Development of a Simulation-Based Platform for Autonomous Vehicle Algorithm Validation

by

Rohan Bandopadhay Banerjee

S.B., Electrical Engineering & Computer Science, Massachusetts Institute of Technology (2018)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2019

Author . . . .
Department of Electrical Engineering and Computer Science
May 24, 2019

Certified by . . . .
Daniela Rus
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Development of a Simulation-Based Platform for Autonomous Vehicle Algorithm Validation

by

Rohan Bandopadhay Banerjee

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2019, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Developing robust algorithms for autonomous driving typically requires extensive validation and testing with physical hardware platforms and increasingly requires large amounts of diverse training data. The physical cost of these hardware platforms makes field testing prohibitive, and the cost of collecting training data limits the size and diversity of this data. Autonomous driving simulation is a promising solution to address both of these challenges because it eliminates the need for a physical testing environment and because it offers environments that are configurable and diverse. However, most autonomous driving simulators are not fully useful for algorithm validation because they lack full integration with fundamental autonomous driving capabilities and because their sensor data is limited in functionality. In this work, we develop and present a simulation-based platform for testing and validation of autonomous driving algorithms that combines an open-source autonomous driving simulator (CARLA) with our existing autonomous driving codebase. Specifically, we describe our software contributions to this platform, including simulated proprioceptive sensors and ground-truth LIDAR road information, and we demonstrate how we used the platform to validate both fundamental autonomous driving capabilities and a point-to-point navigation algorithm in simulation. We also describe how our platform was used to both develop and validate an approach to dynamic obstacle avoidance, a new capability in our codebase. Our platform is a capable tool for both validation and development of autonomous driving algorithms, although open directions remain in the areas of simulator sensor realism and runtime efficiency.

Thesis Supervisor: Daniela Rus
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I would first like to thank Prof. Daniela Rus, my advisor, for her guidance and for steering me in the right direction throughout my MEng. Her advising helped me cultivate my ability to communicate about my work in a succinct manner and become a more independent researcher. My thanks also go to Prof. Sertac Karaman for providing helpful guidance and advice along the way.

I would especially like to thank Teddy Ort, with whom I worked extensively over the course of my MEng, for his helpful feedback, guidance, and mentorship, including but not limited to his familiarity with the autonomous driving codebase and hardware, his help with the CARLA integration process, his advice regarding experimental design, and his help with MapLite. I would also like to thank Thomas Balch for his advice and support on the hardware and software components of the autonomous driving codebase. Additionally, I want to thank Igor Gilitschenski for not only being a capable leader of the autonomous driving group but also for his warmth and accessibility from the time I started my MEng.

I am grateful to every other member of our autonomous driving group, including Brandon Araki, Felix Naser, Alexander Amini, Lucas Liebenwein, Alyssa Pierson, Cristian-Ioan Vasile, Wilko Schwarting, and Tim Seyde, as well as the entire Distributed Robotics Laboratory, for making the lab environment feel like a community.

Finally, I want to extend my thanks to my friends at MIT and beyond, and to my family - especially to my father, for his close mentorship and guidance; to my mother, for her emotional support and encouragement; and to my sister, for her liveliness and accessibility - all of whom gave me the guidance and support that I critically needed to complete this journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern autonomous vehicle software systems are composed of a large number of interdependent components, including modules for perception and sensing, high-level planning and decision making, and low-level control. One instance of such a system is our existing autonomous vehicle framework, which incorporates many of the fundamental components of autonomous driving into a single codebase. This codebase is implemented in a modular fashion using the open-source ROS framework [35] and is shared across three distinct hardware platforms: a full-scale Toyota Prius [28], an autonomous wheelchair, and a 1/10-scale RACECAR [25] vehicle platform. The overarching goal of such systems is to interpret sensor data from the vehicle along with any necessary external information about the environment in which the vehicle is situated, to make a high-level decision about the future behavior of the vehicle, and to determine the low-level control output necessary to achieve this behavior.

As autonomous driving systems have grown larger in scale and are applied to more complex domains, it has become increasingly difficult to rely on hardware platforms alone for testing and validating autonomous driving algorithms. Therefore, researchers and engineers working on the development of autonomous vehicle systems have turned to simulation as a software alternative to hardware testing. A simulator is a software-based environment that contains models for relevant physical components and for the environments in which they reside. An autonomous vehicle simulator will typically include realistic physics models for the agents in the simulation, the ability to construct rich and realistic road environments, and the ability to simulate the sensory inputs that would be present on an actual autonomous vehicle. Typical simulation environments will include static objects such as buildings and road features, as well as dynamic objects such as pedestrians and other vehicles. Such a simulator typically includes motion models for the dynamic objects that enable them to traverse the environment and interact with other static and dynamic objects in a realistic way.

A full-fidelity autonomous vehicle simulator with the aforementioned capabilities would be beneficial to autonomous driving research for a number of reasons. Such a platform would allow for the collection of data from a much larger set of environmental configurations compared to the real world, allowing more control over how much data comes from specific configurations. Such a platform would also allow researchers to prototype algorithms in a safe manner, without incurring the risk of collisions with obstacles or environmental features. For instance, simulators would benefit exploration-based planning algorithms such as RRT* [21] because they would enable a greater exploration of corner cases that might otherwise lead to collisions. Because simulators are not limited by the cost of hardware platforms, multi-vehicle interaction becomes more feasible, opening up the possibility of developing and validating cooperative algorithms. Finally, an autonomous driving simulator could potentially allow algorithms to be implemented and tested at speeds that are much faster than real-time, which would also allow for large-scale collection of diverse data that could be used to much more quickly train learning-based algorithms.

The difficulties in developing effective simulators that can prototype and validate algorithms are manifold. The first gap in simulator efficacy relates to the realism of their simulated data. The quality of simulated perception data hinges upon both the quality of the rendering engine used to generate camera and LIDAR data, and the textures of the physical models that constitute the environment. A more realistic simulator will typically require a larger computational burden, both in terms of time and rendering hardware (such as GPU and graphics driver capabilities), which may negatively impact the latency of the simulator system. The issue is that if the simulated data comes from a significantly different distribution than the real-world data, then perception-based algorithms that are trained on the simulated data will not necessarily generalize to the real world. The quantity and diversity of vehicle data is particularly important for learning-based approaches to autonomous driving to be effective, which includes approaches that apply machine learning models to particular sub-modules (such as perception and planning) as well as fully end-to-end algorithms [5].

Another challenge to developing an effective simulator lies in optimizing the computation time required to model the dynamics of the agents in the world, produce realistic sensor data, and keep track of the states of all of the agents in the simulator. Without a significant speed difference between simulated and real-world dynamics, it becomes more difficult for simulation to achieve the benefits of scale that are theoretically possible with a fully software-based platform. An additional challenge of simulation that is particularly relevant to obstacle avoidance is in developing realistic models for the dynamic agents in the simulation. Programming behavior for these dynamic agents that respects traffic conventions and other vehicles can be as difficult as the problem of general autonomous driving, albeit without the problem of perception (because the ground-truth positions and velocities of all agents are known in the simulator).

The core problem that we seek to address in this work is that of leveraging simulation to more effectively develop and validate perception-dependent autonomous driving algorithms. While there exist simulators [50, 3] that allow for the development and validation of planning and control algorithms, these simulators typically lack the 3D rendering required to generate perception data. And although there are open-source simulators [22, 44, 10] that offer simulated perceptual sensor data, these simulators are typically structured as open-loop configurations that may not include modules that can generate control outputs in response to the perception data.

To our knowledge, none of the existing public autonomous driving simulators are integrated with an autonomous driving stack that includes capabilities such as localization, planning, perception, and control, even though some of these simulators provide the capability for users to interface their own code through ROS. An integrated platform with built-in fundamental autonomous driving capabilities would enable robotics researchers to focus on developing higher-level autonomous driving algorithms without the need to re-implement these capabilities. Another existing limitation with most simulators lies in the quality and efficiency of 3D LIDAR simulation. To highlight a few of these deficiencies, simulated 3D LIDAR sensors typically do not include ground-truth semantic segmentation information (as is typically the case for simulated camera data), they are typically unrealistic because of texture limitations of the underlying physics engine, and they often run much slower than real-time because of the cost of the ray trace operations used to produce the sensor data. All of these factors limit the potential for autonomous driving simulators to be useful tools for the testing and validation of autonomous vehicle algorithms, and for simulators to aid in the development of new algorithms.

Our approach is to develop a full-loop platform that merges an existing open-source simulator with an autonomous driving software stack capable of processing perceptual and vehicle measurement data and generating vehicle control commands. Our platform enables us to take algorithms that have been developed for use on physical autonomy platforms and validate them in simulation. Rather than focus on performance optimization of the simulator or on the realism of the simulated data, our approach focuses on closing the loop by building up the basic functionality of both the simulator and our own software stack. We do this by making engineering contributions to the simulator that include adding novel sensors and augmenting data from the existing sensors. The motivation for developing a full-loop architecture in this way is to emulate the existing interaction between our autonomous driving codebase, which determines how to control the vehicle given perception data, the physical vehicle, which receives control commands and and produces perception data, and the surrounding environment, which includes all other agents and static features in the world. Our goal is for the simulator to encapsulate the responsibilities of the physical vehicle and the environment in a way that seamlessly allows the simulator to interface with our codebase, as illustrated in Figure 1-1.

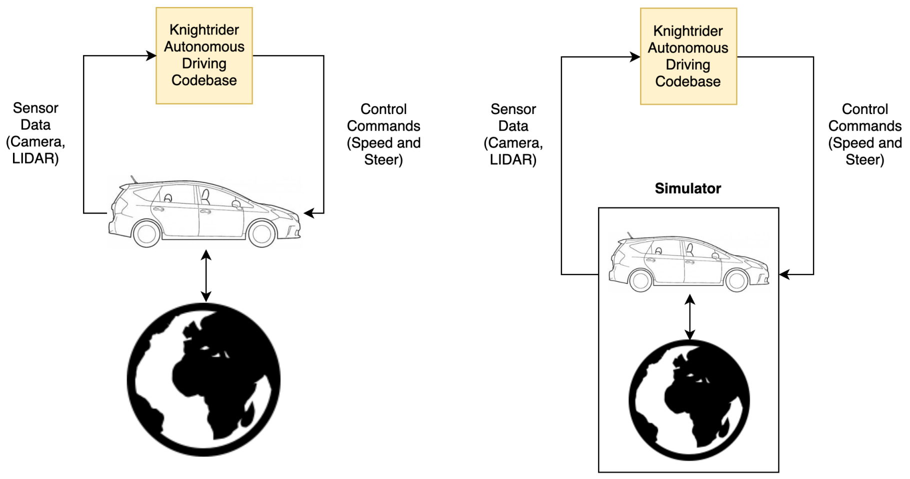

Knightrider Autonomous Driving Codebase
Sensor Data (Camera, LIDAR)
Control Commands (Speed and Steer)
Knightrider Autonomous Driving Codebase
Sensor Data (Camera, LIDAR)
Control Commands (Speed and Steer)
Simulator

Figure 1-1: Architecture comparison between our autonomous driving codebase and the physical vehicle (left) and our proposed simulation-based platform (right), illustrating the interactions between the codebase, ego-vehicle, and the surrounding environment in each configuration.

Even though our platform does not address the issue of simulated data realism, we do augment the existing 3D LIDAR sensor with ground-truth road segmentation information, which makes our platform more effective for validating LIDAR-dependent perception algorithms. In particular, the road segmentation can act not only as an emulator for trained road segmentation models, but also as a source of training data for learning-based road segmentation algorithms that does not require manual annotation. Our platform can also leverage the ground-truth position information from the simulator to provide more accurate translational accuracy benchmarks of navigation algorithms. The fact that the simulator provides a reliable ground truth contrasts with the quality of the Oxford GPS sensor data used to provide ground-truth position information for our vehicle platforms, which is highly dependent on the quality of the GPS satellite reception (and is often not consistent across testing instances). Finally, our approach offers the ability to develop obstacle avoidance methods on a platform that shares the same software infrastructure as our hardware platforms but lacks the risks that are associated with high-speed collisions.

The current iteration of this platform is limited to controlling one vehicle with our autonomous driving codebase, although the autonomous driving simulator involved in our platform is capable of supporting multiple independent agents. Our platform is not currently capable of large-scale testing, as we cannot currently run tests at faster-than-real-time speeds due to the computational burden of rendering the exteroceptive sensors. However, our full-loop software simulation platform enables automated testing by providing more control over the initial conditions of experiments and by avoiding the costs involved with running real-world experiments on our autonomous platforms, such as the risk of collisions and energy resource constraints. Finally, we do not address the challenge of developing realistic models for the other vehicles in the environment. Instead, we leverage the existing baseline models provided by the simulator, which are sufficient for the experiments conducted in this work.

1.1 Contributions

In this work, we develop a simulation-based platform that aims to be an effective substitute for our hardware platforms. We build this platform by developing a complete sensor suite whose data is both realistic and annotated with useful ground-truth information, and by incorporating the capability to process the same control commands that our vehicles would normally receive. The platform that we have developed aims to be a useful tool not only for autonomous vehicle algorithm validation and data collection, but also for algorithm development. We choose to build our platform on top of the CARLA simulator, which we will describe in Chapter 3, because it provides a ROS interface and multiple simulated perception sensors, including multi-channel LIDAR, that are fundamental for integration with the perception and navigation algorithms in our autonomous driving codebase.

This thesis makes the following contributions:

- We build an integrated platform that allows for the testing and validation of autonomous vehicle algorithms in simulation by merging our existing autonomous vehicle platform with an open-source autonomous driving simulator (CARLA), and we demonstrate that this platform is capable of simulation and validation of three fundamental components of autonomous driving: EKF-based proprioceptive localization, pure pursuit control, and RRT* path planning.

- We extend the platform by incorporating ground-truth road segmentation information into the existing LIDAR sensor module in CARLA, and we use this to validate MapLite, a full-stack autonomous navigation framework, in simulation. We also describe our contribution to the MapLite framework that incorporates vehicle stopping behavior at road intersections.

- Finally, we use CARLA as a validation platform for extending the existing dynamic safety zone module in our autonomous vehicle platform to avoid dynamic obstacles by incorporating a proportional distance controller that maintains a safe following distance when driving in traffic.

1.2 Thesis Outline

This thesis is organized as follows. In Chapter 2, we discuss related work in the areas of autonomous vehicle algorithm validation and simulation. Chapter 3 introduces the core of our simulation-based platform and demonstrates our use of the platform to validate three fundamental autonomous driving algorithms. Chapter 4 describes how we extended our platform to validate a point-to-point navigation platform, MapLite, in simulation. Chapter 5 presents another extension of the core platform to develop and validate an approach to avoiding dynamic obstacles in the environment. Finally, Chapter 6 summarizes the capabilities and limitations of our platform and suggests possible directions for future work.

Chapter 2

Related Work

Below we discuss prior work in autonomous driving algorithm testing and validation, autonomous driving simulation, and the application of simulators to validate autonomous driving algorithms.

2.1 Algorithm Testing and Validation

2.1.1 Data-Centric Approaches

One way in which researchers test their algorithms is to rely on existing open-source datasets. A number of these datasets exist for training perception algorithms, which include visual [52, 7] and LIDAR [2, 17] datasets that are annotated using a combination of human and automated annotation methods, and multi-modal datasets [15, 18] that include both camera and LIDAR data. One issue with relying on a fixed dataset is that one is limited to the scenarios that are incorporated within the dataset. Although one could simply augment the dataset with newly collected data, replicating the exact hardware configuration used to create the original dataset can be challenging. Even though the visual datasets include a large number of different weather and scene configurations, they are mainly limited to paved roads in suburban or urban settings. Another issue is that a fixed dataset has a fixed proportion of scenarios that reflects the conditions in which the data was collected. However, it might be useful to train an algorithm on a large amount of data from scenarios that are under-represented in real-world driving but have a disproportionate impact on the performance of the vehicle - for instance, construction zones or inclement weather conditions.

Another approach to generating large datasets is to build models that can produce synthesized data, given only a training dataset that represents a subsample of the true data distribution. Such approaches may use learning-based generative models to synthesize new visual data [37, 47, 19] or point cloud data [49, 24], or may use a combination of classical computer vision and learning to synthesize perturbed data in an online fashion [51, 1]. While these approaches certainly reduce the effort of data collection and provide realistic data that allows autonomous driving agents to learn and generalize, they are still restricted by the scenarios that are present in the training dataset. For instance, even if such a generative model is trained to learn the distribution of images from urban driving, it may be difficult to draw samples that come from a particular type of junction (e.g. a 4-way intersection) if these are not represented in the training dataset. This means that one cannot easily synthesize scenarios with differing topologies or obstacle configurations, which may be crucial for assessing the robustness of a particular algorithm.

Our platform does not rely on an existing perception data repository because it uses the rendering capabilities of the simulator to produce camera and LIDAR data from an underlying 3D representation of the world. It is therefore capable of overcoming some of the limitations of a fixed or synthesized dataset by incorporating differing environmental configurations and underrepresented scenarios into the simulated world. However, the realism of the simulated data from our platform is not currently comparable to that which exists in the datasets that are based on real-world data. Additionally, the semantic complexity of the simulated environment depends on the size and diversity of the 3D models included in the simulator, which can exacerbate the gap between data generated by our platform and real-world data.

