CARLA Leave-Over Documentation

Hunter White

April 2024

1 Overview

This document is intended to serve as a resource for getting started with the "open-source driving simulator for autonomous driving research" known as Car Learning to Act CARLA. It includes a description of CARLA, the steps taken to install and configure CARLA on both lab computers, and the scripts, terminal commands, and workflow for interacting with CARLA that may be useful. A significant effort is made to include solutions to errors that may be encountered during the installation and initialization processes, as well as links to Stack-Overflow/Linux Forums/CARLA Documentation pages that proved useful in the troubleshooting and error correction processes.

2 Creating the Simulation Environment

2.1 Environment setup and package installation

2.1.1 Package and Environment Managers

Definitely Review For Consistency
Anaconda is used as the package and environment manager. Anaconda Navigator vastly simplifies the package and environment management problem by providing a relatively simple, if a little bloated, graphical user interface for managing virtual environments, with shortcuts to launch common programs with the conda environment activated. This proved especially useful when initially exploring the best (read: easiest to integrate with existing machine learning research on GitHub) version of python and CARLA to work with. Most packages can be installed via conda or pip, although some require one or the other. A short comparison between conda and pip is given on the conda blog Special care should be taken to install all possible packages with one installer and then the other if necessary, as conda does not communicate with pip and vice-versa. A getting started guide for conda is available here. When installing packages with conda, it will be useful to add the conda-forge channel. A guide on how to do that is provided here. Note: Occasionally, updating conda before installing all desired packages will result in a deadlocked environment. Unfortunately, the only way to fix the environment is to delete it and reinstall all desired packages

in a new environment. Generally, I never updated conda unless I was about to create a new environment or start a new project.

2.1.2 Required Packages

Be sure to update pip and conda before creating a new environment and downloading packages for CARLA or the provided code.
To run the provided code, the following packages and python version are/is necessary:

Package	Version	requirements filepath
CARLA	0.9.8	N/A
Python	3.7.*	N/A
numpy	1.21.1	(your-carla-filespath)/carla/PythonAPI/examples
pygame	2.1.2	(your-carla-filespath)/carla/PythonAPI/examples
matplotlib	*	(your-carla-filespath)/carla/PythonAPI/examples
open3d	*	(your-carla-filespath)/carla/PythonAPI/examples
pillow	9.4.0	(your-carla-filespath)/carla/PythonAPI/examples
future	0.18.3	(your-carla-filespath)/carla/PythonAPI/examples
networkx	*	(your-carla-filespath)/carla/PythonAPI/carla
distro	*	(your-carla-filespath)/carla/PythonAPI/carla
Shapely	1.7.*	(your-carla-filespath)/carla/PythonAPI/carla
psutil	*	(your-carla-filespath)/carla/PythonAPI/util
py-cpuinfo	*	(your-carla-filespath)/carla/PythonAPI/util
python-tr	*	(your-carla-filespath)/carla/PythonAPI/util
poetry	1.3.2	(your-repository-path)/requirements.txt

Table 1: Required packages for CARLA, CARLA examples, and the provided code

To install all of the above packages, open your prefered command line interface (cmd or powershell for windows, terminal for linux), and activate your conda environment. You can then install the packages in either of the following ways:

1. (Option 1: Individually) All required packages can be installed by navigating to each

   ```
   requirements.txt
   ```

   within your local copy of the GitHub Repository and running the command

   ```
   pip install -r requirements.txt
   ```

   or

```
conda install --yes --file requirements.txt
```

in the command prompt with the desired conda environment activated.

2. (Option 2: Grouped) A single

   ```
   requirements.txt
   ```

   file is provided for easier installation with pip. This is installed the same way as described in Option 1. Alternatively, the conda equivalent

   ```
   <environment_name>.yml
   ```

   file is also provided. To install the

   ```
   <environment_name>.yml
   ```

   file, use the command

   ```
   conda env create -f <environment-name>.yml
   ```

   in the command prompt or terminal.

3. (Note on Poetry) Poetry is included as an alternate package manager, and was used by Idrees Razak when creating the PPO agent and variational autoencoder that I used as a foundation. Poetry's main function in the repository is to download the legacy versions of PyTorch and CUDA that were used to train the PPO agent and VAE. To install these with Poetry, navigate to the Poetry folder using the command prompt or powershell, then run the command

   ```
   poetry update
   ```

   As a disclaimer, I don't think this step is necessary if you would rather install the correct PyTorch and CUDA packages using pip or conda. The legacy versions can be found on the PyTorch website

If you try option 2 and find that packages are missing, please try option 1.

3 How to Download and Install CARLA

NOTE: CARLA requires a dedicated GPU for with at least 6GB of VRAM. This excludes most consumer laptops. More VRAM does not necessarily make CARLA run faster, but it does generally make CARLA more stable when adding large numbers of actors. CARLA also recommends having an addition GPU for any machine learning, although this is not necessary if your GPU has the capacity. (i.e.

12 GB of VRAM and from a recent generation of Vulkan compatible GPUs) CARLA is fairly easy to install, and the process is nearly identical for Windows and Linux (Ubuntu). Because we do not need the additional functionality gained by building CARLA from source, we can follow the quick getting started guide. **NOTE: There are different versions of documentation that correspond to each released version of CARLA. Make sure you use the documentation version that corresponds to the CARLA version you intend to use.** Be sure to make sure that pip and/or conda are up to date before creating the environment.

