

QCar 2

User Manual – Software: Python

v 1.0 – 1st Oct 2024

Quanser Consulting Inc.
119 Spy Court
Markham, Ontario
L3R 5H6, Canada
info@quanser.com
Phone : 19059403575
Fax : 19059403576
printed in Markham, Ontario.

This document and the software described in it are provided subject to a license agreement. Neither the software nor this document may be used or copied except as specified under the terms of that license agreement. Quanser Consulting Inc. ("Quanser") grants the following rights: a) The right to reproduce the work, to incorporate the work into one or more collections, and to reproduce the work as incorporated in the collections, b) to create and reproduce adaptations provided reasonable steps are taken to clearly identify the changes that were made to the original work, c) to distribute and publicly perform the work including as incorporated in collections, and d) to distribute and publicly perform adaptations. The above rights may be exercised in all media and formats whether now known or hereafter devised. These rights are granted subject to and limited by the following restrictions: a) You may not exercise any of the rights granted to You in above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation, and b) You must keep intact all copyright notices for the Work and provide the name Quanser for attribution. These restrictions may not be waved without express prior written permission of Quanser.



This equipment is designed to be used for educational and research purposes and is not intended for use by the public. The user is responsible for ensuring that the equipment will be used by technically qualified personnel only.
NOTE: While the GPIO, ethernet and USB ports provides connections for external user devices, users are responsible for certifying any modifications or additions they make to the default configuration.



The Intel RealSense D435 RGB-D camera is classified as a Class 1 Laser Product under the IEC 60825-1, Edition 3 (2014) internationally and EN 60825-1:2014+A11:2021 in Europe. The camera complies with FDA performance standards for laser products except for conformance with IEC 60825-1 Ed. 3 as described in Laser Notice No. 56, dated May 8, 2019. The RPLIDAR A2M12 reaches Class I laser safety standard and complies with 21 CFR 1040.10 and 1040.11 except for deviations pursuant to Laser Notice No. 50, dated June 24, 2007.

Do not power on the product if any external damage is observed. Do not open or modify any portion of any laser product as it may cause the emissions to exceed Class 1. Invisible laser radiation when opened. Do not look directly at the transmitting laser through optical instruments such as a magnifying glass or microscope. Do not update laser product firmware unless instructed by Quanser.

Table of Contents

A. Overview	3
B.1 Development Details	4
File Structure	5
QCar 2 Examples	5
Quanser Modules	6
Python Libraries	7
High-Level Application Libraries (hal)	7
Python Application Libraries (pal)	8
B.2 Application Modules Setup	8
C. Configure Timing	10
D. Deployment and Monitoring	11
E. Troubleshooting Best Practice	12

A. Overview

Quanser's QCar 2 supports application design in multiple programming languages. This guide describes the process of designing applications in Python.

The examples and files provided for the QCar 2 are tested with **Python 3.11.4** in the Windows PC and **Python 3.8.10** on the QCar 2. Users working with the physical QCar 2 can develop applications on their PC and transfer them to the QCar 2 as described in the [User Manual – Connectivity](#) or develop directly on the QCar 2.

Figure 1 below represents the workflow for developing Python applications. The next sections describe each step in more detail.

