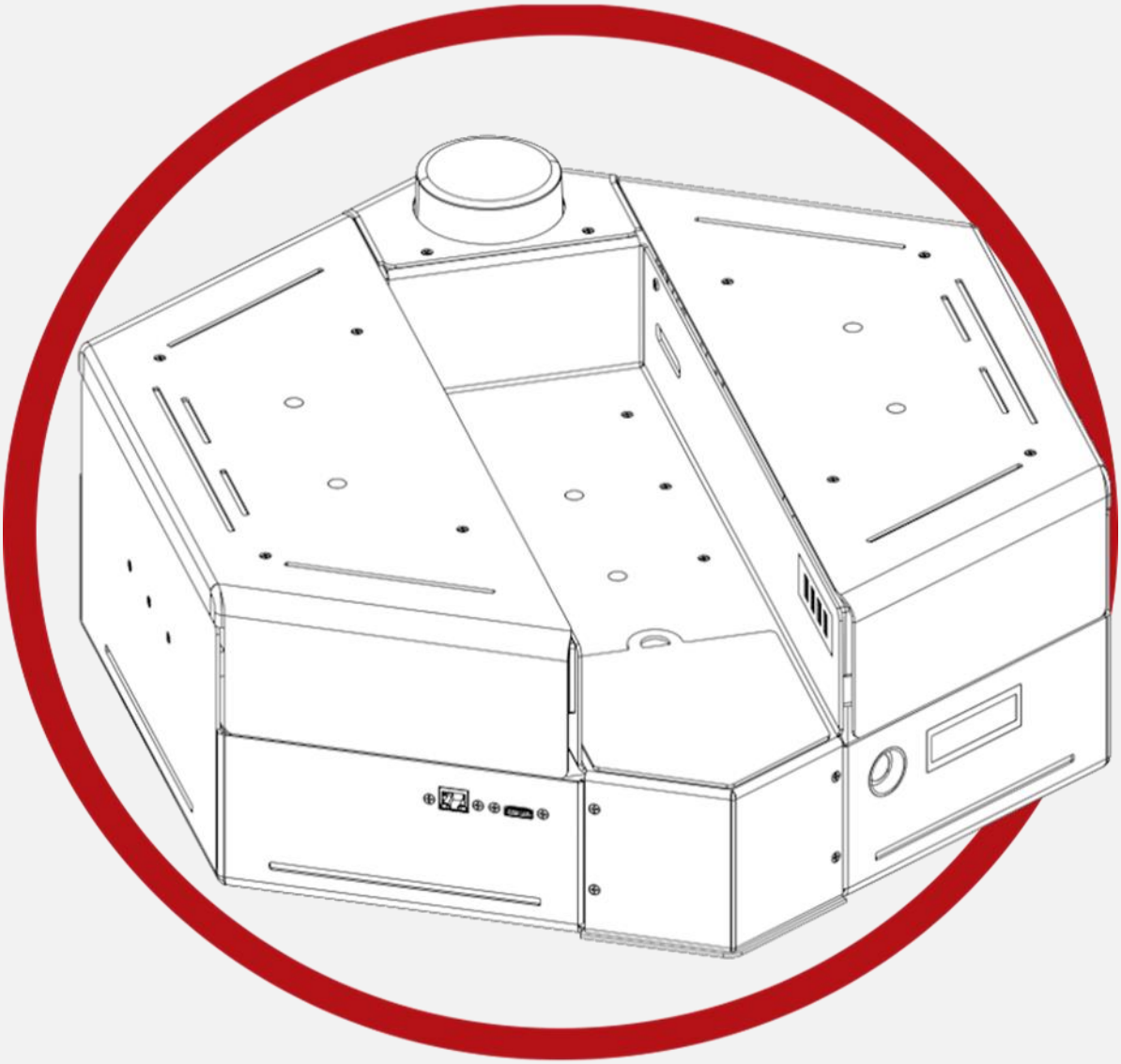
QUANSER
INNOVATE·EDUCATE



QBot Platform

Software User Manual - Python

v 1.0 — 29th Feb 2024

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

Quanser Inc.
119 Spy Court
Markham, Ontario
L3R 5H6, Canada

info@quanser.com
Phone : 19059403575
Fax : 19059403576
printed in Markham, Ontario.

