Lane Assist and Object Avoidance Systems in Urban Settings

Priyanshu Rawat, Koundinya Prabhakar Durbha
Department of Automotive Engineering, Clemson University
International Center for Automotive Research (CU-ICAR)
2021 Fall Course Report: Computing and Simulation for Autonomy, Instructor: Dr. Bing Li

Abstract

Running and optimizing the base code code for lane detection and object identification (for an autonomous vehicle) in an urban setting. Testing and simulating Lane Keeping and Object Avoidance systems in a simulated environment focussing on road congestion.

1. Introduction

2. With the motivation to implement ADAS features to reduce road injuries and accidents caused by traffic congestion. In this project, we have tested ADAS features in a crowded road environment. The initial aim was to run and optimize the baseline codes (for ADAS features like; lane detection and object identification) in CARLA; an open-source simulator focused on autonomous driving.

2.1. Objective

Our initial objective was to test the lane detection and obstacle detection system in CARLA. Later, we focused on our future goals including implementing lane keeping and an obstacle avoidance system into our baseline code and ultimately tested its accuracy in the simulated environment.

2.2. Motivation

Thousands of lives are lost due to traffic congestion on the road. This congestion can be caused by vehicles, pedestrians, or animals. Especially countries/cities with high populations are experiencing a massive surge in accidents because of the overcrowded roads. Other reasons for such accidents can be the jaywalking pedestrian traffic, leniency in traffic rules, stray animals.

As a result, human error needs to be limited in order to reduce accident rates, and this can be achieved by minimizing human involvement in driving the vehicle. For instance, using driver assistance systems like lane keep assist and obstacle avoidance could significantly improve safety while driving around pedestrians and other living beings.

2.3. Related Work

We have currently considered two baseline codes for this project (links of all the relevant sources can be found in the 'References' section). The first baseline code deals with lane detection while the other code covers object detection.

Lane detection is crucial and an initial step towards implementing a lane keeping system. The identified baseline code for lane detection converts real time video recordings captured by manually driving a vehicle in CARLA. It then applies a number of processes on each image frame using the computer vision and other libraries. At last, the algorithm is able to detect lane lines and this information can be used to set control parameters on the vehicle, which in turn can be fed into the vehicle to maintain its current lane.

The second baseline code obtained is regarding object detection. It is essential to have a good object detection ability to perceive the environment around the ego vehicle. This is the first step towards systems like the object avoidance system. This baseline code detects all types of vehicles, and pedestrians and draws bounding boxes around them. This will be used to set control parameters for the vehicle to be able to steer clear or brake if it spots obstacles in its path.

2.4. Setup

For the course of this project we have used a windows 64-bit machine, with configuration as follows -

- Intel i7-11800H
- NVIDIA GeForce RTX 3060
- 16GB RAM

Other than the hardware configuration of the system in use. We have used the 0.9.10 version of CARLA.

CARLA is an open-source simulator developed mainly for autonomous driving research purposes. The software supports development, training, and validation of autonomous driving systems.

On top of that, CARLA also provides open digital assets (urban layouts, buildings, vehicles) that were created for this purpose and can be used freely.

2.5. Installing and Setting Up CARLA

For the installation of CARLA the pre-requisites were -

- 20GB of disk space
- At least 6GB of video memory
- Python 2.7 or higher

Every version of CARLA has its own additional package of extra assets and maps. We have tried installing the latest version of CARLA (0.9.12) and were unable to establish a connection between the client and simulation server. So, we have installed the 0.9.10 version instead for the testing and simulation in the course of this project.

The installation package of CARLA can be easily downloaded from their Github repository. Following the steps listed in the documentation of CARLA, we were able to install and set up the software successfully. [2][3]

Figure 1: CARLA simulator window in manual driving mode

Figure 2: CARLA simulated environment bird's-eye view

2.6. Lane Detection Baseline Code

The baseline code [4] with some modifications was able to identify the lane lines in the simulated environment in CARLA.

