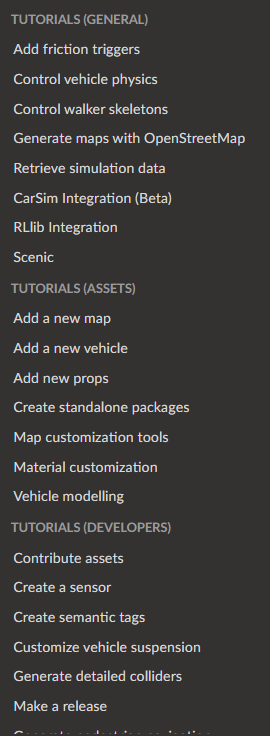
Final Project Proposal

DGMD E-17 Robotics, Autonomous Vehicles, Drones, and Artificial Intelligence

Bruno Janota and James Milliman

# Introduction

Carla is an open-source autonomous driving simulator. It is custom built from the ground up with a modular and accessible API that can be leveraged to work on autonomous driving solutions. That said, the documentation, which can be found at <https://carla.readthedocs.io/en/latest/start_introduction/> is far from comprehensive for applying what we learned in DGMD E-17 to the simulator. The documentation covers CARLA and its concepts for open-source developers, often discussing creating custom sensors, creating new vehicle types, updating the physics engine, adding maps, etc. Even though all of this content is necessary for CARLA to grow, the simple procedure of spawning a vehicle that moves around the environment and collects camera data is not covered in these documents. The documentation focuses on editing and improving the simulator, and the tutorials often are on topics tangential to the use case that would most support a course such as DGMD E-17.



The above image shows a list of the tutorials for CARLA as of its current release. All of these tutorials besides 1, Retrieve Simulation Data, would be out of the scope for vehicular computer vision and control, and lean into asset creation, system development, and physics control.

CARLA is a budding and exciting open-source project that represents an exciting public playground for researchers and hobbyists to jointly develop, test, and improve autonomous driving vehicles, agents, and sensors. The downside of CARLA’s heavy development and documentation is that there is a lot more to the scope of the project than the self-driving agents, there is a framework, assets, and physics engines that also are being heavily worked. That said, the core simulator of CARLA could be an asset to any course similar to DGMD E-17, but there is a barrier to entry given the lack of documentation. Many of CARLA’s secrets are hidden in the source code and example files that can be scavenged on the internet. The goal of our project, JBNav, is to tear down those barriers and show how CARLA can be weaved into the curriculum of DGMD E-17 and used to train self-driving agents, as well as a playground to further test various computer vision and control pipelines.

With that said the goal of our revised project (which we will still call JBNav because it’s a fun name), is to create a framework that leverages all of the various techniques we have learned throughout the course, lane detection, opencv2, object detection, and bring it into the CARLA universe, with revised code examples, libraries, and documentation. The JBNav project will have multiple milestones, each of which build upon the assignments and projects from DGMD E-17 and bring them into the CARLA universe in a reusable way.

JBNav will bring examples and documentation for future courses to leverage and build upon further for leveraging CARLA vehicles and sensors for capturing road level images for custom annotation. Lane line and object detection, recording videos of car maneuvers, and imitation learning will be discussed and demonstrated.

## 1.1 Goal of the Project

JBNav aims to gap-fill the shortcomings and barriers presented by CARLA’s nascent documentation and instructions on how to develop and apply computer vision pipelines and self-driving agents. CARLA represents an amazing supplement to any self-driving course, and the JBNav team envisions using CARLA as another test case, alongside images and video, to deploy and collect results of various pipelines. For example, Assignment 5 covered advanced lane finding, using a more traditional form of computer vision to find and annotate road lane lines in various road conditions, given images and video. Spawning up a simulation and collecting road annotated images from CARLA should be as fast and easy of a process as running the pipeline on a given video. This would add excitement to the course and assignment problems, as well as highlight shortcomings of solutions, since a CARLA environment has large complexity such as intersections, pedestrians, stop signs, etc. JBNav will deliver a code package full of documentation and examples for how to apply pipelines from previous assignments to a CARLA simulation, to be reused and built upon further.

In addition, once CARLA is reverse engineered and applied to DGMD E-17 pipelines, clean documentation and example code/librarie are developed, initial imitation learning pipelines and agents will be developed and deployed. These agents will attempt to learn from images collected from the CARLA simulation to emulate the preset agents CARLA has programmed to navigate the world map.