Waste Electrical and Electronic Equipment (WEEE)

This symbol indicates that waste products must be disposed of separately from municipal household waste, according to Directive 2002/96/EC of the European Parliament and the Council on waste electrical and electronic equipment (WEEE). All products at the end of their life cycle must be sent to a WEEE collection and recycling center. Proper WEEE disposal reduces the environmental impact and the risk to human health due to potentially hazardous substances used in such equipment. Your cooperation in proper WEEE disposal will contribute to the effective usage of natural resources.

Caution

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 to ensure that the equipment will be used by technically qualified personnel only. While the end-effector board provides connections for external user devices, users are responsible for certifying any modifications or additions they make to the default configuration.

Table of Contents

A. Overview

The overall process is described in Figure 1 below. Design your application as you see fit for Python 3. The examples provided are tested with **Python 3.10.4** for the Windows host PC and **Python 3.8.10** on the QBot Platform. The general development flow is described in Figure 1. Seconds B through E cover useful information on development in python 3, configuring timing, deploying your code to the QBot Platform, and debugging common errors. For more details, see the corresponding sections.

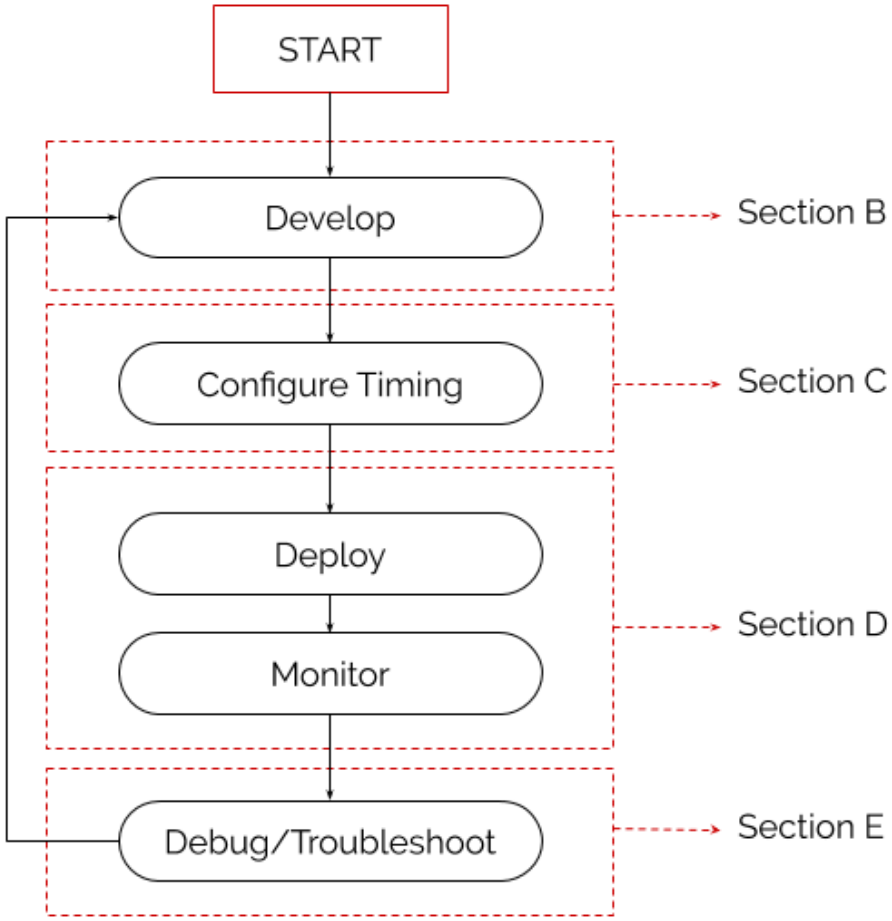


Figure 1. Process diagram for Python code deployment

B. Development Details

Ensure that all the modules required by your application are installed in the location where the script will be deployed. The **QBot Platform** comes equipped with numerous modules already installed. On your Windows development PC, the setup installer included with your resources (setup.bat) will automatically install python packages for you as well. Any additional packages you require can be installed manually as you see fit using the python package manager **pip**. On your computer, use the following command in a command prompt to see what packages you currently have available after installation.

