 10 V!** Verify that the voltages read by the `Scope`, `Digital Meter`, and oscilloscope match.
   You should notice that the motor only spins if the input voltage is greater than 1 V (approximately).

9. Replace the `Constant` block in your Simulink model with a `Signal Generator`. Set the signal generator to output a sine wave with a frequency of 1 and an amplitude of 4. Run the model.
   Make sure the model is stopped before you edit it. After editing, remember to rebuild the model. You should see the input voltage and motor fly-wheel oscillate.

Encoder verification

Next, you will verify that the encoder works properly.

1. Open a new Simulink model and set the WinCon solver options.
2. Create and build the Simulink model shown in Fig. 4
   The block labelled "Count" is actually just a `Scope` *with the name changed.*

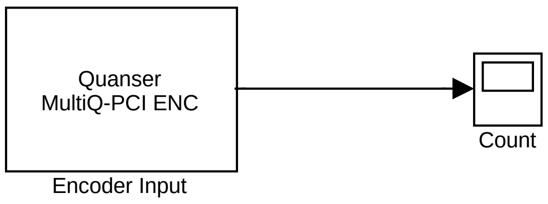


Figure 4: Model for reading the encoder output.

3. Create a `Digital Meter` to read the encoder count and run the model.
4. Manually turn the fly-wheel a full revolution and verify that the count is 4096. Stop the model.
   In this Simulink model there is no block to provide a voltage to the motor, so it will not turn on its own.

Measuring angular speed

To determine the angular speed of the motor, we will need to differentiate the encoder output, which is a measure of the motor's angular position.

1. Create the model shown in Fig. 5.
   The `Transfer Fcn` *block is located in the* `Continuous` *library. The* `Gain` *block can be found in the* `Math Operations` *library.*

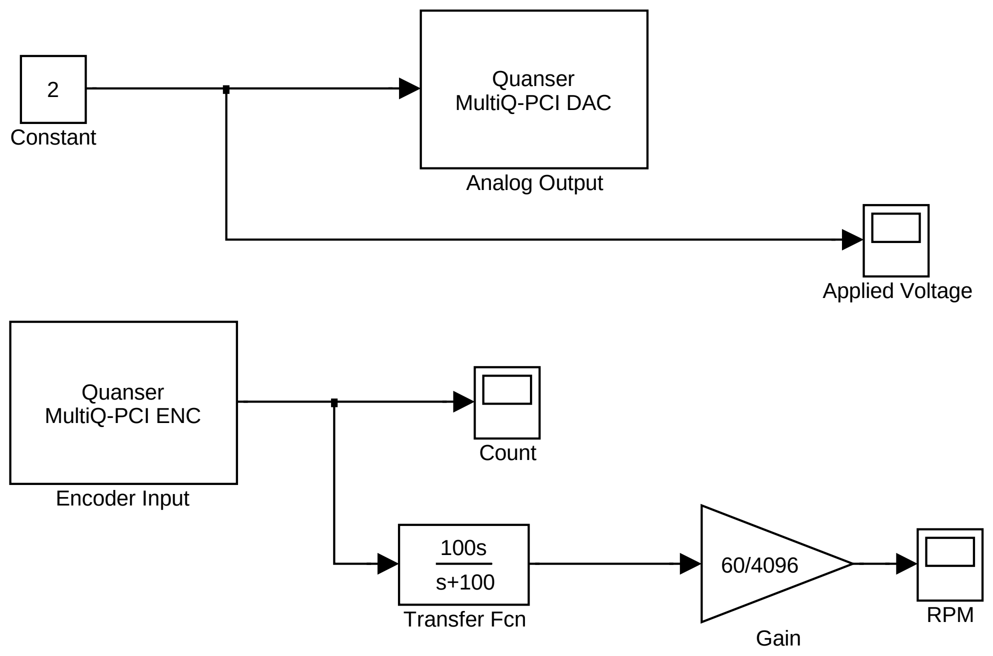


Figure 5: Model for measuring the angular speed of the motor for a given input voltage.

