QUANSER
INNOVATE. EDUCATE.

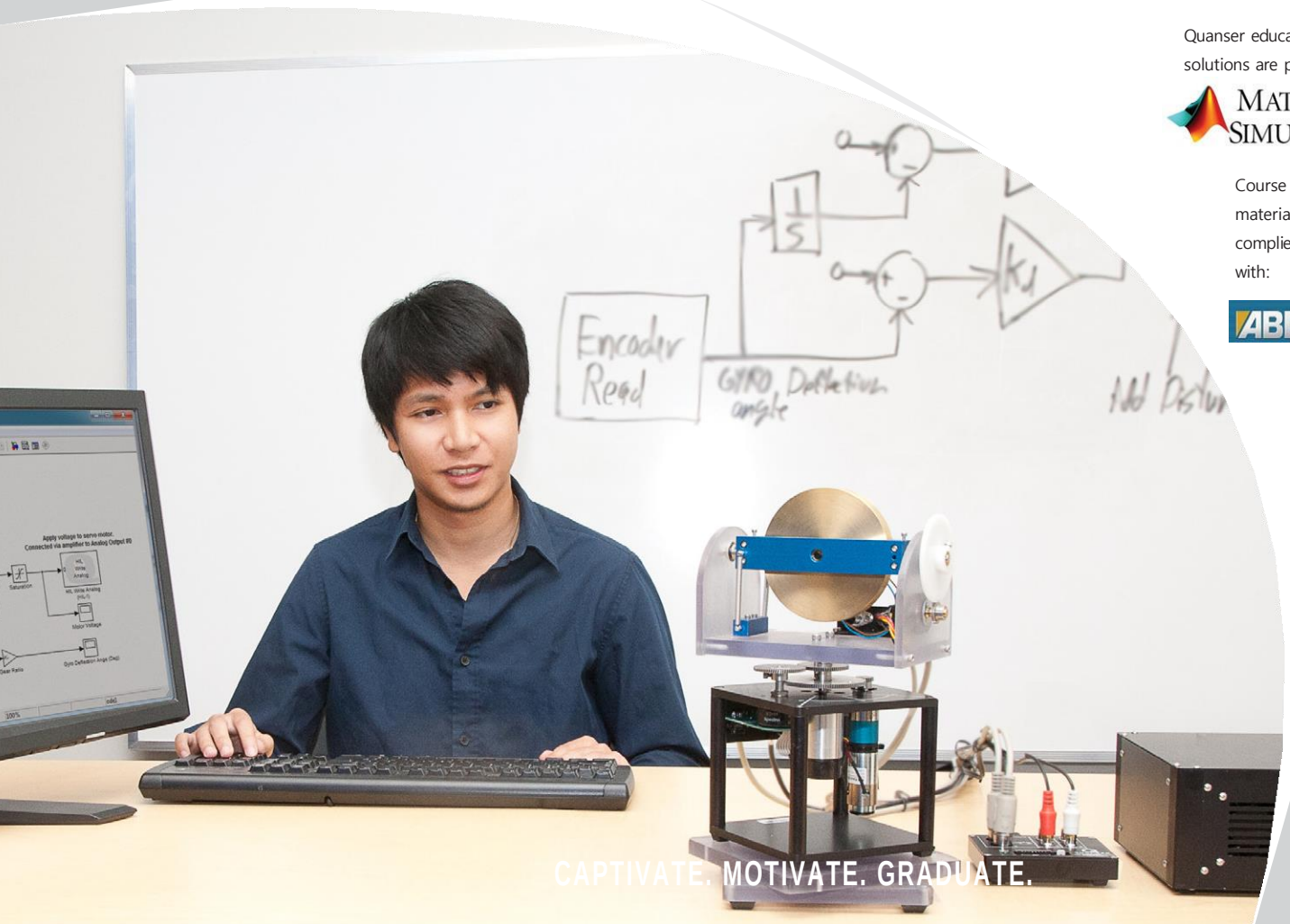
Student Workbook
Experiment 03

Vehicle Speed and Steering Control Experiment

Standardized for ABET* Evaluation Criteria

Developed by:
Jacob Apkarian, Ph.D., Quanser

Modified by:
Hever Moncayo, Ph.D., ERAU
Fall, 2021



Quanser educational solutions are powered by:

MATLAB SIMULINK

Course material complies with:

ABET

CAPTIVATE. MOTIVATE. GRADUATE.

*ABET Inc., is the recognized accreditor for college and university programs in applied science, computing, engineering, and technology; and has provided leadership and quality assurance in higher education for over 75 years.

Quanser Inc.
119 Spy Court
Markham,
Ontario L3R 5H6
Canada
info@quanser.com
Phone: 1-905-940-
3575
Fax: 1-905-940-3576

Printed in Markham, Ontario.

For more information on the solutions Quanser Inc. offers, please visit the web site at:
http://www.quanser.com

Experiment 03: Vehicle Speed and Steering Control

The objective of this laboratory is to develop a feedback system to regulate the speed and direction of a simulated vehicle. The developed controller will compensate also for disturbances to the speed of the vehicle such as the slope of a road.

Topics Covered

- Design of a proportional-velocity (PI) controller for regulation of the speed control of a simulated vehicle.
- Design of a proportional-derivative (PD) controller to control the steering angle of the simulated vehicle.
- Analyze the performance of the controllers by changing the magnitude of the disturbance (slope of the road) to the speed of the simulated vehicle.
- Implementation of the controllers on the Quanser SRV02 device to evaluate their performance.

Prerequisites
In order to successfully carry out this laboratory, the user should be familiar with the following:

- Data acquisition device (e.g. Q2-USB), the power amplifier (e.g. VoltPAQ-X1), and the main components of the SRV02 (e.g. actuator, sensors), as described in References [2], [4], and [6], respectively.
- Wiring and operating procedure of the SRV02 plant with the amplifier and data-acquisition (DAQ) device, as discussed in Reference [6].
- Transfer function fundamentals, e.g. obtaining a transfer function from a differential equation.
- Laboratory described in Appendix A to get familiar with using QUARC® with the SRV02.

1. Background

The Quanser HIL Driving Simulator (QDS) is a modular and expandable Simulink model of a car driving on a closed track. The model is intended as a platform for the development, implementation and evaluation of a variety of control systems. The QDS consists of a variety of components that are integrated together to create a representation of a vehicle being driven on a track. One possible configuration is shown in Figure 1. The model utilizes the QUARC environment to facilitate real-time simulation and hardware-in-the-loop interfacing (HIL). The Quanser Visualization block is also used to create an immersive visual environment for testing and evaluating controllers. Students are expected to observe and think critically about the effects of system parameters on not just the discrete plant, but the overall system.

Some examples of the real-world control problems that can be addressed using the QDS include parking assist systems, radar guided cruise control, active suspension, traction control, and autonomous navigation.

Figure 1:Quanser Driving Simulator

1.1 Speed Control

1.1.1 Electronic Throttle Control

Electronic throttle control (ETC), traction control and cruise control have become standard features on modern cars. More recently, with the advent of radar-guided cruise control, and pre-crash systems electronic vehicle control units have begun to play an increasingly critical role in the real-time control of vehicle speed. Though the implementation of these systems can be complex, the essential system can be viewed as a closed-loop speed controller that sends throttle commands to the engine, shown in Figure 2.

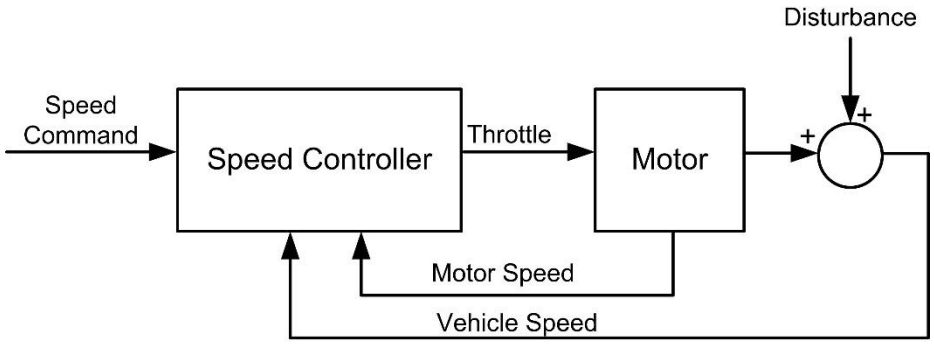


Figure 2:ETC controller

1.1.2 DC Motor Speed Control

For this laboratory, you will use the QDS in conjunction with a Rotary Servo Rotary Servo to develop a Proportional-Integral (PI) speed controller to regulate the speed of the simulated vehicle. This exercise is analogous to developing an ETC for an electric vehicle, shown in Figure 3. In this case, the disturbance to the speed of the vehicle is the slope of the road, which is converted into a voltage and added to the control command. The controller that you will design takes the actual speed of the Rotary Servo motor as the derivative of the encoder measurements, and the commanded speed from the internal driver controller in the QDS. The PI controller then outputs the motor voltage, V_m, which is added to the slope of the road and sent to the Rotary Servo. The slope of the road is provided by the sensors inside the QDS.

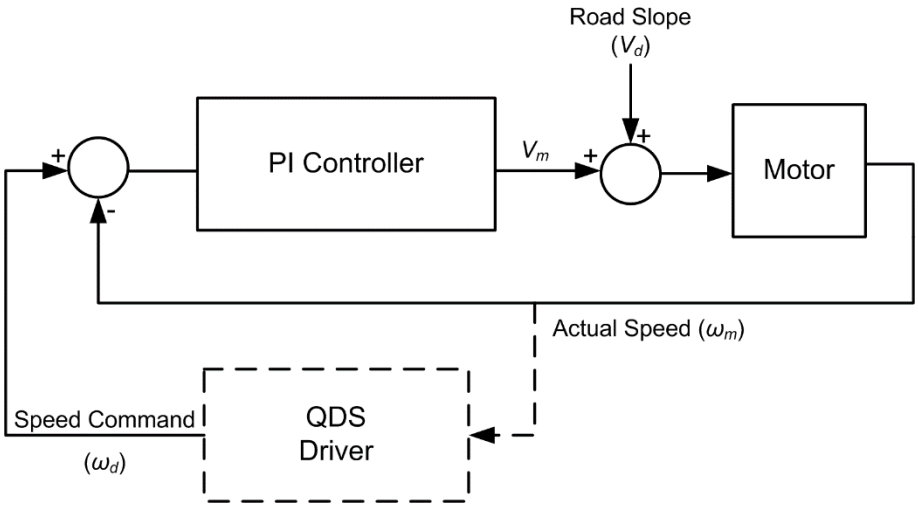


Figure 3:QDS speed controller

1.1.3 PI Control

The PI controller is one of the most common control algorithms. It combines the error reduction of proportional control, with the offset elimination of integral control. The proportional term tracks the instantaneous error, while the integral term controls the offset by tracking the total error over time. The longer there is error in the system, the more the integral term tries to compensate. Though the lack of a derivative term can cause overshoot and a longer settling time, for systems with simple dynamics the algorithm can provide near-optimal performance with no steady-state error.

