ME 360: FUNDAMENTALS OF SIGNAL PROCESSING, INSTRUMENTATION AND CONTROL

Speed Control of a DC Electric Motor

1. CREDITS

Experiment Originated: Professors T-C. Tsao, October, 1995, and Norman Miller, January, 1997
Updated: D. Block, April 2018

2. OBJECTIVE

The objective of this experiment is to study proportional-integral control of the speed of a DC motor. A secondary objective is to become familiar with the use of SIMULINK as a system simulation tool.

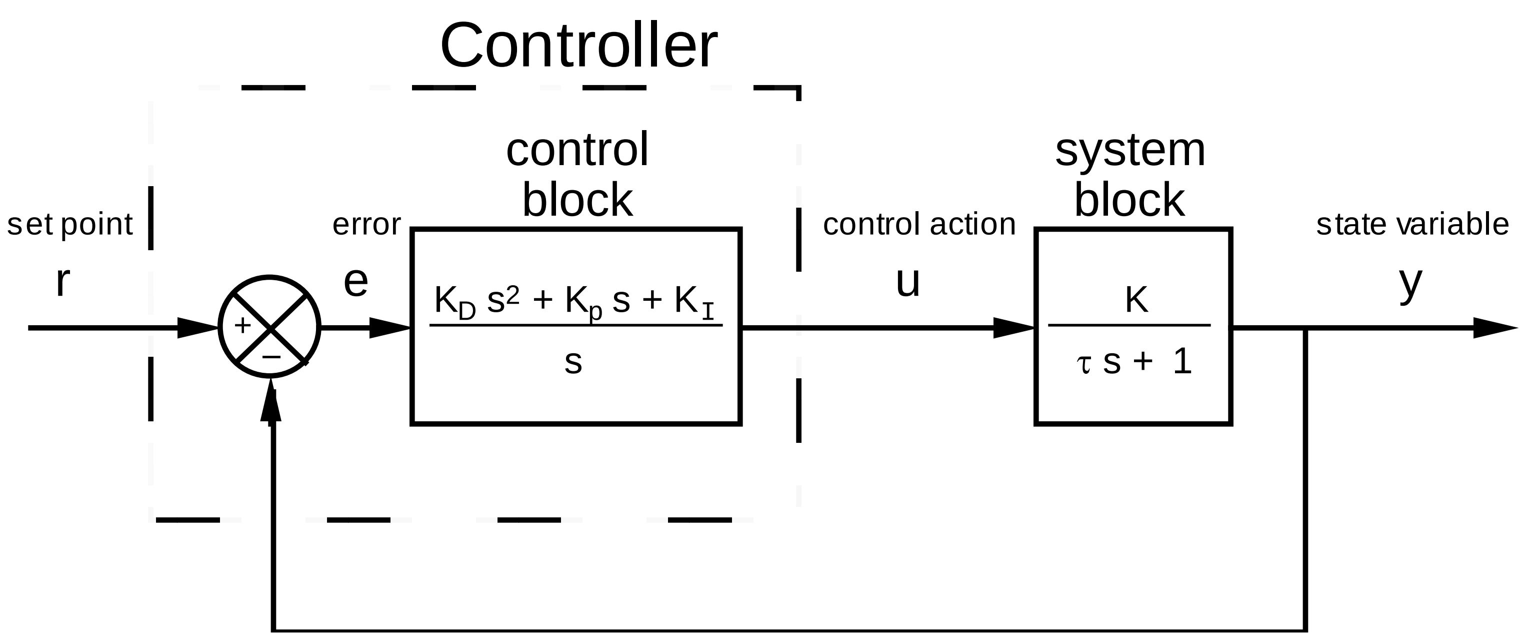
3. KEY CONCEPTS

(a) As we know from previous experiments, a DC motor with voltage as the input and speed as the output behaves as a quasi-linear, first-order system with steady-state gain K and time constant τ. For the motor-generator system used in our laboratory, K is typically between 1.1 and 1.2 depending on the motor and various factors such as bearing wear. τ is typically between 40 and 70 ms.

(b) A quasi-linear system is one that can be modeled as linear for the purpose of control but with dynamic parameters (K and τ) that vary with motor speed and motor age.

(c) We can use the changes in K and τ as the motor ages as a means of diagnosing motor health (bearing condition).

(d) Proportional-integral-derivative (PID) control is one of the most common control methods. A block diagram of PID control is given below. Special cases of PID control include: (i) P (only) control where K_I and K_D are both 0, (ii) PI control where K_D is 0, and (iii) PD control where K_I is 0.



(e) In this experiment, we only consider P and PI control. Derivative (D) control is not considered.

(f) Proportional control is the primary workhorse of PID control. The proportional gain K_p determines how quickly a system responds.

(g) The system generally becomes more responsive (rise time is reduced) as proportional gain is increased.

(h) A drawback of this increased responsiveness is that oscillations die away more slowly. Such oscillations may originate (i) in the system, (ii) in the input, or (iii) from other factors outside the system.

(i) P-only control has *nonzero* steady-state error $e_{ss} = r / (1 + K K_p)$. PI control has *zero* steady-state error.

(j) The primary benefit of adding integral control to this system is elimination of the steady-state error.

4. SYNOPSIS OF PROCEDURE

In this experiment, we add speed control to the motor-generator system used in the previous experiment. We investigate both proportional (P) and proportional-integral (PI) control. We first simulate the motor-generator system on the computer using SIMULINK. We then add speed control to the simulation, and evaluate the performance of the control system. Finally, we test the speed control on the actual motor-generator system and document the differences between simulated and actual system performance.

More specifically, the MATLAB/SIMULINK software and PC-based hardware are used as follows.

(a) The motor-generator system is simulated in software using the steady-state gain and time constant found experimentally in the previous two experiments. The SIMULINK extension of the MATLAB software is used for this purpose.

(b) A proportional-integral-derivative controller is added to the simulation. The effect of proportional gain on the performance of a proportional-only control system is examined. The steady-state error and load sensitivity of proportional control are studied.

(c) The ability of integral control to eliminate the steady-state error and load sensitivity of proportional control is then shown. The effect of integral gain on controller performance is also demonstrated.

(d) Proportional-integral control is tested on the actual motor-generator system using the PC-based hardware to implement the controller digitally. The ability of the controller to maintain constant motor speed under varying load conditions is shown. The effect of the discrete time step on control system stability is also examined.

5. PROCEDURE

Wire the System

Figure 1 shows how the system should be wired. A brief summary of the connections is given below.

<u>DAC Output to Power Amplifier</u>

A gray shielded cable, with three banana plugs on each end, connects Analog Output Channel 0 to the input of the Power Amplifier (labeled "Amplifier" on the patch panel). Red is positive or high, black is negative or low, and white is the shield. At the amplifier end, connect red to red, black to black and white to white. At the DAC end, connect red to red and both black and white to black.

<u>Amplifier Output to Motor Input</u>

A second gray shielded cable connects the output of the Power Amplifier to the input of the motor. Red is positive or high, black is negative or low, and white is ground. Follow red-black-white color coding on each end of the cable.

<u>Generator Output to ADC Input</u>

Two banana-plug patch cords connect the generator output to Analog Input Channel 0 of the ADC. On the generator end, orange is positive or high and gray is negative or low. On the analog-input end, red is positive or high and black is negative or low.

<u>Connections to Oscilloscope</u>

The DAC output is routed to Channel 1 of the oscilloscope, and the generator output is routed to Channel 2 so that the system input and output waveforms can both be displayed. Two cables each with a BNC connector on one end and a pair of banana plugs on the other end are used for this purpose. Red-black color coding is followed.

<u>Connections to Digital Multimeter</u>

A pair of banana plug patch cords are used to connect the inputs of Analog Input Channel 0 to the voltage inputs of the digital multimeter so that the generator output voltage can be measured. Red-black color coding is again followed.

<u>Push Button</u>

The push button from the drawer of the station is connected to the phone jack on the patch panel in the section labeled "Amp Inhibit".

