Lab Procedure for Simulink
Line Following

Setup

1. It is recommended that you review Lab 3 – Application Guide before starting this lab.

2. Launch Quanser Interactive Labs, scroll to the "QBot Platform" menu item and then select the "Warehouse" world.

3. Open the Simulink Model line_following.slx, as shown in Figure 1. The loading of the Simulink model also triggers the setup script for QLab. When the script is run successfully, the QBot and the Environment should be spawn in QLab and User LEDs on the virtual QBot will turn white, as shown in Figure 2.
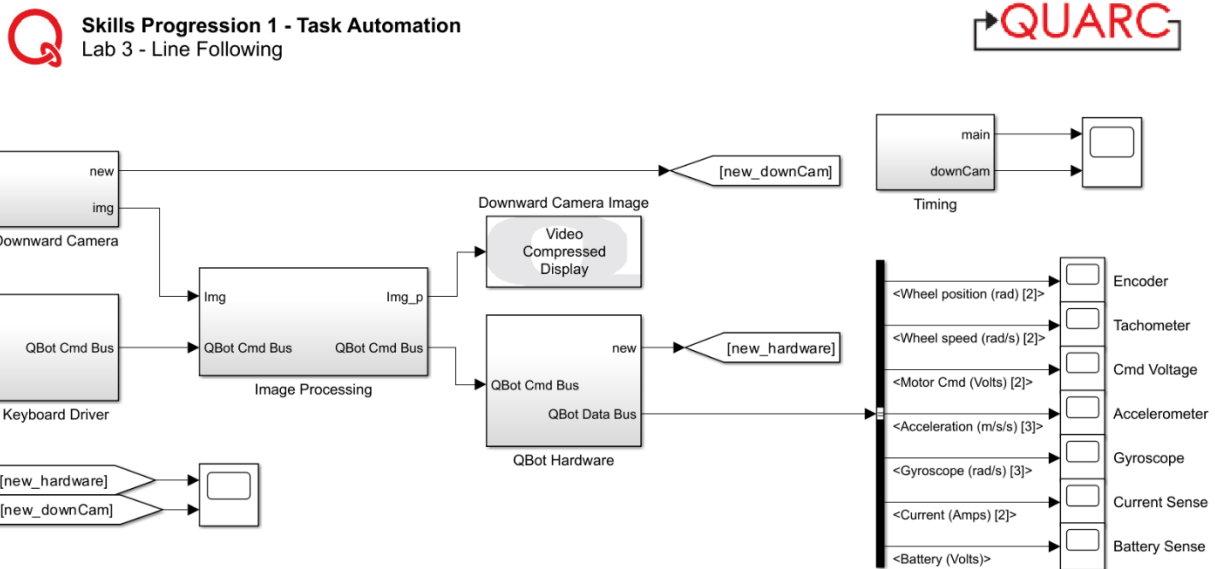

Figure 1. Line Following Simulink Model

Figure 2. Successfully Sat up QLab World

Preparing the Image

1. In this lab you will be focusing on processing the information from the downward facing camera to allow the QBot to drive by itself. This will be done in the **Image Processing** subsystem. At the end of the lab, it should look like Figure 3.
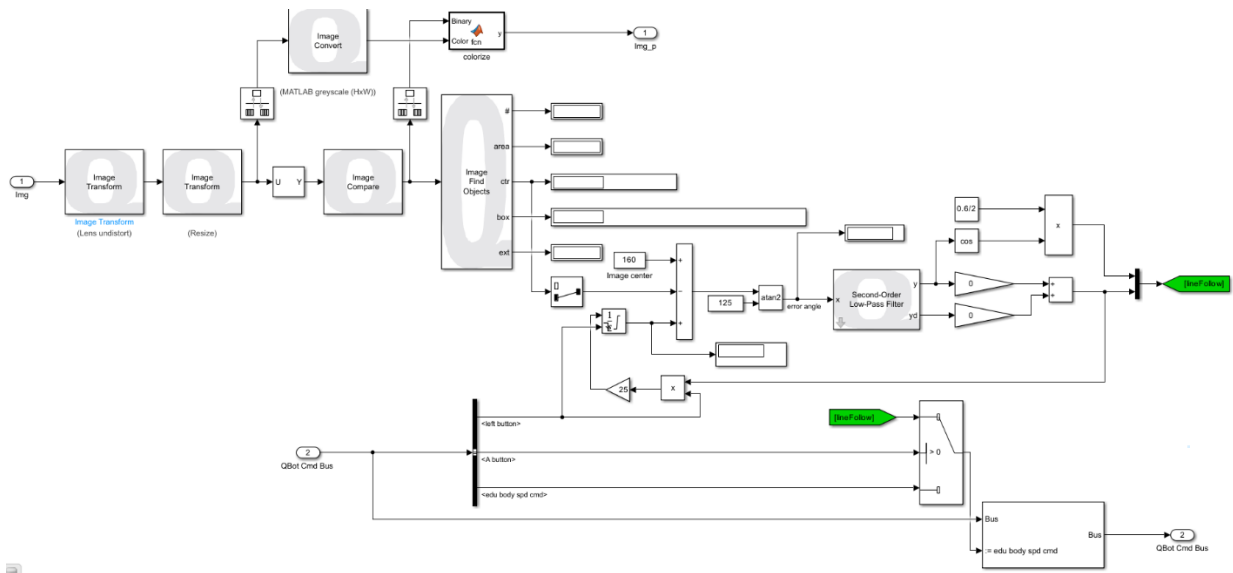