In the time-domain, the linear behavior of a PI controller can be described by:

$$u(t) = k_p\big(r(t) - y(t)\big) + k_i \int_0^t \big(r(t) - y(t)\big)dt \tag{1}$$

where $u(t)$ is the control signal, $r(t)$ is the reference, and $y(t)$ is the measured process output.

1.2 Steering Control

1.2.1 Assisted Steering

Power steering systems have been used for years as a driver aid on production vehicles to make steering easier and safer. More recently, manufacturers have begun to increasingly intervene in the steering process to vary the sensitivity of the steering as a function of the speed of the vehicle. Though true steer-by-wire systems are not available in production automobiles for safety reasons, they are sometimes used in heavy construction or for parking assist systems such as the Toyota Intelligent Parking Assist System.

1.2.2 DC Motor Position Control

For this laboratory, you will use the QDS in conjunction with a Rotary Servo to develop a PD position controller to regulate the steering angle of the simulated vehicle. The controller that you will design takes the steering angle command from the internal driving controller, and the actual steering angle from the Rotary Servo encoder signal. The PD controller then outputs the appropriate motor voltage, V_m, to actuate the Rotary Servo motor. A block diagram of the overall system is shown in Figure 4.

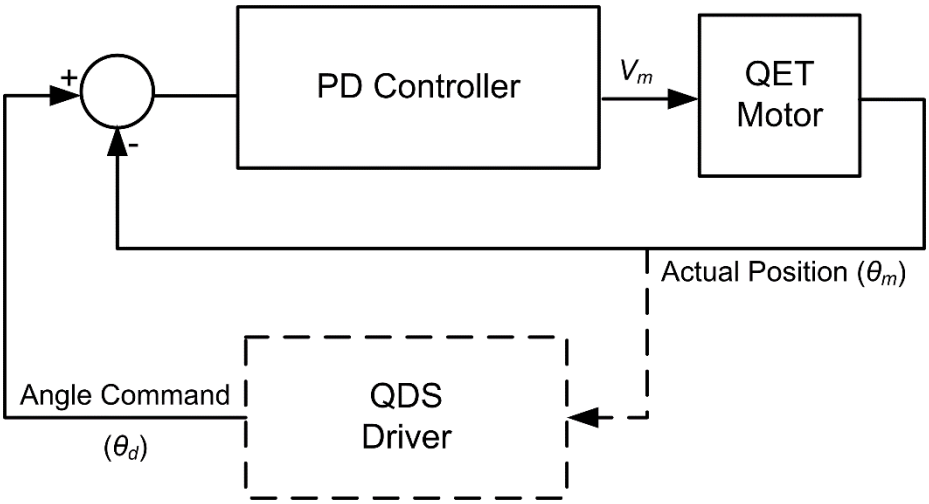


Figure 4:QDS steering angle controller

1.2.3 PD Control

The PD controller is one of the most common control algorithms. For position control, it combines the error reduction of proportional control, with the overshoot elimination of derivative control. The proportional control term tracks the instantaneous error, while the derivative term predicts the response of the system based on the slope of the response. Though the derivative control term is able to reduce overshoot and improve the settling time, the system can be susceptible to steady-state error.

The linear behavior of a PD controller in the time-domain can be described by:

$$u(t) = k_p\big(r(t) - y(t)\big) + k_d\left(\frac{d}{dt}r(t) - \frac{d}{dt}y(t)\right) \quad (2)$$

where $u(t)$ is the control signal, $r(t)$ is the reference, and $y(t)$ is the measured process output.

2. Questions

Before you start the lab experiments given in Section 3, you should study the background materials provided in Section 1 and work through the questions in this Section. These questions will help to prepare for the quiz before starting the experiment and for completing the report of this lab experiment.

1. From Equation 1, determine the PI controller transfer function in the Laplace domain.
2. Determine the steady-state error of the closed-loop system from Equation 1.
3. From Equation 2, determine the PD controller transfer function in the Laplace domain.
4. Why would a PD controller appropriate for the control steering task?

3. Lab Experiments

The objective of this laboratory is to develop afeedback system to regulate the speed and direction of a simulated vehicle. The developed controller will compensate also for disturbances to the speed of the vehicle such as the slope of a road. You will conduct two experiments:

1. Speed cruise control using a PI controller.
2. Steering angle control using a PD controller.

Below is the list of files that will be needed for the experiment:

File Name	Description
speed_model_qds.mdl	The main Simulink@ file that sets the SRV02 motor and sensor parameters and contains the main model for speed cruise control.
steer_model_qds.mdl	The main Simulink@ file that sets the SRV02 motor and sensor parameters and contains the main model for steering angle control.

Figure 5 shows a list of equipment that will be needed for the experiment:

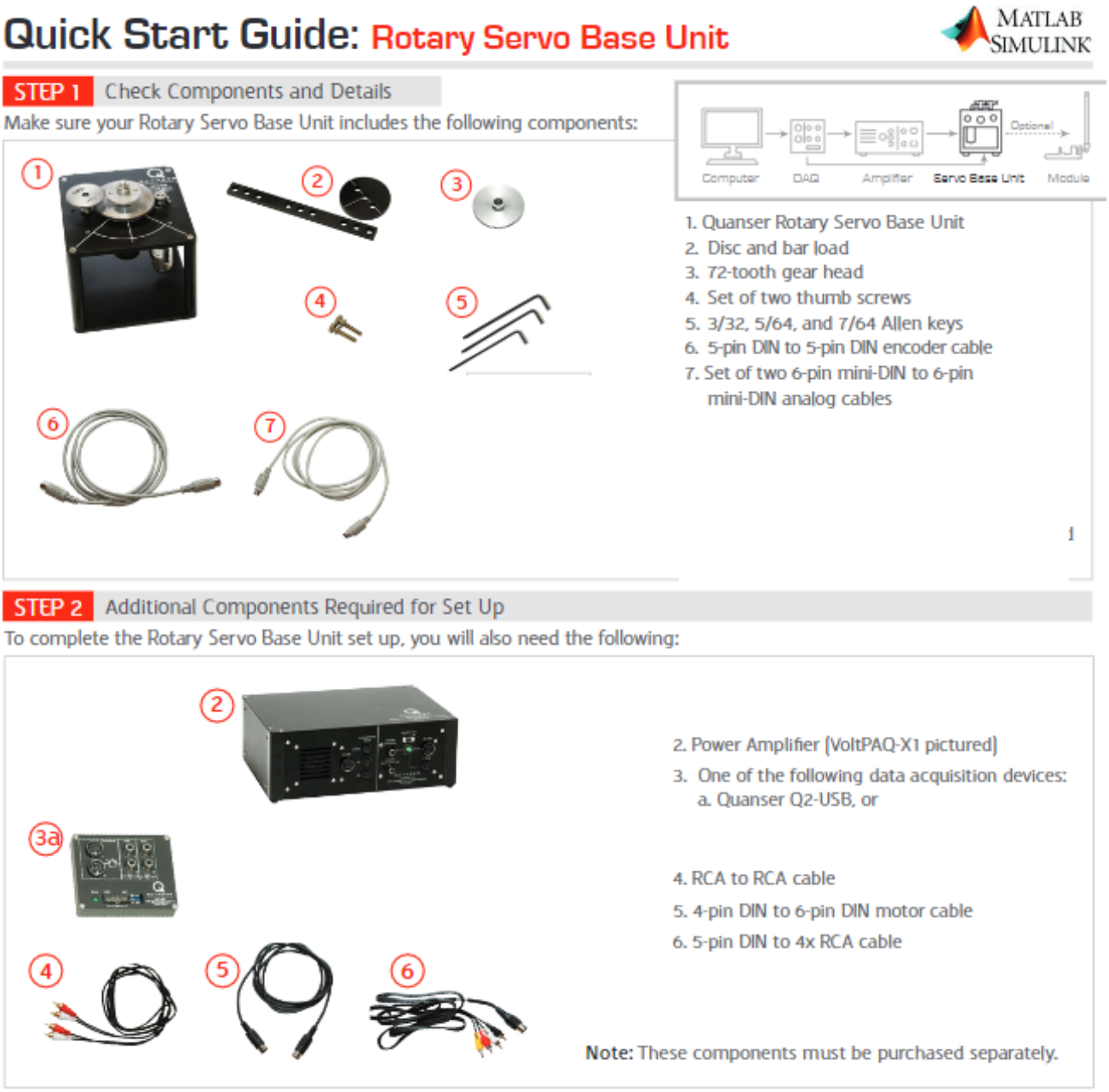


Figure 5: Equipment list

Figures 6 and 7 show how to assemble the equipment. This picture is shown in greater detail in the Quick Start Guide.

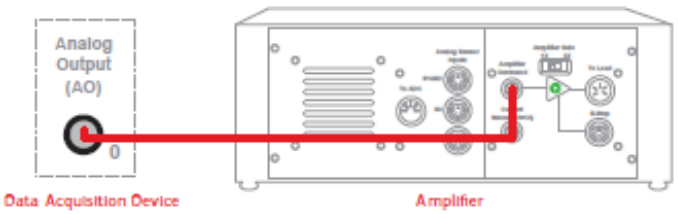
STEP 4 Set Up the Hardware

To set up your Rotary Servo Base Unit, please read the following instructions carefully. The connections shown below are illustrated using a generic data acquisition (DAQ) device and a VoltPAQ-X1 amplifier (you may have a different DAQ or amplifier). For detailed instructions, see the Rotary Servo Base Unit User Manual (enclosed with shipment).

A Before proceeding, set up and test your DAQ device (e.g., Q2-USB). For detailed instructions, see the DAQ device Quick Start Guide or User Manual.

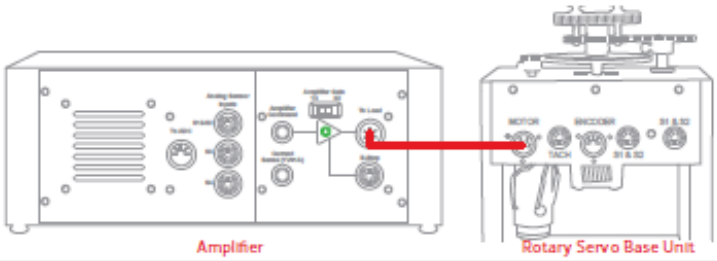
B Make sure everything is powered OFF before making any of these connections. This includes turning off your PC and the amplifier.

