Final project report - AVCP: Autonomous Vehicle Coordination Protocol

Marcelo Paulon J. V.*

Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)

Advisor: Markus Endler

December 2017

Keywords: autonomous vehicles, connected vehicles, smart city, vehicle-to-vehicle, V2V, vehicle-to-infrastructure, V2I, vehicle-to-everything, V2X, movement coordination

Abstract

Over the last years the automotive industry has advanced significantly towards a future without human drivers. Researchers are currently trying to overcome the technological, political and social challenges involved in making autonomous vehicles mainstream. These vehicles need to be safe, reliable and cost-efficient. Connecting them and creating coordination mechanisms could help achieving these goals.

This project proposes a vehicle-to-everything (V2X) coordination protocol for autonomous vehicles (AVCP: Autonomous Vehicle Coordination Protocol) and a testing environment where it will be evaluated. The AVCP aims to significantly reduce travel time and increase security for autonomous vehicles by enabling them to exchange sensing and routing information with each other and with roadside units (RSUs).

*Electronic address: marcelo.paulon@gmail.com

Contents

1 Introduction

Automobile manufacturers such as Ford, General Motors, Tesla, and other companies such as NVIDIA are investing billions of dollars in autonomous vehicle driving research. According to Intel, by 2050, this fast-growing industry will be worth $ 7 trillion [25]. Governments of countries in Europe and of the USA are creating regulations for self-driving cars, as a result of the latest advances of the industry.

The benefits of fully autonomous vehicles can go way beyond removing the need of a human driver. Transportation services such as Uber will start using self-driving cars instead of human drivers, and might become a cheaper and better alternative for end consumers than owning a car. This will represent a shift on the way cities are planned, as fewer parking places will be needed, and most importantly, in a smart city with most of its vehicles being connected and autonomous, traffic optimization will be able to be heavily applied by coordinating movement. This will result in a major decrease on travel time, and it will also save lives as emergency services will be able to reach their destinations faster.

In contrast, there are many problems that need to be addressed regarding autonomous vehicles. Tesla's car autopilot first fatal crash in 2016 [9], for example, brought up discussions about reliability, safety and legal liability regarding autonomous vehicles among researchers. Other important topics, such as security and transparency will need to be heavily discussed as these vehicles become mainstream.

Another problem of this relatively new field is the lack of data sharing between vehicles. Autonomous vehicles are complex systems that rely heavily on technologies such as LIDAR, GPS, high-definition maps, and artificial intelligence for navigation and collision avoidance. This means that each autonomous vehicle is constantly collecting and analyzing a very high volume of data, which according to Intel is about 2.6 terabytes per hour [27]. If this data was shared between the vehicles it could be used for coordinating movement, which could increase safety and also optimize traffic on cities and highways.

2 Autonomous vehicle movement coordination - benefits and usage scenarios

In this section, we will present possible scenarios of autonomous vehicle movement coordination, and what benefits they would bring regarding traffic and security.

2.1 Scenario 1: emergency lane changes

Whether on highways or two-lane roads it may happen that an obstacle will suddenly require a lane change, for example, when an animal crosses the road, or when a landslide occurs. This sudden lane change may represent a dangerous situation, as it is possible that the vehicles on the other lane are approaching from behind in a much higher speed.

By broadcasting a sudden lane change event for the vehicles behind, they will be able to reduce their speed faster, before they can visually detect the car changing lanes, reducing the risk of an accident.

2.2 Scenario 2: early obstacle avoidance

Obstacles such as boulders, broken vehicles, or road work can cause massive traffic as they usually involve obstructing an entire lane. On roads without closed lane signals, by broadcasting the detection of an obstacle on the lane, the vehicles behind would be able to change lanes before reaching the obstacle, reducing traffic.

2.3 Scenario 3: temporary stops

Cars and buses can stop to pickup/drop passengers for short periods of time. As in the last example, this could also increase traffic. These vehicles could send a notification that they will stop for a short period of time, allowing vehicles that are approaching from behind to decide whether or not to change lanes based on their location and on how much time the vehicle is going to be stopped.

2.4 Scenario 4: junctions and intersections

There are many mathematical models for optimizing traffic in junctions and intersections. These models could be optimized by making vehicles that are able

to coordinate their movement.

2.5 Scenario 5: emergency vehicles

An emergency vehicle could broadcast its presence for vehicles ahead, so that they could change lanes early to give priority to it. This will make the emergency vehicles reach their destinations faster, something that could save lives.

2.6 Scenario 6: pile-up avoidance

In the unfortunate event of an accident, vehicles could broadcast a car crash notification that will make vehicles approaching from behind reduce their speed, decreasing the risk of a pile-up. Also, this will allow the other vehicles to reroute, which may reduce the traffic generated by the accident.

3 Current state

Vehicular networks, both centralized and ad hoc (VANET), can highly improve efficiency and safety for transportation. There are many challenges involved in designing such networks, from concurrency and safety problems to high cost and lack of standardization. Surveys such as [5] [21] map the different approaches that are being researched in this field.