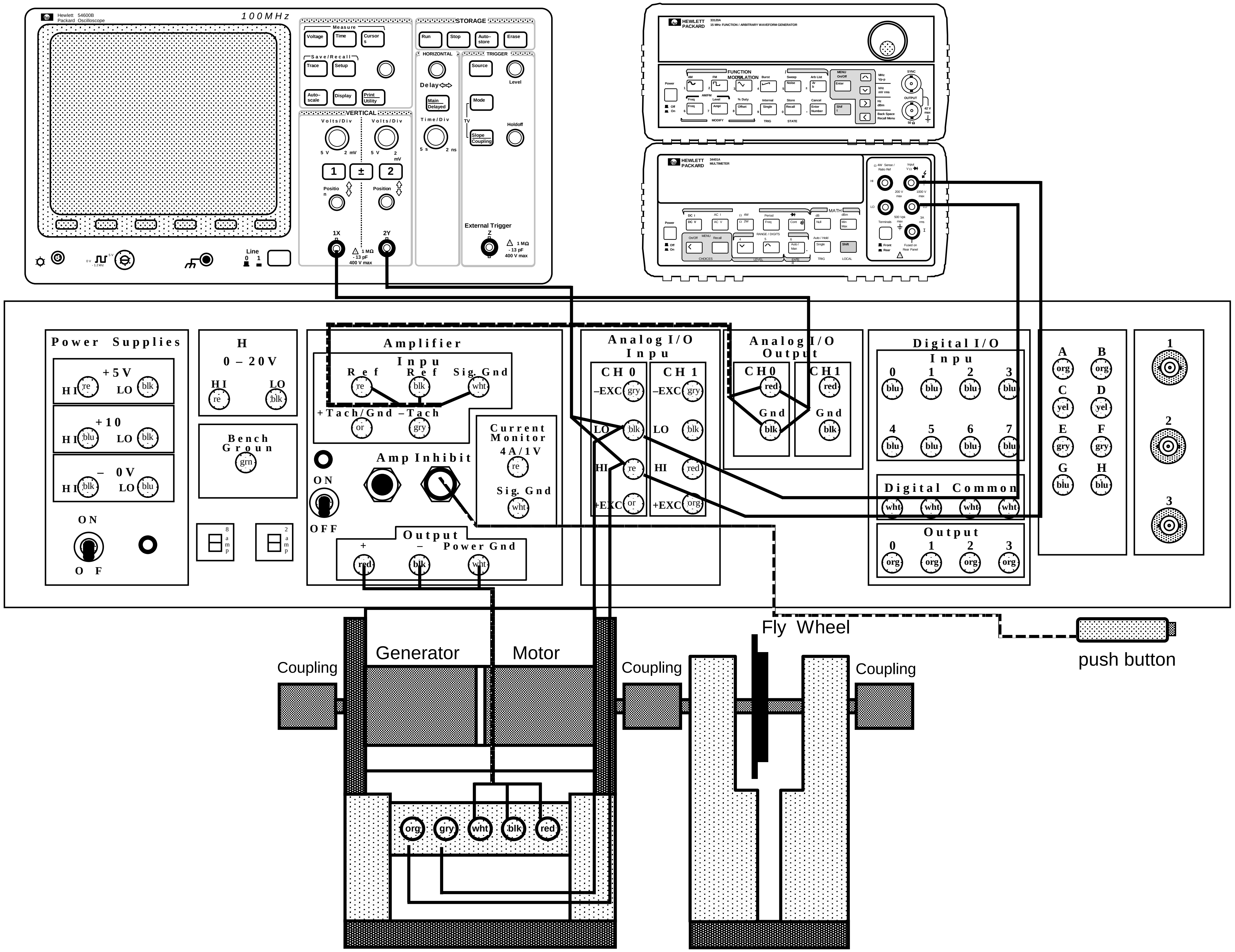


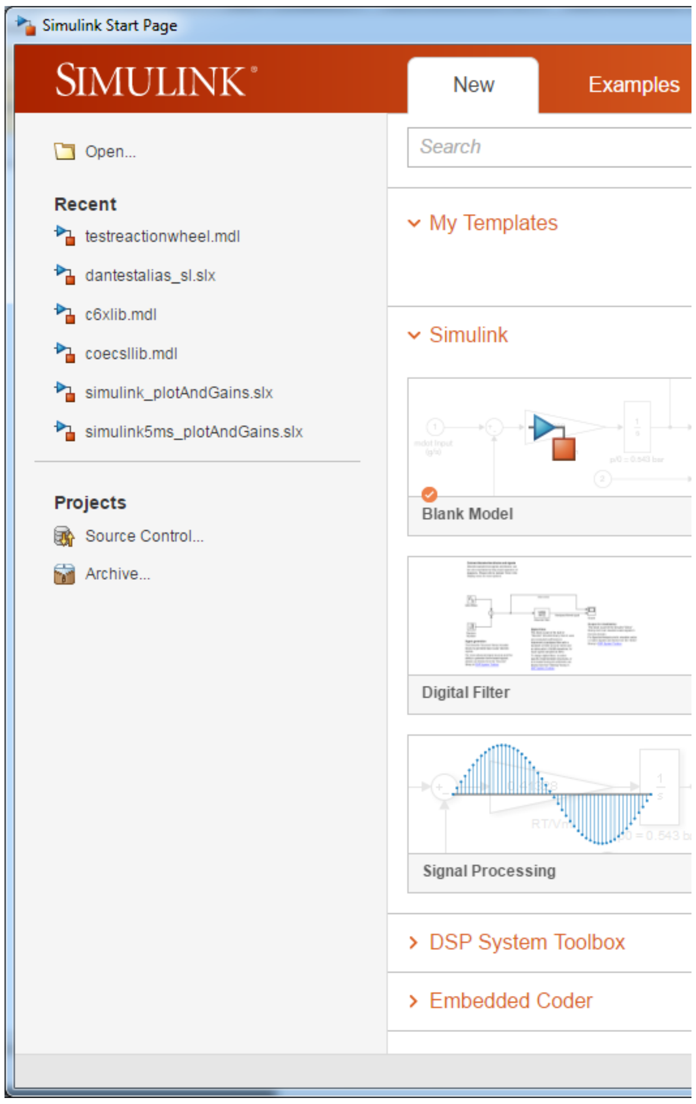
Figure 1. Wiring diagram and equipment layout for speed control experiment.

Part 1. Simulate Motor-Generator System

(a) Power on the station and the computer. Start MATLAB and enter the "`simulink`" command at the MATLAB prompt. The SIMULINK start page should appear as shown to the right.

(b) Create a new SIMULINK model by selecting the "Blank Model"

(c) Construct a block-diagram for the motor-generator system as shown below in the window that appeared in the previous step.



Helpful Tips

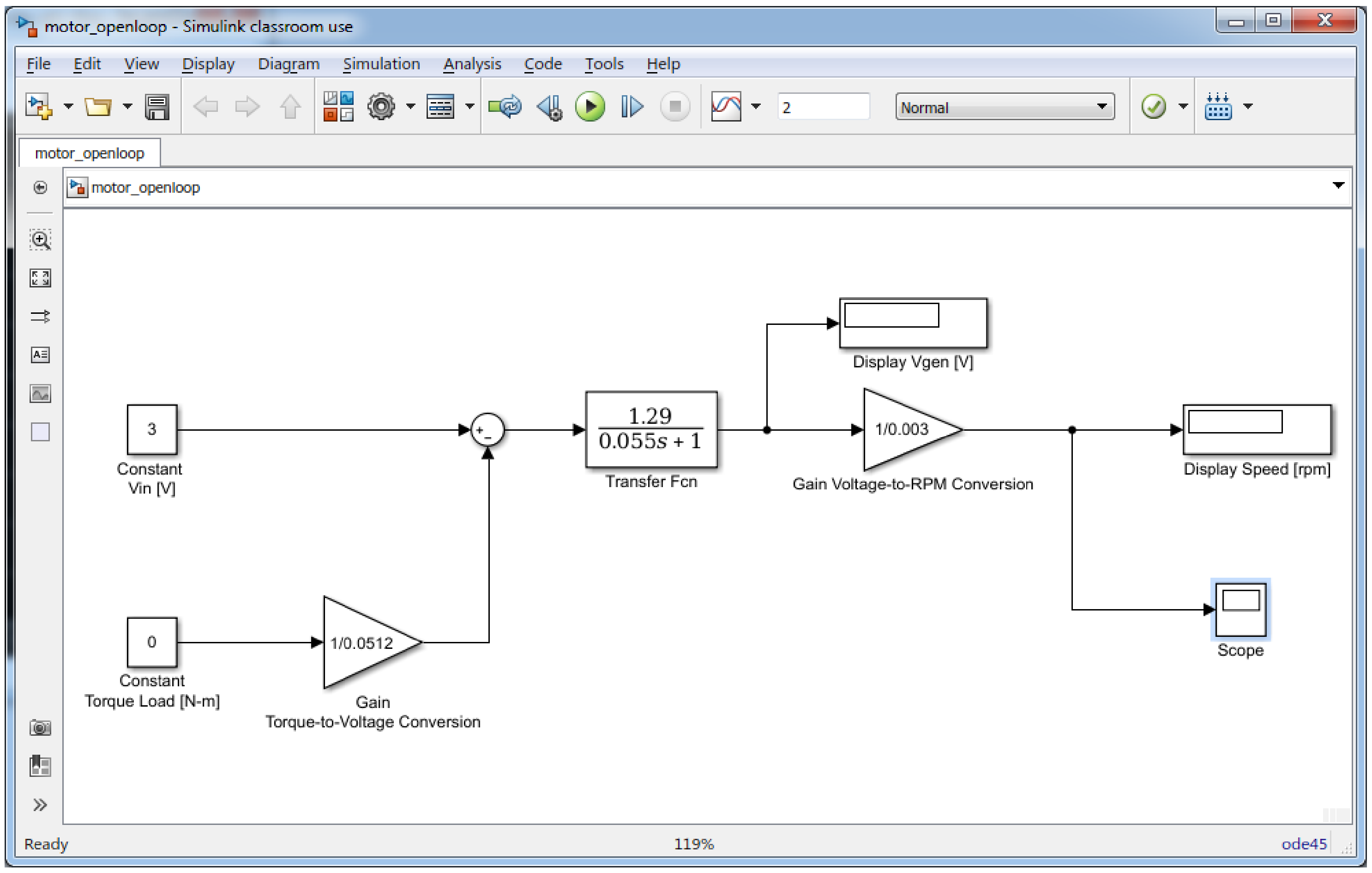
(i) The required blocks can be found from the SIMULINK library under Simulink. Simply drag the blocks into the window using the mouse. Resize blocks and edit text as desired.

(ii) Double-click a block to change its numerical value.

