Lab Guide

Manual Drive

Content Description

The following document describes a Manual Drive implementation in either python or MATLAB software environments utilizing the virtual QCar.

Prior to starting the example please go to the **Cityscape Lite** workspace and run the **qlabs_setup_applications.py** python script to configure the virtual world. The MATLAB example will use the user keyboard to drive the virtual QCar while the python example will utilize the Logitech gamepad which comes with the SDCS studio.

MATLAB

In this example, we will capture commands from a Gamepad and use it to manually drive the QCar platform. The application will also display the percentage battery remaining, power consumption in Watts as well as the car's speed in m/s. The process is shown in Figure 1.

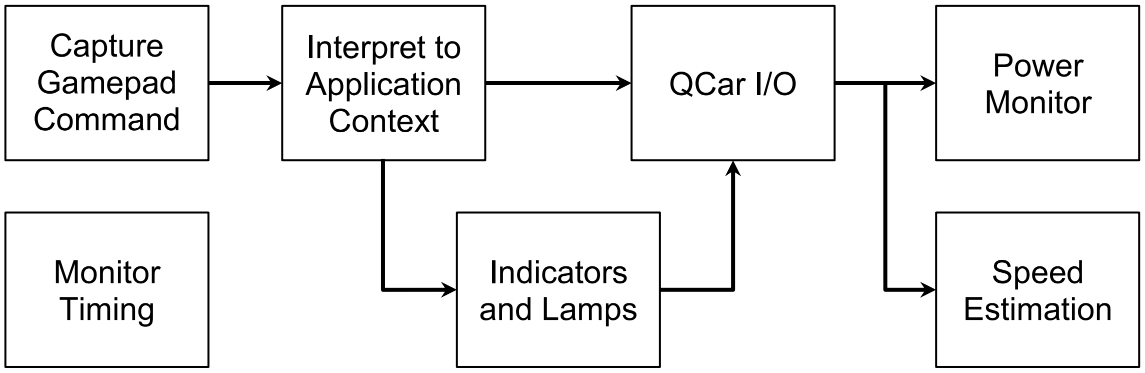
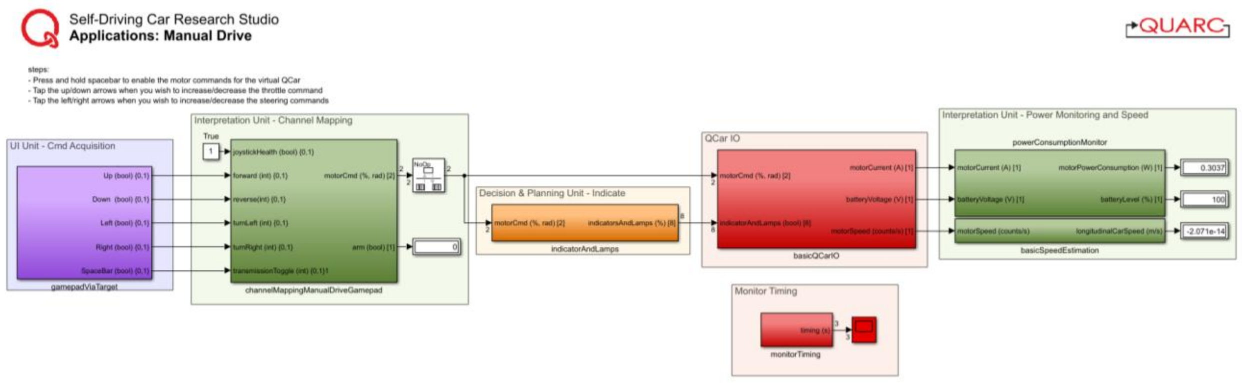


Figure 1. Component diagram

In addition, a timing module will be monitoring the entire application's performance. The Simulink implementation is displayed in Figure 2 below.