2.1.2 Hardware-Centric Approaches

One can get around the issue of testing algorithms on a full-sized vehicle platform by instead using smaller-scale platforms that still have the same sensor suite and control algorithms as the full platform. The RACECAR platform [25] is a small-scale vehicle platform that has LIDAR and camera sensors and is capable of running algorithms implemented in ROS. There also exist controlled platforms consisting of a large number of miniature robots [31, 30] that serve as a means for researchers to prototype and test their algorithms. Some of these robotic platforms [54] also include simulation environments that can be used to train or validate algorithms, although the visual fidelity of these simulators is limited. These systems facilitate testing cooperative algorithms that involve large numbers of vehicles in a controlled environment, whereas testing such algorithms on full-sized vehicles would be more costly and would require a much larger testing environment. Using these platforms may also allow for code sharing between full-sized vehicles and the miniature robotic platforms. Although these systems partially address the cost of hardware by providing a shared, remotely accessible pool of hardware resources, the cost of purchasing a full fleet of these vehicles can still be significant. Additionally, constructing environments that are suitable for these vehicles but also transferable to full-scale environments can be challenging. Our approach is currently limited to validating algorithms on a single independently-controlled vehicle, but the CARLA simulator that we use in our platform is capable of modeling multiple vehicles and pedestrians that operate according to limited motion models.

2.2 Simulation Platforms

Visual simulators such as DeepDrive [36] and the Udacity self-driving car simulator [45] focus on the simulation of multiple camera views that can provide RGB and depth information. Although these simulators provide rich camera information, they are typically limited to modeling a single autonomous agent and do not include other sensor modalities, such as LIDAR, that are useful for autonomous driving. Multiagent control simulators such as TORCS [50] have primarily been applied to problems in reinforcement learning and imitation learning. Although these simulators provide the ability to control multiple agents and learn policies that take vehicle interactions into account, their focus on planning and control means that one cannot test the full range of autonomous driving algorithms that require upstream modules (such as perception and sensing). To fill in this gap between perception-centric simulators and control-centric simulators, a number of full-stack simulators, including Gazebo Citysim [22], Microsoft AirSim [41], and CARLA [10], include elements of perception, planning, and control. These simulators are better suited for integration with existing robotics software stacks because of their ROS integration and sensor suites that include both camera and multi-channel LIDAR. Because our platform aims to interface with our autonomous driving software codebase, and because our focus is on developing and validating perception-dependent algorithms, we chose to incorporate the CARLA simulator into our platform.

A prior version of our autonomous vehicle framework [28] incorporated the Drake simulator [44], which was capable of producing simulated sensor data - both proprioceptive (vehicle odometry) and exteroceptive (2D LIDAR scans) - and receiving control commands from our autonomous driving codebase. However, one of the main limitations of this simulator was that its perception suite was limited to 2D LIDAR. The simulator was also only capable of simulating the behavior of the ego-vehicle, without any capability to simulate the dynamics of other autonomous agents. Although our platform only allows for one independently-controlled vehicle, we can still model external agents such as vehicles and pedestrians, and we are able to develop and validate a larger class of algorithms compared to our prior work. This includes perception algorithms that require camera or 3D LIDAR data, as well as planning algorithms that involve interactions with other agents.

2.3 Simulation-based Validation

Simulators that are built on top of video game engines have been used to benchmark and synthesize training data for a number of vision and LIDAR-based perception algorithms, including object detection [48, 20] and target tracking [27], and SLAM [42]. Although these simulators do provide realistically-rendered visual data, the quality of simulated LIDAR data is dependent on the quality of the textures associated with the 3D models in the simulation and does not necessarily reflect sources of uncertainty that are present in true LIDAR data. The main limitation with these approaches is that they use simulators in an open-loop fashion, where the simulator is used only as a data generation mechanism, rather than a closed-loop configuration where the simulator produces data that is influenced by external control inputs. Our platform is not only capable of data generation but is also capable of validating full-loop autonomous driving pipelines that include perception algorithms as well as planning and control algorithms that produce control outputs to the vehicle.

Simulators have also been used to provide ground-truth semantic segmentation information [39, 38, 40], although these approaches have typically been limited to visual data. Our simulation-based platform provides binary ground-truth road segmentation of 3D LIDAR that directly accesses semantic information from the underlying simulator.

Simulators have been used for training end-to-end algorithms that produce vehicle control outputs directly from sensor data, including for imitation learning [6, 4] and also for validation of modular pipelines that combine perception, planning, and control [10]. Our work is similar to the latter category in that we focus on validating perception-dependent algorithms that produce control outputs, although we do not explicitly validate any learning-based end-to-end approaches in our work. Other approaches leverage hardware-in-the-loop simulation for autonomous driving [9], in which a full-loop simulation combines software models and hardware components to provide control and sensor data. Our approach does not rely on any explicit vehicle hardware other than the resources that are needed to support the simulator and the algorithms in the autonomous driving codebase - namely, a laptop computer with a GPU for rendering sensor data.

Chapter 3

Validation of Fundamental Autonomous Driving Capabilities with CARLA

In this chapter, we describe the development of a simulation-based platform for autonomous vehicle algorithm testing and validation that combines the CARLA simulator with our existing autonomous vehicle codebase. We begin by describing the core components of the CARLA simulator. We then describe our contributions to the CARLA simulator to integrate the simulator with our autonomous vehicle codebase. Finally, we use the platform to demonstrate and validate three fundamental autonomous driving algorithms in our autonomous driving codebase: EKF-based localization, pure pursuit control, and RRT* path planning.

3.1 CARLA Overview

CARLA [10] is an open-source autonomous driving simulator that allows for the simulation of an autonomous ground vehicle in a simulated environment. It is built as a plugin to the Unreal Engine [11], an open-source video game engine, and leverages the features of this engine to simulate the physics of the vehicle and to generate simulated sensor data, including camera and LIDAR. CARLA is also able to model the behavior of other agents in the environment, such as other vehicles, pedestrians, and motorcyclists.

3.1.1 CARLA Architecture

CARLA is structured as a client-server architecture, where a central server exchanges information with one or more clients via a TCP connection. The CARLA server models the dynamics of the world (including that of the ego-vehicle and the other agents in the environment) and produces vehicle measurement and sensor data. A connecting CARLA client receives this measurement and sensor data and can send control commands to the server to change the state of the ego-vehicle. The CARLA client is also responsible for initializing the ego-vehicle in the CARLA environment and for setting the configuration of each sensor.

The CARLA codebase includes a special CARLA client called the CARLA-ROS bridge, which functions as an interface between the CARLA server and any ROS-compatible software framework. The CARLA-ROS bridge uses the CARLA client libraries to receive measurement and sensor data from the CARLA server, and publishes this data in a ROS-compatible format. Additionally, the bridge can subscribe to ROS control messages and use the CARLA client libraries to send this control information to the CARLA server. Figure 3-1 shows the relationship between the server, CARLA-ROS bridge, and the rest of our autonomous driving codebase, which is implemented using ROS.

3.1.2 CARLA Sensor Data

CARLA supports multiple sensor modalities, such as camera and multi-channel LIDAR, but we exclusively use the single-channel 2D LIDAR for the experiments in this section. The current implementation of LIDAR in CARLA generates individual ray traces for each LIDAR channel at constant angular increments to produce the full LIDAR point cloud. This cloud is represented as an unordered collection of points in $\mathbb{R}^3$, with points given in the local coordinate frame of the vehicle.

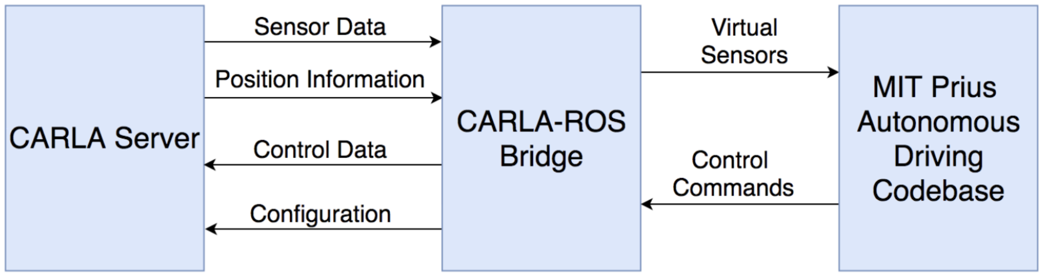


Figure 3-1: System diagram of relationship between CARLA server, CARLA-ROS bridge, and autonomous vehicle codebase.

3.1.3 CARLA Control

CARLA also enables external control of the ego-vehicle, in the form of unitless, normalized throttle and steer data that can be directly sent to the CARLA server. Because our autonomous driving codebase produces ROS commands in the form of steer (rad) and speed (m/s) values, we incorporated a module in the CARLA-ROS bridge to convert the ROS steer and speed values into CARLA throttle and steer data by applying proportional gains to both values. We note that applying a proportional gain to the ROS steer value (a velocity) to produce the CARLA throttle value (an acceleration) does not enable the CARLA vehicle to maintain the commanded velocity over time, and that a PID controller would be a more robust solution to calculate the desired acceleration (as exists in the development version of the CARLA-ROS bridge).

3.2 EKF-Based Localization

The first capability that we demonstrate with CARLA is localization, which allows our ego-vehicle to determine its pose in the world. We choose to use the existing Extended Kalman Filter (EKF) localization algorithm [26] in our autonomous driving codebase, which fuses information from three proprioceptive sensors - GPS, IMU, and wheel encoders - to produce a vehicle pose estimate. The EKF provides this pose estimate as a coordinate transformation between two coordinate frames: the `map` frame, which is a coordinate frame that is fixed at the initial position of the vehicle, and the `base_link` frame, which is a coordinate frame attached to the middle of the rear axle of the vehicle.

We first implemented a GPS sensor that simulates the true Oxford GPS sensor used on the Prius autonomous testing vehicle. To do so, we used the ground truth position of the vehicle with respect to a fixed global frame that is provided by CARLA.

Additionally, we implemented a simulated IMU sensor which provides the orientation, angular velocity, and translational acceleration of the vehicle. The node accesses orientation directly from the ground truth vehicle pose, and accesses the acceleration from the CARLA ego-vehicle measurement information. The angular velocity $\dot{\theta}$ is calculated by differentiating the estimated yaw angle at each time-step and is given by

$$\dot{\theta} = \frac{\theta_t - \theta_{t-1}}{\Delta t}$$

where θ_t is the yaw angle at time t and Δt is the simulation time-step.

Finally, we implemented a simulated wheel encoder sensor that provides the translational velocity of the vehicle in the x and y directions. The forward (x) speed is accessed directly from the CARLA ego-vehicle measurement information, and the tangential (y) speed is set to 0 because we assume that the vehicle does not slip.

Table 3.1 describes the inputs to the EKF provided by the three sensors.

Sensor	Fields
GPS	x, y
IMU	θ, $\dot{\theta}$
Encoder	$\dot{x}$, $\dot{y}$

Table 3.1: Correspondence between sensors and EKF state inputs.

3.2.1 Sensitivity to Proprioceptive Sensors

We now use CARLA to assess the sensitivity of EKF-based localization to the three proprioceptive sensors that we developed in the previous section. In order to determine how important each sensor is to maintaining an accurate pose estimate, we choose to assess the localization accuracy of the vehicle with different combinations of sensors. Because the EKF requires at least one source of position information and one source of orientation information in order to produce a valid pose estimate, we always include the IMU, which is the sensor that provides orientation information, and we include different combinations of the GPS and encoder sensors.

For our particular experiment, we measure how well the vehicle is able to maintain an accurate pose estimate while traversing a straight route in the Town02 CARLA environment. First, we record the ground-truth vehicle poses along this path by recording the output of the GPS sensor with no added noise (which simply transforms the ego vehicle transform into a GPS position). For each sensor combination, we then record the vehicle poses produced by the EKF as the vehicle traverses the path.

To model our prior uncertainty estimates for the sensors, we chose a yaw angle velocity covariance value of $\sigma^2_{\dot{\omega}} = 0.004$ and an x-velocity covariance value of $\sigma^2_{v_x} = 1.0$. All of the covariance values for the GPS message were set to 0, indicating that the GPS provides full ground-truth position.

We selected two representative translational and rotational accuracy metrics to evaluate different aspects of the localization accuracy for each sensor combination. Let $P_{gt} = \{p_{(j)}\}_{j=1}^{N_{gt}}$ be the collection of poses that form the ground-truth path and $P_d = \{x_{(j)}\}_{j=1}^{N_d}$ be the collection of poses that form the driven path. The metrics we used were:

1. Mean translational error:

$$E_{trans} = \frac{1}{N_d} \sum_{i=1}^{N_d} ||x_{(i)} - x^*_{gt,i}||_2$$

   where $x^*_{gt,i}$ is the closest point in the ground-truth path to $x_{(i)}$

2. Mean yaw angle error:

$$E_{yaw} = \frac{1}{N_d} \sum_{i=1}^{N_d} |\omega_{(i)} - \omega^*_{gt,i}|$$

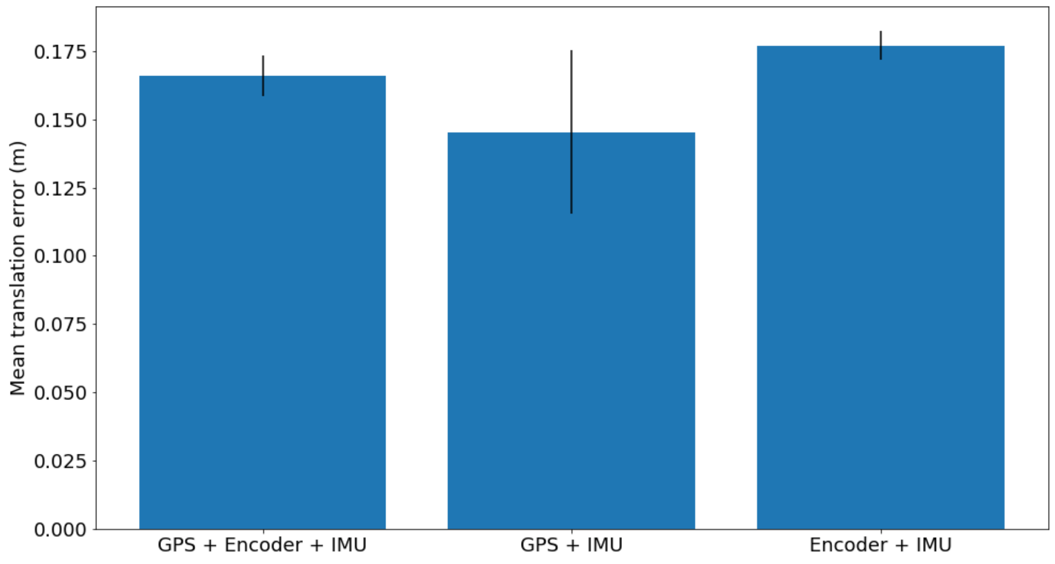


Figure 3-2: Mean translational error.

where $\omega^{*}_{gt,i}$ is the yaw angle of the closest point in the ground-truth path to $x_{(i)}$

Figures 3-2 and 3-3 show results for each of the two described metrics for three different sensor combinations: all three sensors (GPS, encoder, and IMU), just the GPS and IMU, and just the encoder and IMU. We conducted 3 repeated trials for each sensor configuration. We observe that using the GPS alone yields a lower translational error compared to the encoder alone, and that using all three sensors actually leads to an error that falls in between the former two error values. This difference is presumably due to the integration error that the EKF incurs when it infers the vehicle pose (x and y) from the discrete velocity estimates ($\dot{x}$ and $\dot{y}$) that come from the encoder. The yaw angle error values appear to suggest that using all three sensors results in the lowest error, and again that the GPS is marginally better than the encoder. This may suggest that both position (GPS) and velocity (encoder) data are needed to provide an accurate orientation estimate for the vehicle, although we note that the absolute magnitude of yaw angle error for all three sensor combinations is small (on the order of 10^{-4} rad).