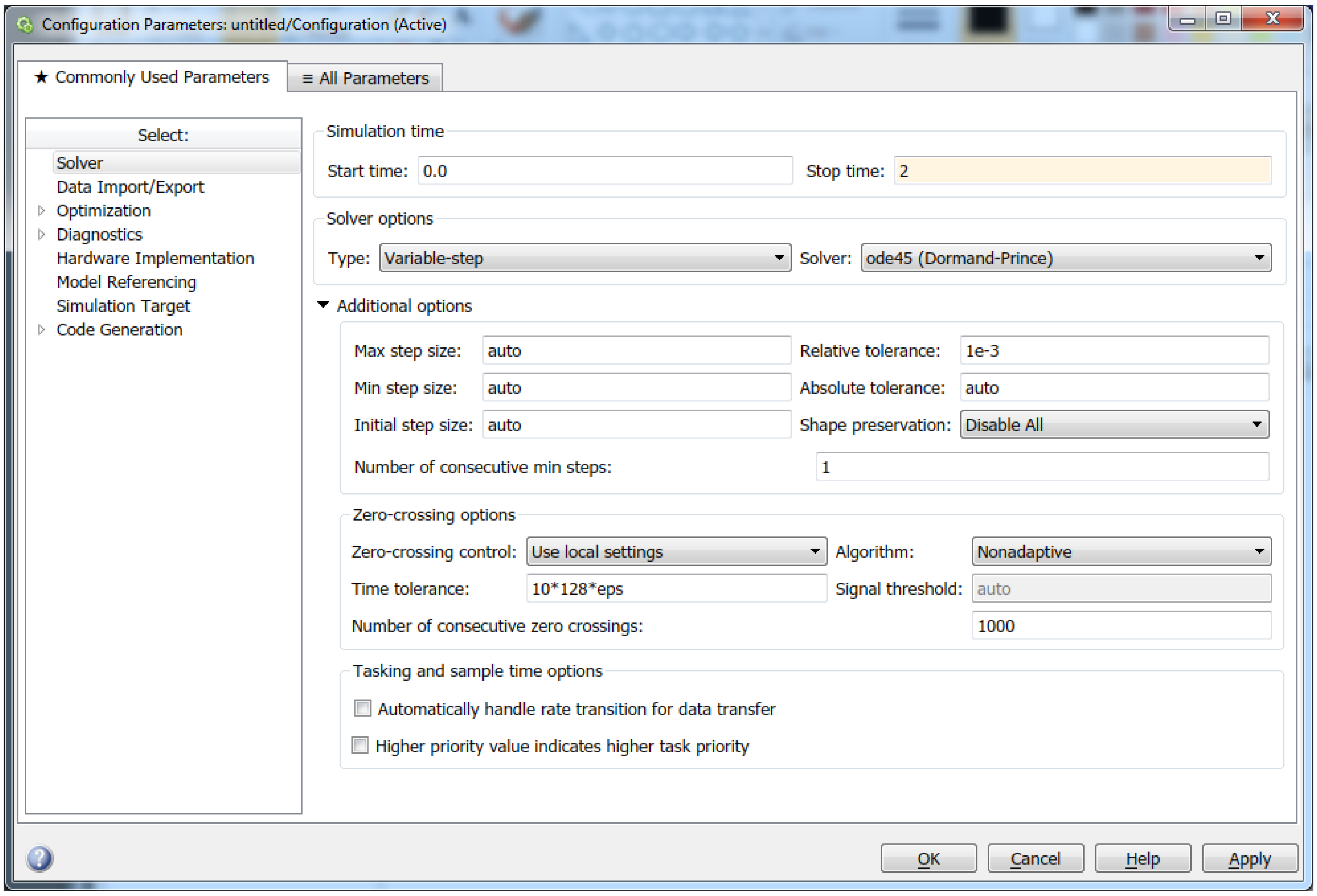
(iii) To draw signal lines between blocks, hold-down the left mouse-button and drag.

(iv) To draw a signal line that branches-off from another signal line, hold-down the right mouse-button and drag.

(v) To change a block's orientation (e.g. to flip a block), select "Flip" under the "Format" drop-down menu.



(d) Choose "Model Configuration Parameters" under the "Simulation" drop-down menu and select “Solver” to bring-up the following window.



Configuration Parameters: untitled/Configuration (Active)

★ Commonly Used Parameters | ≡ All Parameters

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation

Simulation time

Start time: 0.0 Stop time: 2

Solver options

Type: Variable-step Solver: ode45 (Dormand-Prince)

Additional options

Max step size:	auto	Relative tolerance:	1e-3
Min step size:	auto	Absolute tolerance:	auto
Initial step size:	auto	Shape preservation:	Disable All
Number of consecutive min steps:			1

Zero-crossing options

Zero-crossing control:	Use local settings	Algorithm:	Nonadaptive
Time tolerance:	10*128*eps	Signal threshold:	auto
Number of consecutive zero crossings:			1000

Tasking and sample time options

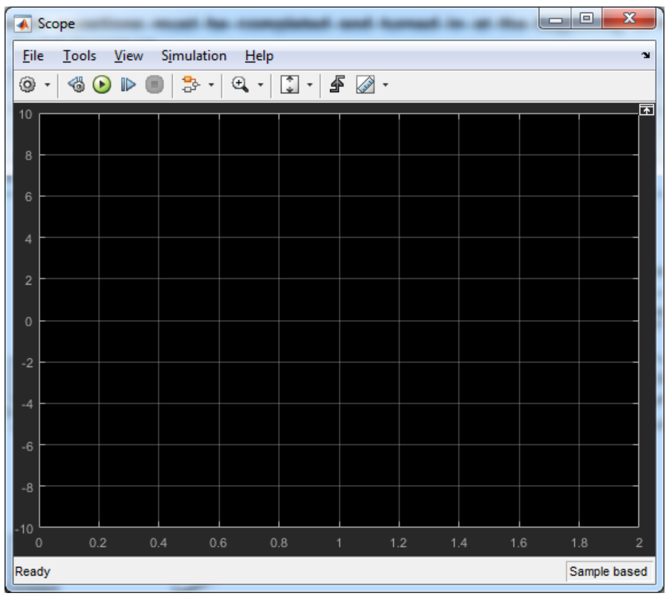
- [ ] Automatically handle rate transition for data transfer
- [ ] Higher priority value indicates higher task priority

OK | Cancel | Help | Apply

(e) Enter "2" seconds for the simulation stop time. In “Solver options”->“Type” select “Variable-step”. In “Solver options”->”Solver” select “ode45 (Dormand-Prince)”. Then click "OK".

Scale Scope Axes

(f) Double-click the scope block in the model. A window should appear similar to the one shown to the right.



(g) Right mouse-click the plot and choose “Configuration Properties”. In the “Display” tab, set the Y-limits (Minimum) value to "0" and Y-limits (Maximum) to "1300".

Create Personal Directory on C: Drive

(h) At the MATLAB command prompt, enter "cd C:\matlab\me360" to change the current working directory.

(i) Enter "cd" to list the current directory. Make sure it reads "C:\matlab\me360".

(j) Enter "mkdir *directory name*" to make a personal directory. Use your university account username in place of the words "*directory name*".

(k) Enter "cd *directory name*" to change to your personal directory.

Save Local Copy of Model on C: Drive

(l) Choose "Save As" under the "File" drop-down menu to bring up the save dialog box.

(m) Move through the directory structure to your personal directory, "C:\matlab\me360\your directory".

(n) Enter a file name for the model and save it in your directory.

Run the Simulation

(o) Click the green “Play Button” to start the simulation.

(p) Record the motor speed and generator output voltage at steady-state.

(q) Observe the response on the scope. Autoscale the Y-axis, if necessary, by selecting the Binocular icon.

(r) The “torque load” constant will be used in later sections to simulate an external load added to the motor’s shaft.

(s) Re-run the simulation for the specified cases on the Data Sheet (torque load should be 0 for all cases) and record the simulated speed.

Part 2. Simulate Motor-Generator System with PI Speed Control

(a) In your motor Simulink simulation file add blocks to create a PI controller.

- Replace the open-loop input “Constant Vin [V]” with your PI controller (multiple blocks).
- The set-point (input to the controller) should be a constant source and have the units of RPMs.
- The feedback to generate the error signal is the speed of the motor in voltage (V_{gen}). That means you will have to convert your set-point to units of volts (V_{gen}) before subtracting the feedback signal. RPM*0.003 = V_{gen} units.
- Note that test case 1 is open-loop, so there will be no feedback. This is to verify that your system is still equivalent to the results from part 1.
- Use the “Gain” block for the proportional and integral gains.
- Use the “Integrator” block to integrate the error signal.
- Also find in the “Discontinuities” group the “Saturation” block. The benches DAC can only output +/- 10V so add the saturation dirently block before the input to the motor transfer function block to make sure no more than +/- 10V is applied to the motor. You will need to edit its parameters to make sure it is limiting +/- 10.

(b) Save a local copy of the model in your personal directory, "C:\matlab\me360\your directory".

(c) Click the green “Play Button” to start the simulation.

(d) Record the motor speed and generator output voltage at steady-state.

