Carla Lab

Team: Virtual Fast and Keyboard Furious

Carla has adapted to a server-client interface system in which the simulation is all handled server-side and the client is used to give commands to the server that spawns a car or sensor and follows these actions and stuff like that.

Server-side needs more computation than the client-side and Ideally, you can run the Server side on a great system and run the client from any moderate system with a network connection.

One problem to this is that there will be a delay depending on the ping of your network which may or may not be appropriate for the project you might be working on.

Carla Installation:

1. There are 2 ways to get Carla working on your system.
    a. Get the prebuilt version: That has unity and source code complied and put in a package.
    b. Build it from Source: Linux Build (There are options for Windows Build as well)

2. Building from Source
    a. Requirements:
        i. Unity
        ii. GPU system
        iii. Minimum 30 Gb of disk space

    b. Unity Installation
        i. To get source code to install unity
            1. Register on Epic
            2. Go to Accounts → Connections → Connect GitHub to Epic account
            3. That will give access to the private repository of Unity.
        ii. Clone the Repository and follow the instructions.
        iii. Set “export UE4_ROOT=~/UnrealEngine_4.24”

    c. Installing Server Side
        i. Git clone the repo.
        ii. Follow steps from Documentation.
        iii. This will build and install Carla and also compile the environment in Unity and takes a while to finish.

d. Installing Client-Side for Carla 0.9.9(Carla Client is referred to as Python API):
    i. Before proceeding to make PythonAPI, make sure you have: Python 2.7 and Python 3.5
    ii. Run the command "make PythonAPI"
    iii. Carefully observe what is the output and make sure it is successful for both python 2.7 and 2.5
    iv. Even if you are just gonna use either 3.5 or 2.7 installation needs to be done with both the requirements met or else the client side will not build correctly.
    v. Once it has all successfully built and installed export python path to the egg file created by make python API command in order for the client-side to work.
    vi. Run examples files to make sure everything is working.

e. Installing Ros Bridge:
    i. Follow steps on ROS Bridge Page.
    ii. There are two options:
        1. User
        2. Source (Developer)
    iii. Choose the appropriate option according to your project and install it.

3. Pitfalls:
    a. Python API not building egg files:
        i. Check Python version and go through the output on the command line and see where it failed
        ii. Even if it fails for one python it does not give an error that installation was aborted rather after that every time you try running the command again it will say installation is complete which is not the case.
        iii. Go through the output find the error and then satisfy that and do "make clean"
        iv. This will remove files from the previous make and then try again.

    b. Import Carla error:
        i. Check the python version you are using and check which python path is given.
        ii. Make sure the same version python path is provided.

    c. Ros bridge carla_msgs not found:
        i. Go to the git repository
        ii. Find the package carla_msgs
        iii. Go to ros-Carla-msgs repository
        iv. Git clone and change name to carla_msgs

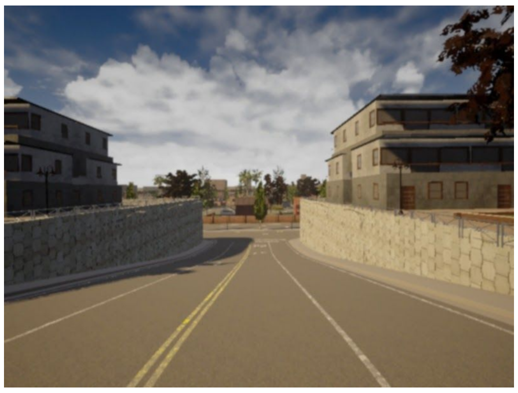
Carla Lane Finding and Following

Given that wall-following and gap finding does not apply to directly to Carla - Urban Driving Environments, our team has contributed and ideated to a new task involving perception based control task of finding road lanes and following center of lanes which is becoming a standard feature of adaptive cruise control in commercial cars.

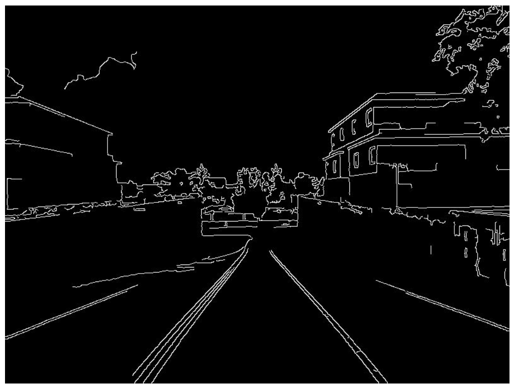
Perception

We implemented lane finding from images of a virtual car's front camera using the OpenCV API's. Following steps give us an image with tracked lanes.