C



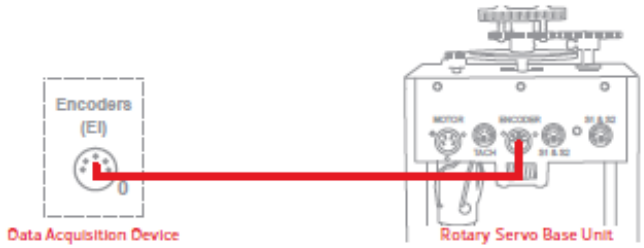
Using the RCA to RCA cable, connect **Analog Output Channel** #0 (AO #0) on the data acquisition (DAQ) device to the **Amplifier Command** socket on the amplifier.

D



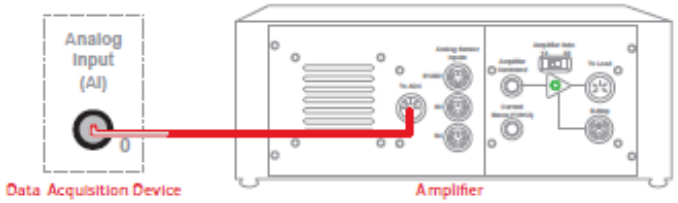
Using the 4-pin DIN to 6-pin DIN motor cable, connect the **To Load** socket on the amplifier to the **Motor** socket on the Rotary Servo Base Unit.

E



Using the 5-pin DIN to 5-pin DIN encoder cable, connect the **Encoder** socket on the Rotary Servo Base Unit panel to the **Encoder Channel #0** (EI #0) socket on the data acquisition device.

F



Using the 5-pin-DIN to 4xRCA cable, connect the 5-pin-DIN connector to the **To ADC** socket on the amplifier and the white RCA connector (S2) to **Analog Input (AI) #0** on the data acquisition device.

Figure 6: How to setup equipment

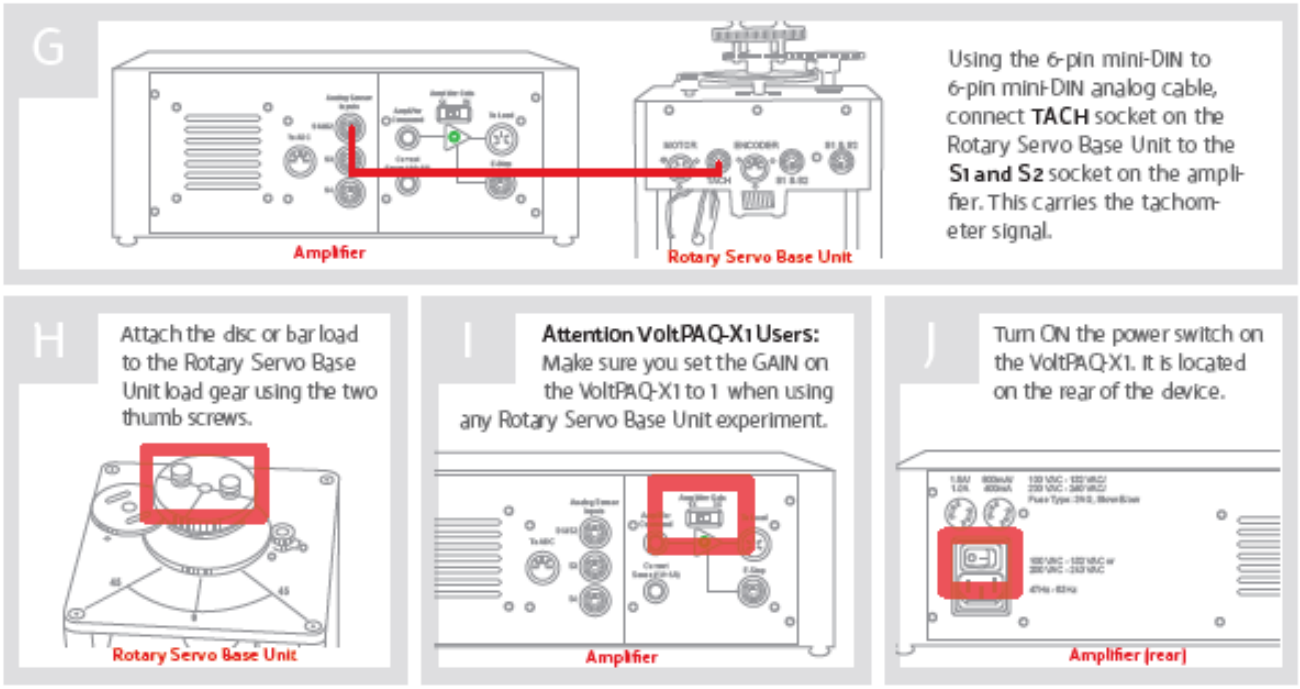


Figure 7: how to setup equipment

Also ensure that encoder is connected to the computer monitor as seen in the Figure 8:

Figure 8: Encoder connection to computer(left) Encoder setup (right)

If the LED light is off, then it is not connected to the computer monitor. It should be on as seen in the picture above

Figure 9 represents the steps to run the controller during the experiment. Notice that the actual plots shown in Figure 9 do not represent what you should obtain in step H.:

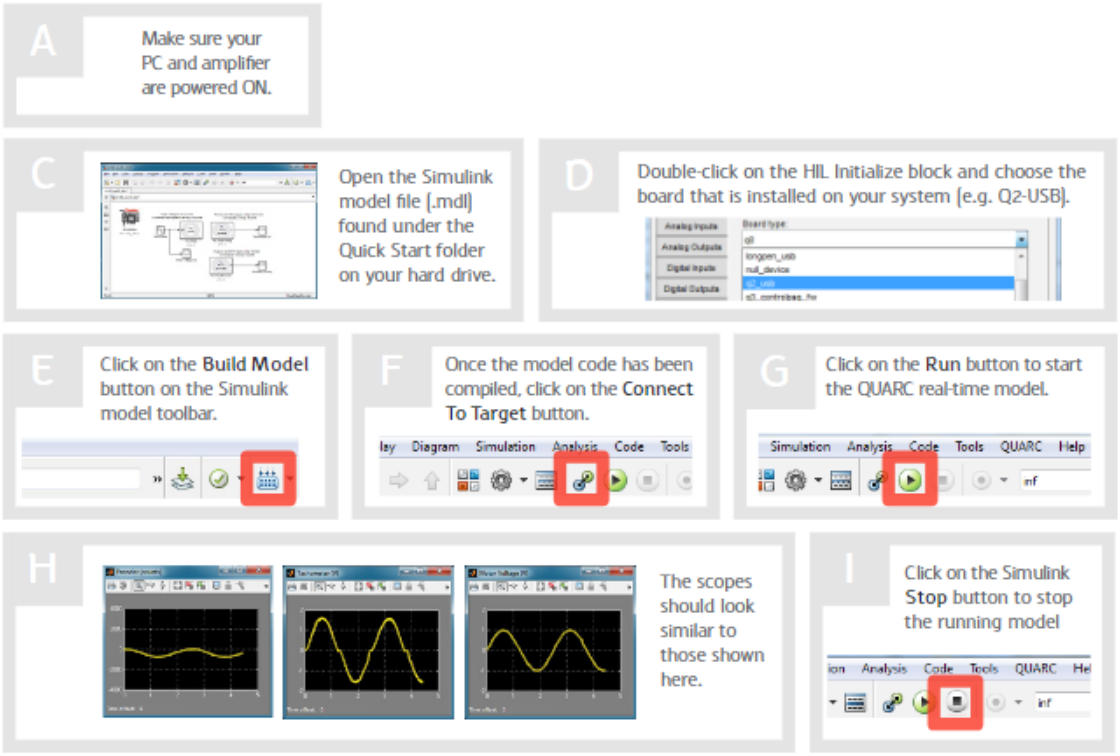


Figure 9: How to run controller during experiment

3.1 Speed Controller Implementation

IMPORTANT: Before you can conduct these experiments, you need to make sure that the lab files are configured according to your SRV02 setup.

1. Set up the hardware in the same configuration as Lab Session Experiment 01.

2. Click right on the visualization folder and add it to the path by selecting “Selected Folders and Subfolders” as shown in Figure 10.

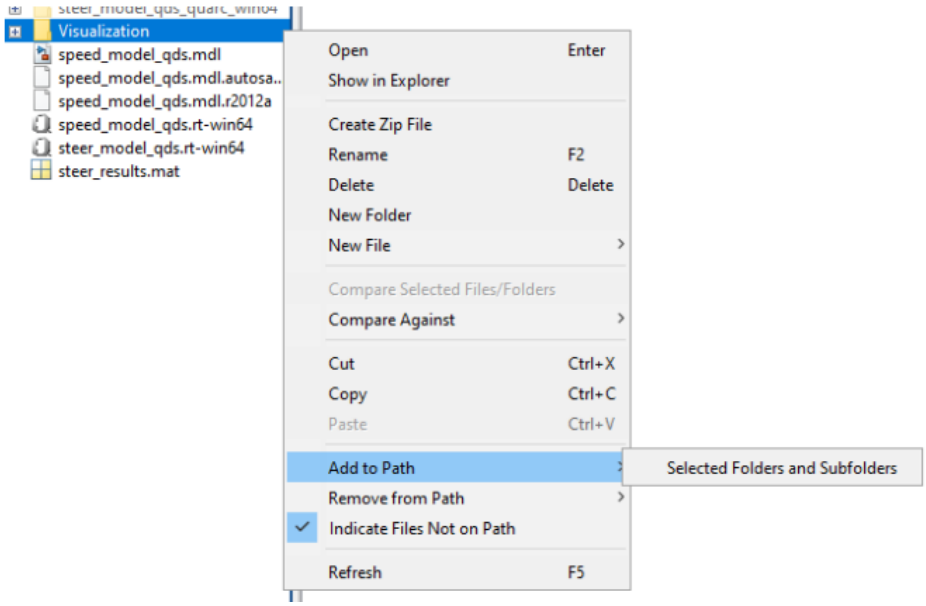


Figure 10:Select Folder and Subfolders

3. Open the Quanser Driving Simulator model *speed_model_qds.mdl*. **Make sure to disable the external driver by moving up the manual switch.**