Some articles such as [17] propose the use of intelligent traffic lights (ITLs) on crossroads. These devices could gather traffic information such as traffic density, and share it with the nearby vehicles so that they can choose a route with less traffic. In case of an accident, the ITLs could also send warning messages to nearby vehicles, which would prevent other collisions.

[7] propose using a RSU to manage traffic on intersections. These RSUs, called intersection managers, can implement different policies, but operate on a common protocol proposed by the authors for managing reservations for traversing the intersection. The vehicles can make, change, or cancel a reservation, and it is up to the intersection managers to confirm, reject or call for emergency-stops. The managers can also propose a counter-offer (crossing the intersection through a different lane, for example). The authors claim that by having a RSU all the information between the agents go through one reliable and monitorable channel.

[1] propose an interesting use for VANETs: infotainment and interactive applications and multi-hop Internet access. The authors address the problem of high

data volume and the consequent increase on communication delay, by giving low preference to more densely crowded routes.

The articles mentioned above have direct or indirect RSU involvement. [2] addresses the high cost of having a centralized authority for efficiently coordinating traffic for autonomous vehicles. The authors propose using formal methods to verify the safety and correctness of a collaborative and decentralized coordination protocol. They claim that vehicles using this protocol do not need to fully agree on a shared state, by using a resource reservation protocol (CwoRIS [22]) to collaboratively negotiate access to the intersection.

In [28], a protocol for vehicle merging on vehicular ad hoc networks (VANETs) is proposed. This protocol calculates the other vehicles future positions instead of using their current position, to support Cooperative Adaptive Cruise Control (CACC).

[24] introduces the Veins simulator [23] and addresses the issues of creating realistic simulations for VANETs. The author claims that many simulators do not take into account driver behavior or characteristics of the urban environment such as traffic lights and merge lanes. One of the examples used in a proof-of-concept for Veins was early obstacle avoidance, described in section 2.2.

4 Proposal and objectives

In this project we propose creating a protocol (AVCP) that specifies how autonomous cars can share their sensor data and use this information to collaboratively optimize traffic and increase safety, without needing a central server. The idea is to propose a standard set of messages for cars and RSUs, and rules for their behavior. These rules will take into account, for each vehicle, its received messages, its intent (e.g. avoid an obstacle, turn left in an intersection), and its sensing information.

The tests for the AVCP protocol will be performed on a simulated environment, a virtual smart city where all vehicles are autonomous. We will simulate the scenarios described in section 2, using simulated self-driving cars, and compare their navigation with and without the protocol being used to support each car's decisions.

Thus, in addition to developing the AVCP specification, the simulation environment for the protocol will need to be created. This will involve modifying an

autonomous car driving simulator to use the protocol. The chosen simulator was written using the C# programming language and the Unity 3D engine, and the path planner that controls the autonomous car was written using the C++ programming language. The simulator will be modified to support controlling multiple autonomous cars, and the path planner will be modified to support the communication between them.

In short, the main goals of the project are:

- develop the AVCP protocol specification
- develop a simulation environment where cars can use the AVCP protocol to communicate
- assess if by using the AVCP protocol the mean travel time of the vehicles is reduced

We will also discuss alternatives to travel time measuring for evaluating the AVCP and other coordination protocols, such as distance driven, brakes usage and chassis stress. At least one of them will be tested with the AVCP protocol.

5 Schedule summary (timeline)

Table 1: Schedule summary (timeline)

	Time period	Task description	Deliverable
1	Jul 1-Sept 30, 2017	Simulator+AI research	Research report
2	Oct 1-Nov 05, 2017	Simulator+AI preparation	Implementation report and code
3	Nov 06-16, 2017	Research on vehicle-to-vehicle coordination protocols	Research report
4	Nov 17-Dec 05, 2017	AVCP protocol draft	Specification
5	December, 2017	Final presentation of the project proposal	Slideshow
6	Dec 06-Jan 31, 2018	Proof of concept - development of the AVCP Simulator / integration with the chosen self-driving car simulator	Implementation report and code
7	Feb 1-7, 2018	Simulation execution and measurements	Testing report
8	Feb 8-Apr 8, 2018	Optimizations on the AVCP specification	Implementation report
9	Apr 9-May 15, 2018	Implementation of the revised AVCP specification	Implementation report
10	May 16-26, 2018	Simulation execution and measurements	Testing report and Paper
11	May 27-Jun 30, 2018	Continuous improvements on the AVCP and the simulator	Implementation report and Specification
12	July, 2018	Final presentation of the project	Slideshow

6 Schedule details

6.1 Phase 1: July 1, 2017 to September 30, 2017
Simulator+AI research

Decide which open-source simulator and AI will be used to test the protocol.

6.2 Phase 2: October 17, 2017 to November 05, 2017
Simulator+AI preparation

Modify the simulator/AI to work in an online environment, allowing us to simulate multiple autonomous cars; Set up the environment where the simulations will be performed.

6.3 Phase 3: November 06, 2017 to November 16, 2017
Research on vehicle-to-vehicle coordination protocols

Gather information about existing V2V standards that could inspire the AVCP development.