Overall JBNav will have multiple milestones and deliverables:

1. Create Example code/libraries for running project pipelines in CARLA and collect results
2. Allow that example code to be given parameters to vary CARLA features, such as lighting, time of day, map, etc.
3. Create documentation for using the above code and libraries that show how to take assignment outputs and run them on carla, how to install carla, etc.
4. Collect examples of running assignments on CARLA and document difficulties and shortcomings, as well as attempt to improve
5. (Extra) Imitation learning on CARLA using deep neural networks (CNN)

## 1.2 Level of Autonomy

For many of the earlier milestones, CARLA will control the vehicle and the JBNav project will be attaching to the output of various camera sensors and applying DGMD E-17 created computer vision pipelines to collect and annotate data in real time.

Once those milestones are completed imitation learning will show a pipeline that takes data from sensors, applies computer vision techniques to understand the environment as well as to make control level decisions, and will show the loop of an autonomous level 4 vehicle. That said the CARLA environment is complex, with intersections, stop signs, pedestrians, and more, and the imitation learning system will most likely have difficulty dealing with all of the complexity of the environment. That said, even with some difficulty the example of applying a computer vision pipeline in realtime to the sensors, saving off annotated sensor data, and making control decisions in a well documented way should lead to an easy project to build upon in future courses.

## 1.3 Description

JBNav aims to lower the barrier of entry for autonomous vehicle students and hobbyists to deploy computer vision solutions to CARLA. CARLA’s fast-paced development and research papers make a simulated environment that would supplement real-world collected video and image data.

Overall there are two types of systems JBNav aims to seamlessly integrate with CARLA. The first is a computer vision system that does not try to control an environment, but does want to attempt to make sense of the world. This could be the lane detection algorithms from earlier in the course, through to image segmentation or neural networks attempting to highlight information in the simulation. JBNav will capture data from CARLA and pass it to the python pipeline, ultimately saving a video of the captured frames. These types of systems would like to know ground-truth data from the simulation, i.e. if their detections are correct, and this may be able to be part of JBNav but may require considerable tinkering with CARLA’s environment. Part of JBNav’s deliverable will be the assignment’s from the course running on CARLA streamed data. The second CV system JBNav wants to more easily integrate with is a computer vision system that also wants to control the vehicle. We will call this an agent and will begin to tinker with agents, from simple agents CARLA provides to imitation learning agents we train and/or can find publicly available.

## 1.4 System overview

CARLA currently has a large number of modules and components that a new user would have to worry about. The documentation covers everything from spawning a world, types of vehicles, physics, extensions, rendering, traffic management, sensor suites, and much much more. That said, a student of DGMD E-17 or a computer vision course would not need to know about most of these features in order to begin to enjoy and use CARLA. These users would simply want a camera feed from a car inside the simulation to apply their pipelines to, and potentially a simple interface for agent creation and usage.

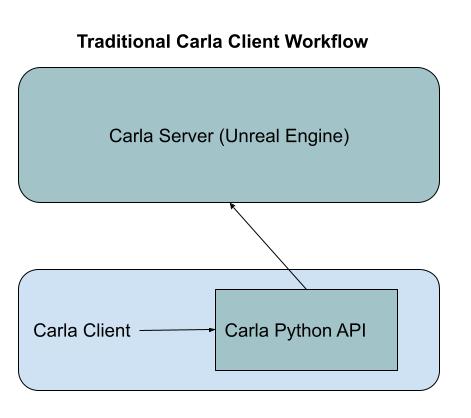


Figure 1: Traditional CARLA workflow

In Figure 1 shown above is the traditional CARLA client workflow using CARLA. A Python API exposes CARLA functionality to a client, that is used to tell the server: what maps to load, time of day, what vehicles to spawn, what sensors to create, what vehicle to control, where to place sensor data, and more. CARLA comes with a lot of power but also comes with a lot of upfront cost to get rolling.