2. Set the `Gain` to 60/4096. Open the `Transfer Fcn` block and set the numerator coefficients to `[100 0]` and the denominator coefficients to `[1 100]`.
   We will learn more about how to specify transfer functions in Matlab later on.

3. What does the transfer function do? What does the gain do? (Hint: What does the encoder measure? How does this relate to what we want to measure?) **Check in with an AI to discuss this before proceeding.**

4. Verify that the measured angular speed is correct using a strobe.

   (a) Create a `Digital Meter` to read the measured angular speed.
   (b) Run the model.
   (c) Shine a strobe light on the fly-wheel.
   If you do not have a strobe at your station, you will have to share with another group.
   (d) Adjust the strobe frequency so that a mark on the fly-wheel appears stationary.
   (e) Verify that the strobe frequency and the `Digital Meter` reading match (approximately).
   (f) Double and halve the strobe frequency. Verify that stationary double and single images are observed, respectively.

Accounting for stick-slip friction

You saw before that the motor does not move if the applied voltage is too small. This is due to stick-slip friction associated with the motor brushes. (Brushless motors are far more linear, but they are more expensive and complicated to control.) Fig. 6 shows the relationship between the steady-state tachometer voltage (which is proportional to the motor speed) and the applied voltage. Note the “dead band” near the origin. Before we can do control, we will need to account for this behavior.

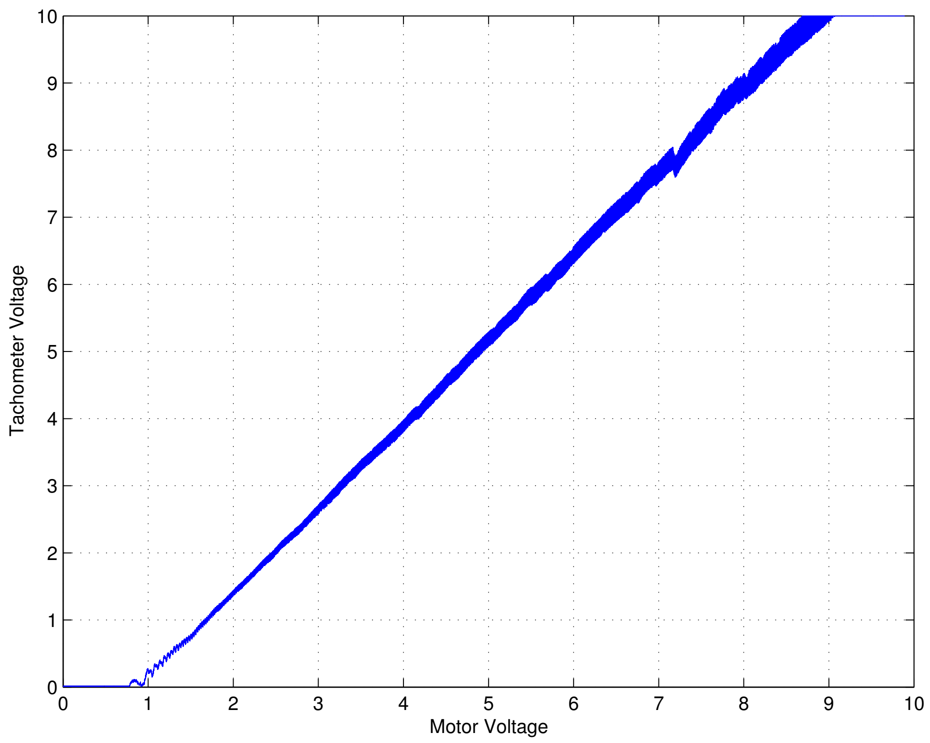


Figure 6: Illustration of the dead band region of a brushed motor. The motor does not move until applied voltage exceeds 0.8 V.

1. Give the motor an applied voltage of 5 V and run it for about 30 seconds to warm it up.
   Use the model from the angular speed section above (Fig. 5). We need to make sure the motor is warm because we only want to model its behavior in its warmed-up state. The friction may be different when the motor is cold.