Figure 3. Image Processing Block

2. Currently, your block should look like Figure 4. It is incomplete and you will fill it as you go through the next sections.
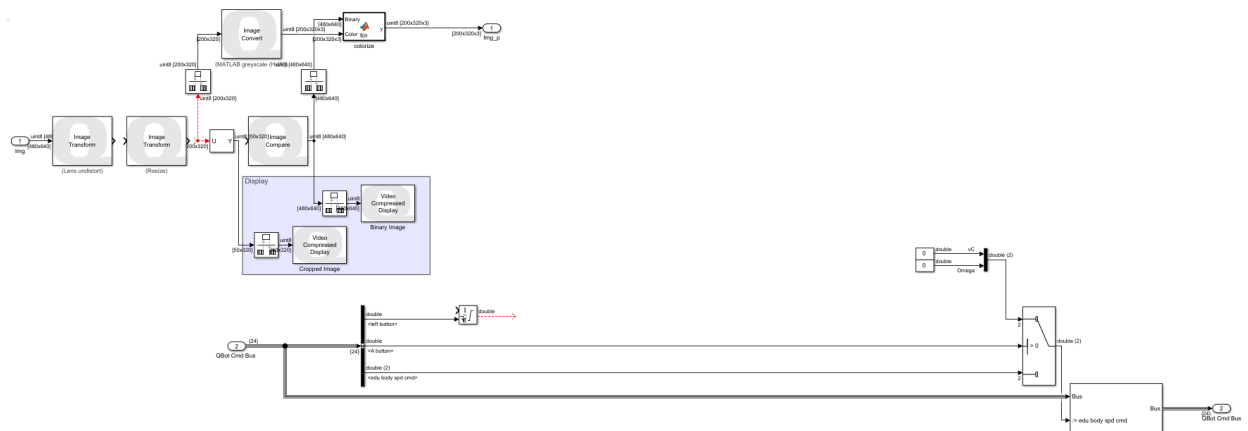

Figure 4. Student Image Processing Block

3. Some blocks have been laid out for you, with the first two **Image Transform** blocks undistorting the wide angle on the downward facing camera as well as resizing it for faster processing.

4. The next block will select the part of the image you are interested in looking at. Open the **selector** block and change the index so that pixels in row 51 to 100 are selected. Your output from that block should be a 50x320 image.

5. The **Image Compare** block will do a comparison to create a binary or bitonal image where the color/area of interest is white (the line), and the rest of the image appears as black.

6. Connect all the blocks up to the **Image Compare**.

7. Click Run ▶ under the Simulation Tab of your model to run the code. When the model is run successfully, the QBot Platform LEDs will turn blue.

8. Use the two **Video Compressed Display** blocks on the purple area to see your input and output image. While the code is running, arm the robot with the Space Bar (LEDs should go green) and move it over a line and straighten the robot such that the line is vertical. As you hover your mouse over the input image, the title ribbon at the top of the window will tell you the RGB and HSV value of each pixel. Particularly, focus on the **Red** value (which applies to grayscale images as well) for the line, as well as the background.

9. Try to tune the parameters in the **Image Compare** block using the values you see in the **cropped image** video display (since you have a grayscale image, you only care about the first range of values under *Primary plane or grayscale image*). Observing the output **binary image**, verify that you are properly highlighting the area of interest. It

should look white while the rest of the image is black. If it is not correct, change the values in the **image compare** block and try again as many times as needed.