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

Note that the QBot Platform has python 3 installed by default.

In a terminal on the **QBot Platform** (using a direct connection or PuTTY terminal if remote) use the following command to look at pre-installed packages, See the Connectivity User Manual for more information on using PuTTY or SSH.

```
nvidia@qbp-XXXXX:~$ python3 -m pip list
```

i. Quanser Modules

After installation your PC and the QBot Platform should also have Quanser modules installed that are used for additional interactions with hardware, devices and stream. On your PC, these packages are installed by the setup installer included with your resources (setup.bat):

quanser-api	2024.x.x
quanser-common	2024.x.x
quanser-communications	2024.x.x
quanser-devices	2024.x.x
quanser-hardware	2024.x.x
quanser-multimedia	2024.x.x

On the QBot Platform, these packages are installed via apt package manager, and updated through it as well. To list the currently installed packages from Quanser, run the following command,

```
nvidia@qbp-XXXXX:~$ apt list | grep quanser
```

To update QUARC runtime on the QBot, run the following commands,

```
nvidia@qbp-XXXXX:~$ sudo apt update
nvidia@qbp-XXXXX:~$ sudo apt upgrade
```

Note: The password for sudo operations is nvidia. A valid internet connection is required. Consider connecting the QBot Platform to a Wi-Fi network with internet connectivity or use a LAN wired connection directly.

The provided **applications**, **teaching content** and **examples** use these packages. To get more information on these packages click here.

If you have manually updated your version of Quanser SDK on your machine, the python packages must be updated as well. This can be done manually by reinstalling the new packages located under the following directory. Use the following commands:

```
C:\...\> cd "C:\Program files\Quanser\Quanser SDK\python"
C:\...\> dir
```

Typing `dir` will indicate the date needed for the next command:

```
C:\...\> python -m pip install --upgrade --find-links . quanser_api-<date>-py2.py3-none-
       any.whl
```

where `<date>` is the date for the API being installed. For example:

```
   sudo python3 -m pip3 install --upgrade --find-links . quanser_api-2022.2.2-py2.py3-none-
                                                any.whl
```

Note: You do not need the exact date. In the command above, you can type quanser_api and hit the Tab key to automatically fill out the rest of the command. Installing quanser_api installs the other five packages as well. The terminal window should indicate that all existing packages were successfully uninstalled, then the new packages were installed.

ii. Quanser Application Libraries

In addition to the **examples** and **teaching content**, the **QBot Platform** also comes with application libraries that are divided into two, high-level (hal) and lower level (pal).

High-Level Application Libraries (hal)

The higher-level python libraries are equipped with a list of python functions commonly used throughout the provided examples and courseware. These **hal** libraries are designed to provide a starting point for developing research and also include solutions for the teaching content. These are not intended to be shared with students as it contains code solutions. Student versions of **hal** require students to complete code as they progress through labs.

There are three sub packages inside of **hal**:

- products
- content
- utilities

Generic classes can be found in the **utilities** folder for PID control, state estimation etc. The **content** directory includes `qbot_platform_functions.py`, which is used by the teaching content, and holds classes related to forward and inverse differential kinematics, image processing and lidar processing.

Python Application Libraries (pal)

The lower-level python libraries, make use of the Quanser Modules and are equipped with a list of python functions commonly used throughout the provided examples and courseware.