2. Determine the smallest applied voltage for which the motor will run. **Be sure to record this dead-band voltage for future use.**
   The motor is not perfectly symmetric, so a certain applied voltage may cause the motor to turn some times, but not others. Do your best to find a minimum voltage that usually gets the motor to turn.

3. Does it matter if you find the dead-band voltage by increasing the voltage until the motor moves versus decreasing it until the motor stops? (Hint: think about different kinds of friction.) **Discuss this with an AI before proceeding.**

4. Add a `Coulomb & Viscous Friction` block to your model, as shown in Fig. 7. Open the block and set the `Coulomb friction value (Offset)` to the dead-band voltage you determined above.
   The `Coulomb & Viscous Friction` *block can be found in the* `Discontinuities` *library. This block will account for the dead band so that small applied voltages will still move the motor. The block will initially contain a vector for the offset value. Simply delete this and enter your dead-band voltage as a scalar.*

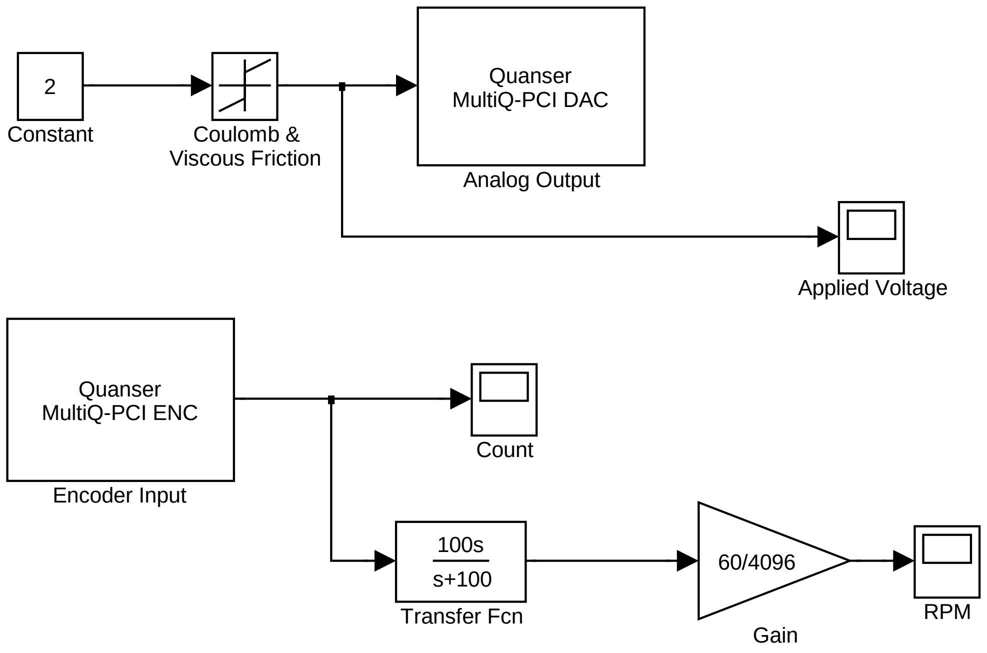


Figure 7: Model that accounts for stick-slip friction. Compare to Fig. 5.

Determining the motor constant

Next you will determine the motor constant $K_{\text{motor}} = \frac{\text{Angular speed}}{\text{Applied voltage}}$, in units of RPM/Volts.

1. For applied voltages of 2, 4, 6, and 8 V, do the following:

   (a) Create a `Scope` to read the angular speed.

   (b) Run the model from Fig. 7 for about 5 seconds.

   (c) Record the steady-state angular speed.
   The angular speed should ramp up and then reach steady-state within 5 seconds. The steady-state value may drift over time. Do your best to identify this value, but don't worry about trying to capture it exactly.