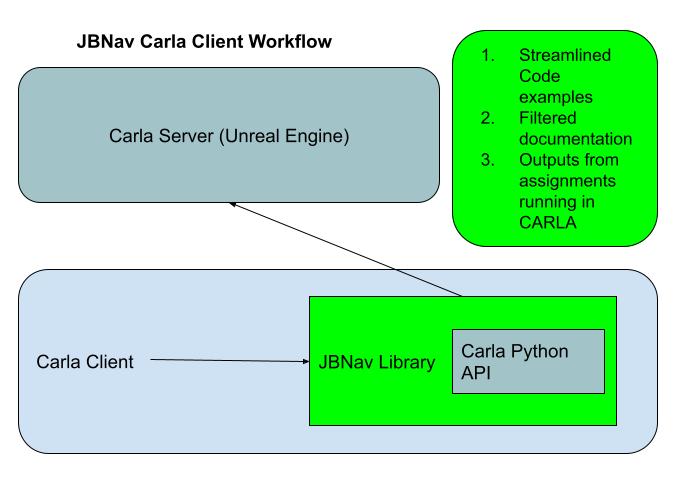


Figure 2: JBNav Client Workflow

Figure 2 shows the JBNav workflow. JBNav will create a wrapper library around the CARLA Python API that exposes a much simpler interface tuned for the use cases of this course and others like it. Alongside that library are examples from the milestone before which include streamlined documentation, and examples and outputs from running the assignments and project code in the CARLA environment.

This library will expose simple ways to deal with computer vision systems that annotated camera sensor feeds and optionally give control to the vehicle. This library will ensure it is flexible enough to supplement and port any DGMD E-17 pipeline to simulated data.

The most complex piece beyond what was above is the imitation learning piece. This piece works as follows in the diagram.

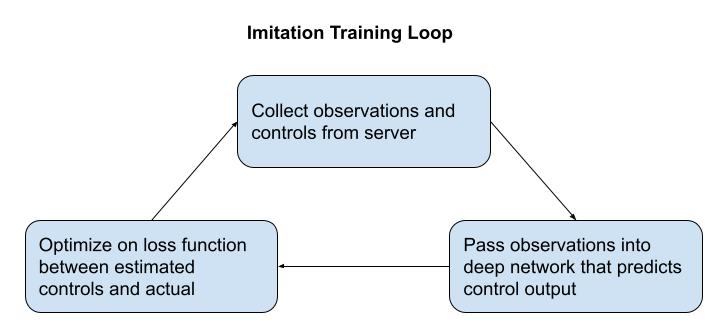


Figure 3: Imitation Training Loop

The above training loop shows how the imitation training agent will learn. Either through “live” collected data, or a dataset prepared ahead of time, a neural network will optimize based on camera sensor images into a convolutional network that will have multiple outputs for the various controls, throttle, steering wheel, brake etc (will be bound by CARLA’s detailed control model). A loss function will take the joint loss of all the predictions and backpropagate them through the network. In theory this agent will learn how to behave according to the car in the simulation, given only camera access (the driver in CARLA knows the state of the road, cars etc. So this will summarize the detailed CARLA agent into a neural network.

# 2. Related Work

CARLA has a research paper that briefly discusses applying various types of learning to autonomous vehicles here: https://arxiv.org/pdf/1906.03199.pdf

# 3. Team Organization

## 3.1 Team Members and Roles

The two team members of the JBNav project have discussed roles and responsibilities, and as of now plan to evenly split the workload. Both will develop and code using a shared github, and will use identical CARLA versions.

As work begins to develop we may split up which modules each member develops, while ensuring interfaces and APIs are communicated so that software components work interchangeably.

# 4. Software and Developing tools

## 4.1 Software:

CARLA, Unreal Engine, Python

## 4.2 Laptop/Desktop setup:

Operating system, Virtual Machine, Docker

## 4.3 Hardware needed:

GPU necessary to run Carla, either on desktop or in the cloud

## 4.4 Simulator needed:

CARLA

# 5. List of Milestones

Make a list of milestones by week as example:

|  |  |
| --- | --- |
| Date | Milestone |
| April 11th, 2021 | CARLA running and controllable |
| April 18th, 2021 | DGMD Projects ported to CARLA, outputs collected |
| April 25th, 2021 | Clean up Code and document |
| May 2nd, 2021 | Prepare presentation, begin imitation learning |

# 6. Team Meeting Schedule

Weekly meetings to discuss progress on assigned tasks, potential blockers, and adjust the direction of technical development (as needed).

# 7. Results

JBNav is a python library that uses CARLA and makes deploying computer vision pipelines to CARLA for testing and vehicle control super easy (i.e. one line of code to run an experiment).