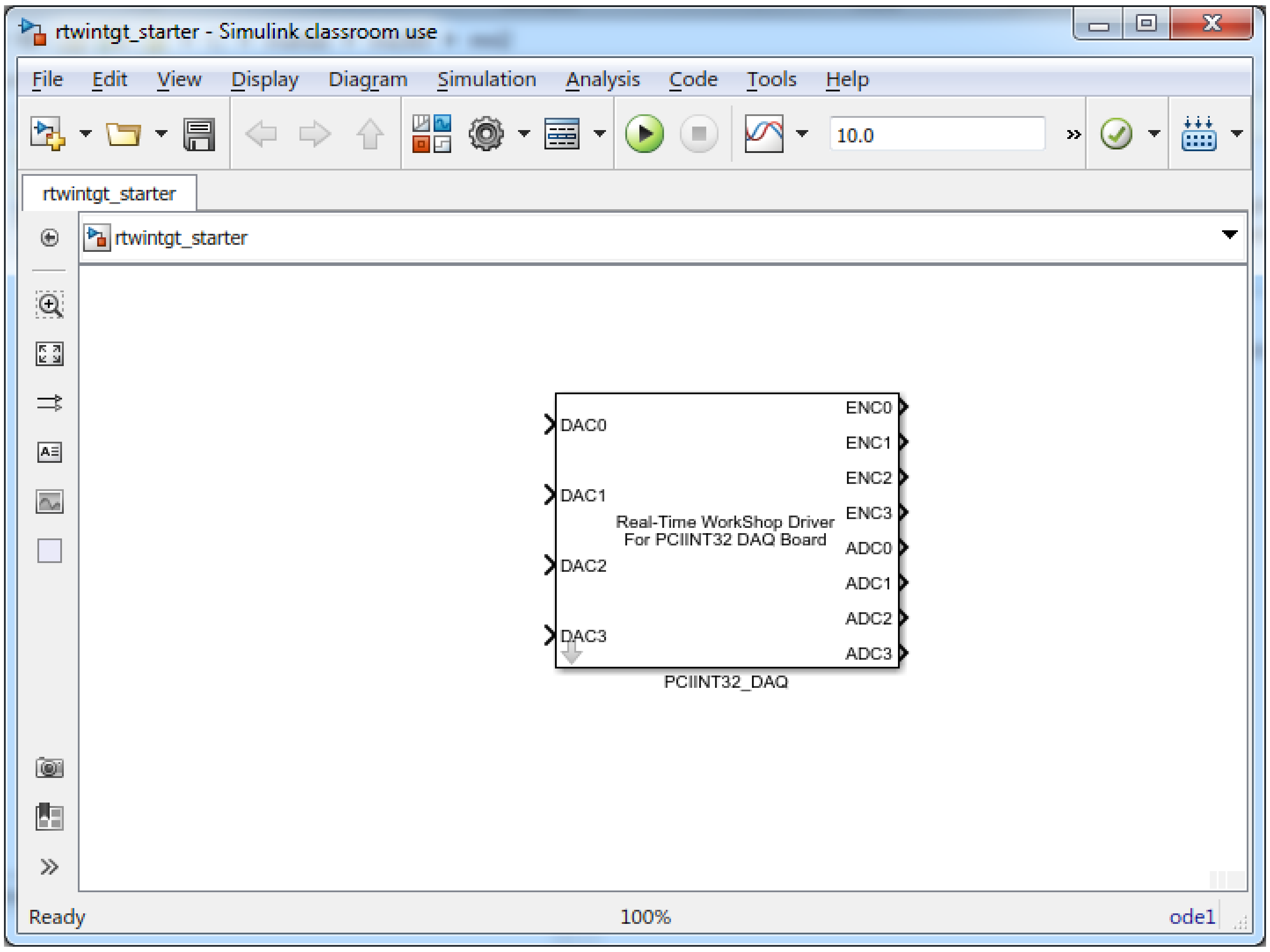
(e) Observe the response on the scope. Select the Magnifying-glass X-icon. Mouse-click the plot to zoom-in and measure the time required for the speed to reach steady state*. If you zoom-in too far or in the wrong region, right mouse-click the plot and choose "Zoom out"*.

(f) Re-run the simulation for the specified cases on the Data Sheet, and record the simulated speed, generator voltage, and response time.

(g) Sketch typical response curves with and without integral gain on the Data Sheet. Label appropriately.

Part 3. Test PI Speed Control on Actual Motor-Generator System

(a) Return to the main MATLAB window and enter "`rtwintgt_starter`" at the MATLAB prompt to open the rtwintgt_starter template window.


rtwintgt_starter - Simulink classroom use
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
rtwintgt_starter
rtwintgt_starter
DAC0
DAC1
DAC2
DAC3
ENC0
ENC1
ENC2
ENC3
ADC0
ADC1
ADC2
ADC3
Real-Time WorkShop Driver
For PCIINT32 DAQ Board
PCIINT32_DAQ
Ready
100%
ode1

(b) Copy your block diagram with PI speed control and paste it into the rtwintgt_starter template window.

(i) Choose "Select All" under the "Edit" menu of the window containing the speed control model.
(ii) Choose "Copy" under the "Edit" drop-down menu.
(iii) Mouse-click the rtwintgt_starter template window.
(iv) Choose "Paste" under the "Edit" drop-down menu of the rtwintgt_starter template window.

(c) We no longer want to simulate the motor, so replace the transfer function block with the PCIINT32_DAQ block. This is your interface to the ADC and DAC of the bench's computer. DAC0 is connected to the input of the motor's amplifier so connect the DAC0 input to the output of the saturation block. ADC0's is your feedback signal connected to the motor's tachometer. Connect it appropriately in your real-time model.

(d) Save a local copy of the real-time model in your personal directory, "C:\matlab\me360\your directory".

Build Executable Version of Model

(e) Now you will build and run this Simulink file. From inside your Simulink file select the menu item Code->C/C++ Code->Deploy to Hardware. This will build your Simulink file into a Real-Time Windows Target application. Once complete, the Play button will turn back to green. A C-code version of the model is first created, and then this code is compiled to produce the executable version.

Important *– The model will NOT build if you do not complete the previous step to save the model.*

Set-up the Oscilloscope

(f) Put the oscilloscope in roll mode and display only the generator output voltage. Set the vertical scale to 500 mV/div and the horizontal scale to 500 ms/div. Move the reference for the channel to the first grid division from the bottom of the screen.

Run Real-Time Workshop

(g) Depress the amplifier-inhibit push button and run the system by clicking the play button.

(h) Observe the response on the oscilloscope. Make use of the oscilloscope "Stop" button to freeze the display.

(i) Measure the steady-state generator voltage on the DMM.

(j) Change the PI control gains, press "Run" on the oscilloscope, and repeat this procedure for each case on the Data Sheet.

(k) Sketch response curves for cases 2, 3 and 5 on the Data Sheet. Label appropriately.

Examine Effects of Integral Windup

(l) What happens if you start your Simulation without enabling (holding down the pushbutton) the motor? What will happen to the integral of the error term in the controller while the motor is disabled?

To experiment with this first perform a normal run with your thumb on the inhibit button before you start your controller. Make sure you have Ki set to a value other than zero. You should see a response that has zero steady state error.

As a second run restart your controller, but this time do not hold the inhibit button. Wait a second or so and then press the inhibit button. Observe what happens. Sketch the response on the data sheet and answer the questions.

APPENDIX A: MOTOR-GENERATOR SYSTEM

For the sake of completeness, we repeat the basic information about the motor-generator system from the previous experiment. After this brief introduction, we examine the effect of external load omitted in our earlier discussion of this system.

The system analyzed in this experiment consists of a DC motor coupled to a DC generator (a motor run backwards) as shown in Fig. A.1. The motor converts a voltage into angular speed, and the generator, in turn, converts this angular speed back into a voltage. The generator acts as a tachometer, a device for measuring rotational speed. Together, the motor-generator combination gives us a system that takes a voltage as an input and produces a voltage as an output. The rotational speed of the shaft serves as an intermediate variable of important physical significance. This voltage-in / voltage-out configuration is ideally suited to the DAC / ADC computer-based instrument available at the laboratory station.

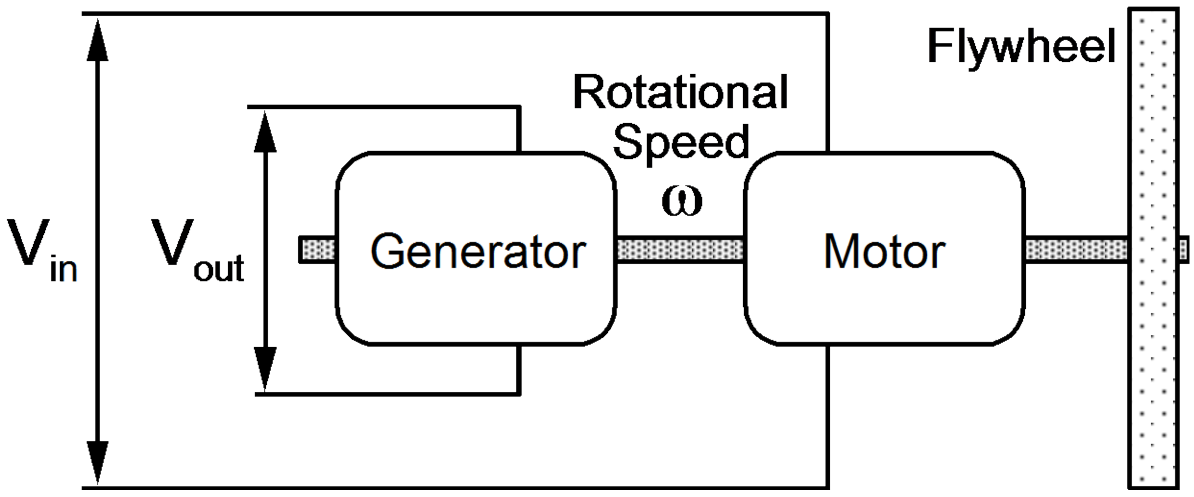


Figure A.1. Motor-generator system.