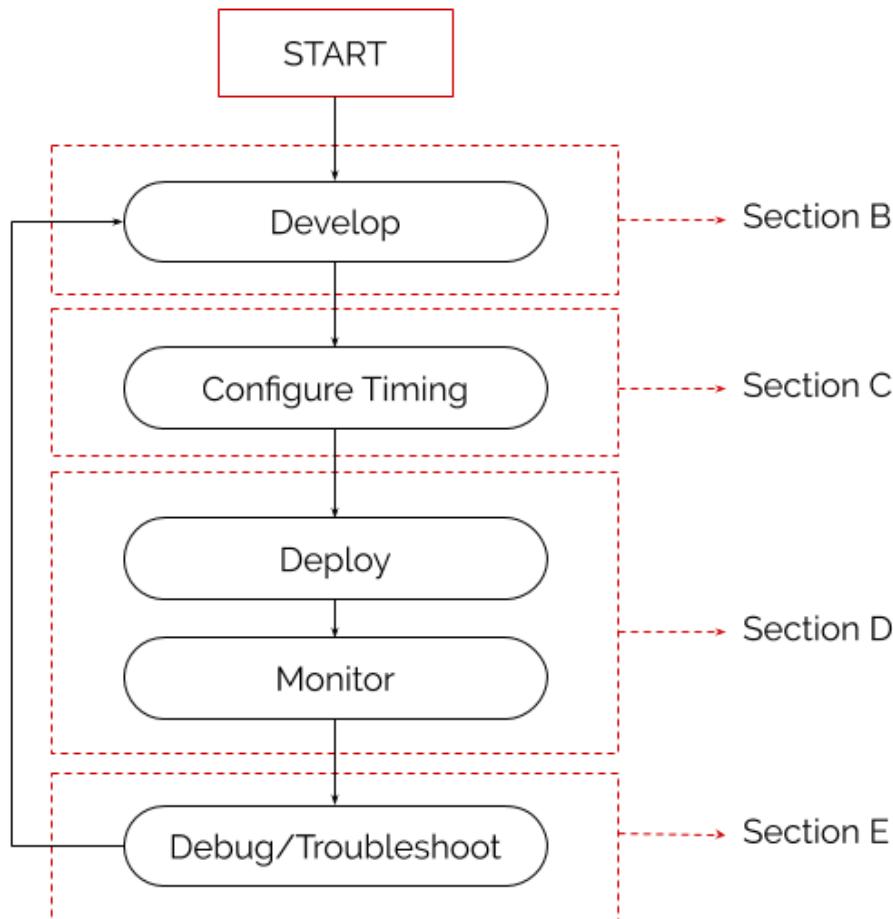


Figure 1. Process diagram for Python code deployment

B.1 Development Details

QCar 2 comes preinstalled with all the necessary packages for running the provided examples and teaching activities. We recommend running commands using **python3** to make sure that Python 3 is being used (**3.8.10**). To see the list of installed packages, connect to the QCar 2, either remotely or directly, and use the following commands in the terminal.

To see the python packages installed with pip, use:

```
nvidia@qcar2-*****:~$ python3 -m pip list
```

To view the complete list of packages installed from Ubuntu, open a terminal window and type the following command:

```
nvidia@qcar2-*****:~$ apt list --installed | more
```

The Ground Control Station (GCS) provided with the Self-Driving Car Studio (SDCS) also comes equipped with Python (**3.11.4**) and all necessary modules needed to run the provided examples and activities. On the GCS use the following command in a command prompt to view the installed packages.