2. In Matlab, plot the steady-state angular speed versus the applied voltage. You should see a linear relationship.

3. Use `polyfit` to identify the slope of this line. This slope is the motor constant K_{motor}.

4. **Record the motor constant for future use. Before proceeding, check in with an AI to make sure your motor constant is reasonable.**

Determining the tachometer constant

Next you will determine the tachometer constant $K_{\text{tach}} = \frac{\text{tachometer voltage}}{\text{Angular speed}}$, in units of Volts/RPM. With this constant, we can use the tachometer voltage as a direct measurement of the motor speed, which is preferable to differentiating the encoder signal, as we did above. The procedure for identifying K_{tach} will be very similar to that used above to identify the motor constant K_{motor}.

1. Create the Simulink model shown in Fig. 8.
   You can simply add the necessary blocks to the model from Fig. 7.

2. Open the `Analog Input` block and set the `Range` to 10.
   This is very important! The Quanser board cannot take more than 10 V of input voltage. You need to do this every time you use an `Analog Input` *block.*

3. For applied voltages of 2, 4, 6, and 8 V, do the following:

   (a) Create a `Scope` to read the tachometer voltage.

   (b) Run the model from Fig. 8 for about 5 seconds.

   (c) Record the steady-state tachometer voltage.
   As before, the tachometer voltage should ramp up and then reach steady-state within 5 seconds. Do your best to identify the steady-state value, but don't worry about trying to get it exact.

4. In Matlab, plot the steady-state tachometer voltage versus the angular speed. You should see a linear relationship.

5. Use `polyfit` to identify the slope of this line. This slope is the tachometer constant K_{tach}.

6. **Record the tachometer constant for future use. Before proceeding, check in with an AI to make sure your tachometer constant is reasonable.**

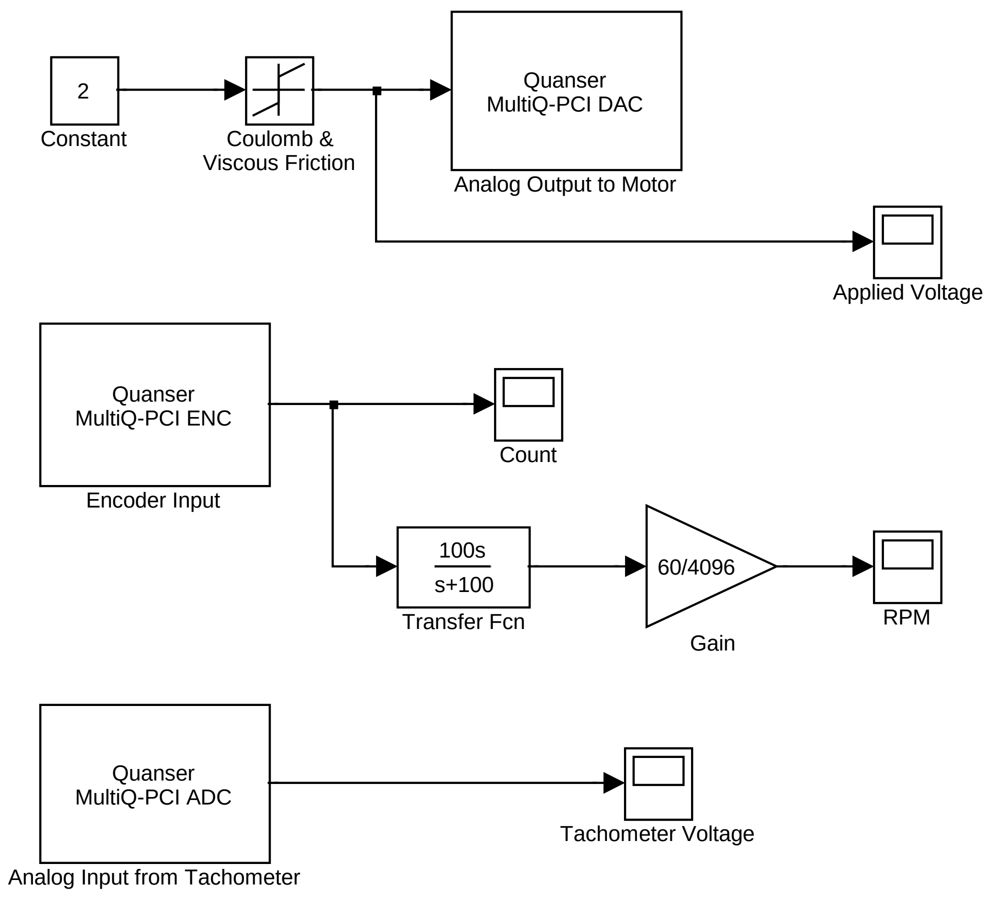


Figure 8: Model for determining the tachometer constant.