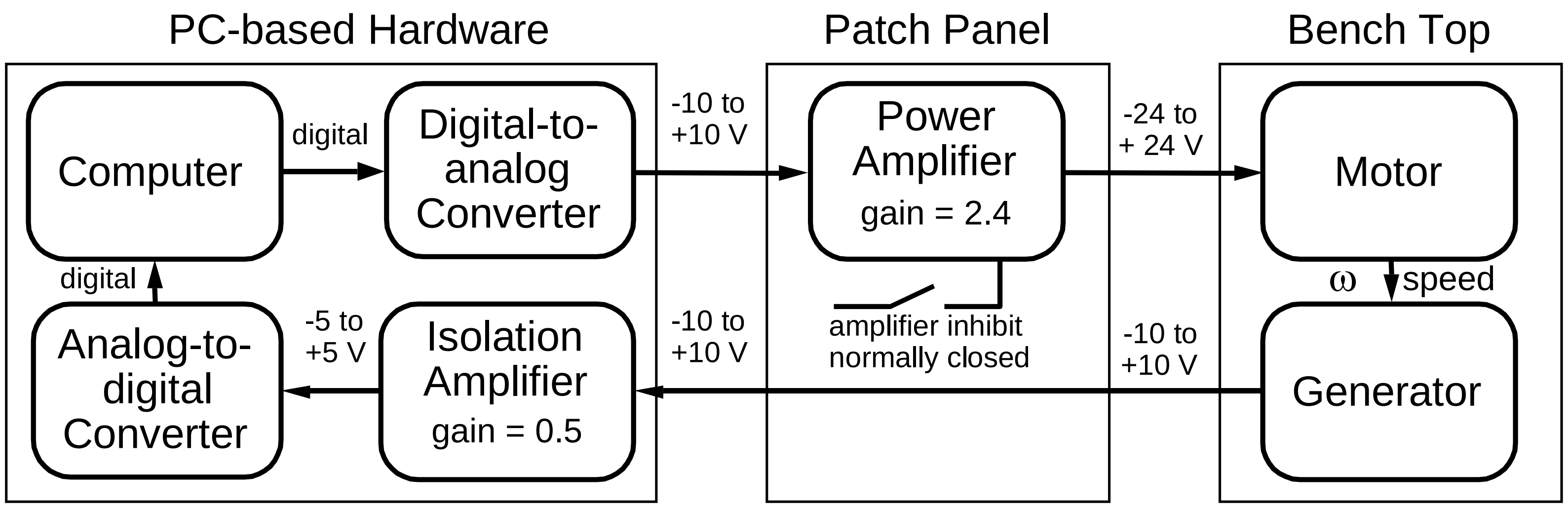

PC-based Hardware
Patch Panel
Bench Top
Computer
digital
Digital-to-analog Converter
-10 to +10 V
Power Amplifier gain = 2.4
-24 to + 24 V
Motor
ω speed
digital
Analog-to-digital Converter
-5 to +5 V
Isolation Amplifier gain = 0.5
-10 to +10 V
amplifier inhibit normally closed
-10 to +10 V
Generator

Figure A.2 Block diagram showing connections between instruments and motor-generator.

The motor and generator are connected to the computer as shown in Fig. A.2. Here, we see that the digital-to-analog converter (DAC) output voltage is routed to a power amplifier rather than directly to the motor. The power amplifier is needed because the current-producing capability of the DAC is insufficient to drive the motor. Although the amplifier does provide a voltage gain of about 2.4, its primary purpose is to meet the power demands of the motor. The need for an intermediate component to handle the power requirements of the system arises frequently in real-world applications and illustrates an important principle in interfacing low-power electronic equipment with high-power mechanical devices. Also, the amplifier-inhibit switch is used to isolate the motor from the power supply in standby mode to prevent inadvertent and potentially damaging inputs to the motor. The amplifier output is normally disabled; the switch must be pressed to allow the motor to run.

Figure A.3 is an expanded block diagram that allows the overall dynamic behavior of the motor to be more easily analyzed. Here, we add an important new consideration; namely, the torque of the external load T_{load}. In the previous experiment, the load on the motor was entirely due to its own "friction". Because this friction is directly proportional to motor speed, input voltage and steady-state speed are directly related. As input voltage increases, motor torque increases. Speed then increases until the friction torque again equals the motor torque. In a real application, the motor also drives an external load. As the load increases, the motor speed decreases. Speed control involves raising the input voltage to offset the effect of increased load. Conversely, as load decreases, speed control lowers the input voltage to again hold speed constant.

We must emphasize that here ω is the rotational speed of the motor and not the frequency used in spectral analysis. This dual use of ω is unfortunate but necessary.

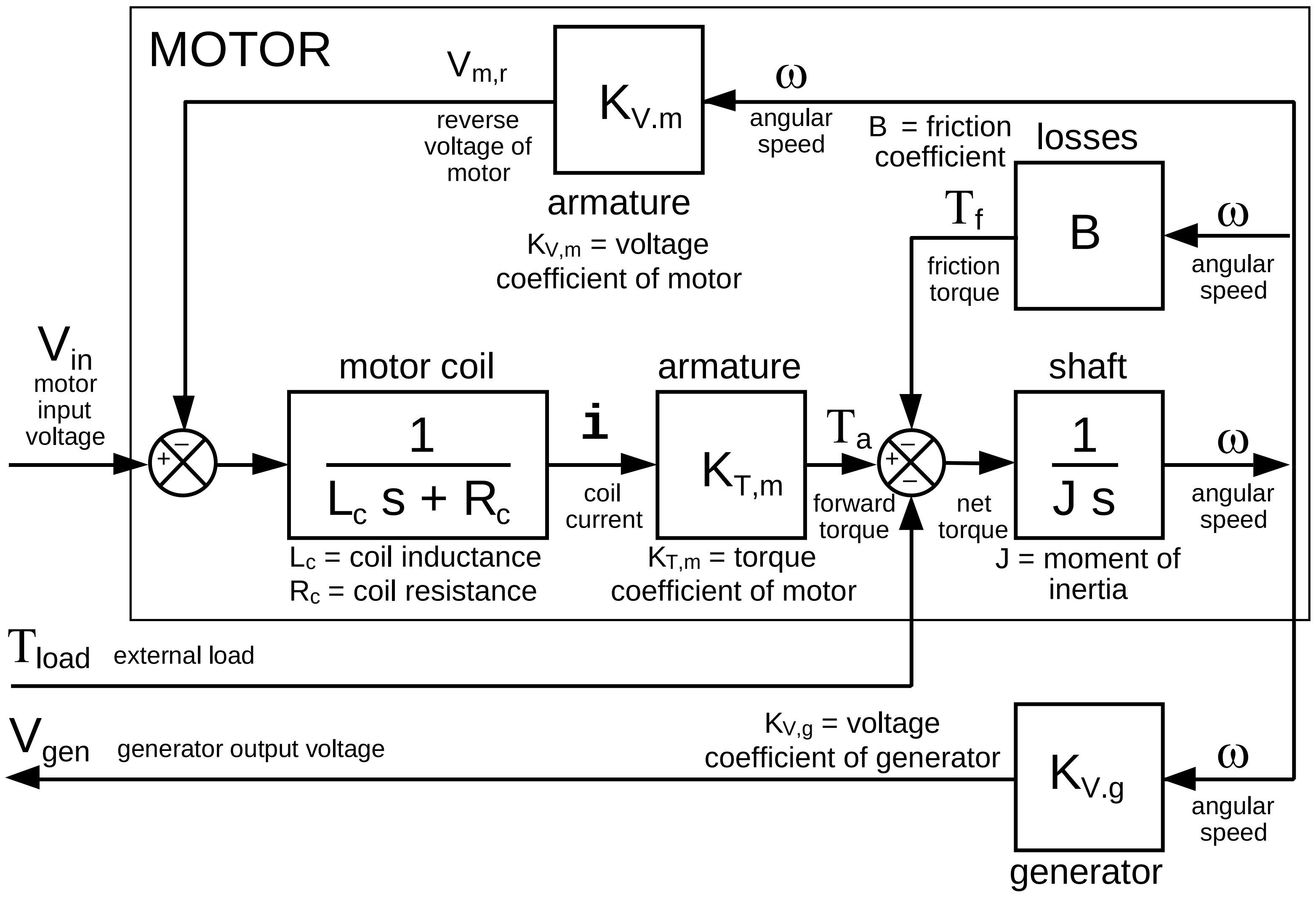


Figure A.3 Block diagram of motor-generator system showing main elements of dynamic response.