6.4 Phase 4: November 17, 2017 to December 05, 2017
AVCP protocol draft

Create a draft of the protocol, containing the data format and simple criterion that can be used to coordinate movement.

6.5 Phase 5: December, 2017
Final presentation of the project proposal

Showcase the problem, the simulator that is being used (and the modifications that were made to the original one), the initial protocol idea, the first proof of concept that was made, and the planned next steps.

6.6 Phase 6: December 06, 2017 to January 31, 2018
Proof of concept - development of the AVCP Simulator / integration with the chosen self-driving car simulator

On the simulator, implement basic data transmission/reception and simple criterion to coordinate movement.

6.7 Phase 7: February 1, 2018 to February 7, 2018
Simulation execution and measurements

Execute the simulations and analyze its results

6.8 Phase 8: February 8, 2018 to April 8, 2018
Optimizations on the AVCP specification

With the initial results of the simulations, we will be able to improve the AVCP by elaborating on the movement coordination criterion.

6.9 Phase 9: April 9, 2018 to May 15, 2018
Implementation of the revised AVCP specification

Update the AVCP simulator to implement the new movement coordination criterion.

6.10 Phase 10: May 16, 2018 to May 26, 2018
Simulation execution and measurements

Execute the simulations and analyze its results. We will write a paper for a conference/journal, detailing AVCP, its implementation and the results that were obtained.

6.11 Phase 11: May 27, 2018 to June 30, 2018
Continuous improvements on the AVCP and the simulator

6.12 Phase 12: July, 2018
Final presentation of the project

Introduce the latest specification of the AVCP and the results of the simulations.

7 Simulator/AI research

7.1 Initial research

The initial research was performed using the Google search engine [11], Google Scholar [10] and the GitHub [12] repository search.

The search queries used were some combinations of the following terms: ”autonomous vehicle”, ”testing environment”, ”simulator”, ”self-driving car”, ”vehicle coordination”. Most of the searches returned 600k-900k results on Google, 5k-6k results on Google Scholar, and fewer than 70 repositories on GitHub (most of them forks from other repositories).

After analyzing the search results, the following projects were selected for comparison, using relevance, popularity and licensing as criterion:

1. ★ Udacity’s Self-Driving Car Simulator [14]
2. Autoware, by Tier IV [15]
3. Veins - Vehicular network simulation framework [23]
4. TORCS - The Open Racing Car Simulator [29]

7.2 Detailed comparison

For comparing the selected projects, the following features/characteristics were taken into account:

(i) Mapping: the project should be able to simulate vehicles, multi-lane roads and obstacles on a map

(ii) Sensoring: the project should be able to simulate vehicle sensor data necessary for autonomous driving, that is, for a given vehicle, detect its own position, speed, the other vehicles around it, and obstacles

(iii) API: the project should provide an API for controlling the vehicle, also providing its sensoring data. Ideally, the API should support controlling multiple vehicles on the simulation

(iv) Graphical interface: the project should provide a graphical interface capable of displaying multi-lane roads, obstacles, vehicles on movement, and occasional lane changes and crashes

(v) Community: projects with larger developer communities are usually more reliable and stable, reducing the risks of using them

(vi) Licensing: the project should have an open license, that allows its use, modification and redistribution for the AVCP project, for free

(vii) Autonomous driving: the project should have existing examples / implementations of autonomous driving, to be used as a starting point for the AVCP testing environment

(viii) Multi-core simulation: ideally, the project should have the ability to execute the simulations using multiple CPU cores.

The table 2, on the next page, summarizes the research results, and the next sections go into further detail on each project that was evaluated.

Table 2: Summary of the research results

Project name	Mapping	Sensoring	API	Graphical Interface	Community	Licensing	Autonomous driving	Multi-core simulation
Udacity's Self-Driving Car Simulator	Yes. Multi-lane roads, other vehicles and obstacles	Yes. Vehicle's information.	Yes	3D	Yes	MIT	Yes	Yes
Autoware, by Tier IV	Yes. Multi-lane roads, other vehicles and obstacles	Partial. Data needs to be collected from real sensors.	No	3D	Yes	New BSD	Yes	Yes
Veins - Vehicular network simulation framework	Yes. Multi-lane roads, other vehicles and obstacles	Yes. Lane number, detected obstacles and traffic lights	Yes	2D	Yes	GPL	Yes	No
TORCS - The Open Racing Car Simulator	Partial. Multi-lane roads and other vehicles	Yes. Vehicle's information.	Yes	3D	Yes	GPL	Yes	Yes

7.3 ★ Udacity's Self-Driving Car Simulator [14]

7.3.1 Mapping

The simulator already has examples of obstacles, multi-lane roads, and other vehicles on the map.

7.3.2 Sensoring

The simulator provides sensor fusion data that contains the other vehicle's information, including their speed and relative position. Non-vehicle obstacles information on sensor fusion would need to be implemented.

7.3.3 API

The simulator provides an API for controlling the vehicle, providing its sensoring data. There is no support for controlling multiple vehicles on the simulation.