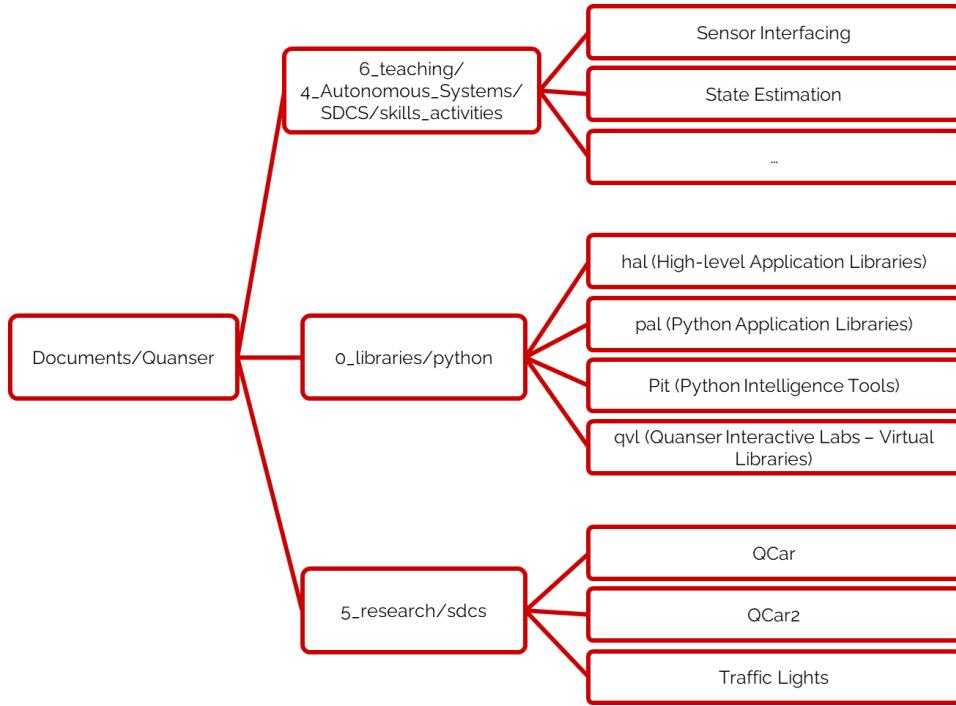
```
C:\...\> python -m pip list
```

If a different PC is being setup, use the resource downloader from our website and follow the instructions. The setup script ensures that an appropriate python version (**3.11.4**) and the necessary python packages are installed, that includes both the Quanser python packages as well as any other dependencies. If you are setting up with multiple installed python versions, you may be prompted to specify the primary python version. Refer to the Python documentation for more information [here](#).

If you need to update any Debian or Python packages in your QCar2, please read the Software section in the [User Manual – Customizing QCar 2](#) or refer to the QCar 2 QUARC documentation that highlights which packages are installed and which ones should not be updated to avoid issues with QCar 2 <https://docs.quanser.com/quarc/documentation/qcar2.html#software> .

File Structure

The main file structure that comes preinstalled in the GCS or on your PC after the resource's installation should look something like this:



This guide will explain examples, libraries and general setup as some of these files will need to be copied to the QCar. For running skills activities, read through this guide and then go to `Documents/Quanser/1_setup/self_driving_car_studio` for the instructor and student guides.

QCar 2 Examples

The supplied GCS or the PC with the installed resources will have examples and hardware tests to help start development. These examples will make use of the Quanser Modules, and the Python libraries described in the next subsections. The process described in section B.2 to set up libraries and examples in QCar 2 will have to be followed.

Under `Documents/Quanser/5_research/sdcs/qcar2/hardware` there are three folders, `hardware_tests`, `applications` and `ros2`.

The `hardware_tests` folder includes examples in both Python and Simulink for testing the IO of the car as well as LiDAR and cameras of the system. To run them, refer to section D. Deployment and Monitoring.

The `applications` folder includes different completed examples to start using QCar 2. These examples include lane following, manual driving as well as applications for LiDAR or Cameras.

If using the teaching resources, you will find 6 skills activities in the following directory: `Documents/Quanser/6_teaching/4_Autonomous_Systems/SDCS/skills_activities`, which guide students through different aspects of Self-Driving.

Quanser Modules

Both the supplied PC (and any PC being used with the downloaded resources), as well as the QCar 2 have Quanser modules installed that are used for interactions with hardware. These packages include,

- `quanser-api`
- `quanser-common`
- `quanser-communications`
- `quanser-devices`
- `quanser-hardware`
- `quanser-multimedia`

To update these packages on QCar 2, use the Software Updater in the vehicle. The instructions are described on [User Manual – Customizing QCar 2](#) section D for software packages.

If you update QUARC or Quanser SDK on the PC, the python packages will need to be updated too. To do this, open a file explorer and type `%QSDK_DIR%python` as shown in figure 2, this will take you to the location of the Python libraries.