4. Open the *Engine Dynamics* subsystem shown in Figure 11.

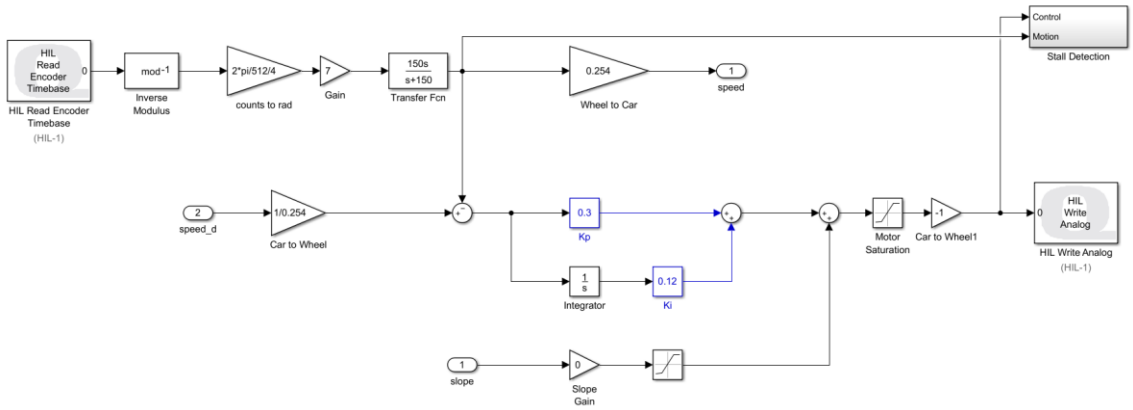


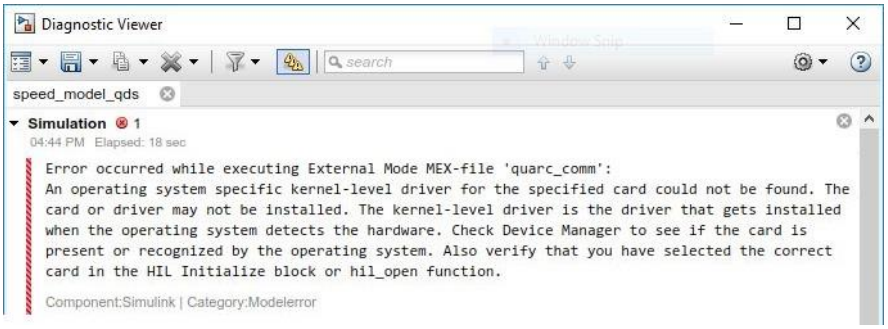
Figure 11:QDS speed controller subsystem

5. Enter a value of 10 in the slope gain block. This simulates a slope road as disturbance to velocity.

6. Enter a k_p gain of 0.3 into the K_p slider gain block, and a value of 0 into the K_i gain block (no integral gain).

7. Build the model, connect to target, then run the simulation.

   If the following error message is encountered, open the "HIL Initialize" block in the Simulink model and change "Board type:" to "q2_usb".



8. Observe the performance of the car as it makes a lap of the track.

9. Add a *Scope* to the subsystem to plot the system response (desired vs. actual speed) without integral action.

10. Enter k_i = 5 into the K_i slider gain block (now we add integral action).

11. Run the simulation.

12. What do you observe about the performance of the vehicle in both cases?

13. Plot the controller reponse (desired vs. actual speed) with integral action.

14. Rebuild and run the simulation.

15. Is the controller able to track the desired speed effectively?

16. Change the slope gain value to 50, and tune the values of K_p and K_i until to obtain an acceptable response in terms of tracking error.

17. Plot the controller response (desired vs. actual speed).

18. Change the gain of the slope disturbance in the *Slope Gain* block to 0.

19. How well is the controller able to track the speed command if the disturbance is eliminated?

20. Delete the connection between the desired speed signal input *speed_d* and the *Car to Wheel* block. Replace the desired speed command with a constant value of 50 m/s.

21. Reset the *Slope Gain* block back to the original gain of 10. Rebuild and re-run the simulation.

22. How well is the controller able to compensate for the slope disturbance? How does this compare to the setpoint tracking performance?

23. If necessary, retune the controller gains to achieve the desired performance.

24. Record the final control gains and response plots.

3.2 Steering Controller Implementation

1. Open the Quanser Driving Simulator model *steer_model_qds.mdl*. **Make sure to disable the external driver by moving up the manual switch.**

2. Click right on the visualization folder and add it to the path by selecting “Selected Folders and Subfolders

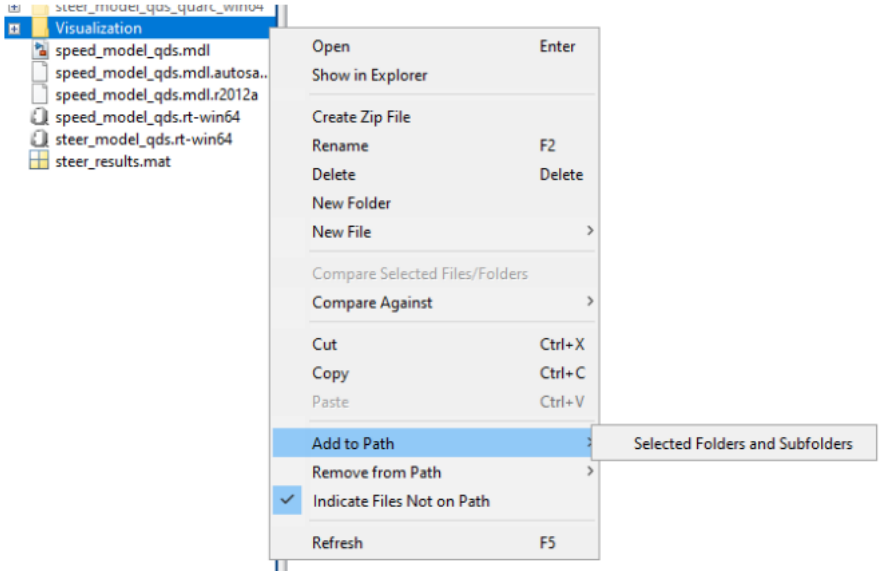


Figure 14: Select Folder and Subfolders

3. Open the *Steering Dynamics* subsystem shown in Figure 15.

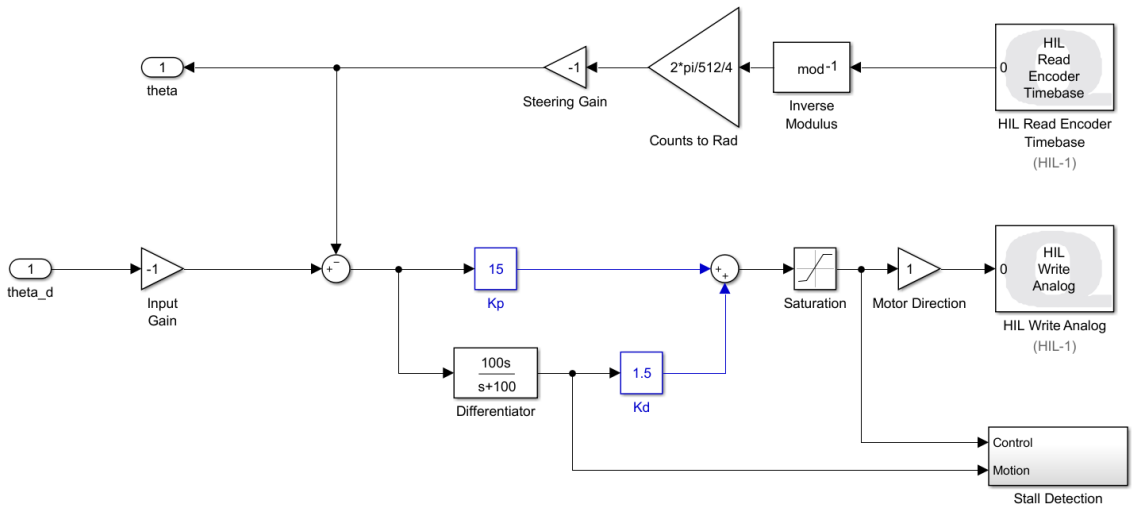


Figure 15: QDS steering control subsystem

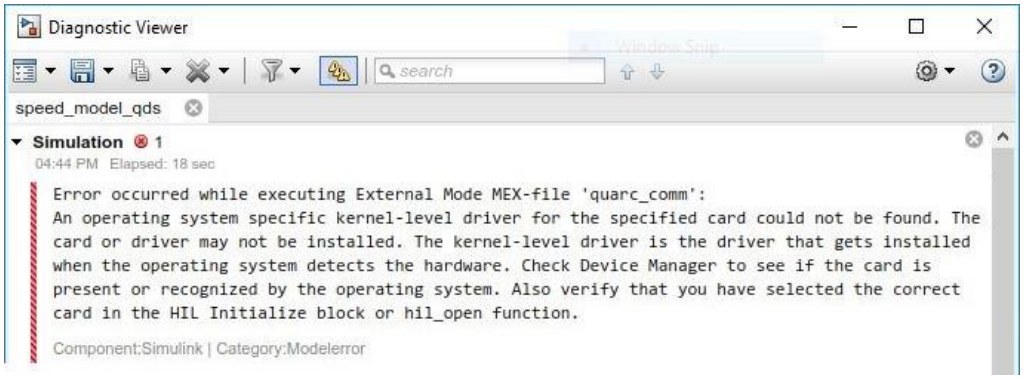
Note: Ensure that the *Input Gain* block is set to -1.

Note: Ensure that the *Steering Gain* block is set to -1, and the *Saturation* block is set to $\pm 10\ V$.

4. Enter the k_p = 15 and k_d= 1.5 gains from into the K_p and K_d slider gain blocks.

5. Build the model, connect to target, then run the simulation.

   If the following error message is encountered, open the “HIL Initialize” block in the Simulink model and change “Board type:” to “q2_usb”.



6. Observe the performance of the car as it makes a lap of the track.

7. What do you observe about the performance of the vehicle?

8. Add a *Scope* to the subsystem to plot the controller error (desired vs. actual steering angle).

9. Rebuild and run the simulation.

10. Is the controller able to track the desired steering angle effectively?

11. What changes could be made to the controller architecture to improve the performance of the steering controller?

12. Tune the controller gains to achieve a very small tracking error as the desired performance.

13. Record the final control gains and response plots.

14. What other applications could a similar system have in the real world?

3.2.1 Steering Controller Implementation- User interaction

1. Change the manual switch from disable to enable in the external driver input of the QDS block.

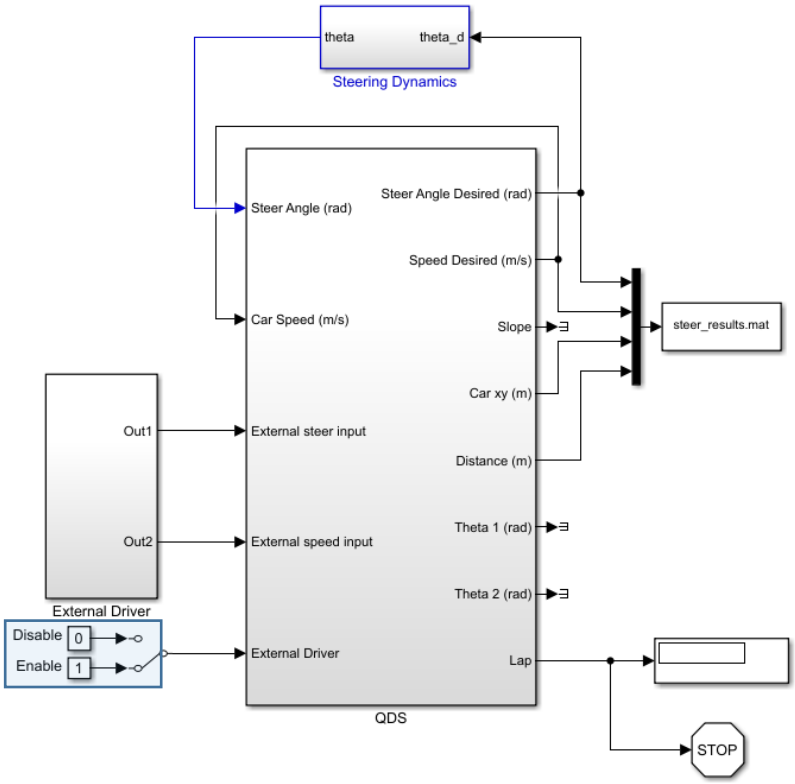


Figure 11: QDS Block

2. Within the External Driver block change the speed constant block to 40m/s. This velocity can be changed for the desired speed of the user.