Graphical user interface, text, application, chat or text message

Description automatically generated

It was envisioned to support two primary use cases:

1. Quickly Deploy computer vision pipelines (such as this classes assignments) to various generated environments and conditions
2. Autonomous Agent and Policy improvement and testing (upcoming)

JBNav can be easily installed and run with the following commands:

Graphical user interface, text

Description automatically generated with medium confidence

Here are the full list of features available in JBNav in addition to the run\_experiment() function signature:

* Weather Presets (Day, night, cloudy, sunny, wet, etc.)
* Traffic generation (pedestrian and vehicle)
* Client synchronous (pipeline drives the frames, meaning you won’t miss frames)
* Storing of training data, processed images, etc
* Autopilot or Agent Controlled Ego Vehicle
* RGB camera (Full HD)
* Auto-Named and stored experiments (TODO: Store metadata in experiment file for replaying experiments)

Text, chat or text message

Description automatically generated

In addition to developing the JBNav module, as part of this final project we developed an Imitation Learning pipeline using a dataset can be downloaded here: [CARLA Imitation Learning (24 GB)](https://drive.google.com/file/d/1hloAeyamYn-H6MfV1dRtY1gJPhkR55sY/view)

The data is stored on HDF5 files where each HDF5 file contains 200 image and control/state information stored in two datasets: “rgb” and “targets”

* The “rgb” datasets contains 200x88 resolution images
  + 657,800 images to train and 74,800 images for validation
* The “targets” dataset includes all the controls and measurements collected:

|  |  |  |  |
| --- | --- | --- | --- |
| Simulation Control Variables and State Information | | | |
| 1. Steer, float  2. Gas, float  3. Brake, float  4. Hand Brake, boolean  5. Reverse Gear, Boolean  6. Steer Noise, float  7. Gas Noise, float | 8. Brake Noise, float  9. Position X, float  10. Position Y, float  11. Speed, float  12. Collision Other, float  13. Collision Pedestrian, float  14. Collision Car, float | 15. Opposite Lane Inter, float  16. Sidewalk Intersect, float  17. Acceleration X,float  18. Acceleration Y, float  19. Acceleration Z, float  20. Platform time, float  21. Game Time, float | 22. Orientation X, float  23. Orientation Y, float  24. Orientation Z, float  25. High level command, int  26. Noise, Boolean  27. Camera (which camera was used)  28. Yaw Angle |

Here are a few examples of the CARLA images and controls for a 30 second scene:

A screenshot of a video game

Description automatically generated with medium confidence

Chart

Description automatically generated

This project focused on predicting the steering angle in radians based on the input images. The steering angle ranged from -1 to 1 and was normally distributed with a mean of 0.

Chart, histogram

Description automatically generated

We tried three different model architectures:

* Convolutional Neural Network with single image input
* Multi-Input CNN/Dense network with images and state information (position, acceleration, etc.)
* VGG19 and ResNet-50 Transfer Learning with image input

We experimented with LeNet style architectures and various filter sizes, depth, and batch normalization. The network below achieved the best validation result with an MAE of 0.0919 rads.

Graphical user interface, text, application, email

Description automatically generated

Here is an example of the original video and the true/predicted steering angle for 1 example CARLA scenario:

Graphical user interface, chart

Description automatically generated

We also leveraged the Tensorflow Functional API to create a multiple input model

* Input 1: Images to be processed by CNN
* Input 2: Position, Acceleration, and Speed numeric vector to be processed by MLP

The outputs from each network were combined and passed to a fully connected output layer in a similar fashion to the graphic below:

A picture containing timeline

Description automatically generated

The results of this model were on par with the CNN predicting on a single input image.

Lastly, we experimented with Transfer Learning using pre-trained weights from ImageNet to fine-tune the steering angle prediction model leveraging both VGG-19 and ResNet-50.

Diagram

Description automatically generated

A few idea for future improvement of the Imitation Learning pipeline are as follows:

* Leverage sequence information (past images, steering angles, change in position, etc.) to predict the next state for steering angle
* Tensorflow’s TimeDistributed and/or ConvLSTM layers which may be helpful
* Implement DAVE-2 architecture from Nvidia’s end-to-end self-driving car paper which includes image preprocessing/augmentation
  + <https://developer.nvidia.com/blog/deep-learning-self-driving-cars/>
* Train with more data (only used 50k images in this project)
* Predict multiple output states (i.e. throttle, brake, etc.)