These are intended to give users the ability to interface with the hardware on the QBot Platform. Unlike **hal** however, this library is intended to be given to students as a starting place for the provided courseware. Just like in **hal**, pal also contains two sub packages:

- products
- utilities

The **products** folder contains a class for the QBot Platform called `qbot_platform.py` which is a specific implementation of the generic classes found in the **utilities** folder. The `qbot_platform.py` file was designed to give users the ability to interact with the standard set of sensors in the QBot Platform from a single location.

If the sensor/peripheral is not defined in the `qbot_platform.py`, the standard utilities can be used for added flexibility. These additional utilities include libraries for using a gamepad, lidar, math utilities, scoping, probing, stream, and vision.

iii. Application Modules Setup

To make use of hal and pal (or include future updates) on the QBot Platform:

1. Make a common directory on the QBot Platform where libraries will be transferred to and from. We suggest for simplicity to put them under Documents/Quanser/ to match where libraries are located on your PC. See the User Manual – Connectivity for how to remotely connect to the QBot Platform and transfer files.
2. The following line has automatically been included at the end of the ~/.bashrc folder on the QBot Platform.

```
export PYTHONPATH=”/home/nvidia/Documents/Quanser/libraries/python”
```

3. Transfer the contents of libraries from your Windows machine to the QBot, as well as your application files using appropriate directory structures. See Section D for running.

iv. Driver Model

Along with the libraries, the QBot Platform also comes with a driver model. The driver model has been created for safety purposes with driving the QBot Platform. We strongly encourage all user applications to go through the driver model for safety purposes. There are several different inputs that get sent to driver model.

Send packet message:

The message format to the drive application includes the following data structure. Use a stream client at 60Hz connected to the server running on the following URL: `tcpip://IP_ADDRESS_OF_QBOT:18888`. The packet structure contains 10 doubles:

Function	Doubles (float64)	Details
`Mode`	1	0 or 1 (teaching mode) | 2 or 3 (research mode) (see Vel Cmd)
`Enable User LED`	1	Enable color override
`User LED color`	3	Color value
`Arm`	1	Arm the motors
`Hold`	1	Not implemented yet
`Vel Cmd`	2	Forward/turn speed (mode 0 or 2) (m/s, rad/s) left/right wheel speeds (mode 1 or 3) (rad/s, rad/s)
`Timestamp`	1	Timestamp signal for loopback

Receive packet message:

The QBot Platform driver also returns a 17 double data packet with useful information. This is summarized below. Use the same stream client to receive this data at 60Hz.

Function	Doubles (float64)	Details
`Wheel position`	2	Wheel encoder positions (in rads)
`Wheel speeds`	2	Wheel tachometer speeds (in rad/s)
`Cmd Voltage`	2	Voltage commanded by onboard controllers (in Volts)
`Accelerometer`	3	Tri-axis Accelerometer data (m/s/s)
`Gyroscope`	3	Tri-axis Gyroscope data (rad/s)
`Current`	2	Left/right motor current draw from the battery (Amps)
`Battery level`	1	Active battery level (Volts)
`Watchdog status`	1	Status information regarding watchdog expiry. If expired, re-arm the QBot Platform to resume function.
`Timestamp`	1	Loopback timestamp signal returned by the driver

Built in Driver LEDs:

The driver and QBot Platform also have some LED colors built in to let you know when the QBot Platform is in different states.

Running the driver model will initially change the QBot Platform lights to pulse on and off white. This shows that your driver model is running, and you can now run your own application to connect to the driver model. If you connect a model to the robot, the lights will turn blue. This shows that the driver application is connected to a client and waiting for commands (but the robot is not yet armed). Arming the robot in this state will cause the lights to turn green, Setting the hold command to 1 in this state will cause the lights to flash warning you that this feature is not yet implemented in the driver.

You can override the blue, green or white pulse colors with your own input by setting the color variable to a RGB color of choice and setting the Enable User LED command to 1.

If your battery is too low, the lights will pulse purple to let you know you need to charge your robot. If your lights pulse yellow, it means you've caused a stall or overcurrent condition. Simply disarm and rearm the robot to continue.