Open-loop control

Now that we have characterized the behavior of the motor, we can apply open-loop control. The goal is to design a control law that will make the motor spin at the desired speed. Rather than work directly with the angular speed, we will instead use the tachometer voltage (which is of course proportional to the speed).

1. Create the Simulink model shown in Fig. 9.
   The block labelled "Motor-Tachometer" is a `Subsystem` *block, found in the* `Ports and Subsystems` *library. You should open it and edit it to look like the model shown in Fig. 10. We hide these blocks in the* `Subsystem` *so that our Simulink model resembles the canonical block diagrams often used in controls.*

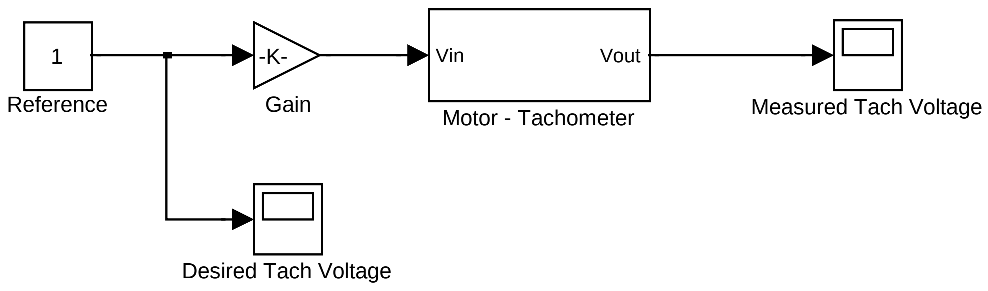


Figure 9: Model for open-loop control.

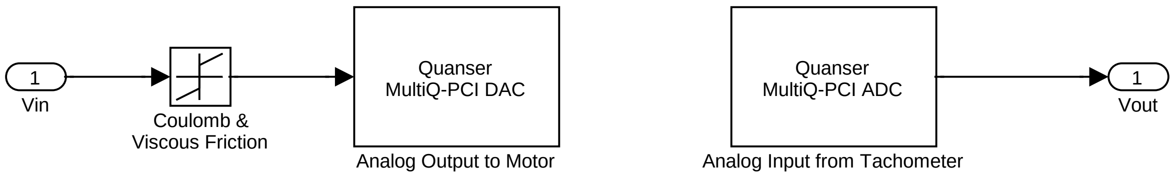


Figure 10: Subsystem block contents for open-loop-control model.

2. Derive an expression for an open-loop gain that will yield perfect reference tracking.
   Remember, we want a gain that will take a desired tachometer voltage and convert it to an appropriate applied voltage. How does this relate to the motor and tachometer constants you identified earlier?

3. Set the gain in your model (K) using the expression you derived.

4. Run your model for several reference voltages, for example 1, 2, 3, 6, and 8 V. Does the system behave as you expect? Is there any difference in the behavior for small versus large reference voltages?

5. **Before proceeding, check with an AI to discuss your results.**

6. Record the step response of the motor with open-loop control.
   (a) Replace the `Constant` block (reference tachometer voltage) with a `Step` block.
   (b) Open the `Step` block and change the step time to 1 second and the step amplitude to 6 V.
   (c) Build the model and create a `Scope` that reads both the `Step` (reference voltage) and the tachometer voltage.
   (d) Run the model for approximately 5 seconds.
   If you allow the simulation to go longer than 5 seconds, the screen will reset and you will be unable to save the initial data. To stop this from happening, go to the `Update` *drop-down menu and click on* `Buffer`*. Change the buffer time to set how long WinCon will wait before resetting the window.*
   (e) Save the data from the `Scope` by going to the `File` drop-down menu and clicking on `Save`, then selecting `Save as MAT-file`.
   You will need to plot this data later.