The steps followed in the baseline code were[5]:

1. Capturing and decoding image frames from CARLA client server to opencv.
2. Each image frame is then converted into grayscale in real-time.
3. Filters are applied to these image frames to reduce noise.
4. Edge detection functions are then applied to this filtered image frame.
5. A region of interest is defined to narrow down the area being worked on.
6. Hough line transform is used to extract features.

These extracted features are then shown in another window.

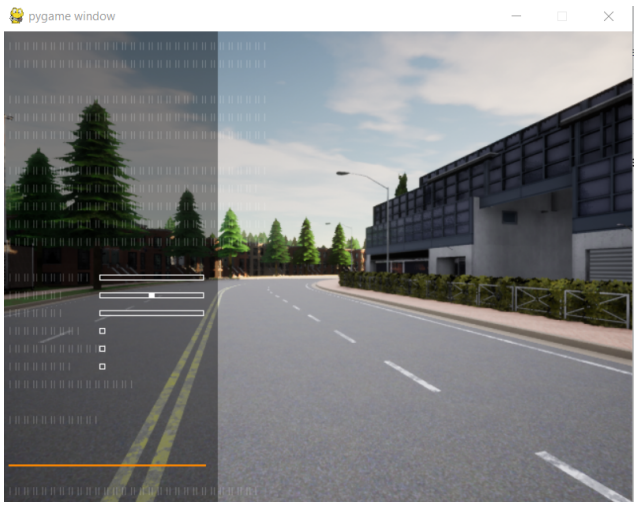

Figure 3: Manual driving environment in CARLA

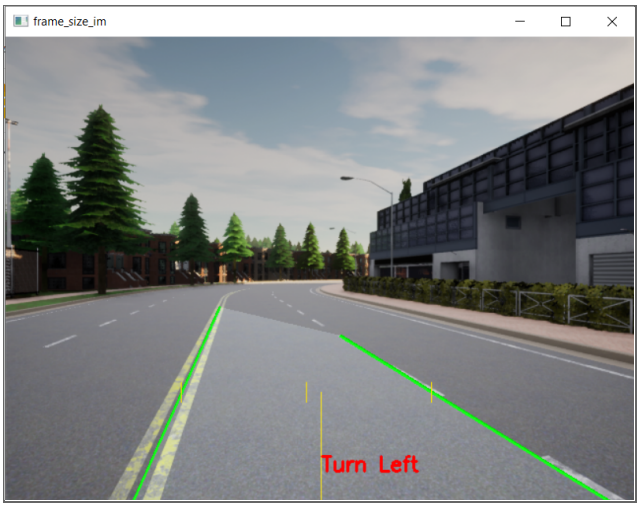

Figure 4: Extracted lane lines obtained from implementing baseline code

The lane center can be found from these two lane lines and the deviation in vehicle pose from the lane center can be used to control the vehicle and maintain trajectory in the desired lane.

2.7. Object Detection- Baseline Code

The object detection baseline code [6] opens a client world on carla and spawns the required number of vehicles and walkers in the simulation. It then switches views to different vehicles and gathers data as snapshots. These snapshots are then used for training and the result is stored as an mp4 file where we can see that different bounding boxes are added for pedestrians and vehicles.

2. Algorithms

We developed algorithms to perform different driver assist tasks like lane keep assist, lane change assist, adaptive cruise control and emergency braking. The final goal being to develop and test a control algorithm capable of obstacle avoidance.
A python script has been used to initialize the code, after which various other functions are called which give inputs to the controller which executes these inputs.
The first step is to make waypoints at allowed spaces. These can be made from the town maps available in CARLA. These waypoints can also be printed onto the simulator.

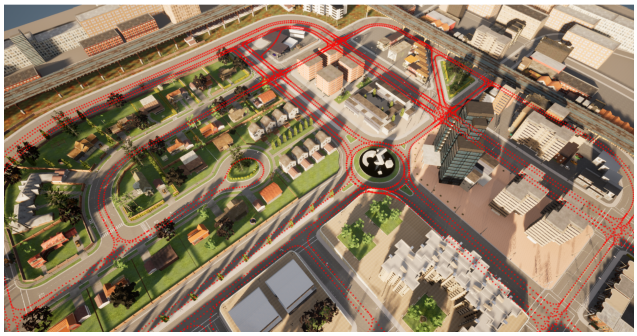

Figure 5: Waypoints loaded onto the simulator.