7.3.4 Graphical Interface

The simulator provides a 3D graphical interface, capable of displaying multi-lane roads, obstacles, vehicles on movement, and occasional lane changes and crashes.

7.3.5 Community

The simulator has a growing community, and has been updated frequently. It is being used as the simulator for a self-driving car online course.

7.3.6 Licensing

The simulator has been released under the MIT License, which enables free use, modification, and redistribution.

7.3.7 Autonomous driving

There are some implementations of autonomous driving using this simulator. These implementations are publicly available on GitHub [12].

7.3.8 Multi-core simulation

The simulator takes advantage of multi-core processing.

Figure 1: Udacity self-driving car simulator

7.4 Autoware, by Tier IV [15]

7.4.1 Mapping

The project already has examples of obstacles, multi-lane roads, and other vehicles on the map.

7.4.2 Sensoring

The project provides real sensor fusion data, but it requires previously collected sensor data to be used as a simulator.

7.4.3 API

There is no API information on the documentation.

7.4.4 Graphical Interface

The project provides a very rich 3D graphical interface, capable of displaying multi-lane roads, obstacles, vehicles on movement, and occasional lane changes and crashes.

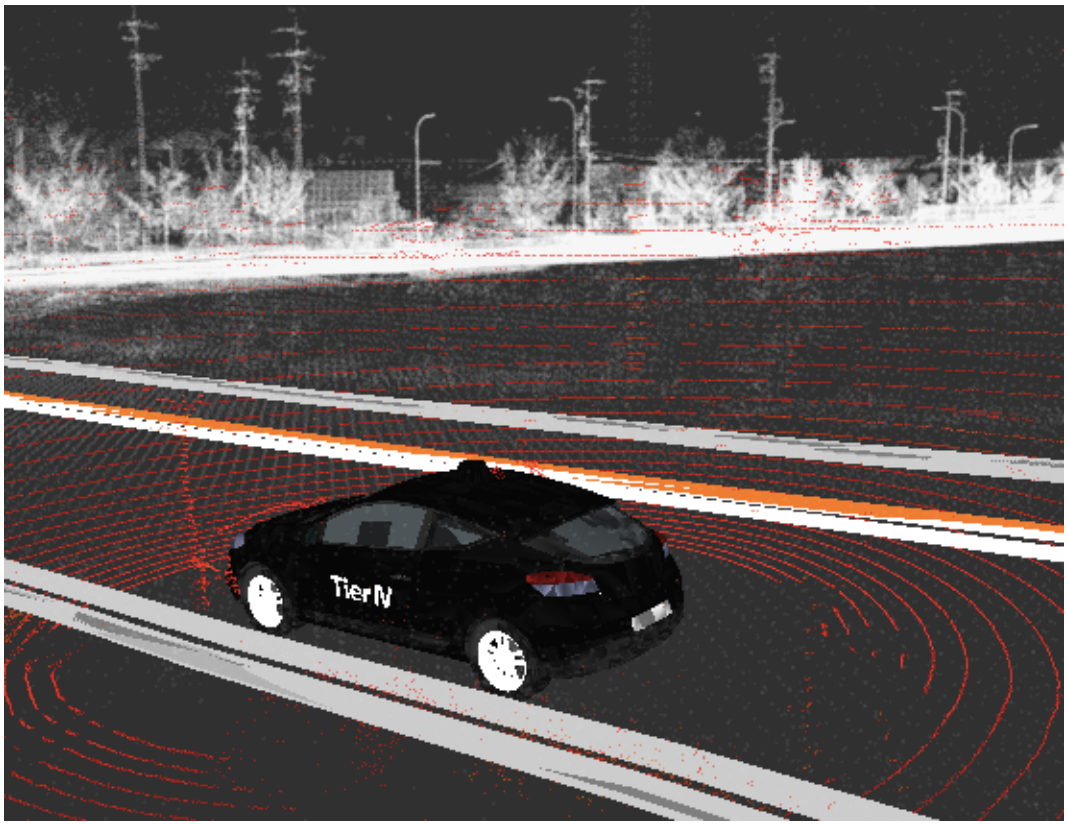

Figure 2: Autoware, by Tier IV

7.4.5 Community

The project is under active development, and has a very wide community of both companies and hobbyists that use it for simulation and field testing.

7.4.6 Licensing

The project has been released under the New BSD License, which enables free use, modification, and redistribution.

7.4.7 Autonomous driving

There are several implementations of autonomous driving using the project, most of them using a real car.

7.4.8 Multi-core simulation

The simulator takes advantage of multi-core processing.

7.5 Veins - Vehicular network simulation framework [23]

7.5.1 Mapping

The project already has examples of obstacles, multi-lane roads, and other vehicles on the map.

7.5.2 Sensoring

The simulator provides information about the vehicle's current position, lane number, detected obstacles and traffic lights.

7.5.3 API

The simulator provides an API for the graphical interface and a Traffic Control Interface (TraCI) that allows different commands such as changing traffic lights states, changing routes and vehicle states.

7.5.4 Graphical Interface

The simulator provides a simple 2D graphical interface, capable of displaying multi-lane roads, obstacles, vehicles on movement, and occasional lane changes and crashes.