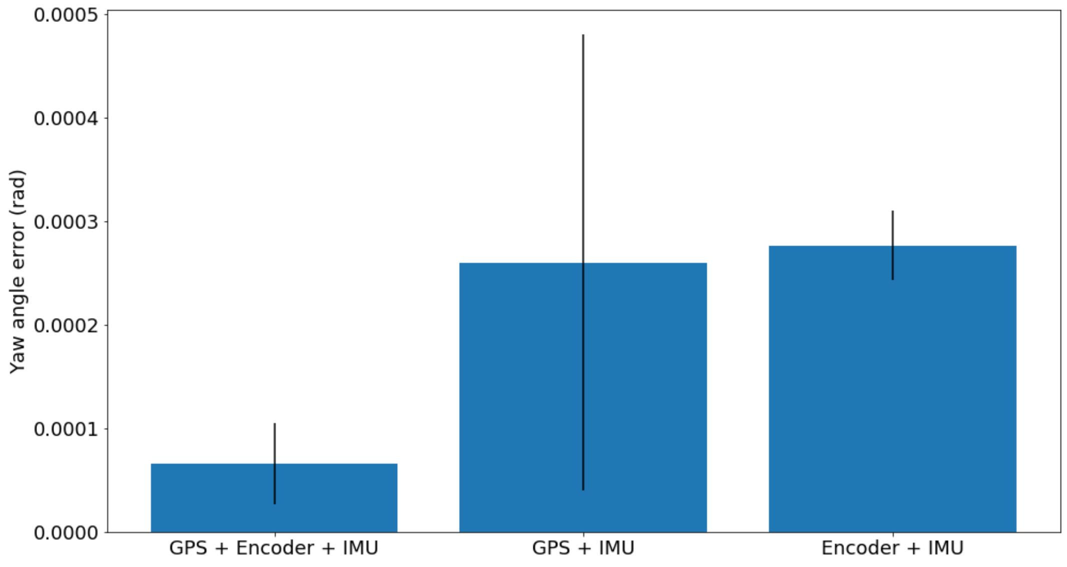


Figure 3-3: Yaw angle error.

3.3 Path Following Using a Pure Pursuit Controller

We now describe how we integrated the pure pursuit controller in our autonomous driving codebase with CARLA, both to demonstrate that the CARLA vehicle could autonomously follow a path in simulation and to validate its performance in simulation.

Pure pursuit [8] is a control algorithm which enables an agent to follow a predefined path. The algorithm takes as an input the path and the current pose of the vehicle relative to the path, and determines the arc of curvature that the vehicle should traverse in order to follow the path. The implementation of the pure pursuit algorithm in our codebase generates a planned path that follows this arc of curvature as well as the steering angle necessary to follow this arc.

Figure 3-4 shows the composition of our system with both the EKF and the pure pursuit controller connected to CARLA. In order to create this system, we provided the pose estimate from the EKF to the pure pursuit controller, along with a predetermined path. The pure pursuit controller generates both a path for the vehicle

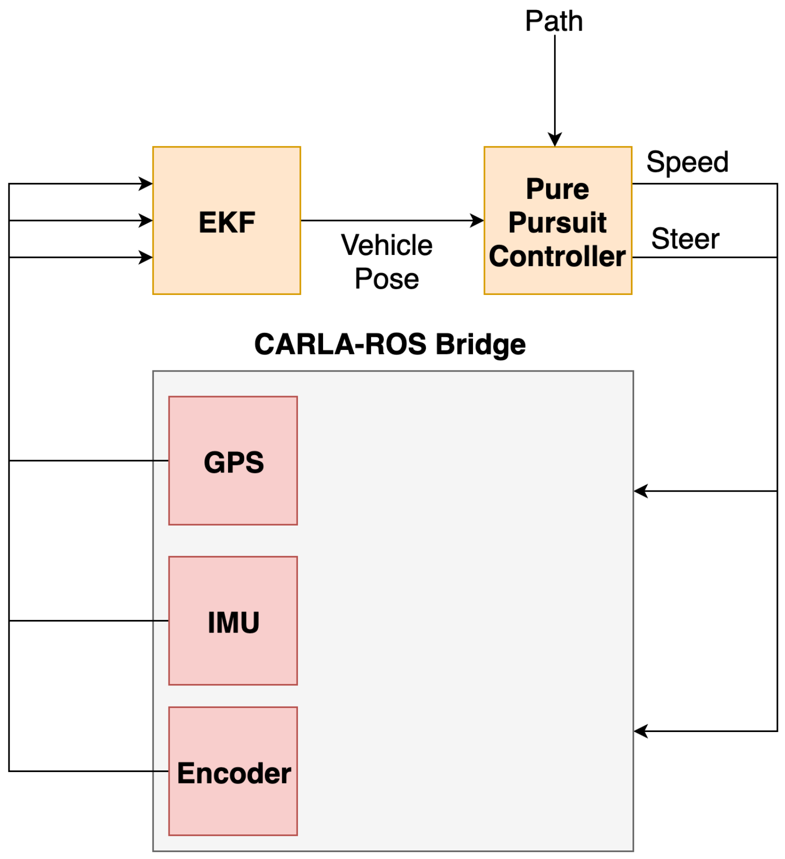


Figure 3-4: Block diagram showing the composition of the CARLA system integrated with the pure pursuit controller. Orange indicates modules that are existing components of our autonomous driving system, blue indicates modules that exist in CARLA, and red indicates our contributions to CARLA.

to follow and the necessary steering angle to follow this path, and it provides this steering angle to an existing speed controller (not shown in Figure 3-4) that calculates the desired speed for the vehicle to follow. These data are received by the CARLA-ROS bridge and are converted into CARLA control messages that are sent to the CARLA server to control the ego-vehicle.

3.3.1 Pure Pursuit Demonstration

Our demonstration of path following using pure pursuit in CARLA involved following a path in the built-in CARLA Town01 suburban environment, shown in Figure 3-5. This environment consists of single-lane roads separated by double yellow lines, which

Figure 3-5: Overhead view of CARLA Town01 environment.

are laid out in a rectangular grid, and includes suburban features such as buildings, houses, and driveways.

The demonstration path consists of a straight road segment followed by a right turn from start location 31 to start location 49, as shown in Figure 3-6. To generate this path, we drove the CARLA vehicle along the path and recorded the simulated GPS position of the vehicle at a frequency of 10 Hz along the path. We then demonstrated that the CARLA vehicle could follow this path using the pure pursuit controller using only its estimated position and orientation from the GPS, IMU, and wheel encoder. Figure 3-9 shows a visualization of the vehicle following the path.

3.3.2 Pure Pursuit Performance in CARLA

We use our simulation-based platform to validate the performance of the pure pursuit control algorithm in CARLA. We focus on investigating the sensitivity of the pure pursuit algorithm to its lookahead distance parameter, which controls the radius from the current position of the vehicle that pure pursuit uses to compute the desired steering command.

We first measure the accuracy sensitivity by using the mean translational error metric described in section 3.2 to quantify the discrepancy between the path driven by

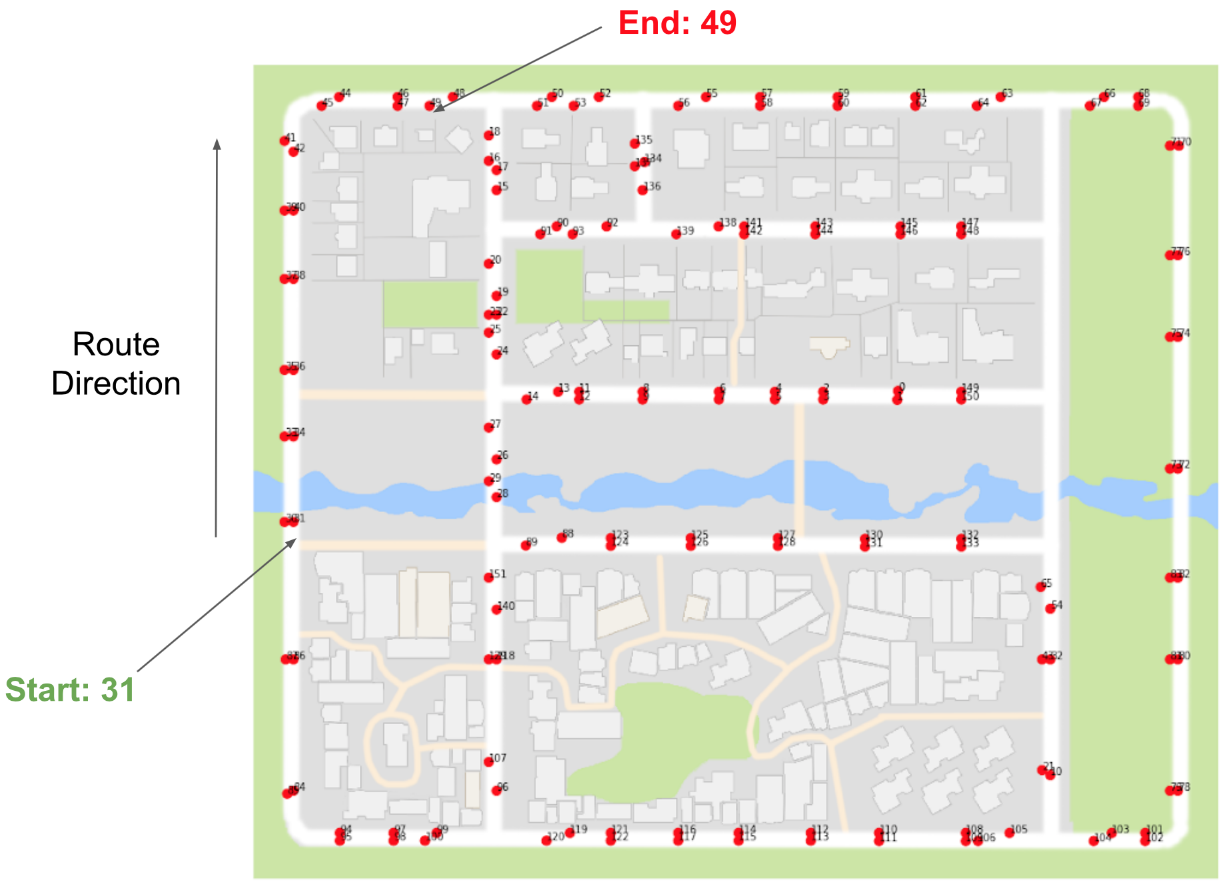


Figure 3-6: CARLA Town01 roadmap with start and end positions of the path indicated. Start and end position numbers are references to the internal position IDs of the Town01 map.

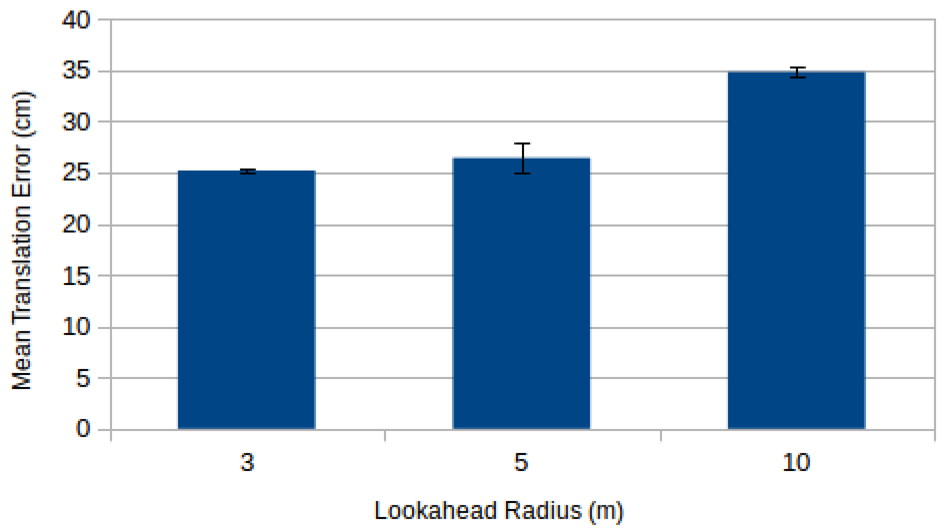


Figure 3-7: Mean translational error between vehicle path and pure pursuit goal path as a function of lookahead radius.

the simulated vehicle and the reference path that the vehicle is trying to follow. Figure 3-7 shows the mean translational error as a function of the pure pursuit lookahead distance, after conducting 3 trials for each lookahead radius parameter setting.

Our expectation was that a smaller lookahead radius would lead the vehicle to oscillate about the reference path but also follow the reference path more closely, leading to a lower overall translational error. In contrast, increasing the lookahead radius would lead the vehicle to traverse the path with fewer oscillations but would also lead to a more gradual convergence to the reference path and to turn undershooting, leading to a higher overall translational error. We observe that increasing the lookahead radius leads to an increased translational error, suggesting that the effect of the gradual convergence dominated the effect of fewer oscillations. It is also possible that we would observe an increase in translational error if we were to decrease the lookahead radius beyond the minimum value tested (3 m), depending on the limits of the steering ability of the vehicle.

We then focus on the sensitivity of the time taken to drive the complete path. Figure 3-8 shows the average time taken across 3 trials for each setting of the lookahead radius. We observe that the time required to traverse the path generally decreases as

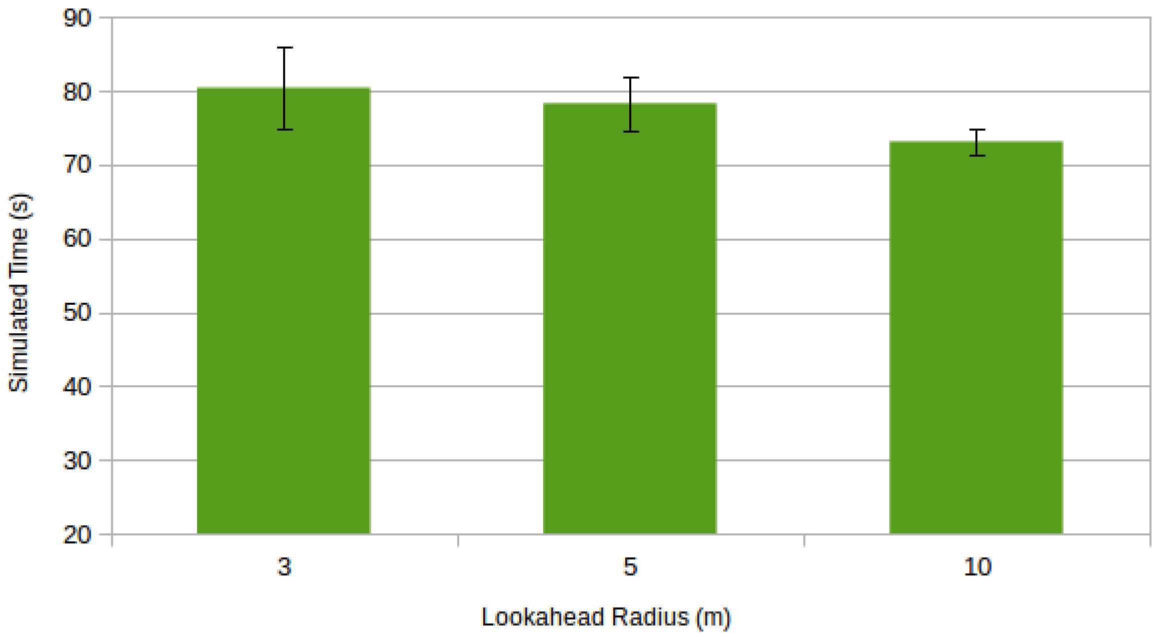
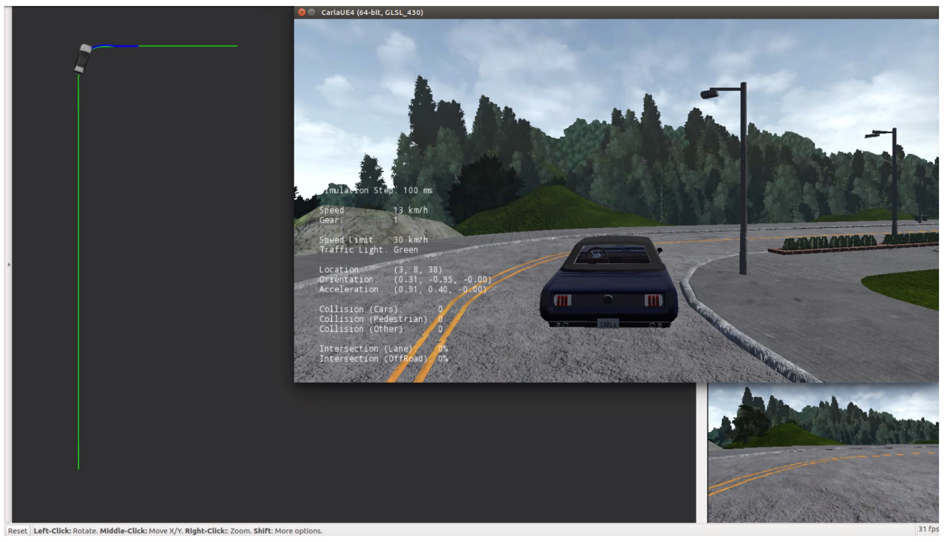


Figure 3-8: Mean simulated time to complete the pure pursuit path as a function of lookahead radius.



CarlaUE4 (64-bit, GLSL_430)
Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Zoom. Shift: More options.
31 fps

```
imulation Step: 100 ms

Speed:          13 km/h
Gear:           1

Speed Limit:    30 km/h
Traffic Light:  Green

Location:       (3, 8, 38)
Orientation:    (0.31, -0.95, -0.00)
Acceleration:   (0.91, 0.40, -0.00)

Collision (Cars):        0
Collision (Pedestrian):  0
Collision (Other):       0

Intersection (Lane):     0%
Intersection (OffRoad):  0%
```

Figure 3-9: RViz visualization of the CARLA vehicle following the challenge path. Reference path is shown in green, and pure pursuit goal path is shown in blue. Also shown are the a third-person view of the CARLA vehicle (upper right) and a first-person simulated camera view from the front of the vehicle (lower right).

a function of this radius. The most plausible hypothesis for this outcome is that the driven path corresponding to a lower lookahead radius contains an increased number of oscillations. Because the speed controller generates a speed that is inversely proportional to the steering angle of the vehicle, driven paths that contain oscillations are traversed in a longer amount of time. In addition, it is possible that the overall distance traveled is lower for a higher lookahead radius, which could result from both the decreased oscillations of the pure pursuit controller and the vehicle undershooting the turn. It is also interesting to note that increasing the lookahead radius decreases the uncertainty in the time required to traverse the path.