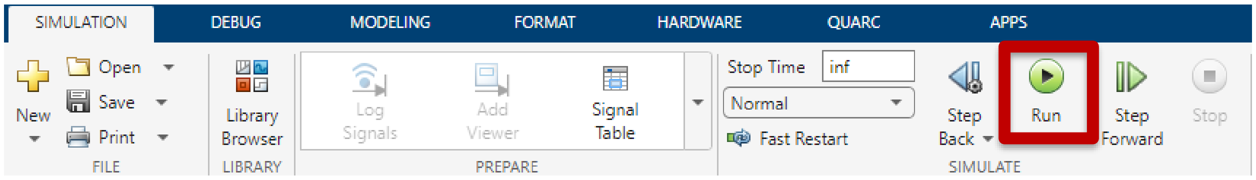

Self-Driving Car Research Studio
Applications: Manual Drive

steps:
- Press and hold spacebar to enable the motor commands for the virtual QCar
- Tap the up/down arrows when you wish to increase/decrease the throttle command
- Tap the left/right arrows when you wish to increase/decrease the steering commands

Figure 2. Simulink implementation of Manual Drive

Running the example

To run examples for virtual QCar please go to the **SIMULATION** tab in the ribbon interface and click on the Run icon.


Stop Time: inf
Normal

Guide

1. Driving manually is mapped to the following keyboard commands:

   a. Press and hold spacebar to enable the motor commands for the virtual QCar
   b. Tap the up/down arrows when you wish to increase/decrease the throttle command
   c. Tap the left/right arrows when you wish to increase/decrease the steering commands

2. The LEDs are in the following states

   a. **Headlamps** are always on. The reverse indicators (white) are on in Reverse.
   b. **Brake lamps** are on when the absolute speed of the vehicle is decreasing.
   c. **Left/right indicators** turn on when the corresponding steering is over a threshold

Python

The application will also calculate the car's speed from encoder counts. The process is shown in Figure 3.

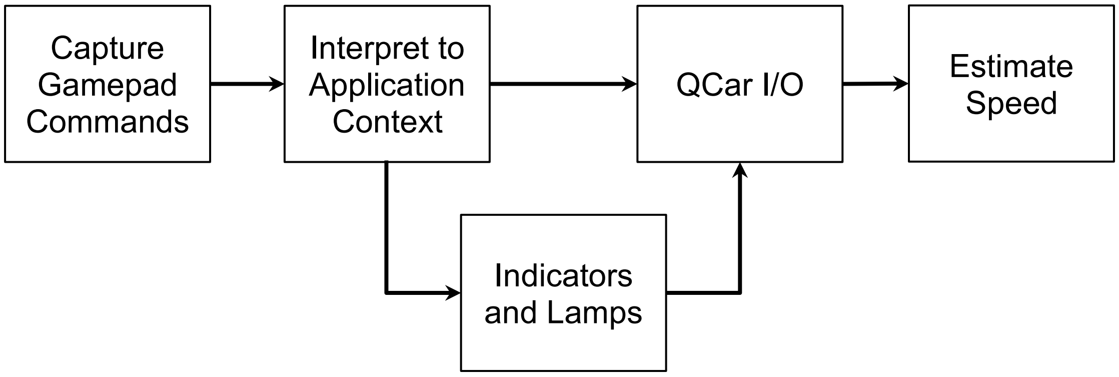


Figure 3. Component diagram

Running the example

1. Check User Manual – Software Python for details on deploying python applications. Run the **task_manual_drive.py** example on your local machine. Please make sure the Logitech gamepad is connected to the host computer prior to running the example.
2. 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**.

Note: If the steering does not appear to work, please ensure the **mode** light on the gamepad is **off**.

Guide

1. Encoder counts to linear speed

   The **pal.utilities.math** module contains the **differentiator_variable** method within the **Calculus** class. This module is used to calculate counts/s as the virtual QCar drives in the environment. Initialized with the **sampleTime**, you can set the actual step size within the main loop when calling the differentiator, accounting for variable sample time. The constant **CPS_TO_MPS** lets us convert the count/s value to meter/s used to estimate the linear speed of the Virtual QCar.

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 ==============
## Estimate linear speed in m/s
encoderCounts = myCar.motorEncoder # Get encoder data from DAQ
encoderSpeed = diff.send((encoderCounts[0], timeStep))
# Convert from  counts/s to meter/s
linearSpeed  = encoderSpeed*myCar.CPS_TO_MPS
```

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('cls')
    print(...)
```