3. Run the simulation and control the steering manually by rotating the servo motor. This will simulate a driver pilot using a steering wheel.

3.3 Results

Fill out Table 1 below with your answers to the Pre-Lab questions and your results from the lab experiments.

Section / Question	Description	Symbol	Value
Question 16, Section 3.1.	Proportional gain Integral gain Slope gain	K_p K_i K_{slope}	
Question 12, Section 3.2.	Proportional gain derivative gain	k_p k_d	

Table 1: Summary of results for the Cruise and Steering Control laboratory.

4. Lab Report

When you are writing your lab report, follow the outline corresponding to the experiment you conducted to build the *content* of your report. Also, in Section 4.2 you can find some basic tips for the *format* of your report.

4.1 Template for Content (Step Response Experiment)

I. PROCEDURE

1. Briefly describe the main goal of this Experiment Lab 04.
2. Briefly describe the simulation procedure when using speed control implementation.
3. Briefly describe the simulation procedure when using steering control implementation.

II. RESULTS

Do not interpret or analyze the data in this section. Just provide the results.

1. Answer to questions 1, 2 and 3 in Section 2.
2. Response plot from step 9 in Section 3.1, *Cruise control system response without integral action.*
3. Response plot from step 13 in Section 3.1, *Cruise control system response with integral action.*
4. Response plot from step 17 in Section 3.1, *Cruise control system response with increased slope disturbance.*
5. Response plot from step 24 in Section 3.1, *Cruise control system response with constant velocity.*
6. Response plot from step 8 in Section 3.2, *Steering control system response with PD Controller.*
7. Response plot from step 13 in Section 3.2, *Steering control system response with tuned PD Controller.*

III. ANALYSIS

Provide details of your calculations (methods used) for analysis for each of the following:

1. Answer to question 4 in Section 2.
2. Step 12 in Section 3.1.
3. Step 19 in Section 3.1.
4. Step 22 in Section 3.1.
5. Step 7 in Section 3.2.
6. Step 11 in Section 3.2.

IV. CONCLUSIONS

Provide a general conclusion from this lab experiment and answer to Step 14 in Section 3.2.

4.2. Tips for Report Format

PROFESSIONAL APPEARANCE

- Has cover page with all necessary details (title, course, student name(s), etc.)
- Each of the required sections is completed (Procedure, Results, Analysis and Conclusions).
- Typed.
- All grammar/spelling correct.
- Report layout is neat.
- Does not exceed specified maximum page limit, if any.
- Pages are numbered.
- Equations are consecutively numbered.
- Figures are numbered, axes have labels, each figure has a descriptive caption.
- Tables are numbered, they include labels, each table has a descriptive caption.
- Data are presented in a useful format (graphs, numerical, table, charts, diagrams).
- No hand drawn sketches/diagrams.
- References are cited using correct format.

Appendix A: SRV02 QUARC Integration

In this section, we explain how to send command voltages to the Quanser@SRV02 and measure the position and speed of its load shaft in real-time using your computer.

System Requirements

Before you begin this laboratory make sure:

- QUARC@is installed on your PC, as described in [1].
- You have a QUARC@compatible data-acquisition (DAQ) card installed in your PC. For a listing of compliant DAQ cards, see [5].
- SRV02 and amplifier are connected to your DAQ board as described in [6].

Prerequisites

The user should be familiar with the following:

- System hardware:
  - Data acquisition card (e.g. Q2-USB in [2])
  - Amplifier (e.g. VoltPAQ in [4]).
  - Main components of the SRV02, i.e. DC motor and sensors such as the potentiometer [6].
- Basics of Simulink@.

A.1 Applying Voltage to SRV02 Motor

Here are the basic steps to apply a voltage to the SRV02 motor using QUARC@:

1. Make a Simulink@model that interacts with your installed data-acquisition device using blocks from the *QUARC Targets* library. This is explained in Section A.1.1,
2. From the Simulink@model, build real-time code as shown in Section A.1.2, and
3. Execute the code as explained in Section A.1.3.

A.1.1 Making the Simulink Model

In this section, we will make a Simulink@model as shown in Figure A.1 using QUARC@blocks to feed a sinusoidal voltage to the SRV02 dc motor. The blocks from the *QUARC Targets* library are used to interact with a data-acquisition board, e.g. Quanser Q2-USB or Q8-USB device.

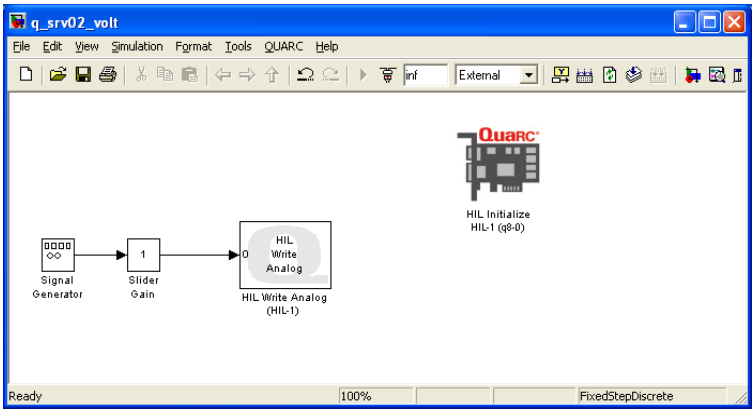


Figure A.1: Simulink model used with QUARC to apply voltage to SRV02

Follow these steps to make the Simulink@diagram:

1. Load the Matlab@software.
2. Create a new Simulink@diagram. To do this, go to File | New | Model item in the menu bar.
3. Open the Simulink Library Browser window by clicking on the View | Library Browser item in the Simulink menu bar or clicking on the Simulink icon.
4. Expand the *QUARC Targets* item and go to the Data Acquisition | Generic | Configuration folder, as shown in Figure A.2.
5. Click-and-drag the *HIL Initialize block* from the library window into the blank Simulink@model. This block is used to configure your data-acquisition device, e.g. the Quanser Q2-USB or Q8-USB hardware-in-the-loop (HIL) boards.
6. In the Library Browser, go to the Data Acquisition | Generic | Immediate I/O category. This contains various blocks used to interact with actuators and sensors.
7. Click-and-drag the *HIL Write Analog* block from the library into the Simulink@diagram. This block is used to output a voltage from an Analog Output channel, i.e. digital-to-analog (D/A) channel, on the data-acquisition device.

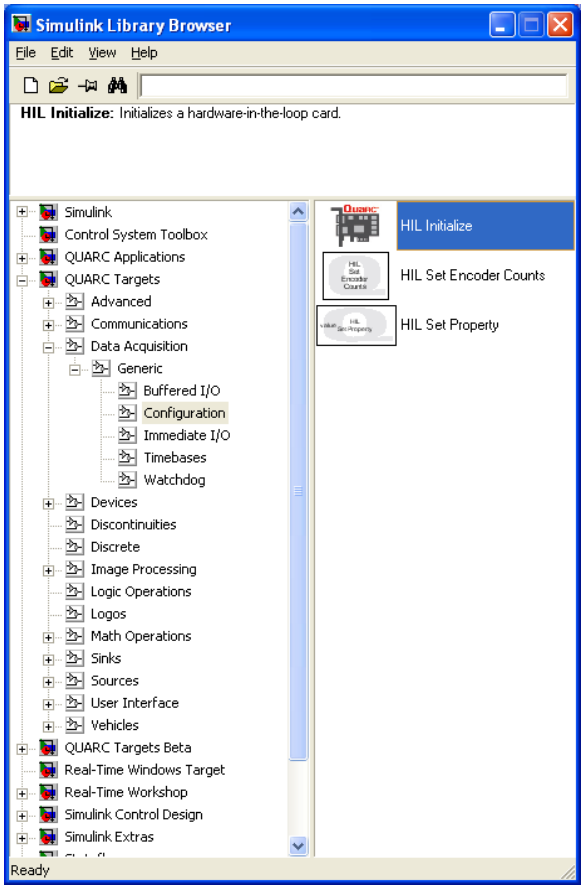


Figure A.2: *QUARC Targets* in Simulink@ Library Browser

8. Add the *Signal Generator* block, found in the Simulink@ | Source folder, and the *Slider Gain* block, from the Simulink@ | Math Operations category, into the Simulink@ model. Connect the blocks as shown in A.1.
9. Double-click on the *HIL Initialize* block:
   - In the *Board type* field, select the board that is installed in your PC. For example, if you have a Quanser Q2-USB board then select *q2_usb*.
   - If more than one of the same board is installed (e.g. two Q2-USB devices), ensure the *Board number* field is set correctly, e.g. if two boards are used, then choose either 0 or 1.
10. Go to the *Analog Output* pane shown in Figure A.3. Make sure the *Analog output channels* field includes 0. This will ensure 0 V is output from Analog Output channel #0 when the QUARC controller is stopped. For more information, click on the *Help* button. Otherwise, click on the OK button and proceed.

    Note: If you are using a NI DAQ device, make sure you enter [0] in the Analog Output channel box. The analog output channels for the NI boards are not selected by default.
11. Double-click on the *HIL Write Analog* block.
    - Make sure *Board name* is set to HIL-1 (i.e. points to the HIL Initialize block).
    - Set *Channels* to 0 (default setting). Recall that, as instructed in Reference [6], the dc motor is connected to Analog Output Channel #0 on the hardware-in-the-loop board. Therefore, *Channels* should be set to 0.
    - Set *Sample time* to -1 (default setting). This implies that the sampling interval is inherited from the previous block.
12. Click on the OK button to save and close the *HIL Write Analog* block properties.
13. Save the Simulink@ mode by selecting the File | Save item in the menu bar or clicking on the Save icon.