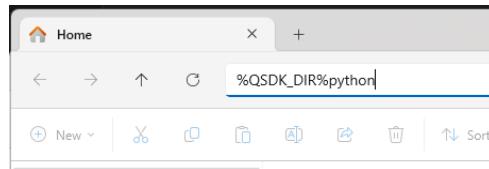


Figure 2. Python libraries in the PC

In that folder, first double click `uninstall_quanser_python_api.bat` to uninstall current libraries. Afterwards, double click on `install_quanser_python_api.bat`, this will ensure the newest Quanser Python libraries are installed.

If the PC being used has multiple Python versions installed and the version you are using to develop is not the one that the system references when you call `c:\...\> python`, you will need to modify the .bat file to run using the correct python version. See the Python documentation on this [here](#). Otherwise, you can delete the other Python versions if not current

Python Libraries

Wrappers around the Quanser Python libraries have been created to simplify the use of the QCar 2. These are all based on the Quanser Modules described in the above sections.

On your GCS, or on your PC after installing the resources, you should have these libraries under `Documents/Quanser/0_libraries/python`. If you have a GCS or downloaded the research resources, you may have files for more Quanser products, however, this guide will focus on the necessary files for running QCar 2 which are shown in figure 3. The pit library will be available with examples nonspecific to QCar, see `examples/pit` for more information.



```
python > hal > content > qcar_functions.py > ...
11
12 > class QCarEKF: ...
148
149 > class QCarDriveController: ...
221
```



```
python > hal > products > qcar.py > ...
/
8 > class QCarGeometry(MobileRobotGeometry): ...
84
```

```
python > pal > products > qcar.py > ...
20
29 > class QCar(): ...
425
426 > class QCarCameras: ...
518
519 > class QCarLidar(Lidar): ...
666
667 > class QCarRealSense(Camera3D): ...
772
773 > class QCarGPS: ...
```

Figure 3. Python libraries

High-Level Application Libraries (hal)

The `hal` library includes our higher-level python libraries, equipped with a list of Python functions commonly used throughout the provided content. Refer to figure 3 for the file structure in this library.

The hal library includes 3 folders: content, products and utilities. This guide will describe the files related to QCar 2 or things necessary to interface with it.

The file **hal/content/qcar_functions.py** includes two classes

- QCarEKF that is specific for the QCar and is built on top of the generic EKF class found under **hal/utilities/estimation.py**.
- QCarDriveController which is used in the skills activities for learning Self-Driving.

The file **hal/products/qcar.py** includes a single class, QCarGeometry that has the frames of reference for important positions in the QCar.

The folder **hal/utilities** folder contains generic high-level application which can be used in any application, available utilities are shown in figure 3.

Python Application Libraries (pal)

The **pal** library is a python application library which makes use of the Quanser Modules. These are intended to give users the ability to interface with hardware on the QCar 2.

The two folders that will be used are products and utilities.

The file **pal/products/qcar.py** was designed to give users the ability to interact with the standard set of IO and sensors in the QCar from a single location. This library is built on top of the generic classes found under **pal/utilities**. Specific classes can be seen in figure 3.

The folder **pal/utilities** has standard utilities that can be used for added flexibility if the sensor/peripheral is not defined in **pal/products/qcar.py**. Available utilities are shown in figure 3. These utilities include things like data streaming, using a gamepad or adding probes.

B.2 Application Modules Setup

To be able to run the provided examples for QCar2, these Python libraries need to be transferred to the car. The following instructions describe how to transfer examples and resources, as well as the python libraries **hal** and **pal** on the QCar 2:

1. Make a directory on the QCar for Quanser resources where files will be transferred to and from. We recommend creating a folder called Quanser under Documents. The following directory should now exist: `/home/nvidia/Documents/Quanser`. To connect to the QCar2 and create this directory, we recommend using WinSCP, as shown in figure 4. (see the [User Manual - Connectivity](#) for how to remotely connect to the QCar).
2. Using WinSCP, on your PC side, navigate to `Documents/Quanser`. On the QCar 2 side, navigate to your new `Quanser` folder. Your WinSCP window should look like figure 4. From your PC side, copy/drag the `o_libraries`, `5_research` and `6_teaching` folders over to your new Quanser folder in the QCar2.