7.5.5 Community

The project is under active development, and has been updated frequently.

7.5.6 Licensing

The project has been released under the GPL License, which enables free use, modification, and redistribution.

7.5.7 Autonomous driving

The simulator can execute predefined VANET models, and it is possible to change the simulation while it is running.

7.5.8 Multi-core simulation

The simulator does not use multi-core processing.

Figure 3: Veins - Vehicular network simulation framework

7.6 TORCS - The Open Racing Car Simulator [29]

7.6.1 Mapping

The simulator already has examples of multi-lane roads, and other vehicles on the map, but no examples of obstacles that aren't vehicles.

7.6.2 Sensoring

The simulator provides other vehicle's information, including their speed and relative position. Non-vehicle obstacles information on sensor fusion would need to be implemented.

7.6.3 API

The simulator allows multiple "robots" to be attached to it, via dynamic-link libraries (Windows) or shared objects (Linux). The API allows access to the map, car position, and other opponent's position

7.6.4 Graphical Interface

The simulator provides a 3D graphical interface, capable of displaying multi-lane roads, obstacles, vehicles on movement, and occasional lane changes and crashes.

7.6.5 Community

The simulator has a large developer community, and it promotes a large annual online contest for developers to build car agents that will race against each

Figure 4: TORCS - The Open Racing Car Simulator

other.

7.6.6 Licensing

The project has been released under the GPL License, which enables free use, modification, and redistribution.

7.6.7 Autonomous driving

There are some implementations of autonomous driving using the project, such as [4] and [8].

7.6.8 Multi-core simulation

The simulator takes advantage of multi-core processing.

7.7 Choosing the simulator

After analyzing the four simulators, the Udacity's Self-Driving Car Simulator was chosen as a base for implementing the AVCP testing environment. Feature-wise, one of the best options were the Veins simulator, but Udacity's simulator simplicity and extensibility for more realistic self-driving implementations were considered more relevant.

8 AVCP Protocol

The AVCP (Autonomous Vehicle Coordination Protocol) is an intent-based vehicle-to-everything (V2X) coordination protocol. It aims to reduce travel time and increase security by enabling self-driving vehicles to exchange sensing and routing information with each other and with roadside units (RSUs).

For example, a route intention, such as stopping at a place to pick up a passenger, would be broadcast by the vehicle so that the vehicles behind can decide early whether or not to change lanes. Sensing information, such as an obstacle detection (e.g. street work, broken vehicle) would also be broadcast and used by vehicles behind to improve their navigation. A scenario that involves the cars ahead would be an emergency vehicle requesting priority. In this case, the cars ahead would give way to the emergency vehicle, so that it arrives faster at its destination. These and other scenarios that the protocol might come into use are described in section 2.

Each message contains the vehicle unique identifier (UID), the AVCP version, the type of vehicle (land, air or sea), its length and width, its location (GPS coordinates - latitude/longitude and altitude when it's an air vehicle), its heading and speed information.

The messages sent by vehicles to other vehicles or controlers (RSUs) nearby can be of the following types:

- ”emergency”: an incident occurred that should make vehicles nearby take emergency procedures defined by the manufacturer (e.g.: find an alternative route, reduce speed)
- ”short-stop”: the vehicle will stop on a specific location for a short period of time. Vehicles nearby can decide whether they should change lanes or not, according to their distance.
- ”request”: a request for information of a nearby controller (e.g.: smart traffic light). The controller will respond only if necessary, by verifying the location of the vehicle that was sent on the request. The vehicle will send this request whenever it detects that a controller can be available to support its decisions (e.g.: when a car is approaching an intersection). The vehicle's destination must be provided in this message.

- ”avoidObstacle”: the vehicles behind should avoid the detected obstacle’s location (specified on the message).
- ”priority”: the vehicle has priority thus the vehicles ahead should give it preference. The vehicle’s authority must be validated using the digital signature of the message. Each manufacturer must update the list of valid authorities, according to each country/region laws.

The messages sent by the controllers to vehicles nearby can be of the following types:

1. ”proceed”: the vehicle can continue
2. ”change”: the vehicle must change its speed and/or direction. The new speed and/or direction information must be provided in the message.
3. ”stop”: the vehicle should stop
4. ”emergency”: the vehicle should take emergency procedures defined by each manufacturer (e.g.: a manufacturer can configure the vehicle to stop or to try to find an alternative route depending on the controller distance)

The details and specifications of the messages are described in section 10.

9 AVCP Simulator: implementation report

9.1 Udacity's Self-Driving Car Simulator state

Udacity's Self-Driving Car Simulator is available on Github [14] and it is used worldwide by students enrolled on Udacity's Self-Driving Car Engineer Nanodegree. The simulator was developed to enable students to test algorithms that are used by car manufacturers such as Mercedes-Benz and Tesla Motors.

The project version that was chosen to be the base for the AVCP testing environment was the one that was used for the third project of Udacity's nanodegree, related to path planning. The code was located on the "term3" branch. The goal of this project was that students would create an algorithm capable of driving a car along a highway with other vehicles, changing lanes when necessary and respecting the speed limit.