3.4 Static Obstacle Avoidance Using RRT* Path Planning

Finally, we demonstrate that a path-planning algorithm can successfully be integrated with CARLA to avoid static obstacles. For our demonstration, we use the RRT* planning algorithm [21], which is a planning algorithm that determines a feasible path for an agent to follow based on the kinematic constraints of the vehicle and the external features of the environment, such as obstacles. RRT* works by growing a tree outwards from the start point of the vehicle by randomly sampling points from the feasible space and connecting them to the tree based on the constraints of the vehicle.

The specific implementation that we tested comes from OMPL [43], an open-source motion planning library written in C++. We demonstrate that the RRT* planner can take 2D simulated LIDAR data from CARLA, a reference path, and the estimated current pose of the vehicle to enable the simulated vehicle to plan and navigate a path that successfully avoids static obstacles. Figure 3-10 illustrates the fundamental components of the RRT* planning module. The RRT* planner first requires a costmap to be generated, which is a map that associates costs with each point in space in a neighborhood of the current position of the vehicle. This costmap

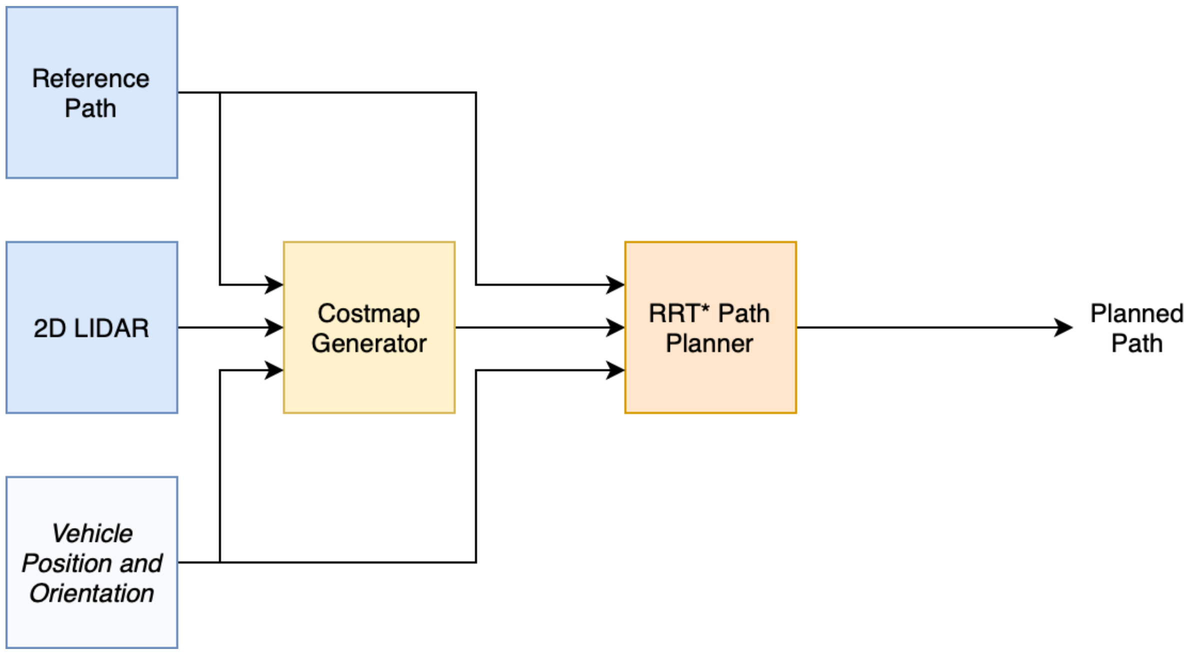


Figure 3-10: Block diagram of RRT* system components.

is built by overlaying multiple layers that depend on a combination of local sensor data and global information about the environment. In particular, the costmap generates an initial segmentation of the surroundings into drivable regions by marking all points that are a certain distance from the reference path as drivable. It also uses the 2D LIDAR to implicitly detect static obstacles and mark the regions that the 2D LIDAR hits as obstacles.

Figure 3-11 shows how the RRT* nodes interface with the CARLA-ROS bridge and with the localization and control modules described in the previous section.

3.4.1 RRT* Demonstration

We first demonstrate our successful integration of CARLA with the RRT* planner by guiding the vehicle to avoid three parked cars in a minimal environment consisting of a straight, single-lane road segment separated by a double yellow line. Figure 3-12 shows a rendering of this environment with three parked cars located in the right lane, along with a visualization of the costmap and RRT* nodes.

We initialized the vehicle in the right lane, and our goal was to show that the

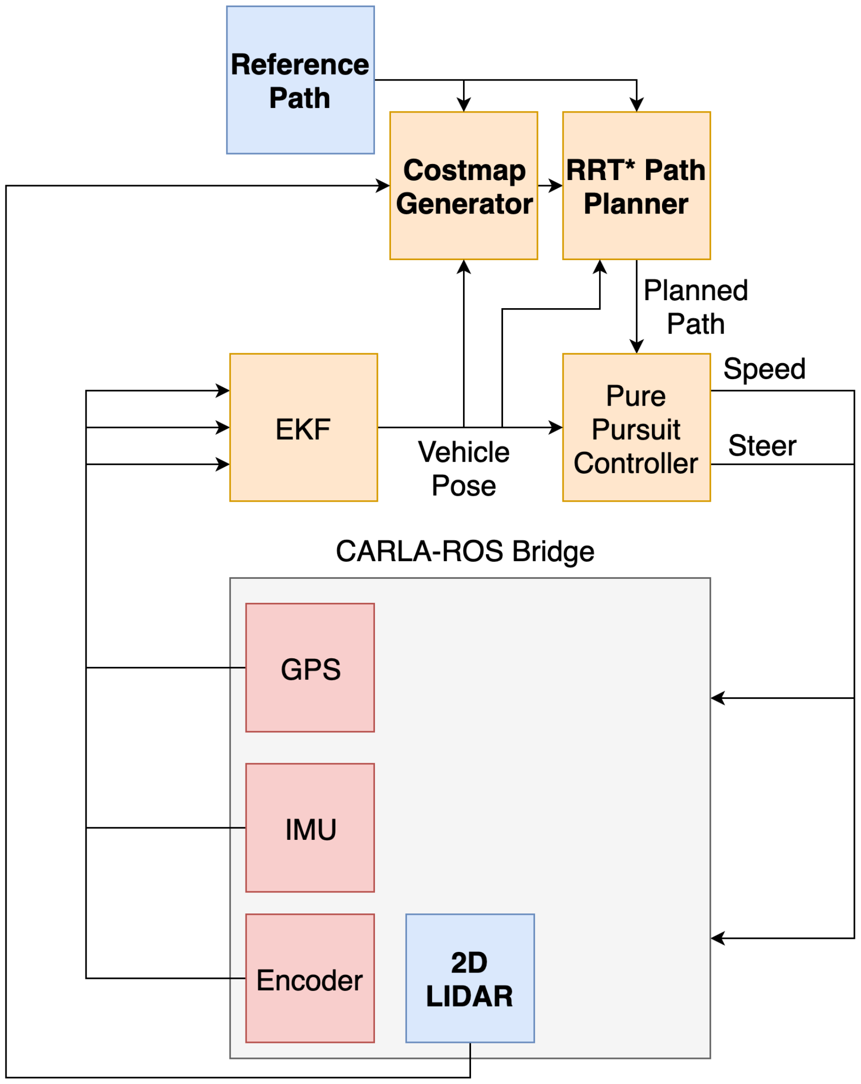
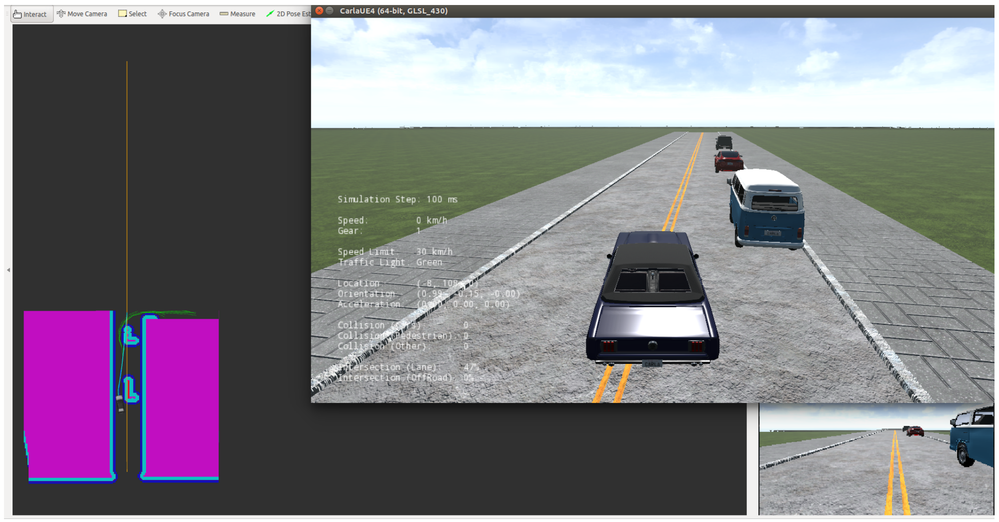


Figure 3-11: Block diagram showing the composition of the CARLA system integrated with RRT* path planning. Orange indicates modules that are existing components of our autonomous driving system, blue indicates modules that exist in CARLA, and red indicates our contributions to CARLA.



```
Simulation Step: 100 ms

Speed:          0 km/h
Gear:           1

Speed Limit:    30 km/h
Traffic Light:  Green

Location:       (-8, 108, 0)
Orientation:    (0.99, -0.15, -0.00)
Acceleration:   (0.00, 0.00, 0.00)

Collision (Cars):          0
Collision (Pedestrian):    0
Collision (Other):         0

Intersection (Lane):       47%
Intersection (OffRoad):    0%
```

Figure 3-12: RViz visualization of the CARLA vehicle avoiding static obstacles in CARLA-RRT* demonstration, with reference path (yellow) and costmap (pink and blue). Upper right shows an overhead view of two-lane road environment, lower right shows first-person camera view from ego-vehicle.

vehicle could avoid these static obstacles by temporarily switching into the left lane when necessary. In order to guide the vehicle to stay in the right lane by default, we provided a reference path that corresponded to the middle of the right lane. We generated this reference path by driving the vehicle down this lane and recording the GPS position of the vehicle at 10 Hz, in a similar manner to what was done for the pure pursuit demonstration in section 3.3.2.

3.4.2 RRT* Performance Assessment

Our first experiment to validate the ability of RRT* to avoid static obstacles in CARLA measured the minimum distance between the CARLA vehicle and the static obstacles as it traversed through the minimal CARLA environment. For each trial, we recorded the path driven by the CARLA vehicle, consisting of a successive collection of vehicle pose estimates (derived from the `map` to `base_link` transform produced by the EKF). We extracted the ground truth positions of the static obstacles from the Unreal Editor. We defined the minimum distance between the path and the static obstacles to be the minimum distance between any point on the path and any of the 2D rectangular bounding boxes of the vehicles.

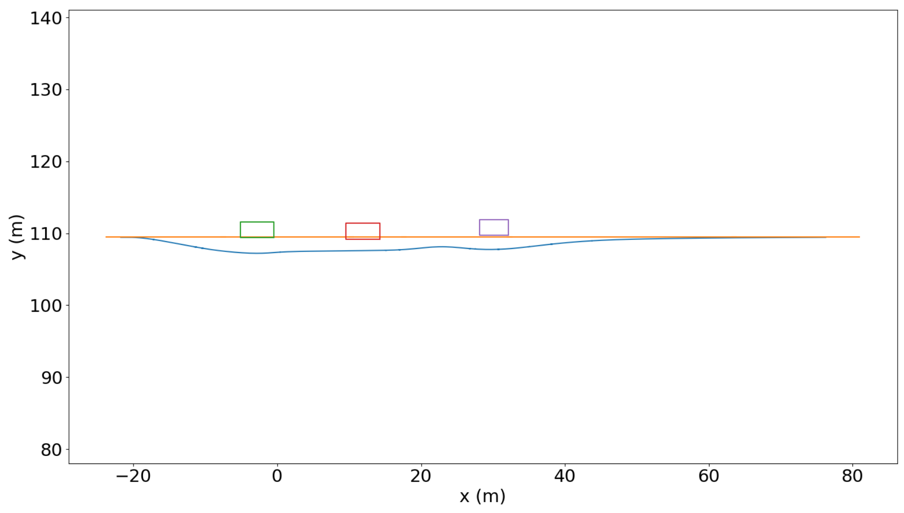


Figure 3-13: Visualization of one challenge path (blue), reference path (orange), and bounding boxes of static obstacles.

After conducting 5 repeated trials of the vehicle navigating through the static obstacle field, we found that the minimum distance from the CARLA vehicle to the static obstacles was 1.52 ± 0.12 m, and the time taken to complete the challenge was 53.96 ± 1.39 s. The average speed of the vehicle while traversing the challenge route was 2.24 ± 0.14 m/s.

Figure 3-13 shows a typical vehicle path, the reference path, and the bounding boxes of the parked cars.

In order to show that CARLA can be used to validate a characteristic of RRT* that would be difficult to assess in the physical world, we assessed the reactivity of the RRT* planner. To do so, we created an environment similar to the one used in the demonstration, which instead had a single static vehicle parked on the road. We then measured the minimum distance from the ego-vehicle to the static vehicle as it traversed the environment, for a range of maximum velocities [1] and starting positions of the ego-vehicle relative to the parked car. Our goal was to assess how sensitive this distance would be to both the velocity and start position. Figure 3-14 shows the

[1]Because the commanded velocity value is converted into an acceleration value for controlling the CARLA vehicle, as described in section 3.1, the maximum velocity parameter is effectively a maximum acceleration parameter for the CARLA vehicle.

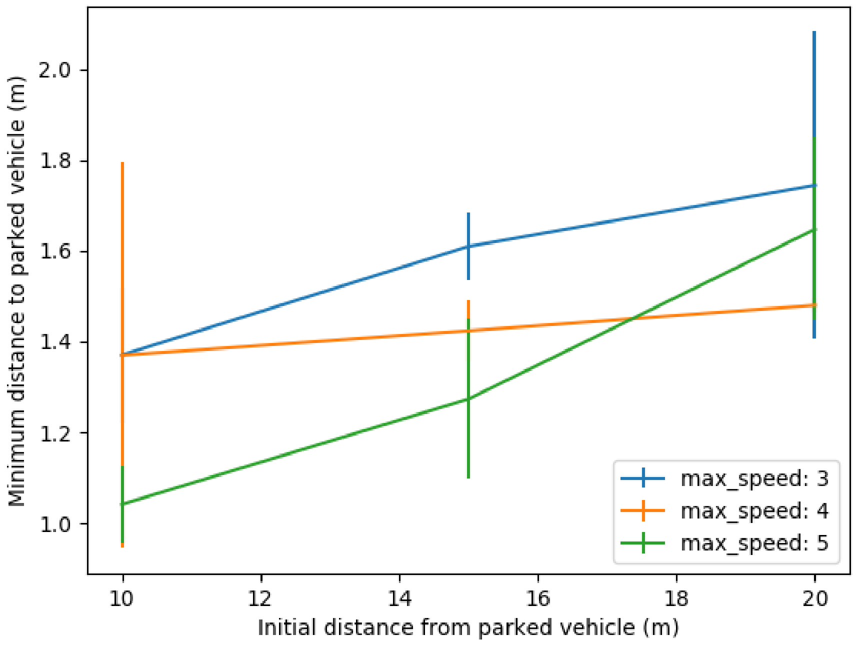


Figure 3-14: RRT* reactivity plot.

results of the experiment for 3 different initial distances and 3 different maximum speed values. We note that an increased initial distance is generally associated with an increased minimum distance from the static vehicle, which is plausible because this increased distance may cause the vehicle to plan a route around the vehicle further in advance. We also note that an increased maximum speed value leads to decreased distances to the static vehicle, which is also plausible because the increased speed limits the ability of the controller to avoid obstacles.

Chapter 4

Validation of MapLite Driving Framework in CARLA

In the previous chapter we demonstrated that we could develop CARLA to be a useful tool for validating fundamental autonomous driving capabilities that are core elements of any autonomous driving system. However, one of our primary goals is to develop CARLA into a platform for testing and validating autonomous driving research algorithms.

We chose to use the MapLite autonomous navigation framework as a case study for validating an autonomous vehicle research algorithm in simulation. We begin by describing the fundamental components of the MapLite navigation framework. Next, we detail the integrated system that includes CARLA and the MapLite framework. Finally, we demonstrate that CARLA can be used to validate the performance of MapLite by presenting results that assess the consistency of MapLite and its differential performance on straight road segments and intersections.