Note: the `libraries` folder may have already existed in the QCar2, in this case, copy over only the subdirectories of `o_libraries` from your PC to `/home/nvidia/Documents/Quanser/libraries` in the QCar2.

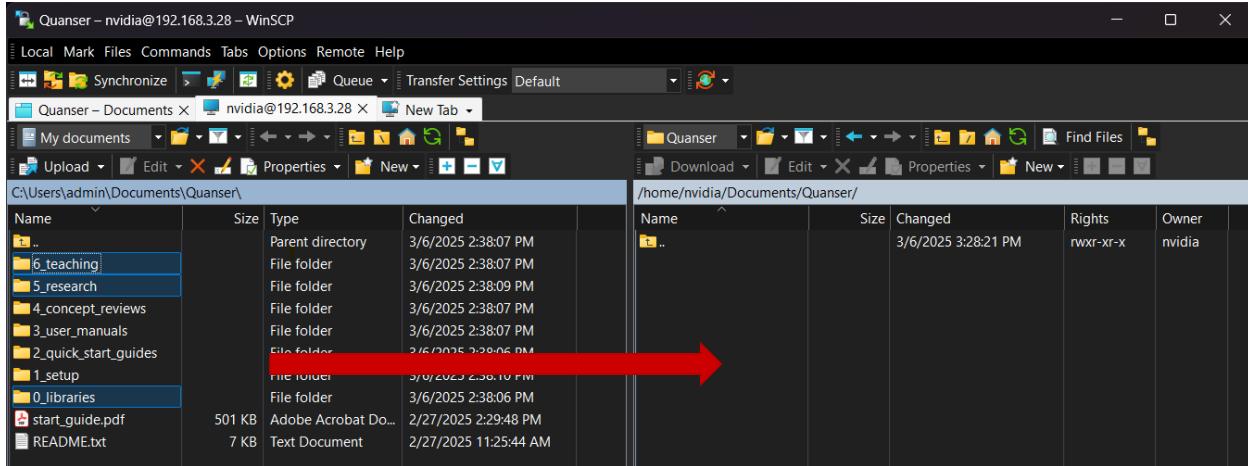


Figure 4. Transferring files to QCar

3. Open `~/.bashrc` file on your Qcar2. To do this using WinSCP, navigate to the `/home/nvidia` folder (two up from the Quanser folder), and click **Ctrl+Alt+H**, this will show the hidden files in the QCar2. You can double click the `.bashrc` file to edit it in the notepad. To do this on the desktop of the QCar 2 using an HDMI connection or remote desktop, go to the nvidia folder in the desktop, click **Ctrl+H**, in the file explorer to show hidden files and double click the `.bashrc` file to edit.
4. Include the following lines at the beginning of the `~/.bashrc` file on the QCar 2 after the existing exports.

```
export PYTHONPATH=$PYTHONPATH:"<PATH TO directory of hal and pal>"  
export QAL_DIR="<PATH TO Quanser Resources>"
```

As an example, if the contents of the `src` folder were copied over to the Quanser folder as shown in figure 5, the environment variables on your QCar 2 should be defined as shown in figure 6:

```
export PYTHONPATH=$PYTHONPATH:/home/nvidia/Documents/Quanser/libraries/python  
export QAL_DIR=/home/nvidia/Documents/Quanser
```