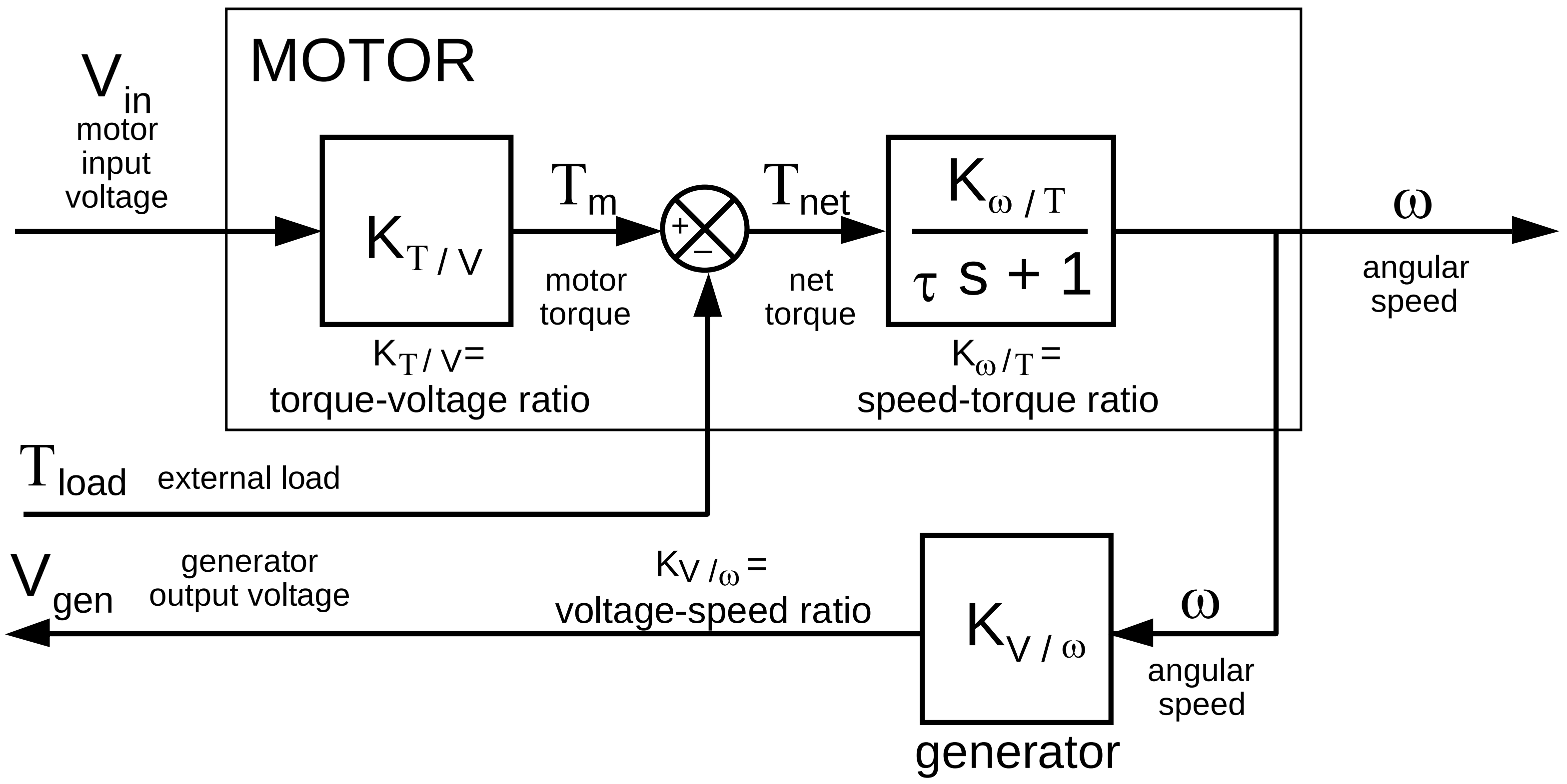


Figure A.4 Simplified block diagram of motor-generator system.

In the two previous experiments, we discovered that the motor-generator system can, in fact, be approximated as a first-order linear system with overall gain K and time constant τ. We found that K varies with input voltage because the system is only quasi-linear. Linearized about a steady-state operating condition of 4 V, the typical range for K is between 1.24 and 1.34. We also found experimentally that the time constant was between 50 and 60 ms depending on the particular motor.

The fact that the motor-generator system behaves as a first-order, linear system allows use to simplify our model as shown in Fig. A.4. The approach used here illustrates another important engineering principle; namely, *use the simplest model that captures the essential behavior of the system.* To break out the important physical variables of torque and speed and thus allow load to be properly included, we split the overall gain K into three factors: (a) the ratio of motor torque to input voltage $K_{T/V}$, (b) the ratio of motor speed to net torque $K_{\omega/T}$, and (c) the ratio of generator output voltage to speed $K_{V/\omega}$. To determine these parameters independently, we need torque and speed measurements as a function of input voltage. Lacking some of these data, we resort to the system specifications provided by the manufacturer and the experimentally determined value for K. The generator output voltage is listed as 3 V per 1000 rpm giving us that $K_{V/\omega}$ = 0.003 V / rpm. Also, an input voltage to the motor of 1 V produces a steady-state speed of 430 rpm at a torque of 0.0152 N-m with the generator and flywheel connected. Thus, $K_{T/V}$ = 0.0152 N-m/V and $K_{\omega/T}$ = (430 rpm) / (0.0152 N-m) = 28290 rpm / N-m. As a check of our calculations, we note that an input of 1 V to the motor produces a torque of 0.0152 N-m, a speed of 430 rpm = (28290 rpm/N-m) $\times$ (0.0152 N-m), and a generator output of 1.29 V = (0.003 V/rpm) $\times$ (430 rpm). The overall gain from input voltage to output voltage is thus 1.29 as desired. Table A.1 summarizes the parameters used our analysis.

Table A.1 Typical values of motor-generator system parameters.

Symbol	Description	Value	Units
K	Overall gain of motor-generator system	1.29	V / V
τ	Time constant of motor-generator system	0.055	S
$K_{\omega/V}$	Speed-voltage ratio of motor (no load)	430	rpm / V
$K_{V/\omega}$	Voltage-speed ratio of generator	0.003	V / rpm
$K_{T/V}$	Torque-voltage ratio of motor	0.0152	N-m / V
$K_{\omega/T}$	Speed-torque ratio of motor	28290	rpm / N-m

With this simplified view of the system, the transfer function relationships become

$$\omega(s) = \frac{K_{\omega/V}}{\tau s + 1} V_{in}(s) - \frac{K_{\omega/T}}{\tau s + 1} T_{load}(s)$$

and

$$V_{gen}(s) = K_{V/\omega}\, \omega(s)$$

where $V_{in}(s)$, $T_{load}(s)$, $\omega(s)$, and $V_{gen}(s)$ represent the transfer function form of the respective variables, i. e., either Laplace or Fourier Transforms as appropriate. The steady-state speed is obtained by setting s = 0 to yield

$$\omega_{ss} = K_{\omega/V}\, V_{in}(s) - K_{\omega/T}\, T_{load}(s)$$

Here, we clearly see that speed increases with increasing input voltage and decreases with increasing external load. We further observe that the system exhibits a first-order (exponential) response with time constant τ to changes in either input variable. Moreover, the generator serves as the speed sensor which we shall see shortly is needed for speed control. The generator has the undesirable characteristic that it also loads the system affecting both steady-state and transient behavior; however, the effect of the generator on motor dynamics is included in our analysis of motor-generator behavior.

APPENDIX B: MOTOR SPEED CONTROL

Figure 6 shows the relationship between the motor, the controller, and the speed sensor. The controller has two inputs and one output. The output is called the control output, and, in this case, it is simply the voltage supplied to the motor. One of the two required inputs is the speed sensor's output ω. The second input is the set point or desired motor speed. The set point can be specified in any of a wide variety of ways. For example, the set point can be specified using a simple knob like the temperature control on a conventional oven. Variations on this input option include slides, thumbwheels, and hold-and-release buttons. The set point can also be specified using (a) a keypad or keyboard with or without a corresponding display, (b) a touch screen, or (c) a stylus. The set point can also be specified remotely by supplying a certain analog voltage (say, between 0 and 10 V), or, more commonly, a current between 4 and 20 mA with 4 mA corresponding to a "zero" set point, and 20 mA corresponding to a "full scale" set point. Still more sophisticated options include digital input of the set point over a parallel or serial communications channel. The options used in practice are too numerous to list completely. In this experiment, we specify our set point by entering its numerical value from the computer keyboard.

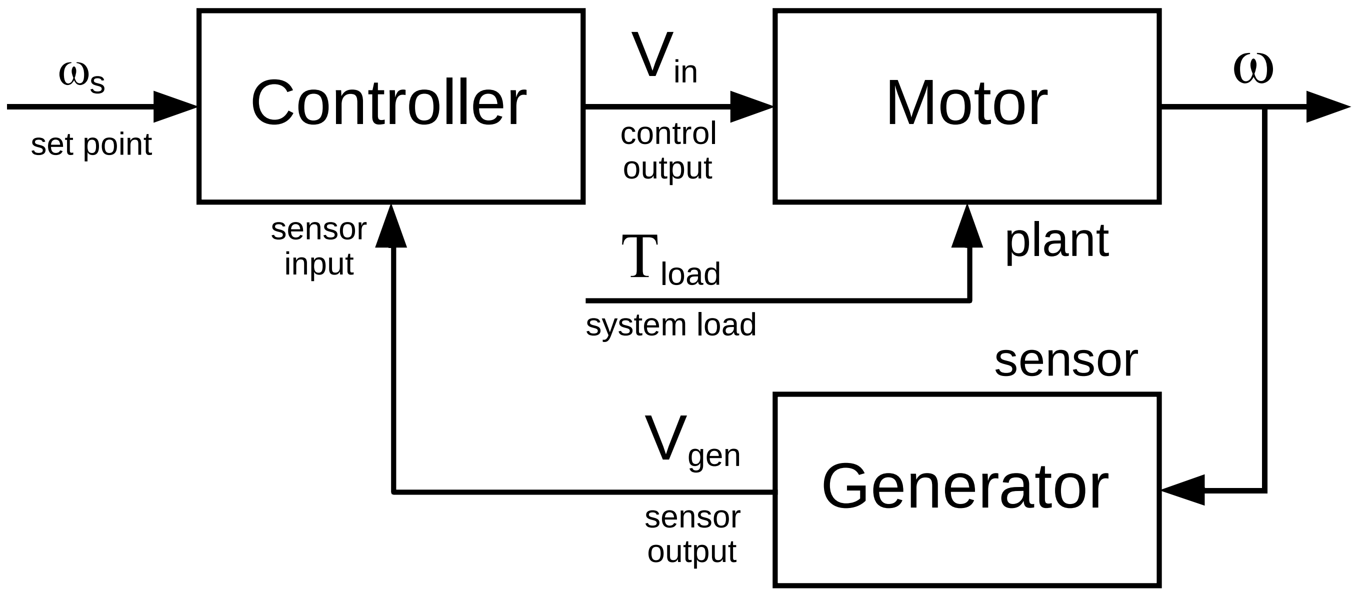


Figure 6. Flow of information between motor, generator and controller.

The controller may have additional inputs and outputs as well as the basic ones noted above. Additional inputs may be either analog or digital and may allow the operating parameters of the controller to be set remotely. Additional outputs may provide for remotely reading the system state, the set point, or both. Various types of displays as well as electrical, pneumatic, or hydraulic signals may be used. Again, the range of options is extensive.