Finally, if your robots' lights are red or flashing red, these states are set by the firmware on the robot. Solid red lights show the robot is on, or the application has stopped normally without any error conditions. Red flashing lights show that the embedded computer has shut down and the robot must be power cycled to work again.

Color	State	Level	Description
White	Pulse	Model	Driver application is healthy and waiting for an incoming client connection
Blue	Solid	Model	Driver application is connected to a client and waiting for commands (the robot is not armed).
Green	Solid	Model	Driver application has armed the robot and motor controllers are active.
Green	Pulse	Model	Driver application has armed the robot and has received the hold command (not implemented).
Other	N/A	Model	Driver application is applying a user LED color.
Yellow	Pulse	Model	Motor overcurrent or stall being detected (this is not an error but indicates strain)
Magenta	Pulse	Model	Low battery warning (this is not an error but indicates that you should stop the model soon and change the batteries).
Red	Solid	Firmware	Driver application was stopped normally without any error conditions, or no application has been deployed yet.
Red	Pulse	Firmware	Embedded Computer has shut down and the robot must be turned OFF using the power switch.

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 not begun 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 or 25 ms. Note that the **time** module's **time()** method returns the current hardware clock's timestamp in seconds.

In Python, the 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.

```python
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

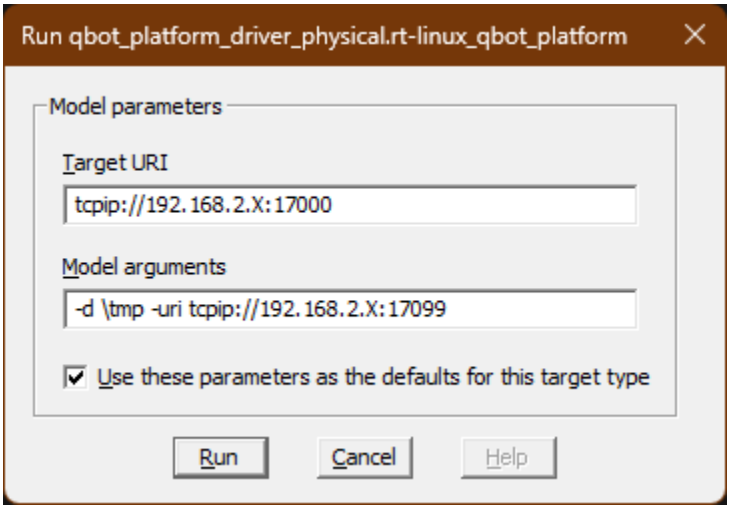
When **developing** code for the QBot Platform there are two main methods:

- Using a personal computer (PC) or a Quanser Ground Control Station (GCS) to develop code, and moving files to the QBot for testing and execution.
- Writing code directly on the QBot Platform.

We recommend that you choose the former so that you run your code on Quanser Interactive Labs first to test for bugs, before proceeding to running the code on hardware. However, to **run** python code on the QBot Platform the two methods stated above are both valid.

When ready to run python code:

1. The QBot Platform has python3 installed. All the examples available will require the use of **python3**.
2. All examples provided also require you to first start the driver model before starting the application. To run the driver model on the physical hardware, right-click on the driver file in your directory, click on 'show more options', followed by 'Run on target'. Use the following settings and hit Run.



3. To run an application in the terminal on the physical QBot Platoform, use the following syntax:

```
sudo PYTHONPATH=$PYTHONPATH python3 <application name>.py
```

As an example, to run the QBot Platform forward kinematics lab file:

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

Troubleshooting Best Practice

To ease debugging during application development, we use the **try/except/finally** structure or the **with** function 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. The **with** function is like the try/catch/finally but simplifies the code when possible. For example, if you specify an incorrect channel number for HIL I/O, a **HILError** will be raised, which can be appropriately handled.

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

            # Basic IO - write motor commands

            # Your code goes here

            # 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)
            counter += 1

except KeyboardInterrupt:
    print("User interrupted!")

finally:
    myQBot.terminate()
```

Solutions for teaching and research. Made in Canada.