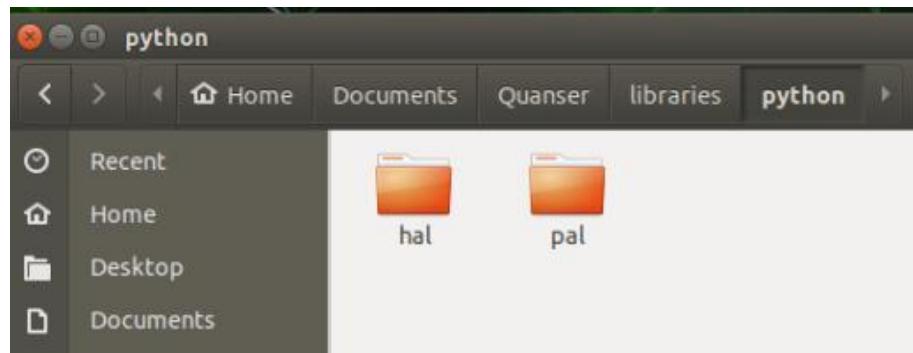


Figure 5. Example folder structure on QCar

```

1 # ~/.bashrc: executed by bash(1) for non-login shells.
2 # see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
3 # for examples
4
5 export PATH=/usr/local/cuda/bin:$PATH
6 export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
7 export PYTHONPATH=$PYTHONPATH:/home/nvidia/Documents/Quanser/libraries/python
8 export QAL_DIR=/home/nvidia/Documents/Quanser

```

Figure 6. `~/.bashrc` file with new environment variables

C. Configure Timing

It is important to maintain a consistent sample rate for real-time applications. Given a sample time, all code in a single iteration must be executed in a time window that is less than the required sample time. In cases where the execution of an iteration is completed in less than the sample time, it is also essential that the next iteration does not begin until a full unit of the sample time has elapsed.

For example, consider an image analysis task that must be executed at 60 Hz, corresponding to a '**sample time**' of 16.7 ms (1/60). If the time taken to execute the analysis code, also referred to as the '**computation time**', is less than the sample time, say 10 ms, then it is important to wait an additional 6.7ms at each time step before proceeding to the next iteration. On the other hand, if the computation time is greater than the sample time, say 20ms, then the sample time cannot be met. In such cases it may be essential to lower the sample rate or increase the sample time, to say 40Hz (25 ms). Note that the **time** module's **time()** method returns the current hardware clock's timestamp in seconds.

In Python, code is executed as fast as possible, and a wait can be inserted using the **time** module's **sleep()** method or the **opencv** module's **waitkey()** method for imaging applications. The following snippet provides a detailed example on how to accomplish this.

```

import time

# Define the timestamp of the hardware clock at this instant
startTime = time.time()

# Define a method that returns the time elapsed since startTime was defined
def elapsed_time():
    return time.time() - startTime

# Define sample time starting from the rate
sampleRate = 100 # Hertz
sampleTime = 1/sampleRate # Seconds

# Total time to execute this application in seconds.
simulationTime = 5.0

# Refresh the startTime to ensure you start counting just before the main loop
startTime = time.time()

```

```

# Execute main loop until the elapsed_time has crossed simulationTime
while elapsed_time < simulationTime:
    # Measured the current timestamp
    start = elapsed_time()

    # All your code goes here ...

    # Measure the last timestamp
    end = elapsed_time()

    # Calculate the computation time of your code per iteration
    computationTime = end - start

    # If the computationTime is greater than or equal
    # to sampleTime, proceed onto next step
    if computationTime < sampleTime:
        # sleep for the remaining time left in this iteration
        time.sleep(sampleTime - computationTime)

```

D. Deployment and Monitoring

When **developing** code for QCar 2 there are two main methods:

- Using the GCS to develop code and then transfer it to the QCar 2
- Writing code directly on QCar 2.

To **run** python code on the QCar, you could connect remotely to the QCar and run code through a terminal or connect directly to the QCar and run the code. Note that QCar 2 does not have python2, and therefore the command **python** and **python3** point to the same python installation.

