Lab Procedure for Python
Obstacle Detection

Setup

1. It is recommended that you review Lab 4 – Application Guide before starting this lab.

2. Turn on the QBot Platform by pressing the power button once. To ensure the robot is ready for the lab, check the following conditions.

   a. The LEDs on the robot base should be solid red.

   b. The LCD should display the battery level. It is recommended that the battery level is over 12.5V.

   c. The Logitech F710 joystick's wireless receiver is connected to the QBot Platform. Before use, **always make sure the switch on top is in the X position and that the LED next to the Mode button is off.**

   d. Make sure your computer is connected to the same network that the QBot Platform is on. If using the provided router, the network should be `Quanser_UVS-5G`.

3. Test connectivity to the QBot. Using the IP displayed in the robot's LCD display, enter the following command in your local computer terminal and hit enter: `ping 192.168.2.x`

4. Open obstacle_detection.py. In Section A change the value of the variable "ipHost" to the IP address of your local Windows machine. To get this IP, open a Command Prompt and enter `ipconfig`. Your IP can be found under IPv4 address and should look like the one the QBot has, `192.168.2.X`.

LiDAR Data Localization

1. In this lab you will be focusing on processing the LiDAR scan data to enable obstacle detection of the QBot. Open obstacle_detection.py. In Section A, right click over **QBotPlatformLidar()** and select "go to definition". This class makes use of the Quanser general **Lidar** object. Right click over **Lidar** and select "go to definition" and take a look at what data is stored in this object. In this lab, only *distances* and *angles* will be used.

2. Go back to the main code, in Section A, right click over **QBPRanging()** and select "go to definition". In this object, the reference frame of the LiDAR data is transferred from the center of the sensor to the center of the QBot. Obstacle detection is also handled

by this object, which we will go through later in this lab. Note: Ensure that Section D – Image Processing has been completed from the previous lab.

3. Notice in **adjust_and_subsample()** under **QBPRanging()**, heading angles from the LiDAR sensor are multiplied by -1 and added to pi/2. This is to correct the headings such that the angle value of 0 corresponds to the front of the QBot, as shown in Figure 1. In addition, the angles are now increasing counterclockwise, consistent with the positive convention of our reference frame. Furthermore, this function returns every 4th element in the distance and angle data, effectively downsampling these signals fourfold. Downsampling is beneficial in the obstacle detection application as it allows for faster computation when high resolution is not necessary.
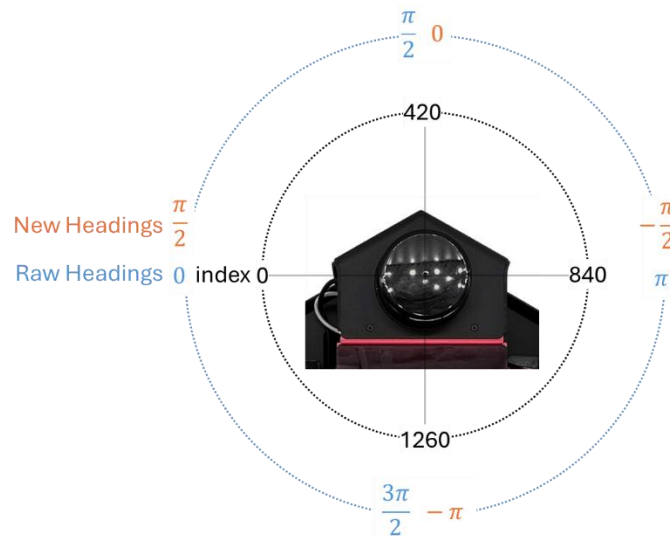

Figure 1. LiDAR measurement correction

4. The **correct_lidar()** function under **QBPRanging()** adjusts the center of LiDAR measurements to the center of the QBot. Use the downsampled range and angle data and the position of LiDAR to complete this function.

5. Use **adjust_and_subsample()** and **correct_lidar()** to finish Section E in obstacle_detecion.py.

Dynamic Monitor Region

1. Go to **detect_obstacle()** function under **QBPRanging()**. This function uses the LiDAR measurement to determine when the QBot should stop to avoid collision with an obstacle. In this lab, you will implement a simple obstacle detection by completing this function.