4.1 MapLite Overview

MapLite [29] is an autonomous navigation framework which enables an autonomous vehicle to navigate through a road environment, given only local sensor data and a sparse topological map of the road network. The abstract formulation of the framework decomposes the navigation problem into three problems: localizing the vehicle on the road network, determining a route in the topological map from the current vehicle position to a destination, and traversing an edge of the road network. The instantiation of the original MapLite algorithm [29] focused primarily on the edge traversal problem, which consisted of a road segmentation module that detected the road boundaries and a trajectory generation module that would then determine a feasible trajectory for the vehicle to drive in free space. However, this instantiation of the MapLite algorithm was not able to navigate on road segments that included intersections, and it was also not equipped to handle point-to-point autonomous navigation because it did not explicitly process the full OSM map.

This motivated the next iteration of the MapLite algorithm, which consists of three core components, shown in Figure 4-1. The first component is the road segmentation module, which classifies the points in an incoming 3D LIDAR point cloud that belong to the drivable road surface. The model used for road segmentation can be any supervised classifier, such as an SVM or a CNN, that can be trained on LIDAR data to provide these pointwise LIDAR road labels. This pointwise road determination is then processed by the map warping module, which deforms the global topological map to be locally metrically accurate based on the drivable road surface. The deformation is performed using a non-linear optimization that determines the optimal transformation of the base topological map that matches the local road information provided by the road segmentation module. Once the topological map has been warped, it is used by the route planner module to determine a path from the current position of the vehicle to an arbitrary destination pose in the space of the graph. The route planner uses Dijkstra's algorithm to find the shortest-cost path from the source node to the destination node.

4.2 MapLite - CARLA Integration

In this section, we describe the contributions that we made to both CARLA and MapLite in order to build an integrated platform that allows for the validation of

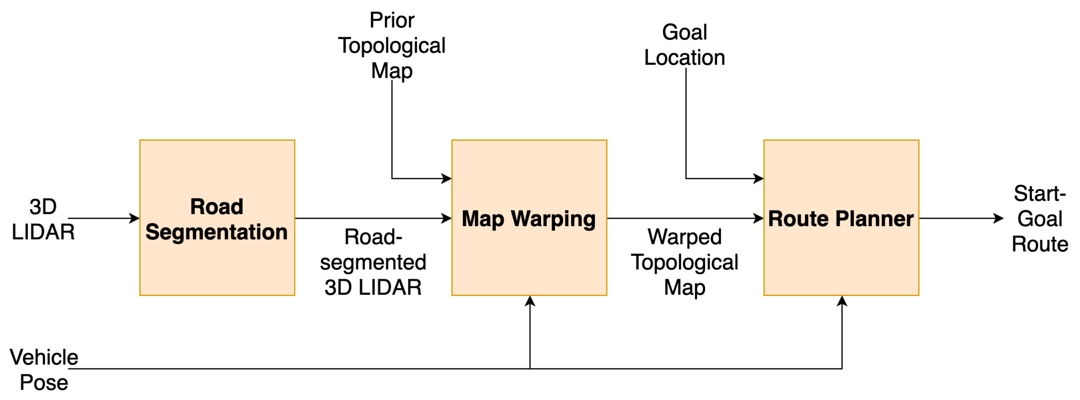


Figure 4-1: Block diagram of MapLite system with road segmentation, map warping, and route planner modules.

MapLite in simulation.

4.2.1 CARLA contributions

Our first contribution to CARLA was to incorporate ground-truth LIDAR road segmentation information into the existing CARLA LIDAR module. We extracted the road identities of the individual ray traces that compose the CARLA LIDAR by accessing a built-in feature of the CARLA server that tags every object in the CARLA environment with its semantic identity. We then modified the LIDAR message types in the CARLA server, the CARLA Python client API, and the CARLA-ROS bridge to accept a new field that contained binary road classifications for each point.

As an ancillary contribution to make the format of the CARLA LIDAR data more realistic, we added fields for both the ring ID and the intensity (the latter being a dummy value because the CARLA LIDAR ray traces do not return intensity information about the objects that they hit). We also assigned 0-range values to the LIDAR returns that were out-of-range rather than excluding them from the point cloud altogether in order to match the data format of real-world LIDAR sensors.

Our second contribution to CARLA consisted of converting the existing topological map from the CARLA environment, which included directed edges corresponding to both sides of the road, into an undirected topological map that contained a single edge for each distinct road segment in the graph. We implemented this conversion in order to make the CARLA topological map consistent with the undirected Open Street Map graphs. Our conversion algorithm, which takes as input a graph G and a distance tolerance ϵ and produces a condensed graph, is presented below:

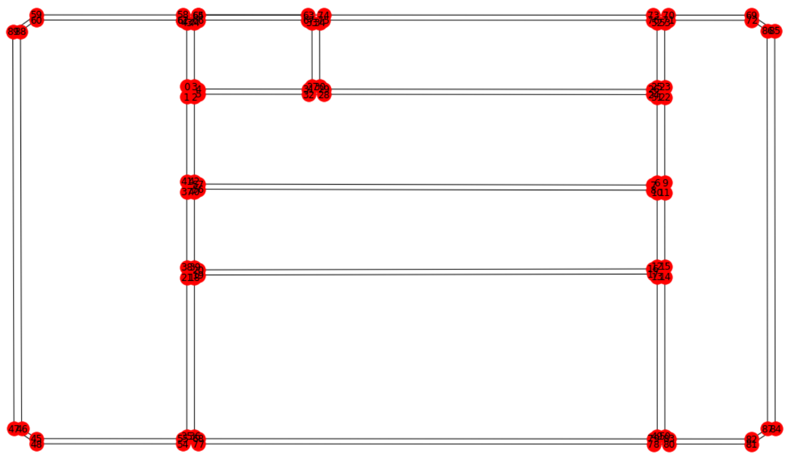
```
function CondenseGraph(G, $\epsilon$)
    while there are nodes to condense in G do
        for node $n$ in G do
            $C$ = GetCloseNodes($n$, $\epsilon$)
            if $|C| > 0$ then
                CombineNodes($n$, $C$)
            end if
        end for
    end while
end function
```

Figure 4-2 compares the original unidirectional map to the processed bidirectional map of the Town01 CARLA environment.

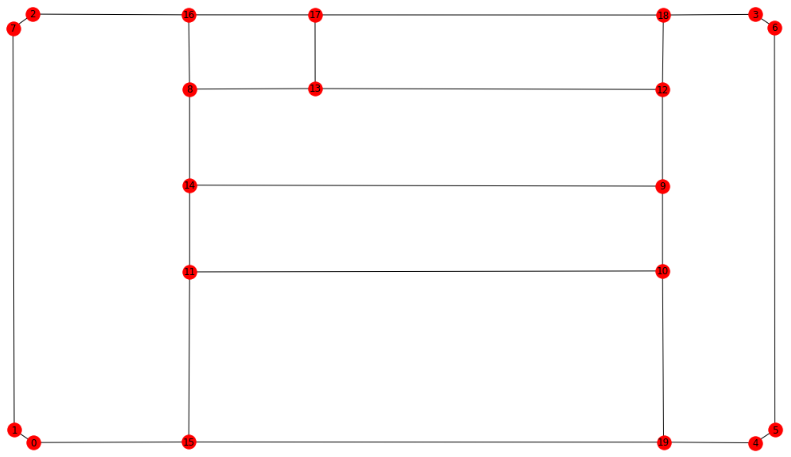
4.2.2 MapLite contribution: stopping at intersections

When we tested our the initial iteration of the intersection-sensitive MapLite algorithm in the Devens rural road testing environment, we found that the vehicle would often fail to clear intersections. This occurred because the time required for the map registration algorithm to update the topological map as it drove through the intersection would exceed the reaction time necessary for the vehicle to plan an updated route that did not violate the constraints of the intersection. To address this issue, we incorporated a routine into the route planner that allows the vehicle to pause at intersections before proceeding along the planned route. This pausing behavior would allow the map registration algorithm to converge to a steady-state topological map before the vehicle proceeds through an intersection.

The intersection stopping routine is implemented on top of the existing route planner as a finite state machine with two states - one to indicate that the vehicle is



(a) Directed (unidirectional) map



(b) Undirected (bidirectional) map

Figure 4-2: Comparison of (a) directed and (b) undirected topological maps of Town01 CARLA environment.

at an intersection (which we will denote `at-int`), and one to indicate that it is not (denoted `not-at-int`). The routine starts in the `not-at-int` state and transitions to the `at-int` state if the total length of the path from the current vehicle pose to the next upcoming intersection is lower than a threshold stopping radius r_{stop}. If this is the case, the route planner will also cease publishing a new path for a duration of t_{stop} seconds. Once in the `at-int` state, the planner will determine whether it has successfully traversed the intersection by checking to see whether the next upcoming intersection in the route is within an intersection tolerance radius r_{tol} of the previously stored intersection pose. If this is the case, the planner sets the goal route to terminate at the intersection immediately *after* the current intersection. If not, the planner transitions to the `not-at-int` state and updates the stored intersection pose to be the new intersection.

This modification to the route planner assumes that the vehicle can actually come to a complete stop before the center of the intersection, but this depends on the behavior of the selected speed controller. If the vehicle is unable to come to a complete stop before the intersection, then the vehicle executes a rolling stop and continues driving towards the next intersection.

4.2.3 Integration

We combined the aforementioned contributions to CARLA and MapLite with the core CARLA-ROS bridge, EKF (without GPS), and pure pursuit modules described in chapter 3 to complete an integrated system that allowed us to simulate the full behavior of MapLite in CARLA. Figure 4-3 shows the composition of the full CARLA-MapLite system, which uses proprioceptive (IMU and encoder) and exteroceptive (LIDAR) simulated sensors along with the global topological map to navigate the vehicle in simulation towards a user-specified goal pose.

One key difference between the integrated system presented here and the standard MapLite algorithm is that the CARLA system does not use a learned model to compute road segmentation directly on the 3D LIDAR, but rather uses ground-truth information to produce that segmentation information. We chose to use this ground

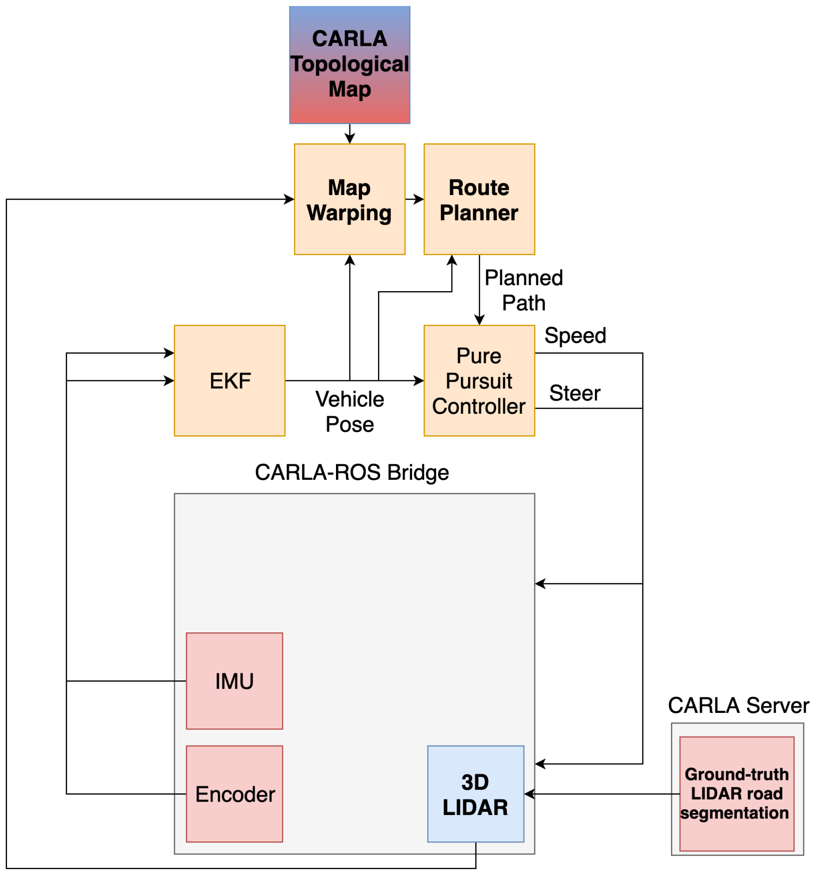


Figure 4-3: Block diagram showing the composition of the integrated CARLA-MapLite system. Orange indicates modules that are existing components of our autonomous driving system, blue indicates modules that exist in CARLA, and red indicates our contributions to CARLA. The topological map is shaded in red and blue to indicate that it was a modification of an existing CARLA feature, rather than a purely novel contribution.

truth information, rather than re-using a road segmentation model from real LIDAR or training a new road segmentation model on the CARLA LIDAR data, because the models trained on real LIDAR data were unlikely to generalize to simulated LIDAR data and vice versa (due to the qualitative differences between CARLA and real LIDAR data). This design choice means that our system is primarily a testing and validation platform for the map warping and route planner modules of MapLite. However, one could introduce noise into the ground-truth road segmentation or train a model on CARLA LIDAR data to extend our system to act as a validation platform for the road segmentation module.

Compared to the demonstration of RRT* planning in chapter 3, we note that the route planner produces a planned path to a particular goal destination, while RRT* produced a local planned path that directed the vehicle to drive towards a global reference path. The map warping node is comparable to the costmap node, in the sense that both provide metrically-accurate but differing representations of the road environment. The costmap node provided an actual metric map (in an occupancy grid format) of the local road environment, whereas the map warping node provides a metrically-accurate sparse topological map of the local road environment (which is a subgraph of the global topological map).

4.3 Performance assessment of MapLite in simulation

We now use the integrated system to assess the performance of MapLite in simulation. We begin by assessing the consistency of the path traversed by MapLite as it proceeds from a fixed starting position to a fixed end position, and we then compare the performance of MapLite on straight road segments to its performance on intersections. We assess the performance of MapLite by comparing the MapLite path traversals to a reference ground-truth path produced by a human manually driving through the simulated CARLA environment. The ground truth path was generated by manually driving the CARLA vehicle using the joystick controller along the center of the road, which in the case of the CARLA Town01 environment corresponds to the double yellow lines that separate each lane. In addition, we compare the performance of MapLite to the performance of a baseline path-following method which uses GPS information to follow a fixed path from the start position to the end position using pure pursuit, which is identical to the approach described in Section 3.3.

4.3.1 Consistency of MapLite

We first assess the consistency of the path traversed by MapLite as it proceeds from a fixed start position to a fixed goal position. We measure consistency by assessing the average accuracy of a collection of traveled MapLite paths compared to a ground-truth reference path, as described earlier. In order to assess the consistency of MapLite, we measure the average deviation from the ground-truth reference path using the mean translational error and mean yaw angle error metrics described in Section 3.2.

We recorded 10 trials of MapLite driving in the CARLA Town01 environment from a fixed start location to a fixed goal location, which we selected to specify a path that included a mix of straight road segments and intersections. Figure 4-4 shows the scenario for this experiment, including the start and goal locations and the initial path in the Town01 topological map. For each trial, we recorded the transform between the `devens` and `base_link` coordinate frames (derived from the transform between the `map` and `base_link` frames produced by the EKF) at regular intervals as the vehicle traversed the route from start to goal. Figure 4-5 shows the 10 trial paths, along with the ground-truth path.

Note that trials 1 and 4 resulted in collisions at the first intersection. These paths do not extend to meet the end of the ground-truth path but rather are truncated at the intersection where the vehicle collided with a static environmental feature. It is not entirely certain whether these collisions were due to fundamental issues with MapLite, or whether these collisions arose because of performance limitations of the hardware platform that we used to run our experiment. Note also that none of the trial paths terminate at exactly the goal location because the speed controller

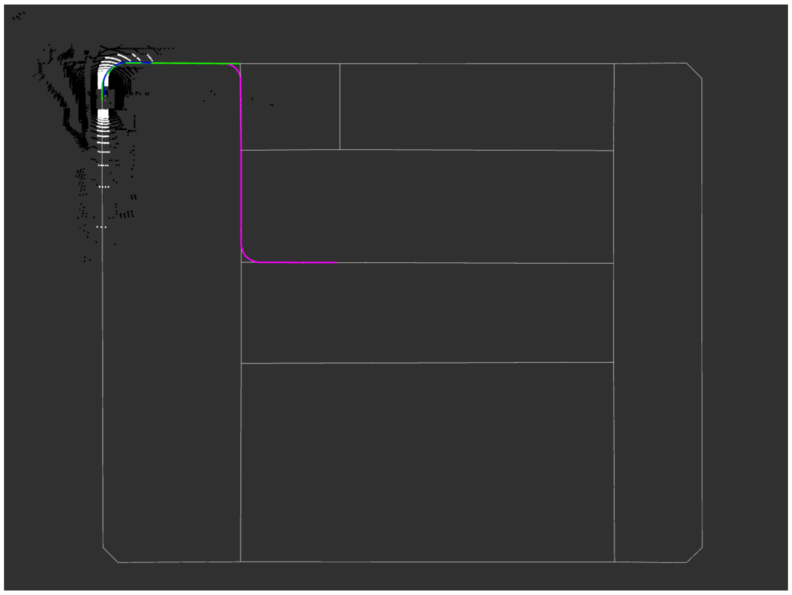


Figure 4-4: Overhead view of MapLite consistency experimental scenario, with initial Town01 topological map (gray), initial planned global route (purple), local route to the first intersection (green), and the start position indicated by the vehicle indicator. Also shown is the ground-truth road segmented LIDAR scan, with road points (white) and off-road points (black).

commands the vehicle to stop once the vehicle reaches a set distance away from the goal position (for this experiment, the stopping distance was set to 20 m).