Now let us look at what goes on inside the controller. One possibility is to hire someone and assign them the job of keeping the speed constant. Their instructions are "Watch the speed. If the speed starts falling, increase the voltage. If the speed starts rising, decrease the voltage." This approach is similar to that used by a driver in maintaining constant vehicle speed. If the vehicle encounters an incline and the speed begins to fall, the throttle is opened and more torque is supplied by the engine. Likewise, if the speed begins to rise, power is reduced. An human controller may sound silly, but operators are an important part of many control systems. However, because this particular task does not require the subjective judgment that a human operator brings to a control problem, we can program a microprocessor to compare the measured speed with the desired speed. If the speed is too low, the input voltage is raised. Conversely, if the speed is too high, the input voltage is reduced. The microprocessor does this job much more quickly, more reliably, and at a much lower cost than the human operator. Moreover, the microprocessor can be programmed to mimic the reasoning used by an operator. For example, the microprocessor can raise or lower the input voltage by a small amount when the speed difference is small and by a large amount when the speed difference is large. Again, this approach may sound silly at first, but intuitive-type controllers of this sort can be easily implemented on relatively low-cost microprocessors, and they work very well for a surprisingly large number of applications.

In this experiment, we investigate a very useful general purpose control method known as Proportional-Integral-Differential or PID control. One key feature of PID control is that the controller has three operating parameters (proportional gain K_p, integral gain K_I, and derivative gain K_D) that can be used to tune the behavior of the controller to meet certain control objectives. These parameters may be given in other terms (for example, the "reset time" $T_I = K_p / K_I$ may be used instead of the integral gain K_I), but the functionality is essentially the same for all PID controllers. Such controllers are widely used to maintain a system in a certain operating state. Here, we use PI control to maintain a constant motor speed. Similar applications of PID control involve (a) holding tank temperature fixed as ambient temperature changes or (b) keeping a boiler operating at a steady exit condition as load changes.

Figure 7 is an exploded view of the motor-generator-controller system for the case of PID control. The block diagram can be simplified to yield the transfer function results given in Table 2 for proportional (P) control, proportional-integral (PI) control, and proportional-integral-differential (PID) control.

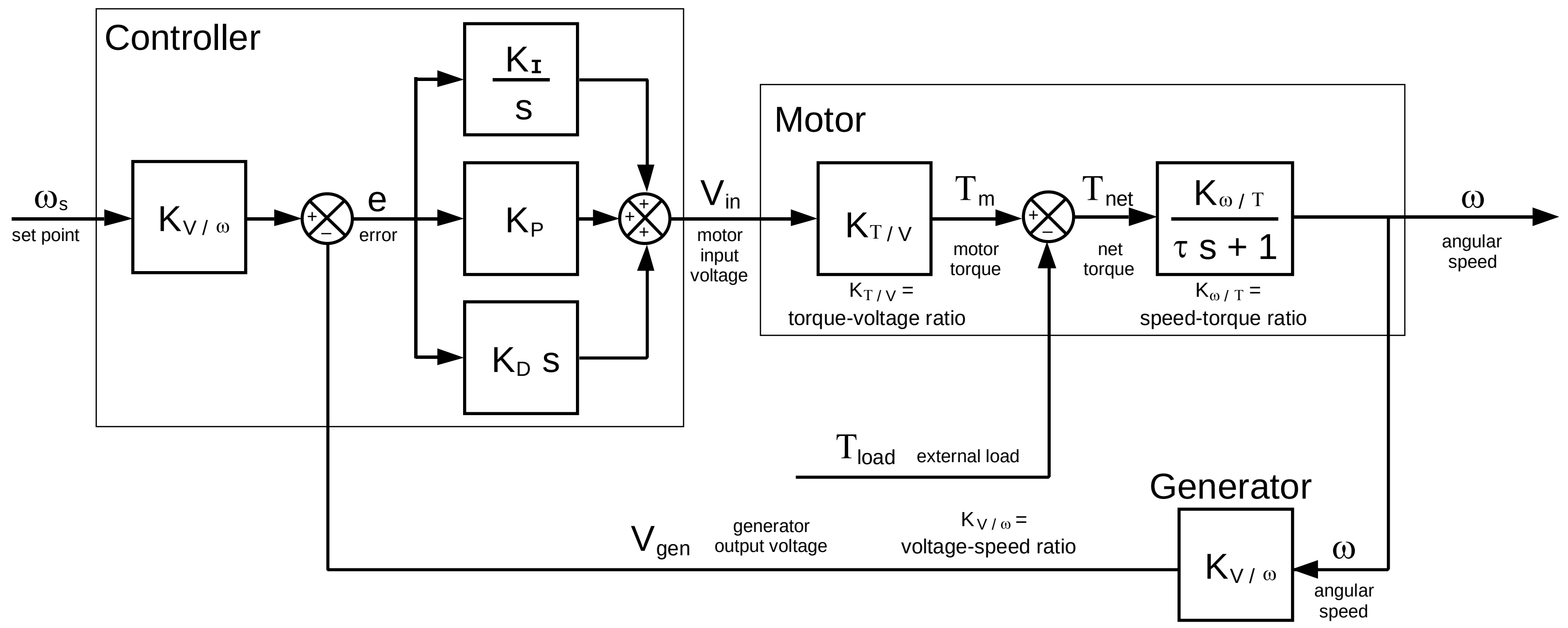


Figure 7. Motor-generator system with PID controller.

In nearly all control applications, we begin with the error between the set point and the current value of the state variable. For example, if the desired motor speed is 1000 rpm and the actual operating speed is 900 rpm, then the error is 1000 – 900 = 100 rpm. In this case, we know that the input voltage to the motor (i. e., the control output) must be increased. Proportional control provides a control output directly proportional to the error. Thus, proportional control provides twice the control output when the error is 200 rpm (800 rpm vs. a set point of 1000 rpm, for example) as when the error is 100 rpm. The ratio of the control output to the error is the proportional gain. Integral control involves adding a control output proportional to the integral of the error, the constant of proportionality being the integral gain. Similarly, derivative gain involves adding a control output proportional to the derivative of the error, the constant of proportionality being the derivative gain. In evaluating the performance of various control options, we must consider (a) the steady-state error, (b) the load sensitivity, and (c) the response time to changes in either set point or load. We now examine each of the three control methods (P, PI and PID) in terms of these performance factors based on the results presented in Table 2.

Proportional Control

1. Unless load is directly proportional to speed as is the case for many "viscous-type" loads, motor speed is sensitive to load even with proportional control in place. Increasing the proportional gain K_p has the positive effect of reducing load sensitivity. Thus, the largest proportional gain possible within the other constraints of the system is desirable if you are looking for the fastest response out of the motor.

2. Even in the absence of a load, proportional-only control produces a steady-state error. Once again, making the proportional gain larger has the positive effect of reducing the steady-state error.

Proportional-Integral Control

1. Motor speed exhibits a second-order response to changes in both set point and load. The typical response is characterized by two real time constants, and thus the sum of two exponentials. The response may be described as a rapid exponential decay to a "steady-state error" as in proportional control followed by a slower decay of the "steady-state error" to zero. Thus, it is useful to think of integral control as "adding" steady-state error removal to the basic proportional control response. Increasing K_p reduces the steady-state error and the initial response time. Increasing K_I increases the rate at which the "steady-state" error dies away. Ideally, both gains are infinite but numerical stability places an upper limit on these gains when digital control is used. Moreover, bandwidth and output voltage limitations further restrict the gain values. Optimal control is often determined empirically.

2. With proportional-integral control, load sensitivity is zero.

3. With proportional-integral control, steady-state error is zero.

Proportional-Integral-Differential Control

1. Derivative control does very little to improve or even change system response characteristics in this case and thus is considered only briefly in the experimental work. Derivative control introduces an additional zero into the closed-loop transfer function. This zero can be used to cancel (or at least partially cancel) the effects of a long time constant in the open-loop transfer function of the plant (i. e., the motor). In this case, the motor has only one very short time constant of 55 ms, so derivative control has little beneficial effect. Moreover, derivative control has the negative side-effect of increasing sensitivity to noise in the output signal from the generator, and, furthermore, our brute-force digital implementation of derivative control with numerical differentiation introduces additional undesirable numerical problems. These negative effects can be partially overcome by "filtering" the derivative control signal which, in effect, shifts the zero location, but the benefit of adding derivative control in this case is still marginal at best.

2. With proportional-integral-derivative control, load sensitivity is zero.

3. With proportional-integral-derivative control, steady-state error is zero.

APPENDIX C: SIMULINK

In this experiment, we simulate motor-generator behavior first without control and then with the three basic types of control (P, PI and PID) discussed above. The software package used for this purpose is SIMULINK, a graphical interface for MATLAB that allows one to build systems graphically by constructing block diagrams comprising a variety of different functional units. The interface is intuitive and therefore needs only a brief overview as it specifically relates to this experiment. Interested students are referred to the User's Guide for a detailed discussion of the SIMULINK package.

Once defined using the SIMULINK graphical interface, system behavior can be simulated in the time domain through numerical integration of the corresponding system equations in MATLAB. Because numerical integration is used, the system need not be linear.

SIMULINK is started by entering "`simulink`" at the MATLAB command prompt (followed, of course, by a "return"). A library browser appears as shown in Fig. 8. A system is built by opening the block libraries and dragging the appropriate functional blocks to the work window. Table 3 and 4 summarize the SIMULINK block libraries and basic operations, respectively.