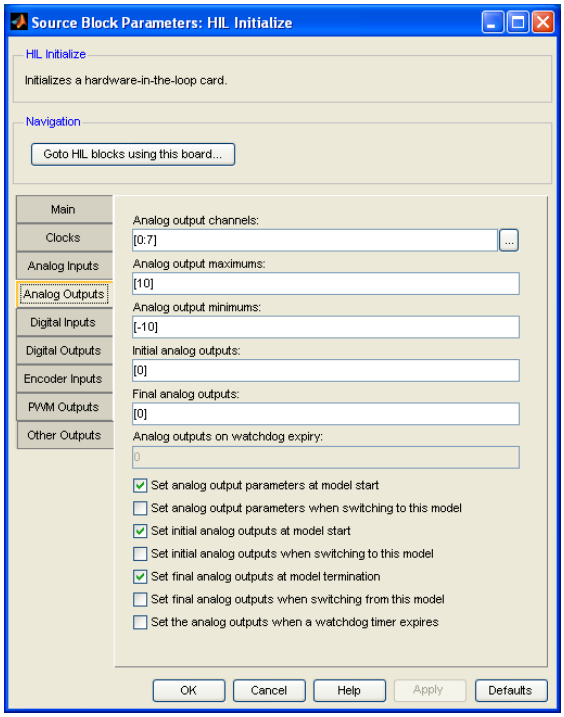


Figure A.3: Configuring Analog Output channel on DAQ device

A.1.2 Compiling the Model

The Simulink@ model we made in Section A.1 can now be used by QUARC to generate code. When you execute this code, a voltage is sent to the SRV02.

Follow these steps to generate code from a Simulink@ diagram:

1. In the Simulink@ diagram made in Section A.1, go to the QUARC | Set default options item to set the correct Real-Time Workshop parameters and setup the Simulink@ model for external use (as opposed to the simulation mode).
2. To view the compiler options shown in Figure A.4, go to QUARC | Options in the Simulink@ model tool bar.
3. Real-Time Workshop pane:
   - *System target file* is set to Target Language Compiler file quarc_windows.tlc.
   - *Make command* is set to make_rtw and the *Template makefile* is set to quarc_default_tmf file.
4. Solver pane:
   - *Stop time* is set to *inf* in order for the code to be executed continuously until it is stopped manually by the user. Alternatively, the *Stop time* parameter can be set to the desired duration (code will cease executing when the stop time value is reached).
   - *Type* parameter is set to Fixed-step. When compiling real-time code, the solver must be fixed-step as opposed to variable step which can be used in simulations.
   - *Solver* is set to discrete. There are no continuous blocks inside the designed Simulink@ model, therefore having a discrete solver is fine. However, if an Integrator block or another continuous system were be added, then the Solver field would have to be changed to an integration method such as *ode1 (Euler)*.
   - *Fixed-step size* field sets the sampling interval, or sampling time, of the controller. By default this is set to 0.002 seconds, which is a sampling rate of 500 Hz.

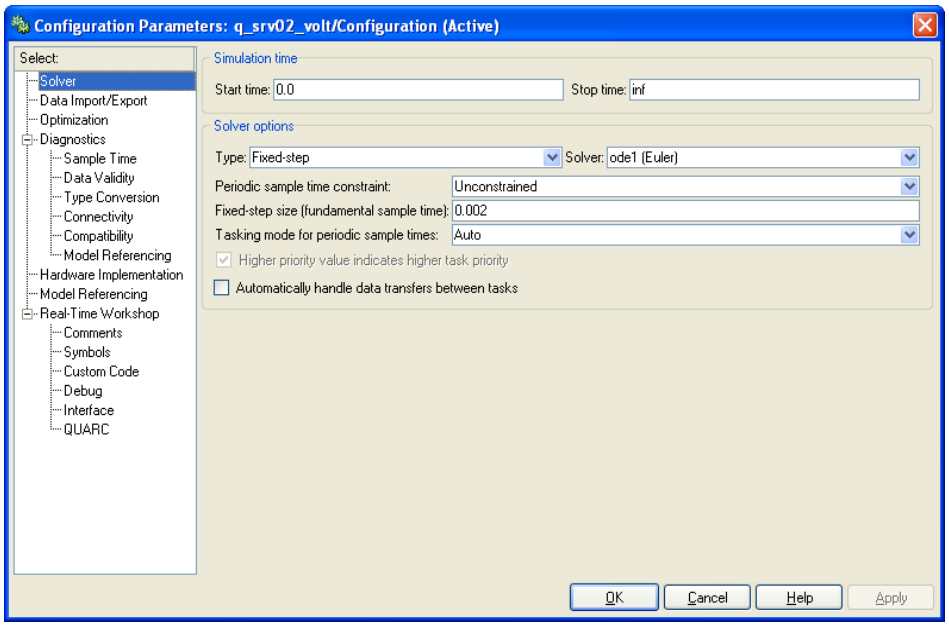


Figure A.4: Default QUARC solver settings

5. Click on the OK button to close the Configuration Parameters window.
6. Select the QUARC | Build item. Various lines in the Matlab@ Command Window should be displayed as the model is being compiled.
7. Once done compiling, a QUARC Windows executable file along with a folder containing various ``C'' and Matlab@ files are generated. Note that once the executable is created, the folder is no longer needed. If you like, you can remove the executable and associated code folder may be removed from the current directory by clicking on QUARC | Clean item.

See [5] for more information about configuring QUARC@.

A.1.3 Running QUARC Code

Once the Simulink@ model has been compiled, the code can be executed and the voltage set in the Simulink@ model can be sent to the SRV02 motor. Here are the steps to follow:

1. Power ON your power amplifier (e.g. Quanser VoltPAQ).
2. To begin executing the code, click on the QUARC | Start item in the Simulink@ model. The SRV02 external gears on the SRV02 should begin rotating back-and-forth. This command actually does two things: it connects to the target and then executes the code. Alternatively, users may decide to go through these steps individually:

   Option 1: In the Simulink@ model tool bar, click on the *Connect to target* icon and then click on the *Run* icon. These buttons are shown in Figure A.5.

   Option 2: Select the Simulation | Connect to target item from the menu bar and the select Simulation | Start Real-Time Code item.

3. Double-click on the Signal Generator block to open its parameter window.
4. Set the *Frequency* field to 0.5 Hz and click on the OK button. Notice how the parameter change effects the SRV02 immediately: the gears on the SRV02 begin to switch back-and-forth slowly.

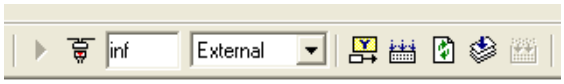


Figure A.5: Simulink@model toolbar: connect to target and compilation

5. Vary the value of the Slider Gain block between 0 and 2. Examine how the angular rate of the SRV02 gears change proportionally with the amplitude of the sine wave.
6. Select the QUARC | Stop item to stop the code execution (or click on the Stop button in the Simulink@model tool bar).
7. Power OFF the amplifier if no more experiments will be run in this session.

A.2 Reading Position using Potentiometer

In this section, the Simulink@model previously designed in Section A.1 is modified to obtain readings from the potentiometer sensor.

A.2.1 Reading the Potentiometer Voltage

Follow the procedure below to design the Simulink@diagram as shown in Figure A.6 to read the potentiometer voltage.

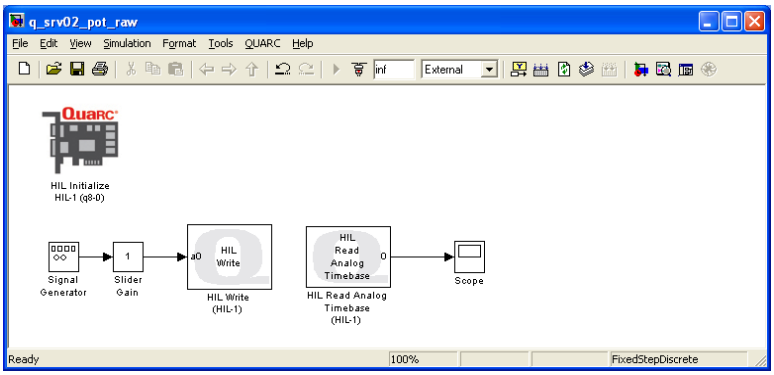


Figure A.6: Simulink@model used with QUARC to read potentiometer

1. From the Generic | Timebases category in the Simulink@Library Browser, click-and-drag the *HIL Read Analog Timebase* block into the Simulink@diagram created previously in Section A.1. This block will be used to read a voltage from an Analog Input channel, i.e. analog-to-digital (A/D), on the data-acquisition device.

   Note: Using a *Timebase* type block causes the running model to be triggered by the hardware timer on the data-acquisition board as opposed to the system clock. This increases performance of the controller by reducing jitter and allowing for greater sampling rates. If you want to use system clock instead (some DAQ cards do not support hardware-based timing, see [5] for details), then use blocks from the *Immediate I/O* category instead.

2. Recall that, as instructed in [6], the potentiometer is connected to Analog Input #0 on the DAQ board. Therefore, the default setting of 0 in the block is fine.

3. Add a Scope block from the Simulink@| Sinks folder in the Library Browser and connect it to the output of the *HIL Read Analog Timebase* block, as shown in Figure A.6.

4. Set the *Frequency* parameter in the Signal Generator block to 1.0 Hz and the Slider Gain block to 1.

5. Double-click on the *Scope* block to open the scope.

6. Save the Simulink@model (if you want to save the model as a different file, go to *Save As*).

7. Power ON the power amplifier.

8. Go to QUARC | Build to compile the code.

9. Click on QUARC | Start to execute the code. As the SRV02 rotates back-and-forth, the Scope should display the potentiometer readings similar to Figure A.7. Notice that the readings shown are the voltage output of the potentiometer and not an angular measurement in degrees or radians.

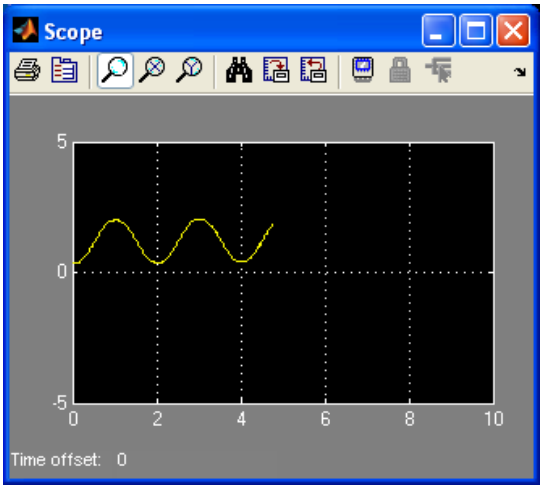


Figure A.7: Reading raw voltage from the SRV02 potentiometer

Note: If the SRV02 gears are rotating but the scope is not plotting any data, then you must re-configure the RTW Signal & Triggering properties inside the Simulink@ diagram. See the *Signal Triggering* section under *QUARC Basics* in [5] for more information.