In order to create the ground-truth reference path, we used the joystick controller module in our autonomous driving codebase to manually drive the CARLA vehicle along the double-yellow lines of the CARLA map, which corresponds to the center of the road (because the MapLite topological map is undirected). We recorded the ground truth path by recording the transform between the `devens` and `base_link` coordinate frames, with only the GPS sensor enabled to provide ground-truth position information, at periodic intervals.

For the GPS path following baseline, we generated the fixed route from the start position to the goal position by taking the initial full route planned by the route planner module of MapLite, and converting the poses in this route to be transforms between the `devens` and `base_link` coordinate frames. We then recorded 10 trials of the GPS following baseline driving in the same CARLA environment and recorded the transform between the `devens` and `base_link` coordinate frames to generate each

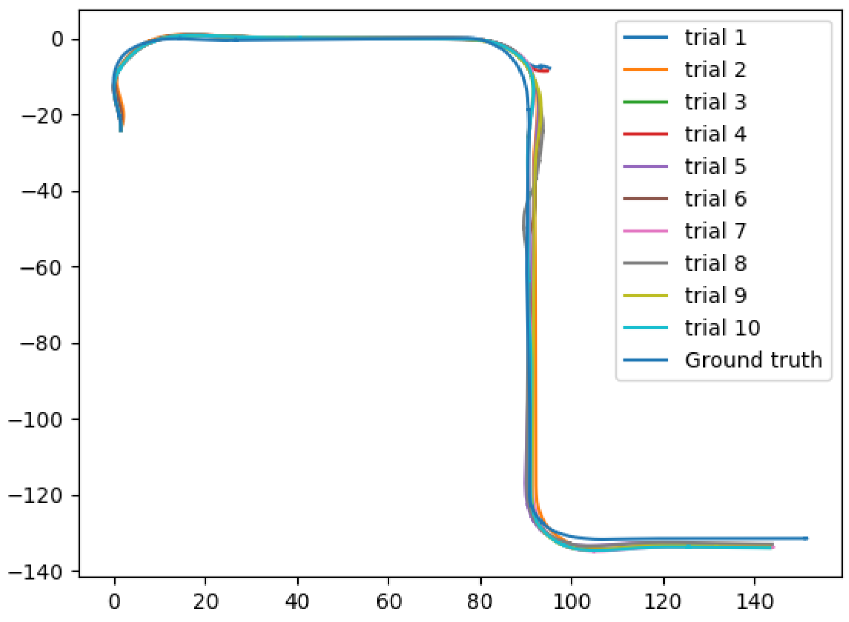


Figure 4-5: Visualization of 10 MapLite trajectories from the same start and goal positions, with ground-truth trajectory shown in blue. Scale is in meters.

trial path. We measured the consistency of the GPS path following baseline in the same manner as MapLite, in which we compared the path followed by the baseline to the same ground-truth reference path as MapLite using the same two accuracy metrics. Figure 4-6 shows the 10 recorded trajectories from the GPS baseline.

Figure 4-7 compares the distribution of accuracies between MapLite and the GPS baseline over all 10 trials for each of the four metrics. We note that both the uncertainty and the absolute error of the paths traversed by MapLite are significantly larger than the corresponding values for the paths traversed by the GPS baseline. Because we are using ground-truth information about the CARLA environment to produce the road segmentation output, our comparison seems to reveal that the performance of MapLite is highly dependent on the efficiency and accuracy of the map warping node, especially as the vehicle traverses intersections. However, despite the fact that there were two incomplete trials that resulted in vehicle collisions, both metrics reflect relatively low absolute error values across all trials for MapLite (although the mean

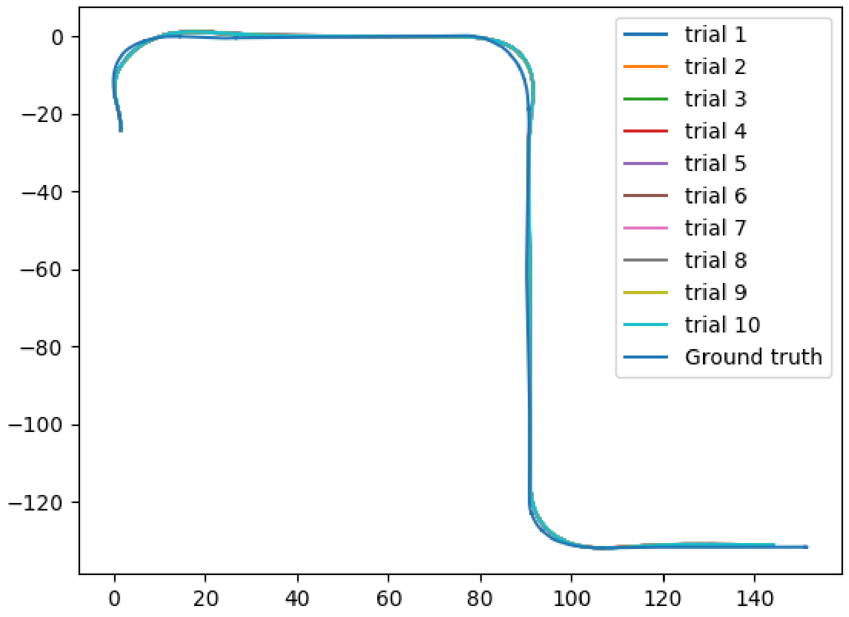


Figure 4-6: Visualization of 10 GPS baseline trajectories from the same start and goal positions, with ground-truth trajectory shown in blue. Scale is in meters.

translational error metric is still large in magnitude, presumably because of the large variance in trajectories observed near the first intersection). This is because both are local error metrics and thus will not take into account whether the vehicle was actually able to complete the route - only whether the vehicle remained close to the ground-truth path for the duration of its driven route.

4.3.2 Straight vs. Intersection Performance of MapLite

Our next goal was to use CARLA to investigate any differences between the performance of MapLite on straight road segments and its performance at intersections.

To assess the performance on straight road segments and intersections, we used the same 10 trials of MapLite and the GPS baseline in the Town01 environment as in the previous experiment. We first partitioned each road trial into straight and intersection regions by extracting the known poses of the intersections along the path from start to goal and marking every pose in the trial path within a specified radius

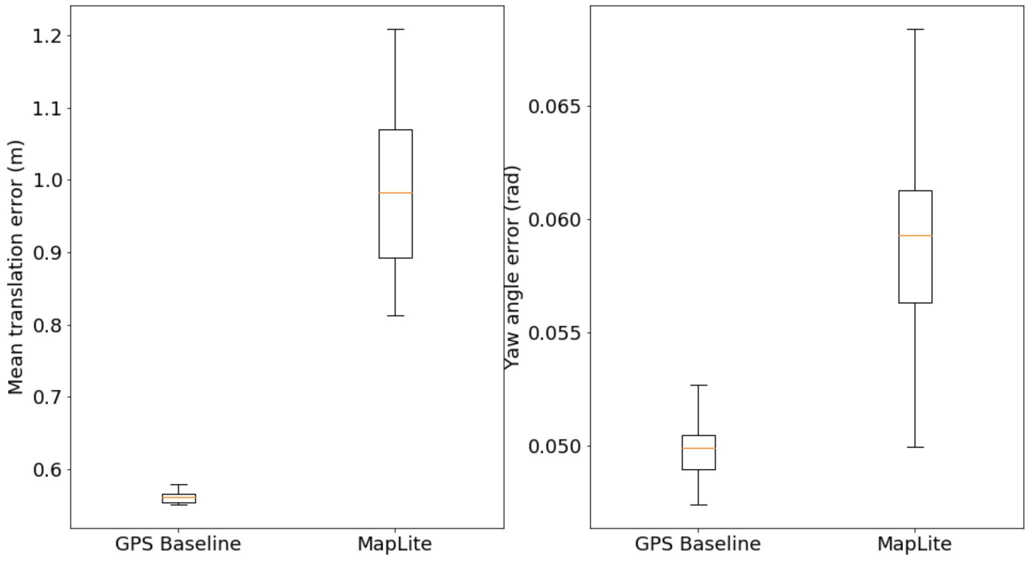


Figure 4-7: Mean translational error and mean yaw angle error for MapLite trajectories and GPS path-following baseline trajectories, compared to ground-truth human trajectory in CARLA.

r_{int} of these intersection poses as belonging to the intersection region. An example partition of a driven route into these regions is shown in Figure 4-8. We then assessed the accuracy of each trial on the two regions separately using the two accuracy metrics used in the previous experiment. Figure 4-9 compares the performance using these metrics across intersections and straight road segments for MapLite and the GPS path-following baseline.

We again observe that the MapLite trajectories have a consistently higher uncertainty and absolute error compared to the GPS baseline across both intersections and straight road segments. At intersection regions, the performance of MapLite is highly sensitive to the quality of the warped map and to the efficiency of the map warping correction as it receives the correct ground-truth information about the road environment. Our hypothesis is that the efficiency of the map warping algorithm is the primary bottleneck and leads MapLite to follow trajectories that deviate significantly from the ideal trajectory at intersections, whereas the GPS baseline follows a fixed trajectory at intersections that is determined only by the lookahead radius of the

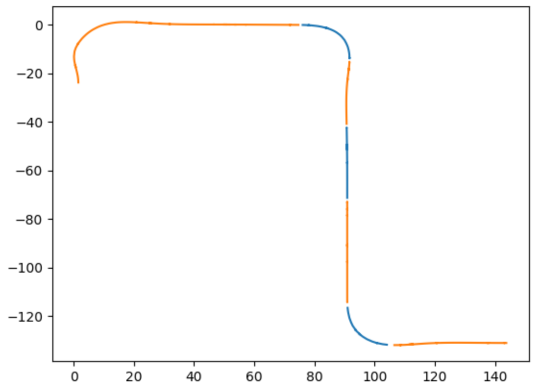


Figure 4-8: Partition of MapLite driven route into regions corresponding to straight regions (orange) and intersection regions (blue).

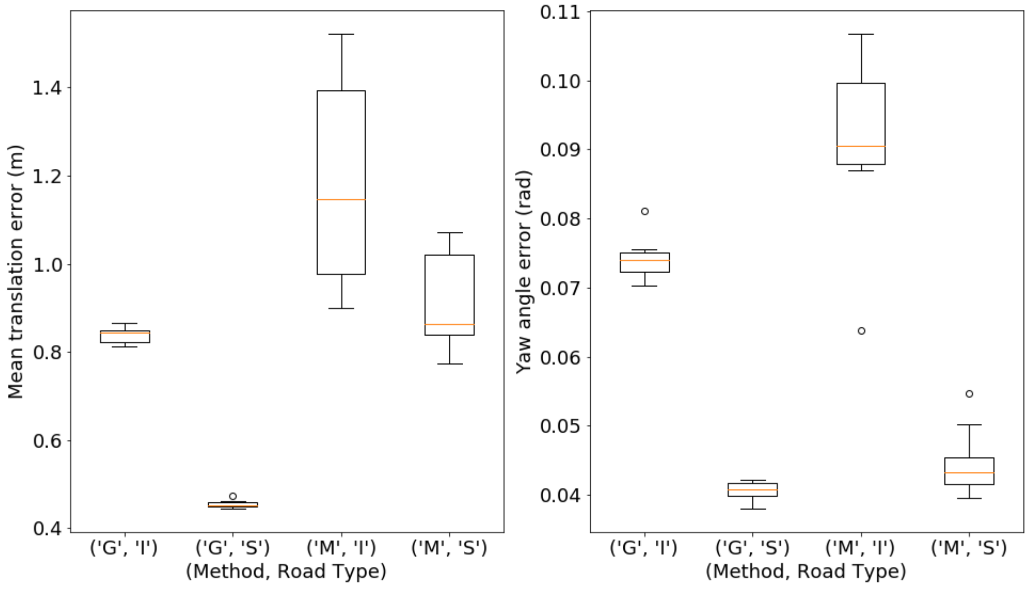


Figure 4-9: Mean translational error and mean yaw angle error for MapLite ('M') and GPS baseline ('G') trajectories compared to ground-truth human trajectory in CARLA, across intersections ('I') and straight road segments ('S').

pure pursuit controller. The reason for the discrepancy across straight road segments is not as clear, although Figure 4-5 suggests that there is a large variance between the traversed straight segments of MapLite after the first intersection. We observe that the error induced upon traversing the first intersection persists throughout the remainder of the trajectory. Even at the end of the trajectory, there appears to be a systematic steady-state error, as all of the traversed MapLite paths are laterally offset from the ground-truth reference path. This steady-state error may also be a limitation of the map warping algorithm, which may have found an optimal transformation of the map that lies within the road region but is not aligned with the true centerline of the road.

Our results for both MapLite and the GPS baseline agree with our prior expectations that the mean translation error and the mean yaw angle error would be higher in the intersection regions compared to the straight regions, even with the intersection stopping routine, because the performance of the map warping algorithm becomes more critical as the vehicle is driving through the intersection (especially when the vehicle is making a turn). In straight road segments, the latency of the warped map should not be as critical because the warped map should converge to a steady-state map as the vehicle drives along the segment, although this steady-state map may not be optimal.

Chapter 5

Dynamic Obstacle Avoidance in CARLA

In chapter 3, we showed that CARLA could be used to validate the ability of the RRT* path planning algorithm to avoid static obstacles by building an integrated platform between CARLA and RRT*. This planning-based approach relied on a costmap representation of the drivable road surface that combined a road width prior with local perception information from the 2D LIDAR sensor. However, this approach alone cannot navigate successfully in an environment with dynamic obstacles because the costmap generation procedure makes no distinction between 2D LIDAR readings that originate from static and dynamic obstacles in the environment.

Therefore, our objectives in this section were two-fold: first, to develop an approach that leveraged existing modules in our autonomous driving codebase to address the problem of dynamic obstacle avoidance, in which a vehicle must navigate through an environment while avoiding obstacles in that environment that may be in motion, and second, to use CARLA as a platform to develop and validate this approach.

We begin by describing our dynamic obstacle avoidance approach, in which we modify the existing dynamic safety zone module inside our autonomous driving codebase to include a proportional controller that maintains a fixed distance from the nearest detected forward vehicle. We then describe how we integrated the dynamic safety zone module with the other components of our autonomous driving codebase to create a validation platform in CARLA for our dynamic safety zone-based approach. Finally, we use this validation platform to assess the sensitivity of our approach to the density of traffic and to the speed of the ego-vehicle in a traffic circle environment.

5.1 Prior Work

The task of enabling an agent to avoid dynamic obstacles in its environment combines the perception problem of detecting objects in the environment, classifying them as either dynamic or static, and possibly tracking or predicting their motion, with the planning problem of selecting an optimal action or planning a future trajectory for the agent.

A number of approaches [32] rely on the explicit detection of objects in the environment, meaning that they require position and velocity information about all dynamic objects in the environment. Explicit object detection and tracking is feasible with a number of sensor modalities such as vision [12] and LIDAR [46, 23, 34], and can either be model-based [12] or model-free [46]. The advantage of relying on this object detection is that it provides a complete picture of the obstacles in the environment and thus allows the agent to plan with a much larger horizon, while the advantage of tracking and prediction is that it allows the agent to anticipate the behavior of dynamic obstacles and use this to improve the quality of planning. However, such approaches are sensitive to the quality of the object detection, which may be dependent on environmental conditions such as illumination (in the case of vision) or sensor characteristics such as resolution (in the case of LIDAR). Other methods are capable of relying on implicit representations of the dynamic obstacles, such as those that use occupancy grids to represent the state of the environment [14, 33]. The advantage of using an occupancy grid representation is that one can model the state of all free space within the vicinity of the agent, although this information is limited by the size and resolution of the occupancy grid. Our implementation relies on an implicit representation of obstacles that partitions our sensor data based on a dynamically-adjusted zone around the vehicle.

Approaches to motion planning for dynamic obstacle avoidance differ in terms of the action space considered. Some approaches select a trajectory from a set of feasible trajectories [21, 53] that respect the dynamic constraints of the agent and the surrounding environment, and convert this trajectory into a low-level control command. Others such as the dynamic window approach [13] and reactive obstacle avoidance approaches [16] directly optimize for a velocity command that respects the environment, even if they may consider feasible trajectories in the process of this optimization. We follow the reactive approach, as we combine obstacle-agnostic path following and trajectory generation with an obstacle-aware guard that modifies the motion commands from the path follower.