Real-Time Windows Target

Real-Time Windows Target provides real-time digital signal processing and control under MSWindows® and the SIMULINK-MATLAB umbrella. Real-Time Windows Target allows us to go one step further and implement speed control on the actual motor-generator system using the PC-based instrumentation. The basic steps in using the Real-Time Windows Target package are:

1. Access the program template by entering "`rtwintgt_starter`" at the MATLAB prompt.

2. Create a new program file derived from the rtwintgt_starter template using a unique file name. The file should be saved in the "C:\matlab\me360\your directory" folder using the "Save As" dialog box under the "File" drop-down menu. For the purpose of this explanation, we refer to this program as "RTmodel" where "RT" designates a real-time program.

3. Copy the simulated system to the RTmodel window from the SIMULINK window where it was created.

4. Replace the simulated motor with data acquisition block. The input of the motor should be replaced with the "DAC0" port, and the output from the generator should be replaced by the "ADC0" port.

5. Save the modified RTmodel program.

6. Select "Build Model" under the "Tools->Real-Time Workshop" drop-down menu. Wait for the program to compile.

7. Depress the push button connected to the amp-inhibit socket on the patch panel, and run the Real-Time Windows Target program by first selecting the menu item "Simulation->Connect to Target" and then selecting the menu item "Simulation->Start Real-Time Code".

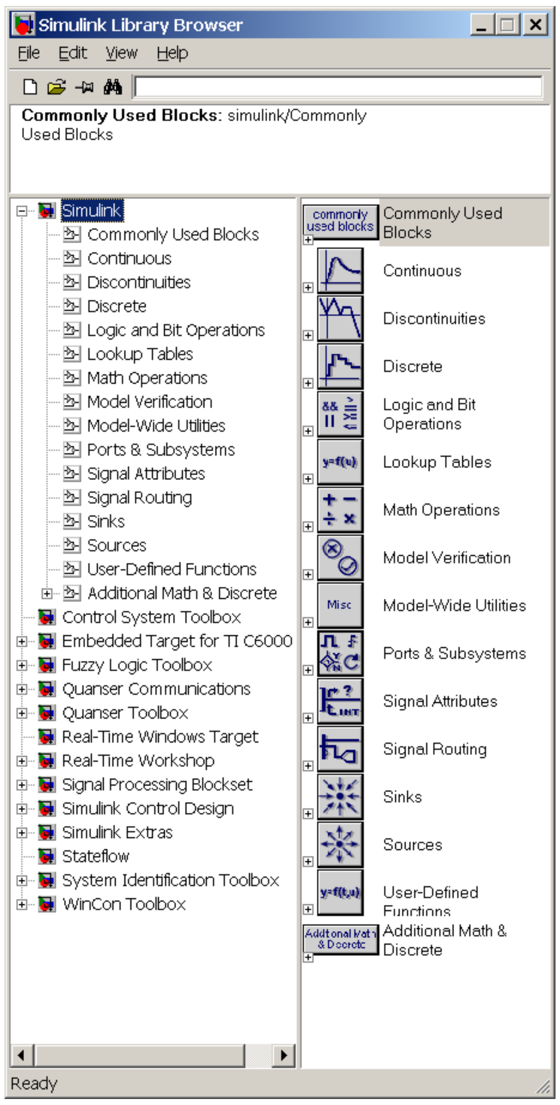
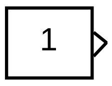
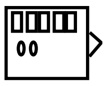


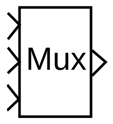
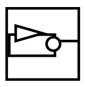
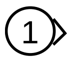
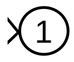
Figure 8. SIMULINK 6.2 block libraries.

Table 3.Selected Blocks from SIMULINK Block Libraries.

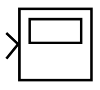
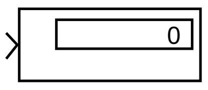
<u>SIMULINK Sources</u> (generate signals)

Block	Description
 Constant	The "constant" block generates a constant input value. Properties include the value of the constant. This block has one output and no inputs.
 Signal Generator	The "signal generator" block generates a sine wave, square wave or sawtooth waveform. Properties include signal frequency and amplitude. This block has one output and no inputs.

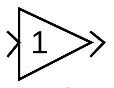
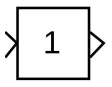
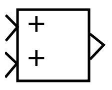
<u>SIMULINK Signals & Systems</u> (connect blocks and build subsystems)

Block	Description
 Mux	The "multiplexer" block combines several scalar signals into a single vector signal. For example, the "mux" block combines several signals for simultaneous display with the scope block.
 Subsystem	To create a subsystem, drag the subsystem block into the work window then open the subsystem block by double clicking its icon. Build the subsystem in the usual manner except use inport and outport blocks for the subsystem inputs and outputs, respectively; close the subsystem window when finished; alternatively, a group of blocks may be combined into a subsystem by selecting the blocks with a bounding box and choosing "Create Subsystem" under the "<u>E</u>dit" drop-down menu
 In1	The "Inport" block creates a subsystem input; the "Inport" block also allows creation of an external input in the work window.
 Out1	The "Outport" block creates a subsystem output; the "Outport" block also allows creation of an external output in the work window

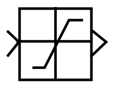
<u>SIMULINK Sinks</u> (display and save signals)

Block	Description
 Scope	The "scope" block displays a waveform in a manner similar to an oscilloscope. Properties include vertical scale and horizontal time base settings and the number of points displayed. The scope display has a tool bar with buttons as follows: (a) "Zoom" enlarge a region of the display (magnifying glass icon), (b) "Zoom X" expand horizontal axis only (magnifying glass with X inside icon), (c) "Zoom Y" expand vertical axis only (magnifying glass with Y inside icon), (d) "Autoscale" scale axes for full screen display (binoculars icon) (e) "Save axis" save axis configuration for later use (scope overlaying plot icon) (f) "Properties" open properties dialog box (miniature properties dialog box icon) (g) "Print" prints a copy of the scope output (MSWindows® printer icon) This block has one input and no outputs. A "mux" connection block can be used to display multiple channels simultaneously.
 Display	The "display" block displays the current value of a signal in a manner similar to a DMM. This block has one input and no outputs.

<u>SIMULINK Math</u> (perform math operations on signals)

Block	Description
 Gain	The "gain" block multiplies the input by a constant. A "mux" connection block can be used create a vector input to a gain block. Properties include the gain which may be a scalar or a vector of the same length as the input. The block has one input and one output.
 Slider Gain	The "slider gain" block is a gain block in which the gain is set using a slider control displayed in a separate window. This control can be used with the "constant" source block to provide an adjustable input. Properties include (a) slider low value, (b) slider high value, and (c) slider current value which may be typed in as well as varied using the slider control. The slider can be changed during a simulation. This block has one input and one output.
 Sum	The "sum" block computes the signed algebraic sum of its inputs. Properties include the number of inputs and the sign applied to each input. An integer value for the input property specifies that number of summation inputs; alternatively, a string of "+" and "–" symbols assigns specific signs to each input; for example, the string "+––+" sets up a block that computes + $V_1 - V_2 - V_3 + V_4$. This block has a variable number of inputs and one output.

<u>SIMULINK Nonlinear</u> (nonlinear systems and math functions)

Block	Description
 Saturation	The "saturation" block sets upper and lower limits for a signal. The properties of the block are the limits. The block has one output and one input. The output equals the input unless the limits are exceeded.

<u>SIMULINK Continuous</u> (continuous systems and math functions)

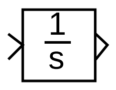
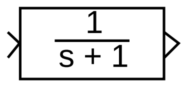
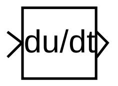
Block	Description
 Integrator	An "integrator" block computes the time integral of the input signal. Properties include (a) the source and value of the initial condition, (b) upper and lower saturation limits, and (c) the numerical integration tolerance. This block has one input for the integrand, an optional second input for the initial condition, and one output for the integral.
 Transfer Fcn	The "transfer function" block implements an ordinary differential equation describing the dynamic characteristics of a functional unit in transfer function form. Alternative methods of specifying dynamic response include the "zero-pole" block and the "state-space block". Properties include the coefficients of the numerator and denominator in decreasing powers of "s" specified using MATLAB vector notation. This block has one input and one output.
 Derivative	The "derivative" block outputs the time rate of change of the input. This block has one input and one output. Implemented is by numerical differentiation which is the crudest form of derivative and often problematic for control system use.

Table 4.Basic SIMULINK Operations

Task	Procedure
add block to work window	open one of the block libraries by double clicking its icon then drag the appropriate block to the work window
duplicate block within window	right-click on block and drag duplicate to new location
select blocks for modification or deletion	(i) click on block icon in the work window, OR (ii) click first block then shift-click each additional block, OR (iii) form a bounding box containing the blocks by clicking in one corner of the intended box and then dragging to the other corner
delete block from work window	select block(s) to be deleted then press the delete or backspace key
copy block(s) from one work window to another	select block(s) then standard copy, cut and paste operations
modify block properties	double click block then use properties dialog box OR select block and use Format drop-down menu options
flip location of inputs and outputs	select block, then select "Flip Block" under "Format" drop-down menu OR use ctrl-f keystroke
change block name	click name then edit text
modify block size	select block(s), then drag block handle
modify block format	select block(s), then use "Format" drop-down menu and associated dialog boxes to make changes
connect blocks with signal line	click output (input) of first block and drag to input (output) of second block; multi-segment lines can be created by releasing mouse button at end of each segment; a complete connection is indicated by a large, solid arrow head; an incomplete connection is designated by a small, skeleton-style arrow head
create connection branch	right-click on signal line then drag to branch destination
label signal line	select line and type text label
move blocks and lines	click and drag to new location
create subsystem	select blocks with bounding box then use "Create Subsystem" under "Edit" drop-down menu OR use "Subsystem" block in "Signals & Sys."