10. Vary the sine wave frequency between 0.1 and 2 Hz and the Slider Gain value between 0 and 2. Notice how the controller is updated in real-time and the measurement in the scope changes with the new parameters.

    Note: The potentiometer has an electrical range of 352 degrees across $\pm$ 5 V [6]. As a result, the potentiometer outputs a discontinous voltage signal as seen in Figure A.8. Then, a disadvantage is that you cannot surpass the $\pm$ 180 degree range when controlling the position or take the derivative of this sensor to control velocity.

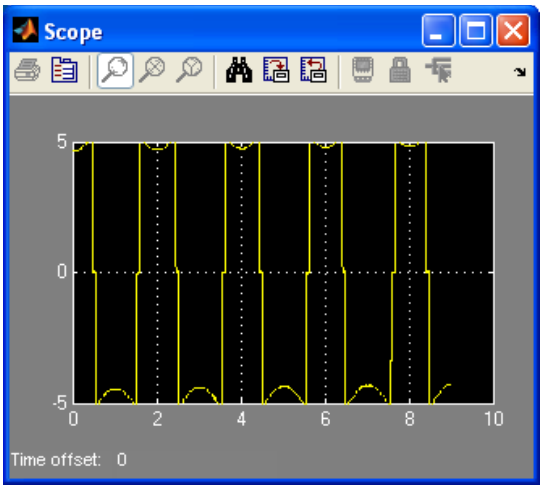


Figure A.8: Potentiometer reading when encountering its discontinuity

11. Stop the code by clicking on QUARC | Stop.

A.2.2 Measuring Position

The Simulink@ model will now be modified to read the *load angle* from the potentiometer and the voltage commanded to the motor as shown in Figure A.9.

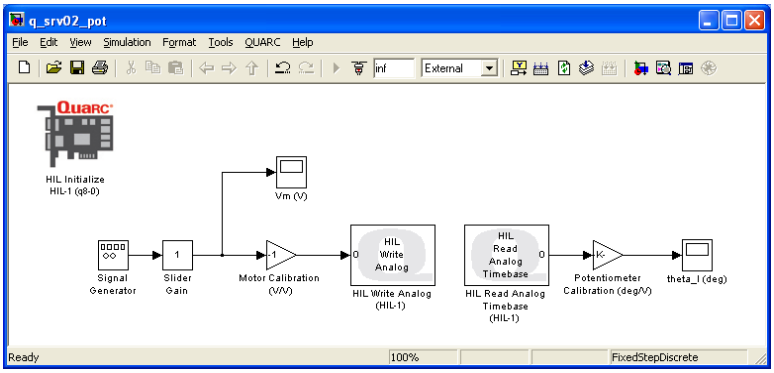


Figure A.9: Simulink@ model used with QUARC to send voltage to the SRV02 and read its load shaft angle using potentiometer.

Modify the Simulink@ diagram built in the previous section following these steps:

1. Label the Scope block that is presently connected to the *HIL Read Analog Timebase* block to *theta_l (deg)*. This will display the angular measurement of the load gear in degrees.
2. Add a Scope block from the Simulink@ | Sinks folder in the Library Browser and connect it to the output of the Slider Gain block, as shown in Figure A.9, above. Label the Scope block *Vm (V)* which stands for the input motor voltage.
3. From the Simulink@ | Math Operations category in the Library Browser, add two Gain blocks into the Simulink@ model.
4. Connect the gain blocks as illustrated in Figure A.9.
    - One gain block between the *Amplifier Pre-Compensation* and *HIL Write Analog* blocks and label it *Motor Calibration (V/V)*.
    - Another gain block between the *HIL Read Analog Timebase* and *theta_l* Scope blocks and denote it *Potentiometer Calibration (deg/V)*.
5. Re-compile the Simulink@ model, i.e. click on QUARC | Build.
6. Click on QUARC | Start to run the code.
7. Open the *Vm (V)* and *theta_l (deg)* scopes.
8. In the Signal Generator, set the *Frequency* parameter to 0.25 Hz.
9. Set the Slider Gain block to 1.
10. Currently when the voltage goes positive the load gear rotates in the clockwise direction. However, the desired convention is for the load gear to rotate in the counter-clockwise direction when the voltage goes positive. Thus, set the *Motor Calibration (V/V)* block to -1.
11. As described in [6], the potentiometer outputs between $\pm$ 5 V when it rotated 352 degrees. Enter the value 352/10 in the *Potentiometer Calibration (deg/V)* block.
12. In the *theta_l (deg)* scope, click on the *Autoscale* icon so the y-axis range is increased and the full signal can be viewed.
13. Set the Slider Gain block to 0.
14. Rotate the load gear manually and examine the corresponding response in the *theta_l (deg)* scope. Confirm that, indeed, the correct measurement is being taken.
15. Position the load gear such that 0 is read in the *theta_l (deg)* scope .

16. Set the Slider Gain block to 1.

17. Examine the relationship between the input voltage and load position. When the input voltage increases in the positive direction, the potentiometer angle decreases. Add a negative sign to the *Potentiometer Calibration (deg/V)* block so the entered value becomes -352/10. Now, the angular position of the load gear, displayed in the *theta_l (deg)* scope, reads increasingly positive when the commanded motor input voltage, i.e. signal in the *Vm (V)* scope, is positive.

    Note: The theoretical model assumes that the velocity increases when the voltage is positive. The feedback control design is based on the model. Therefore, it is imperative that the actual system follows the same conventions as the model that will be developed.

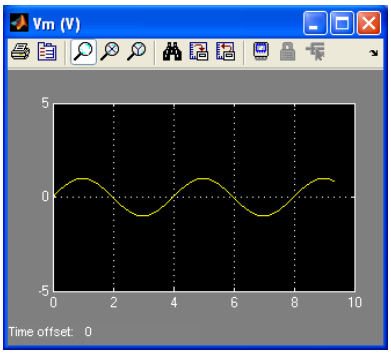


Figure A.10: Sinusoidal input voltage.

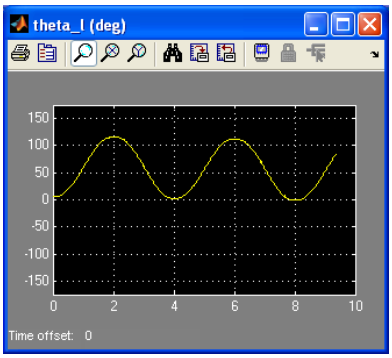


Figure A.11: Load gear position reading using potentiometer.

18. Select the QUARC | Stop item to stop the code from running.

19. Power OFF the amplifier if no more experiments will be run in this session.

A.3 Measuring Speed using Tachometer

In this section, we will modify the Simulink@ diagram designed in Section A.2 to include the readings from the tachometer.

Before continuing, please ensure that the SRV02 unit being used has a tachometer, i.e. SRV02-T option.

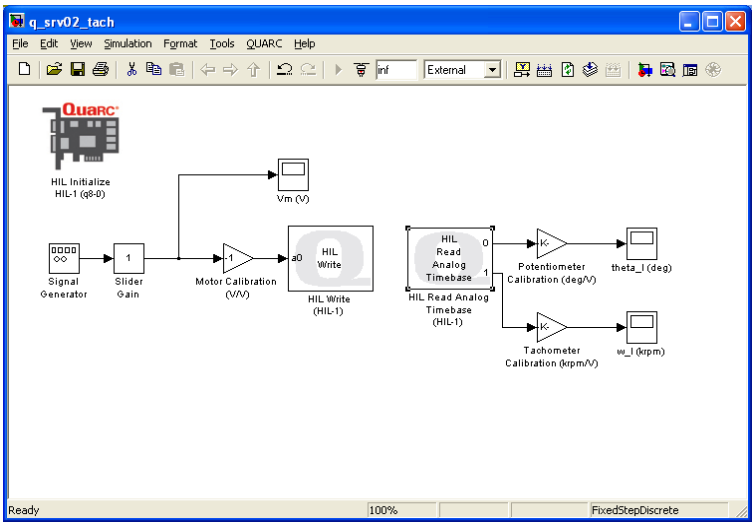


Figure A.12: Simulink@ model used with QUARC to send voltage to SRV02 and read the potentiometer and tachometer sensors.

Using the Simulink@ diagram built in Section A.2, go through this procedure to add the tachometer functionality:

1. Double-click on the *HIL Read Analog Timebase* block to open its properties.
2. As detailed in [6], the tachometer is connected to Analog Input #1 on the hardware-in-the-loop board. To add a channel, set the *Analog channels* field to [0,1] and click on the OK button.
3. Add a Scope block and a Gain block from Library Browser into the Simulink@ model.
4. Connect the Scope and Gain blocks as depicted in Figure A.12, above.
    - Connect the Gain block to the output of Channel #1 from the *HIL Read Analog Timebase* block and label it *Tachometer Calibration (krpm/V)*.
    - Connect the output of this gain block to the input of the Scope block and label the scope *w_l (rpm)*. The speed of the load gear is denoted by the ω_l. This scope will display the measured speed of the load gear in thousand revolutions per minute.
5. Set the Signal Generator *Frequency* parameter to 1.0 Hz and the Slider Gain block to 1.
6. Open the *Vm (V)* and the *w_l (krpm)* scopes.
7. Power ON the amplifier.
8. Go to QUARC | Build to compile the code.
9. Click on QUARC | Start to execute the code. As the SRV02 rotates back-and-forth, the *w_l (rpm/V)* scope should display the tachometer readings. Since the *Tachometer Calibration (krpm/V)* gain has not been configured yet, the scope is displaying the tachometer output voltage, which is proportional to the speed of the load shaft.

10. The back-emf constant of the tachometer sensor is 1.5 mV/rpm. However, the measurement is taken directly from the motor itself (see [6]). Thus, to read the velocity of the gear the tachometer calibration gain must be divided by the gear ratio. Enter 1 / 1.5 / 70 in the *Tachometer Calibration (krpm/V)* gain block when using the SRV02 in the high-gear configuration (or 1 / 1.5 / 14 if using the low-gear configuration).

    Note: The measurement will be very small. Click on the *Autoscale* icon in the scope to zoom up on the signal. Alternatively, the y-range of the scope can be set manually. To do this, right-click on the y-axis, select Axes Properties from the drop-down menu, and set the desired y-range values.

11. Examine the relationship between the input voltage and load speed. When the input voltage increases in the positive direction, the tachometer velocity decreases. Similar to the potentiometer, the speed of the load shaft should go positive when the input voltage is positive. To fix this, add a negative sign to the *Tachometer Calibration (krpm/V)* block so the entered value becomes -1 / 1.5 / 70 (or -1 / 1.5 / 14 for low-gear). The input voltage and load velocity scopes should read as shown in Figure A.13 and Figure A.14.