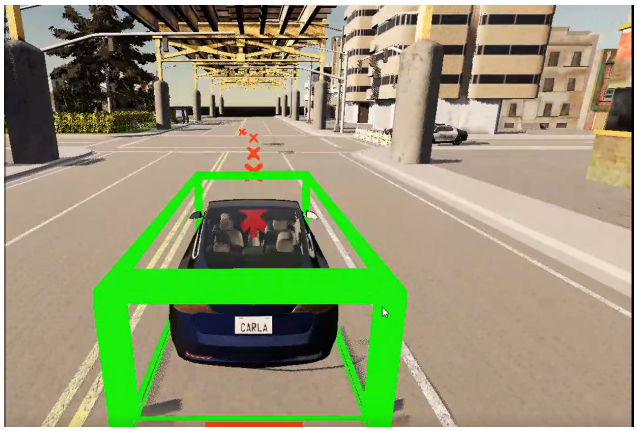

Figure 6: 3D waypoints printed onto the simulator.

The control flow of the algorithm is shown in figure _ below.

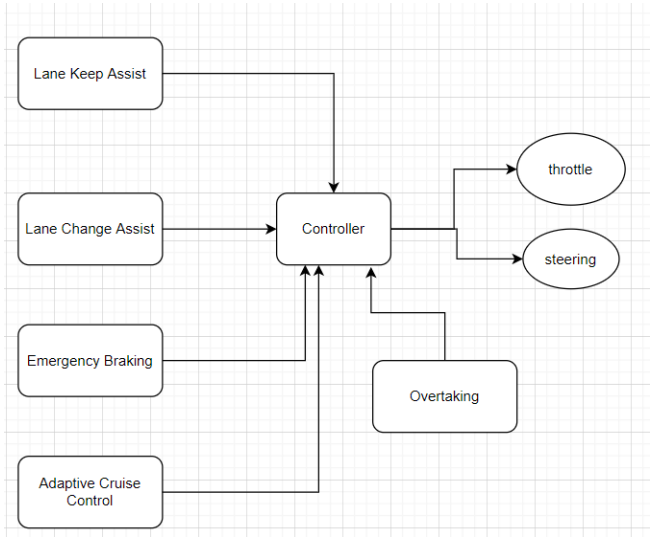

Figure 7: Flow of the code used.

2.1. Controller

A PID controller has been implemented for this project. It takes in target speed and steering angle from functions like the lane keep, lane change, ACC functions and

implements them, which are reflected in the resulting vehicle pose in the simulator. Two PID controllers have been used- one each for longitudinal and lateral control. the tuning parameters obtained for them are as follows:
Longitudinal:
kp=0.8, ki=0.5, kd=0.09
Lateral:
kp=0.15, ki=0.07, kd=0.05.

2.2. Lane Keep Assist
The Lane Keep Assist module checks to see if there are waypoints in the right side of the lane. If so, it can find out the centerline of the current lane and send it to the controller as a target pose, which the controller then tries to achieve.

2.3. Adaptive Cruise Control
This module lets the ego vehicle maintain a desired distance from a vehicle ahead of it. The ego vehicle can obtain the positions of other vehicles around it and compare them to its own. It can identify vehicles on it's own lane and in front of it, whose speed it can follow to maintain the desired distance from the next vehicle.

2.4. Emergency Braking
This module bypasses the controller to change throttle to -1 i.e. braking immediately if the vehicle is less than a certain defined distance from a pedestrian or a vehicle. the controller takes control again once the vehicle reaches a safer distance.

2.5. Overtaking
The overtaking module works by checking the proximity of vehicles in both the self lane and the next lanes. If there is no vehicle in a safe distance in the adjacent lane and if the vehicle in front is travelling at a slower speed than the ego vehicle and the speed limit on the self road, the vehicle shall trigger an overtake.The target pose is set to the next lane and the vehicle safely changes lanes and then speeds up.