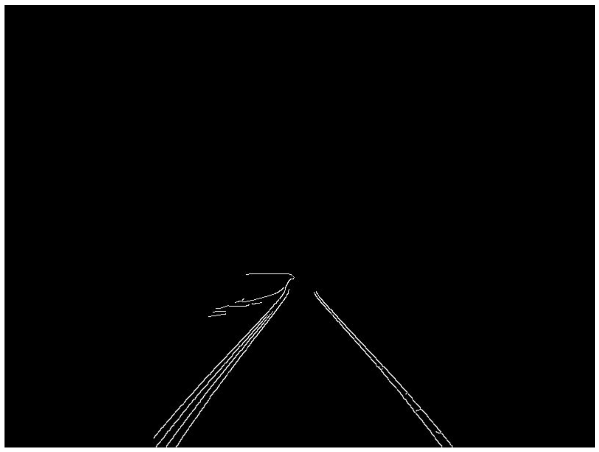
1. Adding Gaussian noise for blur effect, Convert Color to grayscale Image
2. Canny edge detector to detect edges
3. Selecting a Region of Interest (ROI) (to limit
4. Hough transform to find lines from the detected pixels from edge detection
5. Extrapolation of lines using and finding two lines left and right line from the lines found using hough transform



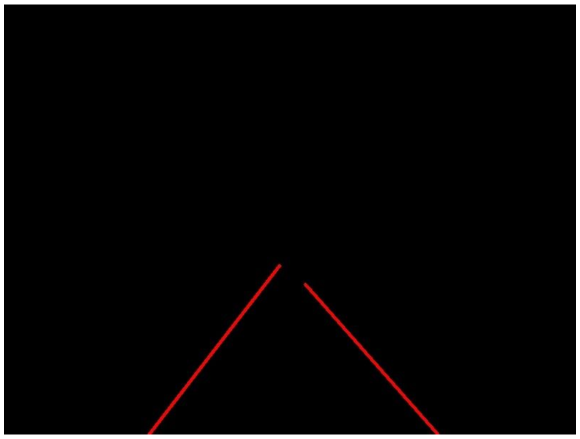
Original Image

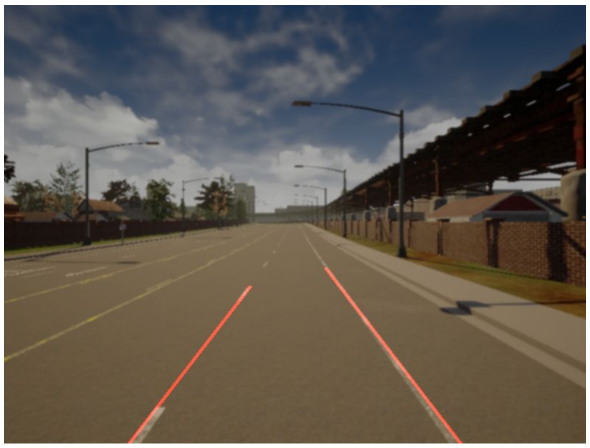


Canny Edge Detections



Clip extra edges using ROI Selection


Finding Single Lanes using Hough Line Transform


Overlaying Lanes on Original Image

Points on the lane are taken as reference to keep the car at the center.
To find corresponding points on lane in image space to 3D World Coordinates, we transform them using depth information.

Log normal Depth image from Front Depth Camera.

The camera calibration information is given cx,cy and fx,fy.
Given x,y in pixel coordinates and corresponding ‘z’ from log-depth image, We can get the real world coordinates with this transformation.

```
realworldz = z
 realworldx = (x - cx) * z * (1/fx)
 realworldy = (y - yy) * z * (1/fy)
```

Ttransformed points on the lane to world coordinates to is used to find center of lane. Using the center of the lane, we used the concept from Wall following lab to keep the desired distance of 1m from the right lane(some places our lane detection was not functioning for left lane). Ideally it would midway from both lanes. We also use a lookahead distance given the car is in motion. Error between actual distance and desired distance gives us control inputs to determine steering angle and speed and we tune the PID accordingly.

Overall, the system works but is not robust. We show in videos a lane following for straight stretch of road, however the lane detection fails at turns and crossings. Overall, we were out of time and luck due to remote colloboration and a single computer running Carla. In this report, we bring value by showing some pitfalls in carla-ros-opencv installation and near-working demo of lane following.