4 Verifying CARLA installation

CARLA provides a number of examples that are useful for checking that CARLA and its required packages were installed correctly. These provided examples are also useful for understanding the CARLA Python API, and give some ideas on how to structure code that will interact with CARLA. The latest version of the Python API Reference can be found here.

4.1 Opening a CARLA server instance: Windows

To open a carla instance on Windows, first open the terminal by pressing "ctrl+x" and selecting terminal. Depending on your windows preferences and configuration, this will open either a powershell instance or a command promt instance. For our purposes, they are largely interchangeable, although powershell may be more familiar if you have previous experience. In your terminal, nvaigate to the CARLA folder by using the change directory function, abbreviated cd.
I have extracted CARLA to

```
C:\WindowsNoEditor
```

.
insert command line figure showing cd to carla directory. If you are having trouble navigating to your CARLA directory, you can copy the path to your directory by finding the directory in file explorer and right clicking on the "TOP MIDDLE WINDOW THINGY?" and selecting copy address as text. You can then paste the address into your terminal after the call to change directory. Alternatively, you can navigate through your files and directories by using a combination of ls, which lists the files and directories within the current directory, and cd.

To start an instance of CARLA after navigating to the CARLA directory, simply run the command

```
.\/CARLAUE4.exe
```

in the terminal. A black window should pop up on the screen. After a few moments, the window should load to the default town for your version of CARLA. In CARLA 0.9.8, it should look like this:
Insert default carla window here for 0.9.8

You can use the WASDQE keys to navigate the spectator camera around the default map. There's not much to see, as we haven't loaded in vehicles, pedestrians, etc. (a.k.a. "Actors"), but this will give you a feel for how smoothly CARLA runs on your device.

4.2 Opening a CARLA server instance: Linux (Ubuntu 20.04)

TO BE WRITTEN

4.3 running an example script

To add in some actors, we will need to open an additional terminal window, and make sure the conda environment is activated. The easiest way to do this is to install the powershell/command prompt shortcut in anaconda navigator, or by typing conda activaite ¡your CARLA environment name¿. You will see the name of your environment in parenthesis in your terminal prompt if you have successfully activated your environment. Next, navigate to the Python API examples directory within your CARLA directory using the terminal window with your activated conda environment. the examples directory should have a path similar to :

```
<your_carla_directory>\PythonAPI\.examples
```

from here, running any of the example scripts should work. My personal favorite is

```
manual_control.py
```

, which allows the user to manually control a vehicle using the keyboard. This example, and the corresponding

```
manual_control_steeringwheel.py
```

, are also exceptionally useful for understanding how to integrate PyGame and outside control commands into a CARLA environment. Experimenting with the example scripts is a great way to understand how to interact with CARLA and can serve as a great template for writting your own scripts. Other scripts included in the

```
<your_carla_directory>\PythonAPI\.carla\agents\navigation
```

directory can be useful for understanding the basics of path planning and route creation.

5 Driving Simulator

The steering wheel and pedals have been modified from a Logitech G920 Driving Force Racing Wheel and Pedals for Xbox. These modifications and the manufactured base for the wheel and pedals were created by a previous senior design team. Their accompanying documentation can be found as a .zip file here. **NOTE: I have changed the microcontroller from the Atmel ATSAMC21N18A to an Arduino Uno Rev3**. I had issues understanding and using the Atmel microcontroller, but you are welcome to reintegrate it if you so desire. The rest of the user manual and parts lists/files are unchanged and are still valid.

5.1 Arduino Configuration and Code Walkthrough

The Arduino code for interfacing between Python, CARLA, and the Driving Simulator can be found here.
To give a quick overview: The script begins by defining the pins connecting the arduino and the motor controllers. It is important to make sure that these pins are correct, and the connections are good. Max displacement values are assigned based on an initial calibration. When changing the length of string attaching the pedals to the motors, the settings on the motor controllers, or after any catastrophic failure type event, these values should be re-confirmed. in setup(), we assign pins as outputs, and set their initial states. We open the serial port corresponding to the Arduino board and set its parameters. This is also how we interact with python. The loop and accompanying functions generally do the following:

1. Print the current displacement of the wheel and pedals if it has been longer than printInterval milliseconds since the last check. This is not guaranteed to be the true position, and the system should be recalibrated to zero before interfacing with CARLA/Python.

2. Checks if there is information waiting in the input buffer on the serial port corresponding to the Arduino Uno.

3. If there is data available on the buffer, read the buffer until the newline character, Otherwise, return to (1)

4. Parse the data returned: seperate the read data into 3 values, seperated by a comma.

5. Constrain the returned values between and their respective minimum and maximum values

6. Check the current state of each direction pin, and change it if necessary.

7. While any of the [APP,BPP,SW] are not equal to the desired position, pulse the corresponding stepPins until all [APP,BPP,SW] are at desired position

8. Return to (1) This loop generally runs very fast, and is generally limited by the response time of the motor controller. setting delayMicros() too small leads to some awful sounds, and the incorrect end position of the motor. To require fewer pulses per degree of rotation, consider modifying the motor controller settings instead of or in addition to reducing the pulse delay.