2. Notice that in addition to the LiDAR measurements, QBot body speed commands are also used as input to the function. The additional inputs allow our obstacle detection

algorithm to dynamically change monitor region and safety threshold, as shown in Figure 2.
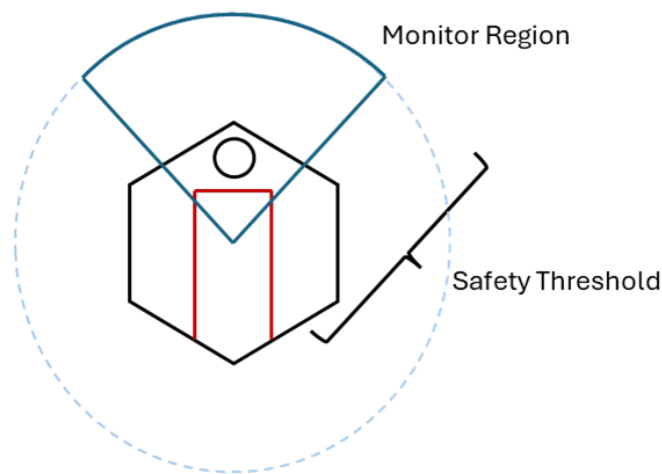

Figure 2. Dynamic Scanning

3. In Section 1 of **detect_obstacle()**, The variable *startingIndex* corresponds to the westmost angles of the monitor region. First, normalize *turnSpd* using the turn speed limit (3.564). Then, derive an equation for *startingIndex* in terms of normalized *turnSpd* and *turnSpeedGain* and complete section 1.

4. Similarly, normalized *forSpd* using the forward speed limit (0.35), then derive an equation for *safetyThreshold* in terms of normalized *forSpd* and *forSpeedGain* and complete section 2.

5. Use **detect_obstacle()** to finish Section F in obstacle_detection.py.

6. Save the change to **QBPRanging()** and go back to obstacle_detection.py. Verify that Section D – Image processing is already completed for you. However, feel free to change the gains in the PD controller or copy over your own functions from the previous lab.

7. Run the code as per the steps in the Running Code section.

8. When observer.py is running, observe the two figures that are opened. They should display the LiDAR scan in the monitor region, as well as a uniform arc representing the safety threshold.

9. Without arming the QBot, move the joystick around. Notice that the monitor region and safety threshold may not be changing with the body speed commands. This is most likely due to *forSpeedGain* and *turnSpeedGain* being too small to produce any noticeable responses. Press the right button (RB) to stop the code.

10. Change the two gains in Section F and re-run the code as per Running Code. Observe how the monitor region and safety threshold dynamically change in response to the varying speed commands. Iterate through this process as much as you need until you are satisfied with the monitor region responses. Press the right button (RB) to stop the code.

Running Code

Note: You will be coding on your local Windows machine and then transfer the code to the QBot Platform. You must go through the steps in this section every time a change is made in the code.

1. Open WinSCP, enter the IP address of the QBot Platform for Hostname, and enter "nvidia" for both Username and Password, then click the login button.

2. In the WinSCP window, transfer required files to run the application:

   a. navigate to "*~\Documents\Quanser\Mobile_Robotics*" and create a new folder "*Lab4*", as shown in Figure 3. Then copy the updated obstacle_detection.py to "*Lab4*". In addition, copy the qbot_platform_driver_physical.rt-linux_qbot_platform over. (You can just drag the files over).
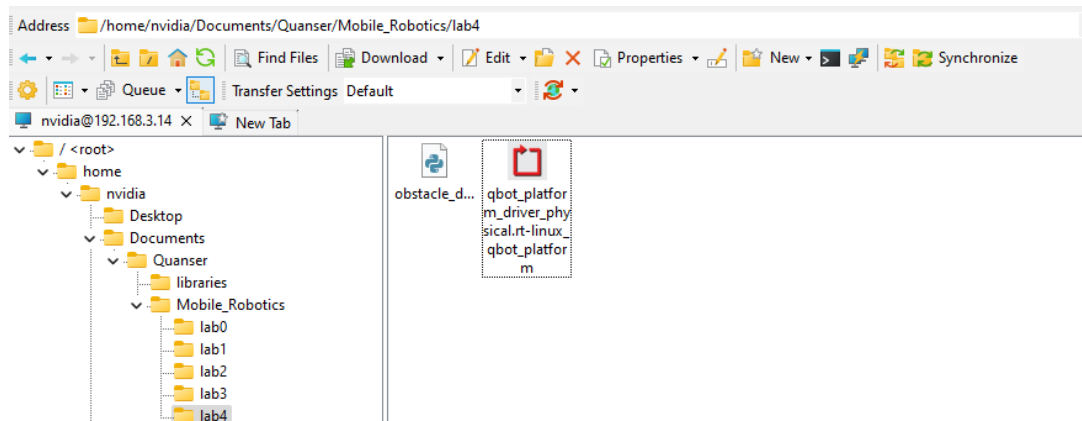