When ready to run python code:

1. All the examples available will require the use of **python**.
2. To run an application, use the following syntax:

```
python <application name>.py
```

As an example, to run a QCar hardware test, if your terminal is in the correct folder, you could run:

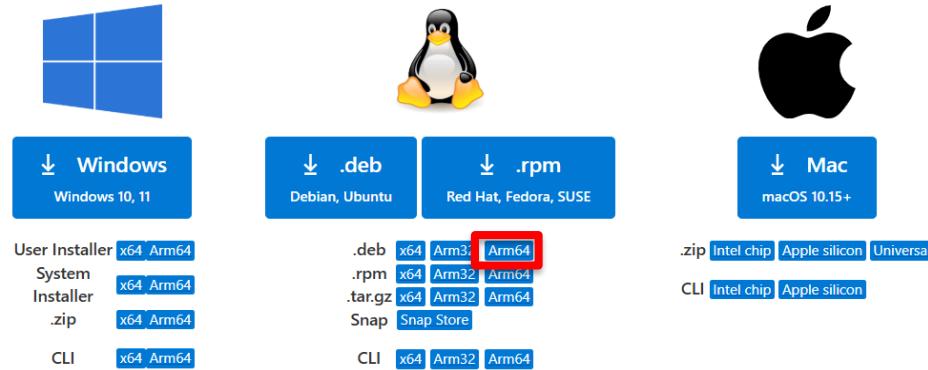
```
python QCar2_hardware_test_basic_io.py
```

You can connect directly to the QCar or remote into it as described in [User Manual – Connectivity](#) to be able to see the output of the applications if needed.

For QCar2, it is recommended to connect to the QCar remotely via Remote Desktop and develop directly on the QCar using **VS code**. However, it is not pre-installed in the QCar. To install, first download the Debian package from [Visual Studio Code](#), and select the "Arm64" option of the Debian package, as shown below.

Download Visual Studio Code

Free and built on open source. Integrated Git, debugging and extensions.



Then, to install, navigate to the directory where the Debian package is downloaded to and run the following command in the terminal:

```
sudo apt install ./<file>.deb
```

E. Troubleshooting Best Practice

In order to ease debugging during application development, we use the **try/except/finally** structure to catch exceptions that otherwise terminate the application unexpectedly. Most of our methods in the Quanser library have this structure built in. After configuration and initialization, scripts begin with **try**. If an unexpected error arises, it will be captured by the **except** section instead. This can ensure that code in the **finally** section still gets executed and the application ends gracefully.

For example, if you specify an incorrect channel number for HIL I/O, a **HILError** will be raised. However, you still want to call the **terminate()** method to close access to the HIL board, without which, opening it on the next script call may fail.

If your code does not terminate properly and you have problems running your code again due to the 'card cannot be opened' error. We recommend turning the car off and on. Make sure the code can catch an error and call terminate properly.

```
## Main Loop
try:
    while elapsed_time() < simulationTime:
        # Start timing this iteration
        start = time.time()

        # Basic IO - write motor commands
        mtr_cmd = np.array([ 0.2*np.sin(elapsed_time()*2*np.pi/5),
                            -0.5*np.sin(elapsed_time()*2*np.pi/5)])
```

```

LEDs = np.array([0, 1, 0, 1, 0, 1, 0, 1])

current, batteryVoltage, encoderCounts = myCar.read_write_std(mtr_cmd, LEDs)

# End timing this iteration
end = time.time()

# Calculate computation time, and the time that the thread should pause/sleep for
computation_time = end - start
sleep_time = sampleTime - computation_time%sampleTime

# Pause/sleep and print out the current timestamp
time.sleep(sleep_time)
print('Simulation Timestamp :', elapsed_time(), ' s, battery is at :', 100 - (12.6
-batteryVoltage)*100/(2.1), ' %.')
counter += 1

except KeyboardInterrupt:
    print("User interrupted!")

finally:
    myCar.terminate()

```

© Quanser Inc., All rights reserved.



Solutions for teaching and research. Made in Canada.