    Note: If you want the measurement to be in RPM instead of kRPM, enter -1000 / 1.5 / 70 gear (or -1000 / 1.5 / 14 for low-gear).

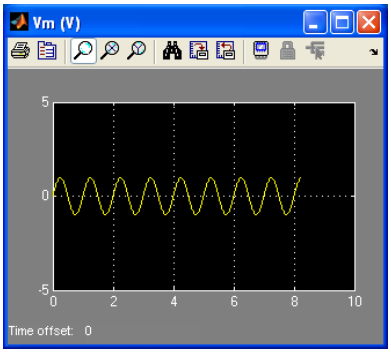


Figure A.13: Sinusoidal input voltage.

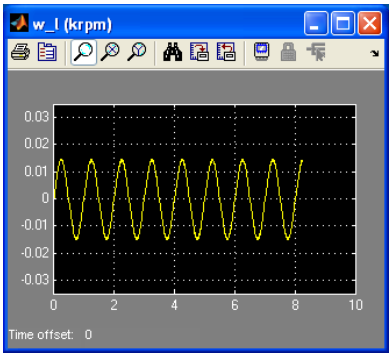


Figure A.14: High gear velocity reading using tachometer.

12. Select the QUARC | Stop item to stop the code from running.

13. Power OFF the amplifier if no more experiments will be run in this session.

A.4 Measuring Position using Encoder

The Simulink@ diagram designed previously is modified to include an encoder measurement, as illustrated in Figure A.15 below.

Before continuing, please ensure that the SRV02 unit being used has an encoder, i.e. SRV02-E option.

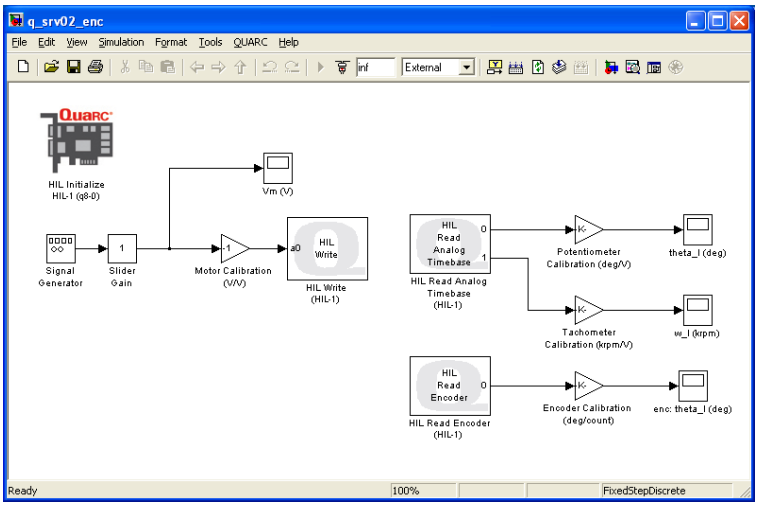


Figure A.15: Simulink@ model used with QUARC to send voltage to SRV02 and reads the potentiometer, tachometer, and encoder sensors.

Using the Simulink@ model designed in either Section A.2 or A.3, follow this procedure to add encoder functionality:

1. From the QUARC Targets | Data Acquisition | Generic | Immediate I/O category in the Library Browser, add a HIL Read Encoder block.
2. Recall that, as instructed in [6], the encoder is connected to Encoder Input #0 on the data acquisition board. The HIL Read Encoder block is already configured for to read channel 0 and the default encoder configurations in the HIL Initialize block are fine (but keep in mind that these can be changed).
3. Add a Gain block and a Scope block from the *Math Operations* and *Sinks* folders in the Library Browser, respectively, into the Simulink@ model.
4. Connect the scope and gain blocks as depicted in Figure A.15.
    - Connect the HIL Read Timebase block output *e0* to a Gain block and label it *Encoder Calibration (deg/count)*.
    - Connect the output of this gain block to the input of the Scope block and label the scope *enc: theta_l (deg)*. This scope will display the measured angular position of the load gear in degrees.
5. Set the *Frequency* parameter in the Signal Generator block to 1.0 Hz and the Slider Gain block to 1.
6. Open the *Vm (V)* and *enc: theta_l (deg)* scopes.
7. Save the Simulink@ model (you may want to save the model as a different file).
8. Power ON the power amplifier.
9. Go to QUARC | Build to compile the code.

10. Click on QUARC | Start to execute the code. As the SRV02 rotates back-and-forth, the *enc: theta_l (deg)* should display the encoder readings. Since the *Encoder Calibration (deg/count)* gain has not been configured yet, the scope is displaying the number of counts from the encoder output, which is proportional to the position of the load shaft.

    Note: The measurement will be very large. Click on the Autoscale icon in the scope to zoom out and view the entire signal. Alternatively, the y-range of the scope can be set manually. To do this, right-click on the y-axis, select Axes Properties from the drop-down menu, and set the desired y-range values.

11. As discussed in [6], the encoder outputs 4096 counts for every full revolution. To measure the load gear angle, set the *Encoder Calibration (deg/count)* gain block to 360 / 4096 degrees per count.

12. The measurement will be very small. Click on the *Autoscale* icon in the scope to zoom up on the signal or adjust the range of the y-axis manually. The input voltage and position scopes should appear similarly as shown in Figure A.16 and Figure A.17. Note that no further calibration is needed since the encoder position increases when the input voltage goes positive.

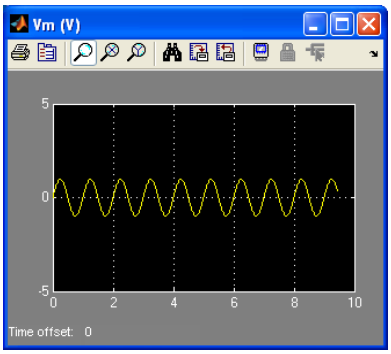


Figure A.16: Sinusoidal input voltage.

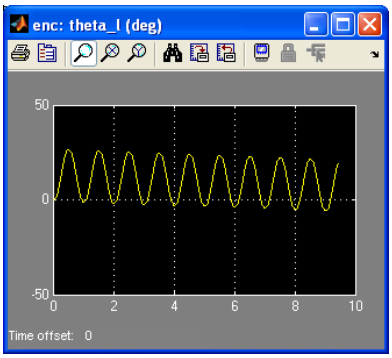


Figure A.17: Position reading using encoder.

13. Select the QUARC | Stop item to stop the code from running.

14. Power OFF the amplifier if no more experiments will be run in this session.

A.5 Saving Data

The scopes in the Simulink@model can be configured to save variables in the Matlab@workspace. For instance, to configure the load gear position scope *theta_l (deg)* in the Simulink@model from Section A.2 perform the following:

1. Open the *theta_l (deg)* scope.
2. Click on *Parameters* icon and select the *Data History* tab, as shown in Figure A.18.
   - Select the *Save data to workspace* check box.
   - Set the *Variable name* field to a desired variable, e.g. *theta_l*.
   - Set Format to Array.

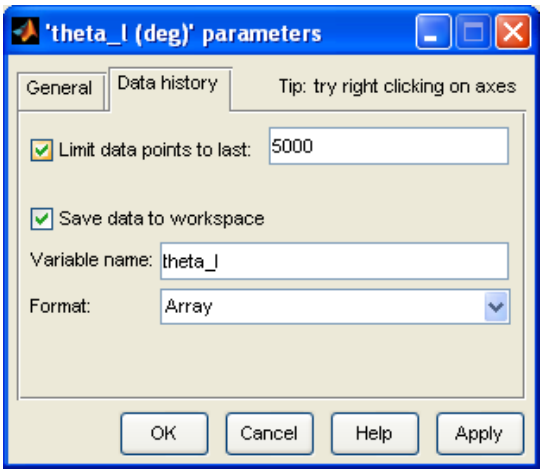


Figure A.18: Adjusting scope parameters to save data

Note: By default, the Limit data points to last box is set to 5000. This means that only the last 5000 points of data will be saved in the variable. Thus, given that the controller runs at 500 Hz, this implies that the last 10 seconds of data shown in the *theta_l (deg)* scope will be captured.

3. Click on the OK button.
4. Save the Simulink@model.
5. Select QUARC | Build to rebuild the model.
6. Click on QUARC | Run.
7. Run the controller for a few seconds and stop QUARC.
8. When the controller is stopped, the variable *theta_l* is saved to the workspace. The variable is a two-dimensional array with a maximum of 5000 elements. The first vector is the running time and the second vector is the position of the load gear. You can plot the data into a Matlab figure using a script like:

   ```
   t = theta_l(:,1); th_l =
   theta_l(:,2); plot(t,th_l,'r-
   ');
   ```

   You can then add Matlab@commands such as *xlabel*, *ylabel*, and *title* to desribe the data and units you are plotting.

   Note: If the controller has not run for the full 10 seconds then, it will have t_f/T_s number of elements, where t_f is the duration of the controller and T_s is the sampling interval. For instance, if you ran QUARC for 4 seconds then there will be $4/0.002 = 2000$ elements.

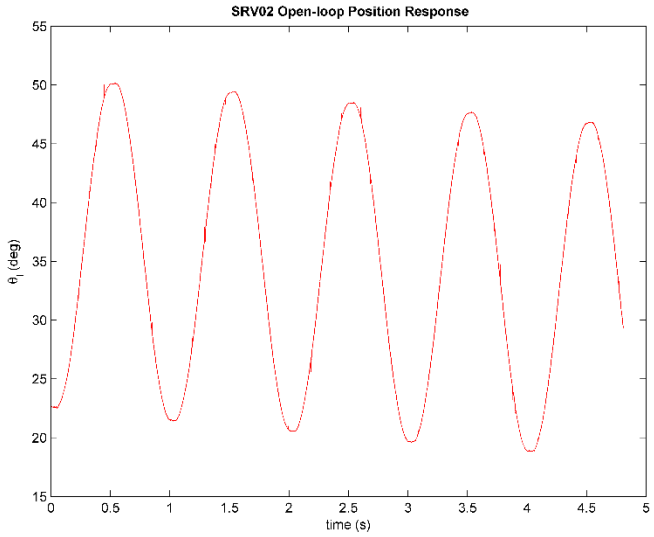


Figure A.19: Plotting saved data

9. Running the script generates a Matlab figure as shown in Figure A.19.

   Note: Use the Matlab@command *ginput* to measure points directly from the Matlab figure.

There are many different ways to save data for offline analysis, e.g. saving data to a Matlab MAT file. Go to [5] in the QUARC Basics | Data Collection category for more information.

BIBLIOGRAPHY

[1] Quanser Inc. *QUARC Installation Guide*, 2009. [2] Quanser Inc. *Q2-USB User's Manual*, 2010.

[3] Quanser Inc. *VoltPAQ User's Manual*, 2010. [4] Quanser Inc. *QUARC User Manual*, 2011. [5] Quanser Inc. *SRV02 User Manual*, 2