The project consists of two main parts:

1. the simulator interface [14], written in C# and using the Unity game engine [26]
2. the path planning algorithm implementation [13], written in C++

9.1.1 Simulator interface (C# / Unity)

The simulator interface starts up with the car to be controlled positioned on a two-way six-lane highway with other cars passing by. There is a button that enables the "Manual Mode", in which the user can control the car manually using the arrow keys. The project uses the RoadArchitect library [19], that is able to generate complex roads, bridges and tunnels in Unity, and the SocketIO library for Unity3d [**SocketIO**] library for WebSockets communications.

The data provided from the simulator to the C++ path planner/controller via a "telemetry" WebSocket message was, according to the documentation [13]:

- "x": The car's x position in map coordinates
- "y": The car's y position in map coordinates
- "s": The car's s position in frenet coordinates
- "d": The car's d position in frenet coordinates

- ”yaw”: The car’s yaw angle in the map
- ”speed”: The car’s speed in MPH
- ”previous_path_x”: The list of x points previously given to the simulator
- ”previous_path_y”: The list of y points previously given to the simulator
- ”end_path_s”: The previous list’s last point’s frenet s value
- ”end_path_d”: The previous list’s last point’s frenet d value
- ”sensor_fusion”: A 2d vector of sensor data about the cars nearby (car’s unique ID, car’s x position in map coordinates, car’s y position in map coordinates, car’s x velocity in m/s, car’s y velocity in m/s, car’s s position in frenet coordinates, car’s d position in frenet coordinates).

The car received from the path planner a list of points (x,y) that it visited every .02 seconds. The spacing of the points, in meters, determined the speed of the car.

9.1.2 Path planner/controller (C++)

The path planner needed to read a .csv file that contained the highway map waypoints information.

The project [13] made available by Udacity contained only the skeleton code that the students were to use to communicate with the simulator interface. Thus, it was necessary to choose a solution from one of the nanodegree students, and request the author for authorization. The chosen solution [30] was used as a starting point for developing the AVCP implementation.

The path planner was organized in the following classes:

- PathPlanner: path planning orchestrator
- Map: map waypoint information object
- Track: track information object (represented by a Spline)
- tools: helper functions and plotting utilities

The idea of this solution was basically to keep the car following the lane until another car on the front was closer than the so-called ”secure distance”. This secure distance got higher as the car speeded up, and when it was violated the

planner entered a state to change lanes. The software fitted a spline from the current lane to the targeted lane to make a smooth transition.

9.2 Modifications

The following modifications have been performed on Udacity's Self-Driving Car Simulator that resulted in the AVCP Simulator:

1. Controlling multiple cars on the road
2. Routing
3. Vehicle communication

Their implementation, available on GitHub [20] and released under the MIT License, will be detailed in the next sections.

9.2.1 Controlling multiple cars on the road

Udacity's simulator was designed to support controlling only one vehicle at a time, via Web Sockets communication between the Unity/C# interface and the C++ path planner/controller.

This modification involved changing this communication for "telemetry" and for "control" messages. Both of them now contain an array with the information for each car (and each car is assigned an UID).

9.2.2 Routing

Udacity's simulator used a closed track, thus it could model the entire car trajectory using spline interpolation. For the AVCP testing environment, we needed to consider intersections and alternative paths, thus, a routing system was implemented, with the cars having different goals, such as arriving at a waypoint α, or arriving at another waypoint β.

The first step was to enable the path planner independent of a static .csv file, by creating a message of type "mapData" sent by the simulator, that contained the map's waypoints information. This required changing the RoadArchitect library to support exporting this data programmatically.

Then, on the PathPlanner side, it was necessary to save the waypoints using proper data structures. The waypoints were stored on a K-d tree [3], for fast nearest neighbor search. The connections between the waypoints were stored on a

directed graph. After finding the nearest waypoint to the car, the planner could then calculate the fastest route to its goal, using the Dijkstra's algorithm [18]. Currently, this feature is still being implemented.

9.2.3 Vehicle communication

Udacity's simulator did not have any support for vehicle-to-vehicle (V2V) or vehicle-to-infrastructure (V2I) communication, thus, it was necessary to implement it. Currently, this feature is still being implemented.

9.3 Results

Until now, we were able to execute the original path planning solution simultaneously on multiple cars. We were able to control one of the cars manually and trigger lane changes by the other vehicles by moving to the front of them on the highway.

10 Appendix A: AVCP protocol draft

10.1 Definitions

- UUID: Universal Unique Identifier
- OBU: On-Board Unit, a device inside the vehicle that allows its wireless communication with other OBUs and RSUs
- RSU: Road-Side Unit, a device on the roadside that communicates with OBUs

10.2 Syntax specification

10.2.1 File format and encoding

Every AVCP payload must be in the ECMA-404 / JavaScript Object Notation (JSON) format [16]. The data should be encoded using UTF-8 without BOM (byte order mark), and can be compressed using gzip [6]. For security purposes, only digitally-signed requests should be accepted, signed by each manufacturer.

