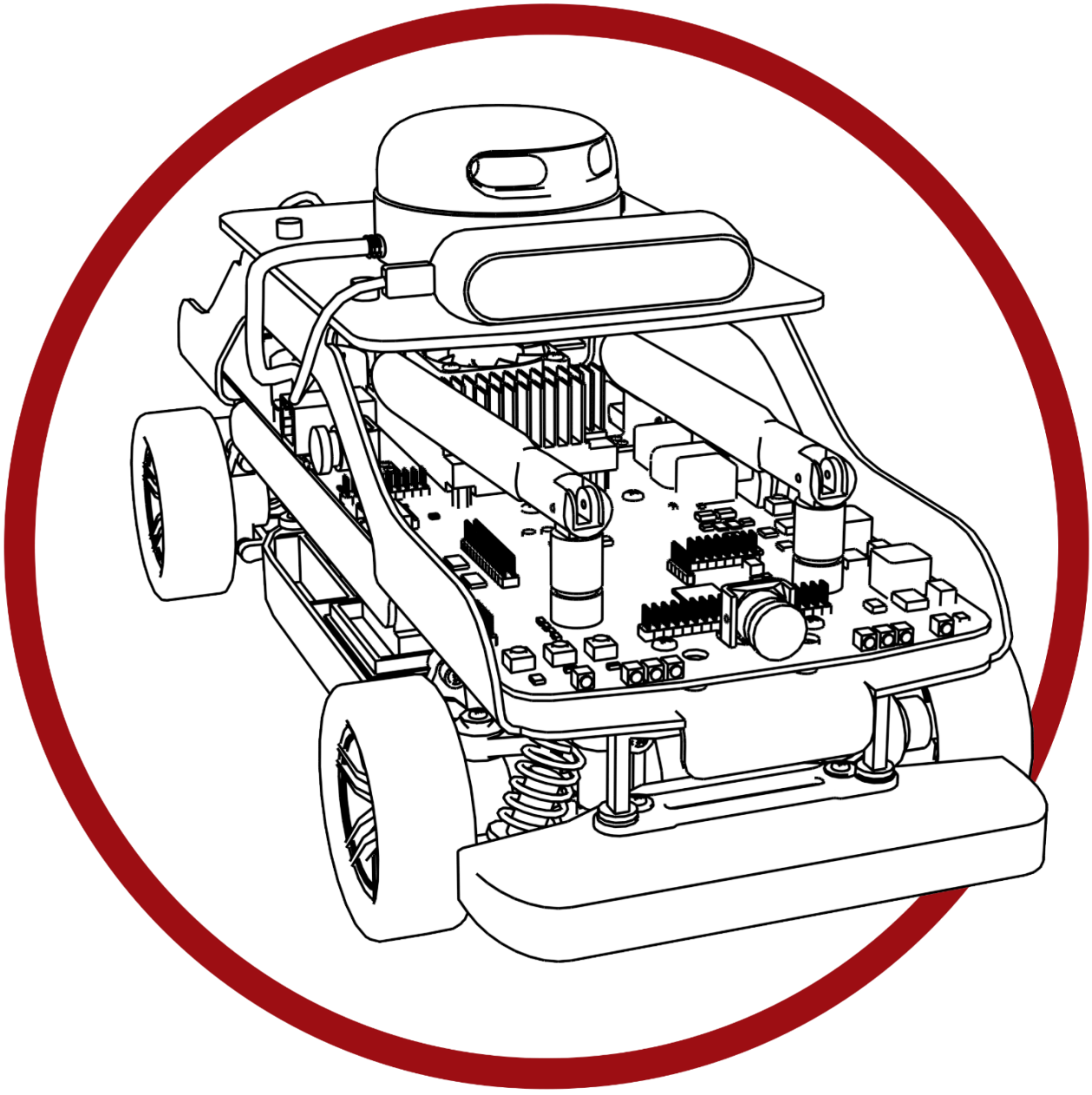
QUANSER
INNOVATE·EDUCATE

Self-Driving Car Research Studio



Manual Drive - Python

V 1.1 (November 2020)

Table of Contents

I. System Description

In this example, we will capture commands from a Gamepad and use it to manually drive the QCar platform. The application will also calculate the car's speed from encoder counts. The process is shown in Figure 1.

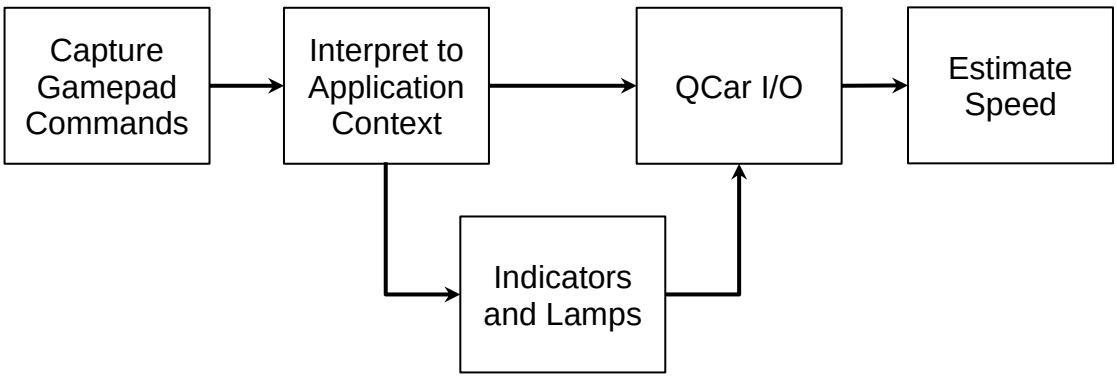


Figure 1. Component diagram

II. Running the example

Check the user guide **V - Software - Python** for details on deploying python scripts to the QCar as applications.

Once you have set the controller number correctly for the gamepad (see Python Hardware Test documentation for details), run the script using a **sudo** flag. There are two ways to drive the car. If users set the configuration to **3** in the script, the Gamepad's **right stick** longitudinal axis is used for the throttle, the **left stick** lateral axis is used for steering, and the **LB** button is used to arm the QCar. If users set the configuration to **4** in the script, the **right trigger RT** is used to provide positive throttle in the forward/reverse directions based on the state of the **button A**. please check **Manual Drive** documentation - **Simulink** for more details.

III. Details

1. Encoder counts to linear speed

   The **q_interpretation** module contains the method **basic_speed_estimation** to convert from encoder speed (counts/s) to linear speed (m/s) based on the QCar's differential and spur parameters. However, the HIL I/O provides encoder counts, which must first be differentiated. This is accomplished by using the **differentiator_variable** method within the **Calculus** class of the **q_misc** library provided. Initialized with the **sampleTime**, you can set the actual step size within the main loop when calling the differentiator, accounting for variable sample time.

   Note: Don't forget to initialize the differentiator using the **next** method!

```
# Set up a differentiator to get encoderSpeed from encoderCounts
diff = Calculus().differentiator_variable(sampleTime)
_ = next(diff)
timeStep = sampleTime

# Inside main while loop
   encoderSpeed = diff.send((encoderCounts, timeStep))
```

2. Performance considerations

   We run the example at 50 Hz. The **os** module is used here to clear the terminal screen whenever new gamepad updates are received. Although this is expensive and isn't the best thing to do, we account for the slower sample rates by using the variable sample time differentiator instead of a static one. Without accounting for the variable sample times, the differentiator will underestimate sample times and thereby overestimate the speed. Comment out the system screen clear as well as the print statement (similar to the snippet below) to improve performance up to 500 Hz.

```
if new:
    os.system('clear')
    print(...)
```