Closed-loop control

Next we will implement closed-loop control and see how the performance compares to open-loop control. Again, we will specify a reference tachometer voltage and try to match it.

1. Create the model shown in Fig. 11.
   Use the same `Subsystem` *as in the model for open-loop control. The circular object is a* `Sum` *block, which can be found in the* `Math Operations` *library.*

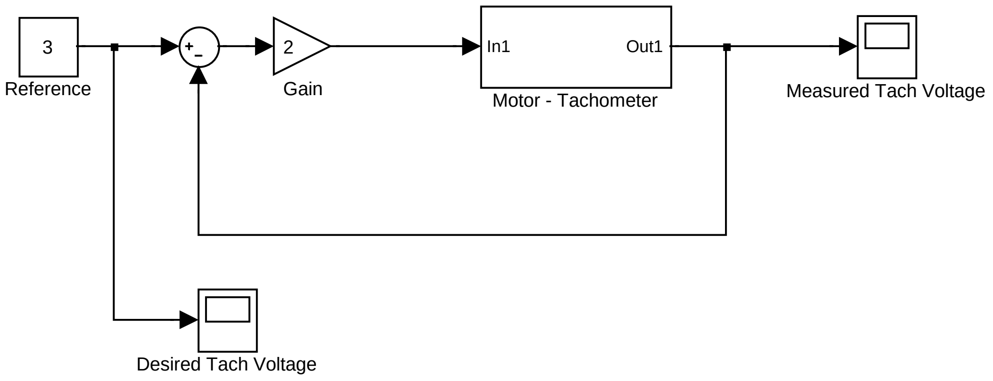


Figure 11: Model for closed-loop control. Subsystem contents are shown in Fig. 10.

2. Determine an expression for any steady-state error that you expect from this system. **Discuss this with an AI before proceeding.**

3. For a given reference voltage (< 10 V), try different controller gains K (≤ 4). How does the performance change? Does it match your prediction from above?

4. Try a high controller gain $K > 4$, **but only for a few seconds**. The motor should whine and buzz. Can you explain why this is happening? (Hint: how does noise affect the system?) *Don't let the motor whine and buzz for too long, or it will overheat!*

5. Put a low-pass filter in the feedback loop (transfer function $200/(s + 200)$). Run the system with a high gain $K > 4$. How is the performance different from before? Can you drive the steady-state error to zero? **Discuss this with an AI before proceeding.**

6. Record the step response of the system with closed-loop control, for controller gains of 2, 4, and 50:

   (a) Replace the `Constant` block (reference tachometer voltage) with a `Step` block.

   (b) Open the `Step` block and change the step time to 1 second and the step amplitude to 6 V.

   (c) Set a controller gain $K = 2$.

   (d) Build the model and create a `Scope` that reads both the `Step` (reference voltage) and the tachometer voltage.

   (e) Run the model for approximately 5 seconds.
   Again, don't let the window reset or you will lose the initial data. Set the buffer length as necessary to avoid this.

(f) Save the data from the `Scope` to a `MAT` file as before.
You will need to plot this data later.

(g) **Repeat this for a controller gains $K = 4,\ 50$.**
For very large K you should notice that the tachometer voltage "clips" (saturates) at 10 V.

5 Deliverables

- A single plot of the four step-responses. There should be four curves on this plot: the step input, open-loop step response, closed-loop step response with $K = 2$, closed-loop step response with $K = 4$, and closed-loop step response with $K = 50$.
- A plot of the data and curve fit used to determine the motor constant K_{motor}
- A plot of the data and curve fit used to determine the tachometer constant K_{tach}

Make sure that the values of your constants are clearly noted. Turn in one report per group.