10.2.2 Vehicle/OBU Messages

```
{
    "uuid": String,
    "avcpVersion": "1.0.0",
    "operationType": OperationType,
    "lengthMeters": Double,
    "widthMeters": Double,
    "cruiseInformation": CruiseInformation,
    "location": Location,
    "vehicleMessageType": VehicleMessageType
}
```

Notes:

1. The UUID must be unique for each vehicle
2. It is recommended that each manufacturer prefix it with its name (e.g.: ”ManufacturerName-ABCXYZ123QWERTY”)

10.2.3 Controller/RSU Messages

```
{
    "uuid": String,
    "operationType": OperationType,
    "location": Location,
    "controllerMessageType": ControllerMessageType,
    "toUUID": String
}
```

Notes:

1. The UUID must be unique for each controller
2. It is recommended that each manufacturer prefix the UUID with its name (e.g.: ”ManufacturerName-ABCXYZ123QWERTY”)
3. ”toUUID” can be a wildcard ”*” to match any vehicle UUID

10.2.4 Structures

10.2.4.1 VehicleMessageType

There are 5 types of messages that can be sent by vehicles:

1. Emergency: an incident occurred that should make vehicles nearby take emergency procedures defined by the manufacturer (e.g.: find an alternative route, reduce speed).

```
{
    "action": "emergency"
}
```

2. Short stop: the vehicle will stop on a specific location for a short period of time. Vehicles nearby can decide whether they should change lanes or not, according to their distance.

```
{
    "action": "short-stop",
    "currentTimestampUnixGMT": Long,
    "timeoutMilliseconds": Long
```

```
}
```

3. Request: a request for information of a nearby controller (e.g.: smart traffic light). The controller will respond only if necessary, by verifying the location of the vehicle that was sent on the request. The vehicle will send this request whenever it detects that a controller can be available to support its decisions (e.g.: when a car is approaching an intersection).

```
{
    "action": "request",
    "destination": Location
}
```

4. Avoid obstacle: the vehicles behind should avoid the detected obstacle's location.

```
{
    "action": "avoidObstacle",
    "location": Location
}
```

5. Priority: the vehicle has priority thus the vehicles ahead should give it preference. The vehicle's authority must be validated using the digital signature of the message. Each manufacturer must update the list of valid authorities, according to each country/region laws.

```
{
    "action": "priority"
}
```

10.2.4.2 ControllerMessageType

There are 4 types of messages that can be sent by controllers:

1. Proceed: the vehicle can continue

```
{
    "action": "proceed"
}
```

2. Change: the vehicle must change its speed and/or direction

```
{
    "action": "change",
    "cruiseInfo": CruiseInfo
}
```

3. Stop: the vehicle should stop

```
{
    "action": "stop"
}
```

4. Emergency: the vehicle should take emergency procedures defined by each manufacturer (e.g.: a manufacturer can configure the vehicle to stop or to try to find an alternative route depending on the controller distance)

```
{
    "action": "emergency"
}
```

10.2.4.3 OperationType

Land operation:

```
"land"
```

Sea operation:

```
"sea"
```

Air operation:

```
"air"
```

10.2.5 Location

Land and sea operation:

```
{
    "latitude": Double,
```

```
    "longitude": Double,
    "gpsMaximumErrorMeters": Double
}
```

Air operation:

```
{
    "latitude": Double,
    "longitude": Double,,
    "gpsMaximumErrorMeters": Double,
    "altitudeMeters": Double,
    "altitudeMaximumErrorMeters": Double
}
```

10.2.6 CruiseInformation

Land and sea operation:

```
{
    "speedMetersPerSecond": Double,
    "headingDegrees": Double
}
```

Air operation:

```
{
    "speedMetersPerSecond": Double,
    "headingDegrees": Double,
    "verticalSpeedMetersPerSecond": Double
}
```

References

[1] N. Alsharif, S. Céspedes, and X. S. Shen. “iCAR: Intersection-based connectivity aware routing in vehicular ad hoc networks”. In: *2013 IEEE International Conference on Communications (ICC)*. June 2013, pp. 1736–1741. DOI: 10.1109/ICC.2013.6654769.

[2] Mikael Asplund et al. “A Formal Approach to Autonomous Vehicle Coordination”. In: *FM 2012: Formal Methods: 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings*. Ed. by Dimitra Giannakopoulou and Dominique Méry. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 52–67. ISBN: 978-3-642-32759-9. DOI: 10.1007/978-3-642-32759-9_8. URL: https://doi.org/10.1007/978-3-642-32759-9_8.

[3] Jon Louis Bentley. “Multidimensional Binary Search Trees Used for Associative Searching”. In: *Commun. ACM* 18.9 (Sept. 1975), pp. 509–517. ISSN: 0001-0782. DOI: 10.1145/361002.361007. URL: http://doi.acm.org/10.1145/361002.361007.

[4] Chenyi Chen et al. “Deepdriving: Learning affordance for direct perception in autonomous driving”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2722–2730.