5.2 Dynamic Safety Zone

Our approach builds on the existing dynamic safety zone module of our autonomous driving codebase. The dynamic safety zone module acts as a safeguard for the ego-vehicle by identifying infractions in a dynamically-adjusted zone in front of the vehicle and modifying the commanded speed of the vehicle accordingly. This dynamic zone is a rectangular region that is generated in front of the ego-vehicle and whose shape follows that of the projected future trajectory of the ego-vehicle. The length of this dynamic safety zone depends on both the length of the projected future trajectory and the current speed of the vehicle. The motivation for this dependence on the current speed of the vehicle is that a faster-moving vehicle will require a longer duration of time to come to a complete stop, and therefore the zone should be longer in order to account for obstacles that may be within the path of the vehicle.

The dynamic safety zone module detects infractions by using the 2D LIDAR to detect points that fall within the zone. The module then processes these infractions and uses them to calculate a desired safe speed for the vehicle. Figure 5-1 outlines the key components of the dynamic safety zone.

The prior version of the dynamic safety zone module would command a fixed

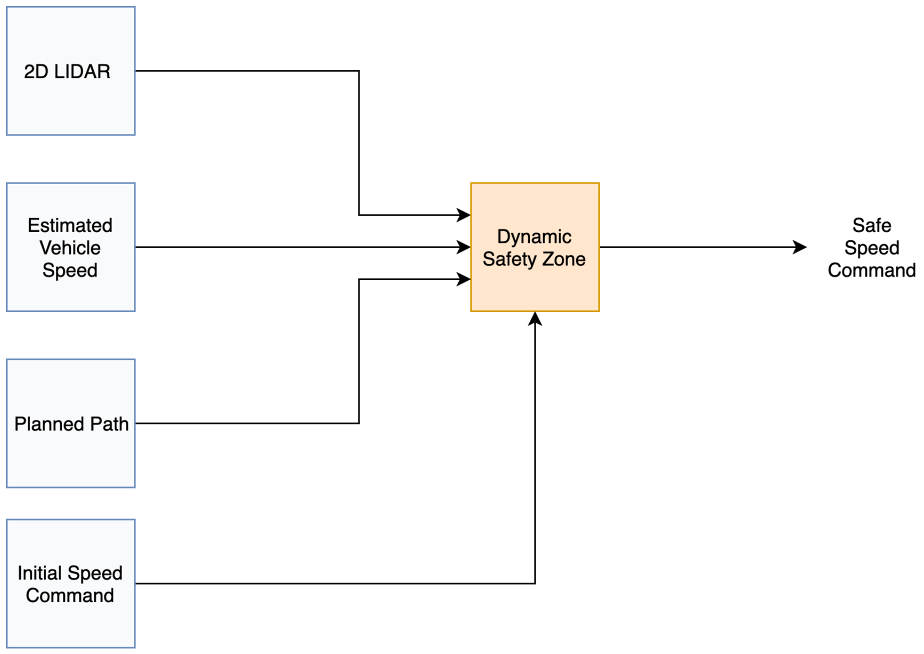


Figure 5-1: Block diagram of existing dynamic safety zone module in our autonomous driving codebase.

velocity v_{safe} if an infraction was detected in the dynamic zone, regardless of the distance of the inferred obstacle from the ego-vehicle. Our contribution was to incorporate a proportional velocity controller into the dynamic safety zone to ensure that vehicle would more smoothly adjust its speed in response to dynamic obstacles.

The modified dynamic safety zone module first infers the distance to the nearest forward obstacle d_{obs} by extracting this information directly from the 2D LIDAR returns that fall into the dynamic zone. The module then sets a commanded velocity v_{cmd} that is proportional to the difference between d_{obs} and an ideal set-point distance d_*. This commanded velocity is clipped to remain within the range $[0, v_{max}]$, because we wish to disallow velocities that are negative and velocities that are larger than a maximum safe velocity value v_{max}, which is an independent parameter. The formula that describes v_{cmd} is as follows:

$$v_{cmd} = \begin{cases} 0 & \text{if } d_{obs} < d_* \\ K(d_{obs} - d_*) & \text{if } d_* \leq d_{obs} \leq d_* + \frac{v_{max}}{K} \\ v_{max} & \text{otherwise} \end{cases}$$

If no obstacle is detected within the dynamic safety zone, then the initial speed command is passed through as the safe velocity command.

5.3 Validation in CARLA

5.3.1 Validation System Overview

In order to validate the ability of the modified dynamic safety zone to avoid dynamic obstacles, we integrated the dynamic safety zone with other modules in our autonomous driving codebase and with our other contributions to CARLA to build a full-loop validation platform, shown in Figure 5-2. Our validation scenario involved the vehicle following a pre-determined trajectory and adjusting its speed in response to dynamic obstacles using the dynamic safety zone module.

We began with the core CARLA implementation described in chapter 3 used to validate autonomous path-following. This core includes three proprioceptive sensors (GPS, IMU, and encoder) to localize the vehicle, the existing pure pursuit module to calculate the desired steering angle and projected path, and an existing speed controller to calculate an initial desired speed to follow this path. We incorporated the dynamic safety zone as a safeguard on top of the pure pursuit controller by passing the raw speed command from the speed controller to the dynamic safety zone. We also passed the planned path generated by the pure pursuit controller as the estimated future trajectory, and we passed the inferred vehicle velocity from the localization EKF. Finally, we incorporated the existing 2D LIDAR sensor from CARLA to provide perception information to the dynamic safety zone.

5.3.2 Demonstration Overview

We then used this validation platform to demonstrate dynamic obstacle avoidance in a CARLA environment. We chose the existing CARLA Town03 environment, a 370 m x 370 m urban environment that contains multiple intersections and road types. In particular, Town03 environment includes a two-lane traffic circle with four

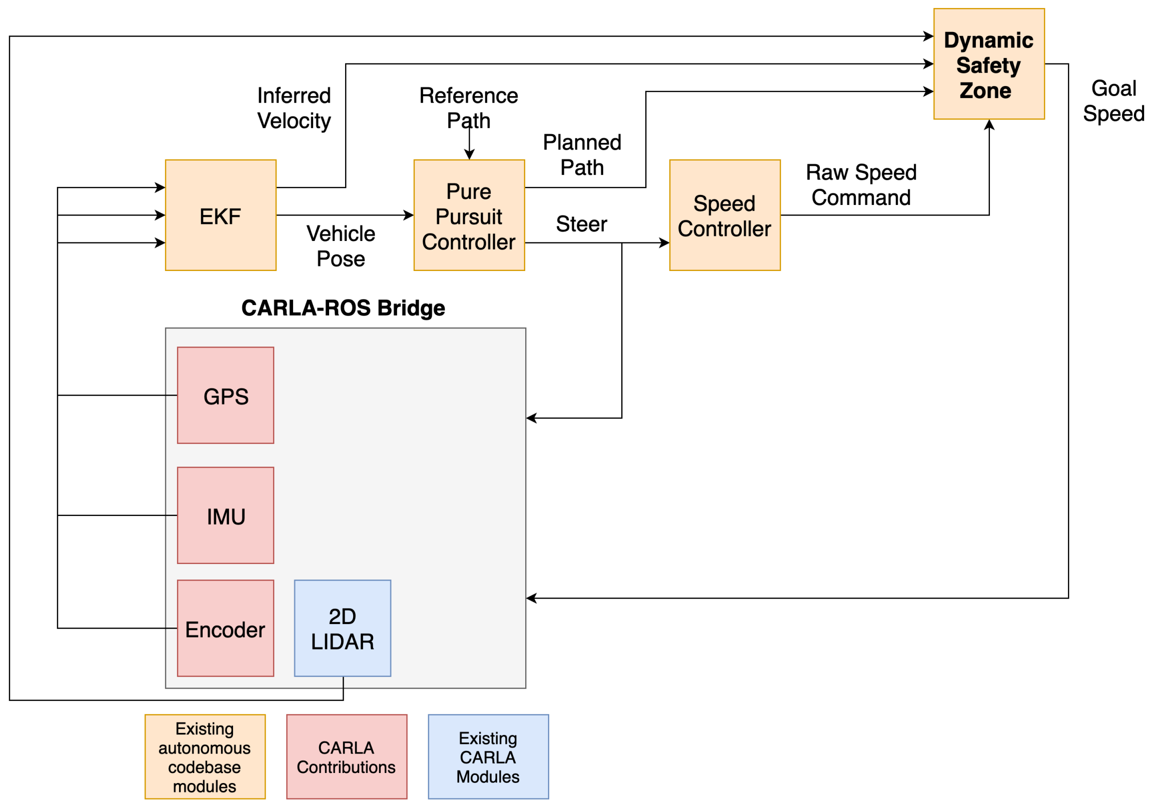


Figure 5-2: Block diagram of complete CARLA validation pipeline for dynamic obstacle avoidance using the dynamic safety zone.

incoming and outgoing bidirectional roads, which we chose to use as a case study for validating the effectiveness of the ego-vehicle to avoid other vehicles. Figure 5-3 shows an overhead view of the traffic circle environment along with our starting location for the ego-vehicle (start location 115 in the Town03 environment).

We first recorded a path that followed the outer lane of the traffic circle by recording the simulated GPS position of the vehicle at a frequency of 10 Hz along the path, in the same manner as described in section 3.3. Because the pure pursuit algorithm follows a fixed path, we effectively constrained the vehicle to remain in the outer lane, only adjusting its speed in response to detected dynamic zone infractions.

We then populated the environment with a desired number of vehicles by randomly initializing these vehicles at spawn locations distributed across the entire Town03 environment (which are not limited to being near the traffic circle). The vehicles navigate through the environment using the built-in CARLA autopilot. The autopilot keeps the vehicle within the lanes which are defined in the Town03 map specification, and makes a random route decision when the vehicle arrives at an intersection. The autopilot also contains an obstacle detector that sends ray casts in the direction of the current vehicle orientation and stops the vehicle if any of these ray casts intersects with an object. However, the autopilot does not incorporate any higher-level behaviors, such as following the rules of the road, negotiating with other vehicles, or changing lanes in response to a static obstacle.

Figure 5-4 shows a visualization of the pure pursuit controller working in conjunction with the dynamic safety zone to avoid dynamic obstacles in the Town03 environment.

The main limitations of our algorithm are that we only consider a zone in front of the ego-vehicle and not the sides or the blind spots, that we do not distinguish between static and dynamic LIDAR points, and that we guide the vehicle to follow a fixed path. First, because our approach only looks in front of the ego-vehicle, we found that the ego-vehicle would sometimes collide with vehicles that were trying to enter the traffic circle too closely. This occurred because the dynamic safety zone would only detect an infraction and attempt to slow down when the entering vehicle

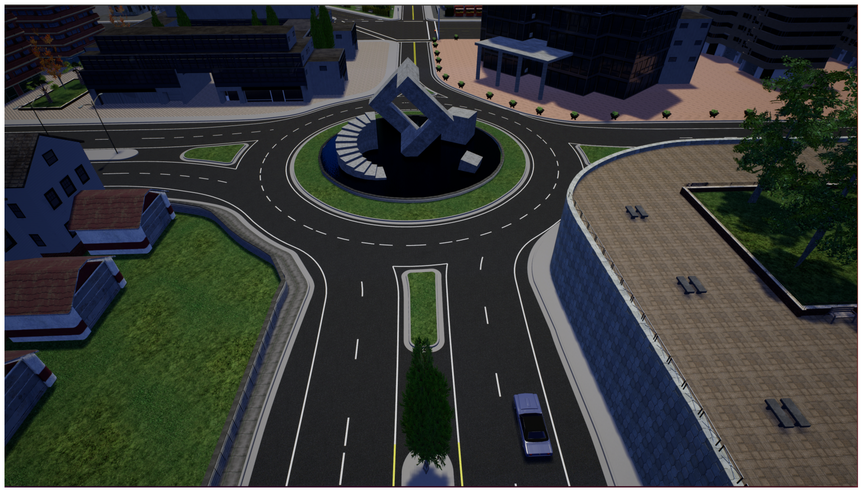
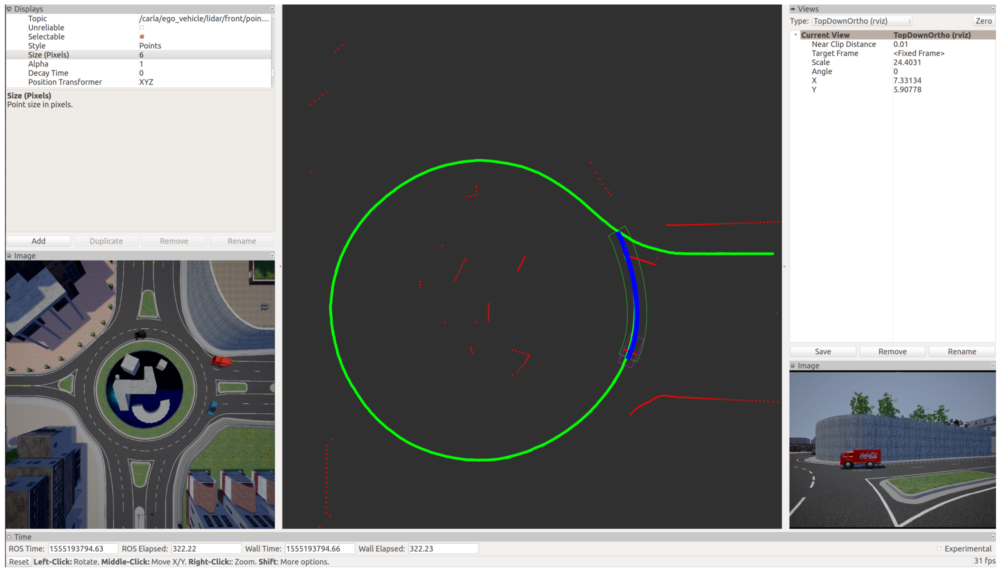


Figure 5-3: Overhead view of traffic circle in CARLA Town03 traffic circle environment with ego-vehicle (in blue) initialized at the starting location.



Topic	/carla/ego_vehicle/lidar/front/poin…
Unreliable	☐
Selectable	☑
Style	Points
Size (Pixels)	6
Alpha	1
Decay Time	0
Position Transformer	XYZ

Size (Pixels)
Point size in pixels.

Type: TopDownOrtho (rviz)

Current View	**TopDownOrtho (rviz)**
Near Clip Distance	0.01
Target Frame	<Fixed Frame>
Scale	24.4031
Angle	0
X	7.33134
Y	5.90778

ROS Time: 1555193794.63 ROS Elapsed: 322.22 Wall Time: 1555193794.66 Wall Elapsed: 322.23

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Zoom. **Shift:** More options.

Figure 5-4: Visualization of dynamic safety zone module and pure pursuit controller in CARLA Town03 traffic circle environment with 2D LIDAR scan (red), reference path (light green), pure pursuit planned path (blue), and dynamic zone (green).

was already too close to the ego-vehicle for the ego-vehicle to stop safely. Second, the ego-vehicle would sometimes brake in response to static objects that were detected on the sides of the road because these objects would fall into the dynamic zone and would be marked as infractions. Finally, the vehicle would not be able to respond to situations where the planned route lane was blocked by static vehicles (such as in the case of a traffic jam) because the vehicle attempts to follow the planned route and because the dynamic safety zone does not actually affect the steering behavior of the ego-vehicle (only its speed).

5.4 Experimental Evaluation

5.4.1 Sensitivity to Traffic Density

Our first experiment was to assess how sensitive the dynamic safety zone approach was to the density of traffic in the CARLA Town03 environment. In order to quantify how well the vehicle was able to avoid dynamic obstacles, we recorded the amount of time elapsed until the vehicle first collided with another moving vehicle or a static feature in the environment (such as a building wall). We started the vehicle at a fixed starting location in the Town03 environment on the right lane of one of the roads entering the traffic circle, as shown in Figure 5-3. We then allowed the vehicle to enter the traffic circle before populating the CARLA environment with the autopilot vehicles. Figure 5-5 illustrates the relationship between the number of vehicles in the CARLA environment and the average time until the first collision, with 10 trials per density setting. Our average duration metric truncates trial runs that lasted longer than a duration of t_{max} seconds, for an arbitrary choice of threshold t_{max}. In particular, given a collection of durations $\{d_i\}_{i=1}^N$, the average duration that we report is $\frac{1}{N}\sum_{i=1}^N \min(d_i, t_{max})$. The reason why we truncated the trial runs with this threshold was to prevent trials from lasting indefinitely, which would occur in situations where the effective probability of collisions was so low that the expected time until the first collision would be very large.

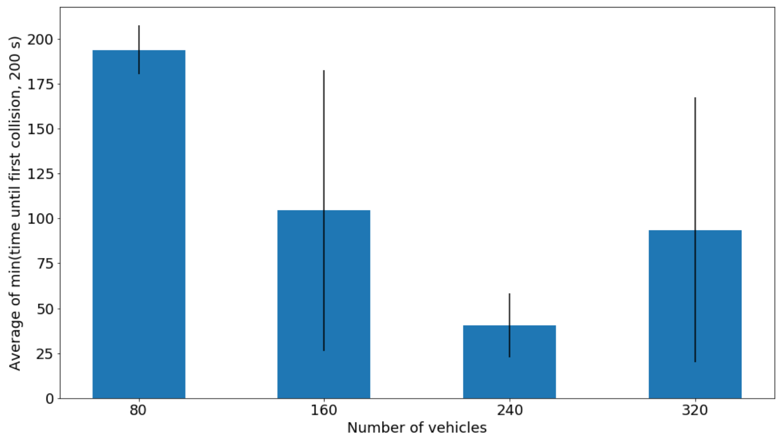


Figure 5-5: Relationship between number of vehicles and average time until the first collision between the ego-vehicle and another vehicle.