10. You should now have properly highlighted your area of interest. Stop the code and delete the purple Display area and all 4 blocks in it. The blocks at the top of this subsystem create the **Downward Camera Image** block shown as part of figure 1. This output will highlight the area of interest as red on the full undistorted original image. Run your code to verify this.

Find Objects of Interest and Line Follow

1. Add an **Image Find Objects** block to find the blob of interest. Set the block settings as follows: `Number of objects` to `1`, `Sort order` to `Sort by number of pixels` and only select the `Find largest objects` in the options. Using your previous reading on connectivity, select the connectivity you see fit as well as a minimum number of pixels.

2. Using the `ctr` output (center of the blob) from the **Image Find Objects** block, find the error representing the difference between the center of the line and the center of the camera feed. Consider the image is 320 pixels wide and you want the distance between the blob center and the image center. Figure 4 will help you visualize this.
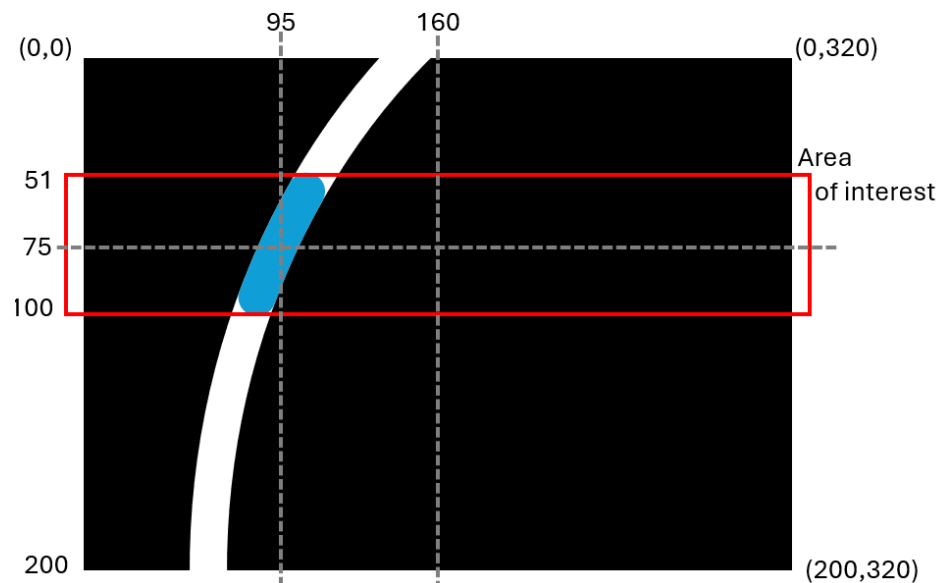

Figure 5. Image with the blob of interest in blue.

3. Find the error angle of the blob center based on the bottom center of the image. Use Figure 5 as a reference to find the correct geometric function to use. The angle error angle will be used for determining the turn speed command for the QBot.

4. Put the error angle through a **Second Order Low-Pass Filter** block to get rid of noise. Set the cut-off frequency to $2\pi * 5$, the damping ratio to 1, and 0 as the initial condition.

5. Set the forward speed, **vC**, as $0.3 * \cos(y)$ where **y** is the output from the **Second-Order Low-Pass** filter block. This will make sure that the speed is inversely proportional to the angle error and the QBot drives faster when there is less error.

6. Add gains with a value of **0** on the **y** and **yd** outputs of the **Second-Order Low-Pass Filter** block. Add these values and use its output as the turn speed input for your QBot. These two gains create a proportional-derivative controller for the robot (also referred to as Visual Servoing). You will tune these gains as your robot moves. Make sure your Omega output does not exceed 1.78 for a maximum angle error.

7. Use your new Omega to compensate for curves when finding the center. The Space Bar should reset values when needed. Use the completed subsystem from Figure 3 as a reference. What does this code do? Take notes.

8. Run the model again. While you keep the QBot armed, drive the robot so it sees the line to follow. Then press and hold the "**7**" key, and your line following code will be enabled.

9. Does your robot turn when it sees a line? Why not? Stop pressing the "**7**" key and using the keyboard, bring the robot back on top of the line.

10. Try increasing the gains after the low-pass filter. Slowly increase these gains to tune your PD controller and test until you like the result. If the robot ever misses the line, let the "**7**" key go, drive over the line again manually and repeat.

11. Observe your line following behaviour and let it run it for a bit. What does the turn speed feedback to the integrator feeding into the error estimate do when following lines turning in different directions?

12. When you are happy with the line following behavior, stop the Simulink model. Ensure that you save a copy of your completed files for review later. Close Quanser Interactive Labs.