[5] P. S. Nithya Darisini and N. S. Kumari. “A survey of routing protocols for VANET in urban scenarios”. In: *2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering*. Feb. 2013, pp. 464–467. DOI: 10.1109/ICPRIME.2013.6496522.

[6] P. Deutsch. *GZIP file format specification version 4.3*. RFC 1952 (Informational). Internet Engineering Task Force, May 1996. URL: http://www.ietf.org/rfc/rfc1952.txt.

[7] Kurt Dresner and Peter Stone. “A Multiagent Approach to Autonomous Intersection Management”. In: *J. Artif. Int. Res.* 31.1 (Mar. 2008), pp. 591–656. ISSN: 1076-9757. URL: http://dl.acm.org/citation.cfm?id=1622655.1622672.

[8] Adithya Ganesh et al. “Deep Reinforcement Learning for Simulated Autonomous Driving”. In: ().

[9] The Guardian. *Tesla driver dies in first fatal crash while using autopilot mode*. 2016. URL: https://www.theguardian.com/technology/2016/jun/30/tesla-autopilot-death-self-driving-car-elon-musk (visited on 07/08/2017).

[10] Alphabet, Inc. *Google Scholar*. URL: https://scholar.google.com (visited on 07/08/2017).

[11] Alphabet, Inc. *Google search engine*. URL: https://google.com (visited on 07/08/2017).

[12] GitHub, Inc. *GitHub*. URL: https://github.com (visited on 07/08/2017).

[13] Udacity, Inc. *CarND-Path-Planning-Project*. URL: https://github.com/udacity/CarND-Path-Planning-Project (visited on 07/08/2017).

[14] Udacity, Inc. *Udacity's Self-Driving Car Simulator*. URL: https://github.com/udacity/self-driving-car-sim (visited on 07/08/2017).

[15] Tier IV. *Autoware*. URL: https://github.com/CPFL/Autoware (visited on 07/08/2017).

[16] *The JSON Data Interchange Format*. Tech. rep. Standard ECMA-404 1st Edition / October 2013. ECMA, Oct. 2013. URL: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

[17] G. S. Khekare and A. V. Sakhare. "A smart city framework for intelligent traffic system using VANET". In: *2013 International Mutli-Conference on Automation, Computing, Communication, Control and Compressed Sensing (iMac4s)*. Mar. 2013, pp. 302–305. DOI: 10.1109/iMac4s.2013.6526427.

[18] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005. ISBN: 0321295358.

[19] MicroGSD. *Road Architect*. URL: https://github.com/MicroGSD/RoadArchitect (visited on 07/08/2017).

[20] Marcelo Paulon. *avcp-self-driving-sim*. URL: https://github.com/marcelopaulon/avcp-self-driving-sim (visited on 12/08/2017).

[21] J. Rios-Torres and A. A. Malikopoulos. "A Survey on the Coordination of Connected and Automated Vehicles at Intersections and Merging at Highway On-Ramps". In: *IEEE Transactions on Intelligent Transportation Systems* 18.5

(May 2017), pp. 1066–1077. ISSN: 1524-9050. DOI: 10.1109/TITS.2016.2600504.

[22] M. L. Sin, M. Bouroche, and V. Cahill. “Scheduling of Dynamic Participants in Real-Time Distributed Systems”. In: *2011 IEEE 30th International Symposium on Reliable Distributed Systems*. Oct. 2011, pp. 245–254. DOI: 10.1109/SRDS.2011.37.

[23] Christoph Sommer. *Veins - Vehicular network simulation framework*. URL: http://veins.car2x.org/ (visited on 07/08/2017).

[24] Christoph Sommer, Reinhard German, and Falko Dressler. “Bidirectionally coupled network and road traffic simulation for improved IVC analysis”. In: *IEEE Transactions on Mobile Computing* 10.1 (2011), pp. 3–15.

[25] The Telegraph. *Self-driving cars will add $7 trillion a year to global economy, says Intel*. 2017. URL: http://www.telegraph.co.uk/business/2017/06/02/self-driving-cars-will-add-7-trillion-year-global-economy-says (visited on 07/08/2017).

[26] Unity Game Engine. “Unity game engine-official site”. In: (). URL: https://unity3d.com (visited on 07/08/2017).

[27] Kathy Winter. *For Self-Driving Cars, There’s Big Meaning Behind One Big Number: 4 Terabytes*. https://newsroom.intel.com/editorials/self-driving-cars-big-meaning-behind-one-number-4-terabytes. Accessed: 2017-09-30.

[28] W. K. Wolterink, G. Heijenk, and G. Karagiannis. “Constrained geocast to support Cooperative Adaptive Cruise Control (CACC) merging”. In: *2010 IEEE Vehicular Networking Conference*. 2010, pp. 41–48. DOI: 10.1109/VNC.2010.5698268.

[29] Bernhard Wymann et al. *TORCS, The Open Racing Car Simulator*. 2014. URL: http://www.torcs.org (visited on 07/08/2017).

[30] yosoufe. *CarND-Path-Planning-Project*. URL: https://github.com/yosoufe/CarND-Path-Planning-Project (visited on 07/08/2017).