Our expectation was that the average duration until the first collision would monotonically decrease as the number of vehicles increased, because an increased density of vehicles would lead to a higher probability that the ego-vehicle collides with any one of them. We did observe this monotonic decrease up to a certain point, but with a large number of vehicles we actually observed a sharp increase in the survival time of the ego-vehicle. We observed that with 80 external vehicles, the ego-vehicle does not frequently encounter other vehicles because they are diffusely spread throughout the entirety of the Town03 the environment. In the situations with 160 and 240 vehicles, we observed that the frequency of vehicle-vehicle interactions increased as expected, leading to a lower ego-vehicle survival time. However, at the high density of vehicles (320 vehicles), we hypothesize that the frequency of inter-vehicle interactions increased to the point that all vehicles proceeded at a slower speed due to the behavior of the CARLA built-in autopilot, which causes each vehicle to brake if another vehicle is detected in front of or next to the ego-vehicle. The reduced speed of all agents in the environment likely prevented the external vehicles from colliding with the ego-vehicle.

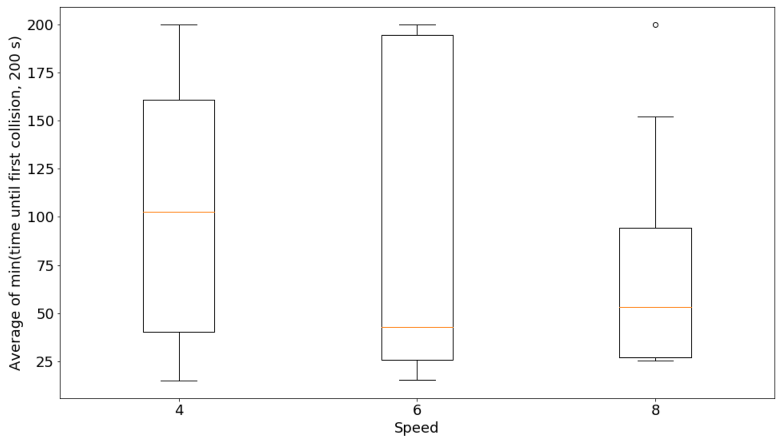


Figure 5-6: Relationship between speed of the ego-vehicle and average time until the first collision between the ego-vehicle and another vehicle.

5.4.2 Sensitivity to Ego-Vehicle Speed

Next we sought to gauge the effect of the speed of the ego-vehicle on its ability to avoid dynamic obstacles using the dynamic safety zone approach. We again measured the performance of the vehicle using the average of truncated durations metric described in the traffic density experiment, although with a different value of t_{max}.

We tuned the pure pursuit lookahead radius separately for each speed in order to provide a fair comparison between the vehicle performance at different speeds. The reason for this is that at higher speeds, the lookahead radius typically needs to be increased to avoid oscillations around the desired path. To standardize the comparison between speeds, we set the lookahead radius to be numerically equal to the vehicle speed (thus making the lookahead radius equivalent to 1 second of driving time).

Figure 5-6 indicates the relationship between the speed of the ego-vehicle and the average time until the first collision with another vehicle, with 10 trials per speed setting. Our expectation was that the ego-vehicle may have been able to avoid collisions with the other vehicles if it was traveling at higher speeds. The reason for this is because at higher speeds, the ego-vehicle would avoid situations where vehicles entering the traffic circle would cut off the ego-vehicle as they entered. If the ego-vehicle is traveling at a speed that is much lower than that of the other vehicles, then it is likely that the other vehicles would not detect the ego-vehicle as they entered the circle, thus leading to a higher probability of collision.

However, our findings indicated that the average time until the first collision did not noticeably change with higher ego-vehicle speeds - in fact, the median survival time for the lowest speed setting (4 m/s) was significantly higher than that of the higher speed settings (6 m/s and 8 m/s). We offer two possible hypotheses - one related to the dynamic safety zone algorithm itself and the other related to CARLA - that may explain the large number of trials (especially for the higher speeds) that lasted below 50 s. The first is that the ego-vehicle would often collide with another vehicle that was entering the traffic circle closely, which is a limitation of the dynamic safety zone (which only examines LIDAR points in a zone that follows the current projected path of the vehicle). The second is that we observed a transient effect that occurred upon initializing the CARLA environment with the external vehicles, in which the apparent update rate of the sensor data, projected path, and dynamic safety zone would decrease to a value of less than 1 Hz. This would sometimes lead the ego-vehicle to significantly deviate from the traffic circle, leading the pure pursuit algorithm to sharply correct the ego-vehicle's trajectory and possibly collide with another vehicle.

Chapter 6

Conclusion and Future Work

In this work, we described the development of a simulation-based platform that combined an autonomous driving codebase with an open-source driving simulator. The purpose of this platform was to serve as both a validation platform for the fundamental capabilities of autonomous driving and as a development and validation platform for autonomous driving research capabilities. Even though some of these capabilities could be tested on offline sensor data, the strength of our platform is that it is a full-loop testing environment that models the feedback between the sensor data produced by the vehicle and the control outputs produced by our autonomous driving algorithms.

With the incorporation of proprioceptive sensors into CARLA, we were able to validate the performance of the existing localization, pure pursuit, and path planning algorithms inside our autonomous driving codebase. This enabled us to build the core of our integration between our autonomous driving codebase and CARLA, which allows the vehicle to autonomously follow a reference trajectory using the proprioceptive sensors, and which additionally allows the vehicle to avoid static obstacles using the CARLA 2D LIDAR.

We then showed that CARLA could be used to validate a point-to-point autonomous navigation algorithm (MapLite) by incorporating ground-truth road segmentation into the CARLA 3D LIDAR and modifying the route planner to stop at intersections. We demonstrated that CARLA could be used to assess the dispersion of the paths generated by MapLite and to compare the performance of MapLite at different topological regions of the road. Our hope is that CARLA can continue to serve as a useful platform for assessing the performance of MapLite and improving different components of the algorithm through iterative validation in simulation and real-world testing, although more investigation is needed into the sources of uncertainty in CARLA itself before it can be a fully effective testbed.

Finally, we used CARLA to develop an extension to the dynamic safety zone module of our autonomous driving codebase and validate the ability of this extension to avoid dynamic vehicles in a controlled traffic environment. Our extension was still limited by the fact that it lacked explicit object detection, only searched a limited region of the vehicle's environment for any type of obstacle, and could not distinguish between static and dynamic obstacles. Nevertheless, we demonstrated the potential of CARLA to be used as a platform for developing algorithms for new capabilities that did not previously exist in our codebase.

6.1 Lessons Learned

Working closely with the CARLA simulator and our autonomous driving codebase provided many crucial insights into software engineering and the management of a large, modular codebase shared by multiple researchers that will be broadly applicable to future robotics and autonomous driving research endeavors. I learned the importance of effectively using software version control tools such as Git to keep track of the multiple versions of CARLA used for the different contributions outlined in this thesis. I also learned the value of ensuring that the integration between CARLA and the autonomous codebase was up-to-date with new features that were included in the codebase. In addition, I learned a great deal about autonomous driving simulation and a selection of its strengths - such as its ability to reveal corner cases and hidden assumptions in our algorithms, as was the case with our costmap implementation - as well as some of its drawbacks - such as its sensitivity to the hardware requirements of the computing workstation used (including GPU and graphics driver requirements).

My work on dynamic obstacle avoidance helped me understand the value of Occam's razor when developing an initial solution to a time-constrained engineering problem. The dynamic safety zone, even with the proportional controller modification, cannot address all of the possible situations that can be encountered with other dynamic vehicles. However, it was an adequate solution for the constrained problem that we defined, which assumed that the other dynamic vehicles obey traffic laws while the ego-vehicle keeps to its lane. Finally, I learned a great deal about conducting more methodical scientific investigation as I developed this thesis - including the importance of concretely defining the research problem, of defining the scope of the contributions that I made, and of designing effective experiments that aimed to uncover new knowledge about the systems and algorithms that we developed.

6.2 Future Work

Two important future directions for our platform are improving the quality of the CARLA sensor data and improving the efficiency of running validation experiments in CARLA. Because most of the algorithms highlighted in this work rely on LIDAR, we wish to adapt the CARLA LIDAR module to be more realistic by modeling the sources of uncertainty in actual LIDAR and incorporating this uncertainty into the CARLA LIDAR. Having a realistic LIDAR module would enable CARLA to be an effective data augmentation platform, which would allow us to explore perception problems such as improving the quality of LIDAR-based road segmentation for MapLite. We also hope to develop a more efficient implementation of the 3D LIDAR that either exploits parallelization to speed up the computation of the individual ray casts or does not rely on generating these ray casts. Future directions for dynamic obstacle avoidance include using the dynamic safety zone with a partition of the 2D LIDAR scan into dynamic and static regions, or incorporating an RRT-based planning algorithm with obstacle detection [32] and additional vehicle behaviors such as lane changes.

Our hope is that CARLA can be a useful tool for researchers who want to prototype and test algorithms for autonomous driving and for those who want to generate large-scale, diverse datasets for learning-based autonomous driving algorithms.

Bibliography

[1] Alexander Amini, Igor Gilitschenski, Jacob Phillips, Julia Moseyko, Sertac Karaman, and Daniela Rus. Data-driven simulation for learning end-to-end autonomous vehicle control. In *International Conference on Machine Learning (ICML); Autonomous Vehicle Workshop*, 2019.

[2] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Juergen Gall. A dataset for semantic segmentation of point cloud sequences. *arXiv preprint arXiv:1904.01416*, 2019.

[3] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. Sumo–simulation of urban mobility: an overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.

[4] Alex Bewley, Jessica Rigley, Yuxuan Liu, Jeffrey Hawke, Richard Shen, Vinh-Dieu Lam, and Alex Kendall. Learning to drive from simulation without real world labels. *arXiv preprint arXiv:1812.03823*, 2018.

[5] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[6] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[7] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[8] R. Craig Coulter. *Implementation of the Pure Pursuit Tracking Algorithm.* Jan 1992.

[9] Weiwen Deng, Yong H Lee, and Annie Zhao. Hardware-in-the-loop simulation for autonomous driving. In *2008 34th Annual Conference of IEEE Industrial Electronics*, pages 1742–1747. IEEE, 2008.

[10] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[11] Epic Games. Unreal Engine. https://www.unrealengine.com.

[12] Andreas Ess, Konrad Schindler, Bastian Leibe, and Luc Van Gool. Object detection and tracking for autonomous navigation in dynamic environments. *I. J. Robotic Res.*, 29:1707–1725, 12 2010.

[13] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, 4(1):23–33, March 1997.

[14] C. Fulgenzi, A. Spalanzani, and C. Laugier. Dynamic obstacle avoidance in uncertain environment combining pvos and occupancy grid. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1610–1616, April 2007.

[15] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[16] Chad Goerzen, Zhaodan Kong, and Bernard Mettler. A survey of motion planning algorithms from the perspective of autonomous UAV guidance. *Journal of Intelligent and Robotic Systems*, 57(1-4):65, 2010.

[17] Timo Hackel, Nikolay Savinov, Lubor Ladicky, Jan D Wegner, Konrad Schindler, and Marc Pollefeys. Semantic3d. net: A new large-scale point cloud classification benchmark. *arXiv preprint arXiv:1704.03847*, 2017.

[18] Xinyu Huang, Xinjing Cheng, Qichuan Geng, Binbin Cao, Dingfu Zhou, Peng Wang, Yuanqing Lin, and Ruigang Yang. The apolloscape dataset for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 954–960, 2018.

[19] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[20] Matthew Johnson-Roberson, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rosaen, and Ram Vasudevan. Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks? *arXiv preprint arXiv:1610.01983*, 2016.

[21] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *Robotics Science and Systems VI*, 104:2, 2010.

[22] N. Koenig and A. Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, Sep. 2004.

[23] Bo Li, Tianlei Zhang, and Tian Xia. Vehicle Detection from 3D Lidar Using Fully Convolutional Network. *arXiv e-prints*, page arXiv:1608.07916, Aug 2016.

[24] Chen-Hsuan Lin, Chen Kong, and Simon Lucey. Learning efficient point cloud generation for dense 3d object reconstruction. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[25] MIT. MIT RACECAR. https://mit-racecar.github.io .

[26] Thomas Moore and Daniel Stouch. A Generalized Extended Kalman Filter Implementation for the Robot Operating System. In Emanuele Menegatti, Nathan Michael, Karsten Berns, and Hiroaki Yamaguchi, editors, *Intelligent Autonomous Systems 13*, pages 335–348, Cham, 2016. Springer International Publishing.

[27] Matthias Mueller, Neil Smith, and Bernard Ghanem. A benchmark and simulator for uav tracking. In *European conference on computer vision*, pages 445–461. Springer, 2016.

[28] Felix Naser, David Dorhout, Stephen Proulx, Scott Drew Pendleton, Hans Andersen, Wilko Schwarting, Liam Paull, Javier Alonso-Mora, Marcelo H Ang, Sertac Karaman, et al. A parallel autonomy research platform. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 933–940. IEEE, 2017.

[29] Teddy Ort, Liam Paull, and Daniela Rus. Autonomous vehicle navigation in rural environments without detailed prior maps. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2040–2047. IEEE, 2018.

[30] Liam Paull, Jacopo Tani, Heejin Ahn, Javier Alonso-Mora, Luca Carlone, Michal Cap, Yu Fan Chen, Changhyun Choi, Jeff Dusek, Yajun Fang, et al. Duckietown: an open, inexpensive and flexible platform for autonomy education and research. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1497–1504. IEEE, 2017.

[31] Daniel Pickem, Paul Glotfelter, Li Wang, Mark Mote, Aaron Ames, Eric Feron, and Magnus Egerstedt. The robotarium: A remotely accessible swarm robotics research testbed. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1699–1706. IEEE, 2017.

[32] Alyssa Pierson, Wilko Schwarting, Sertac Karaman, and Daniela Rus. Navigating congested environments with risk level sets. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8, 2018.

[33] Alyssa Pierson, Cristian Ioan Vasile, Anshula Gandhi, Wilko Schwarting, Sertac Karaman, and Daniela Rus. Dynamic Risk Density for Autonomous Navigation in Cluttered Environments without Object Detection. In *IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May 2019. (accepted).

[34] B. Qin, Z. J. Chong, S. H. Soh, T. Bandyopadhyay, M. H. Ang, E. Frazzoli, and D. Rus. *A Spatial-Temporal Approach for Moving Object Recognition with 2D LIDAR*, pages 807–820. Springer International Publishing, Cham, 2016.

[35] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[36] Craig Quiter and Maik Ernst. deepdrive/deepdrive: 2.0, March 2018.

[37] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. 11 2016.

[38] Stephan R. Richter, Zeeshan Hayder, and Vladlen Koltun. Playing for benchmarks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2232–2241, 2017.

[39] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *European Conference on Computer Vision*, pages 102–118. Springer, 2016.

[40] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3234–3243, 2016.

[41] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics*, 2017.

[42] John Skinner, Sourav Garg, Niko Sünderhauf, Peter Corke, Ben Upcroft, and Michael Milford. High-fidelity simulation for evaluating robotic vision performance. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2737–2744. IEEE, 2016.

[43] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. http://ompl.kavrakilab.org.

[44] Russ Tedrake and the Drake Development Team. Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems, 2016.

[45] Udacity. udacity/self-driving-car-sim, Jan 2019.

[46] Dominic Zeng Wang, Ingmar Posner, and Paul Newman. Model-free detection and tracking of dynamic objects with 2d lidar. *The International Journal of Robotics Research*, 34(7):1039–1063, 2015.

[47] Xiaolong Wang and Abhinav Gupta. Generative image modeling using style and structure adversarial networks. In *European Conference on Computer Vision*, pages 318–335. Springer, 2016.

[48] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1887–1893. IEEE, 2018.

[49] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in neural information processing systems*, pages 82–90, 2016.

[50] Bernhard Wymann, Christos Dimitrakakisy, Andrew Sumnery, and Christophe Guionneauz. TORCS: The open racing car simulator, 2015.

[51] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9068–9079, 2018.

[52] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling. *arXiv preprint arXiv:1805.04687*, 2018.

[53] Qiuming Zhu. Hidden Markov model for dynamic obstacle avoidance of mobile robot navigation. *IEEE Transactions on Robotics and Automation*, 7(3):390–397, 1991.

[54] Julian Zilly, Jacopo Tani, Breandan Considine, Bhairav Mehta, Andrea F Daniele, Manfred Diaz, Gianmarco Bernasconi, Claudio Ruch, Jan Hakenberg, Florian Golemo, et al. The ai driving olympics at neurips 2018. *arXiv preprint arXiv:1903.02503*, 2019.