Figure 3. Correct directory for python script

   b. navigate to "~\Documents\Quanser\libraries\python" on the QBot Platform as shown in Figure 4, create these directories if they don't exist. Copy over the "hal" and "pal" folder from your local Windows machine, replacing existing files on QBot.
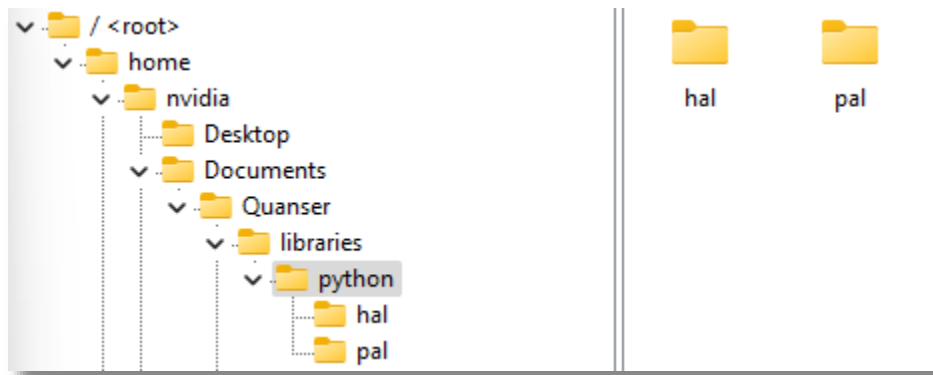
Figure 4. 'hal' and 'pal' folder in correct directory

3. Run observer.py on your local Windows machine first to initiate receiving data feeds from the QBot Platform.

4. Run obstacle_detection.py on QBot Platform:

   a. Open a PuTTY session by clicking ⬛ in the top bars of the WinSCP window.

   b. In the PuTTY terminal, enter the password "nvidia".

   c. Navigate to the python script directory as shown in step 2a by using the `cd` command.

   d. Run the script using the following command:

   ```
   sudo PYTHONPATH=$PYTHONPATH python3 obstacle_detection.py
   ```

   e. When the script is run successfully, User LEDs will turn blue.

Obstacle Detection

1. Go to section F in obstacle_detection.py. Notice that *minThreshold* has already been define for you. This variable represents the radius of the bounding circle of QBot.

2. Go to **detect_obstacle()** function under **QBPRanging()**. In section 3, the total number of points that lays between *minThreshold* and *safetyThreshold* will be computed and compared to *obstacleNumPoints*. When total number of obstacle point exceeds *obstacleNumPoints*, *obstacleFlag* should be set to true. Complete section 3 and save your changes.

3. Use **detect_obstacle()** to complete section F.

4. Run the code as per Running Code.

5. <u>Without arming the QBot</u>, gradually move an obstacle towards the LiDAR sensor to trigger *obstacleFlag*. When an obstacle is detected, the User LED will turn magenta.

6. How sensitive is your obstacle detection algorithm to obstacle? Does it trigger too early or too late?

7. Tune *obstacleNumPoints* until you are satisfied with the obstacle detection response.

8. While arming the QBot, drive it to a line on the mat, and press A button to start line following. Observe the obstacle detection response to different scenarios.

    a. The obstacle is directly on the line.

    b. The obstacle is on the side of the line such that the QBot would narrowly drive pass the obstacle.

9. Does the QBot stop in time to avoid collision? Does the QBot stop when it could have moved pass the obstacle?

10. Can you move the QBot closer to obstacle after *obstacleFlag* is triggered?

11. Tune the *forSpeedGain*, *turnSpeedGain*, and *obstacleNumPoints* until the QBot can response correctly to the two obstacle configurations in step 7.

12. Stop the code when complete by pressing the right button (**RB**). Ensure that you save a copy of your completed files for review later.

13. Turn OFF the robot by single pressing the power button (do not keep it pressed until it turns off). Post shutdown, all the LEDs should be completely OFF.