Through the combination of these modules, the vehicle should be able to perfectly navigate an empty simulation environment, and also navigate in a dynamic environment with moving vehicles and pedestrians.

3. Simulation Environment

CARLA has some in-built town maps, and one can also custom build a world map.For this project, an in-built map in CARLA, Town 03. It is an urban city setting in line with our motivation. It has city roads with speed limits of 30 kmph and also highways with speeds reaching 90 kmph. It also has a round-about which could test the controller smoothness.

Figure 8: Snapshot of a populated CARLA Environment.

Tests were conducted in this environment in different stages. First was to test the autonomous navigation capacity of the vehicle in a world without vehicles or pedestrians. The objective here was to test if the vehicle is able to navigate an empty world without crashing. this is the first step to truly autonomous driving.

4.Results
There were four simulations run:

1. Autonomous navigation on an empty world.
2. Navigation with only pedestrians to test safe braking.
3. Navigation with vehicles to test cruise control.
4. Navigation with vehicles to test overtaking.

In case 1, we found that the vehicle could navigate around with ease.it did not crash into any static object. It also maintained a trajectory at the lane centerline, and safely changed lanes when required. However, it was noted that it's performance on roundabouts was not satisfactory. It seemed to often jump between the inner and outer lanes. It might be due to insufficient PID tuning leading to unpredictable behaviour on curvatures, not observed on normal lanes but evident on roundabouts.

Moving on to case 2, we tested the vehicle in an environment populated with pedestrians.The car ignored traffic rules while the pedestrians were made to follow them, to create more interaction between the vehicle and pedestrians, as the vehicle tries to jump a red signal when pedestrians would be crossing the road. The vehicle could detect pedestrians and stop once it reached a set proximity to them. The emergency braking and pedestrian detection were a success.

Figure 9: Vehicle waiting for pedestrians to cross.

In the third case, we tested the vehicle for automatic cruise control and the emergency braking for vehicles. The world was heavily populated by vehicles in frequent blocks and

traffic jams, and the vehicle had to travel safely without any collision. The vehicle was able to navigate through the traffic but it was not as robust. In heavy traffic, it could not properly react to sudden braking and speeding up, and crashed a few times. However, cruise control and emergency braking did work in most cases.

The fourth case was that of overtaking. The vehicle did not overtake the vehicles in front despite meeting the set conditions. Instead, it would follow adaptive cruise control and brake if too close.

2.8. 5. Conclusion

By the end of this project, we were able to successfully employ lane keeping, lane change, emergency braking and adaptive cruise control maneuvers in CARLA. However, we weren't able to successfully implement overtaking, which slowed our progress towards the final goal of a smooth and robust obstacle avoidance system which could safely navigate high density traffic routes.

In terms of future work, the first thing to complete would be the overtaking function. Furthermore, other controllers like the Stanley controller and pure pursuit controller could be used and their performances compared with the PID controller to find out the strengths and weaknesses of the three.

This project looked at autonomous vehicle functionalities only from a modular perspective, where tasks are divided into perception, path planning, controls and the like. However, there are other learning techniques like imitation learning and reinforcement learning which completely bypass this modular perspective and get good results.

Once the present modular model is robust, it could also be compared to these end to end methods to find a good balance between the two, which would ultimately help us reach our goal of a smooth and robust obstacle avoidance mechanism in vehicles, and the subsequent larger goal of autonomous driving technology.

References

[1] https://www.baileyjavinscarter.com/what-percentage-of-auto-accidents-are-caused-by-human-error/
[2] https://carla.org/
[3] https://carla.readthedocs.io/en/latest/start_quickstart/
[4] https://automaticaddison.com/the-ultimate-guide-to-real-time-lane-detection-using-opencv/
[5] https://github.com/priyanshurawat1509/Final_Project/blob/main/Lane_Detection.py
[6] https://github.com/AlanNaoto/carla-dataset-runner
[7] https://proceedings.mlr.press/v78/dosovitskiy17a/dosovitskiy17a.pdf
[8] https://arxiv.org/pdf/2010.11063.pdf