Decision Making Protocol in Autonomous Vehicles for Optimal Routing and Safe Control

by

Lakhan Shiva Kamireddy

B.E.(Hons), Birla Institute of Technology and Science, 2015

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Masters of Science

Department of Electrical, Computer, and Energy Engineering

2019

This thesis entitled:
Decision Making Protocol in Autonomous Vehicles for Optimal Routing and Safe Control
written by Lakhan Shiva Kamireddy
has been approved for the Department of Electrical, Computer, and Energy Engineering

_____

Prof. Fabio Somenzi

_____

Prof. Sriram Sankaranarayanan

Date _____

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Lakhan Shiva Kamireddy, (M.S., ECEE)

Decision Making Protocol in Autonomous Vehicles for Optimal Routing and Safe Control

Thesis directed by Prof. Fabio Somenzi

Abstract

We analyze two significant decision-making problems namely planning and control in an autonomous driving scenario with an objective of not only improving the safety of drives but also enhancing the vehicle's capability to make better decisions. We often see human-drivers deciding to travel on a preferred route by selecting them from the available choice set at an intersection based on information about the real-time delays and congestions on routes reported by apps like Google Maps, Waze. When we have autonomous vehicles on the roads, this decision problem still exists, but the self-driving vehicles can be modeled as rational and intelligent agents. We, therefore, model this decision-making problem and solve for an optimal route when a vehicle approaches an intersection. The proposed solution attempts to balance the congestion in the traffic network, considering the selfish nature of the vehicles. We also analyze decision-making at the level of control and communications. We model this problem in an autonomous driving scenario for taking high-level control decisions. We compare our protocol with a related work that approaches the problem differently. The choices of the framework we use, to model the communication layer of the protocol makes it easier for us to verify and model check the protocol. For example, we implement the process algebra description using Promela language, and we prove the protocol's correctness guarantees using SPIN model checker. We use Reinforcement Learning (RL) algorithm in the design of control protocol. RL enables the vehicles to get better at taking control decisions as they learn from experience in a safe simulation environment. They are based on a Markov Decision Process (MDP) formulation. We use neural networks as function approximators in the protocol for better performance, and formally verifying neural networks involving images is still an open research problem.

Dedication

To my parents, siblings, and my teachers

Acknowledgements

This dissertation would not have been possible without the support, guidance, and encouragement that I have received from many people. Firstly, I would like to thank my advisors Prof. Fabio Somenzi and Prof. Sriram Sankaranarayanan for their support and encouragement. I want to thank Prof. Ashutosh Trivedi for all the discussions we had on approaching such problems.

I would also like to thank the faculty whose classes made me get closer to my goal of successfully completing the program. I want to thank my friends without whom attending graduate school and completing this thesis would not have been as enjoyable. I would also like to extend my warm wishes to Adam Sadoff, graduate advisor of the ECEE department for supporting me throughout the program. I want to extend good wishes and acknowledge the people and friends with whom I worked during my stay here, for being wonderful and making life all the more pleasant.

Finally, I express my gratitude to my parents Aparna and Jeevan Reddy for providing me with unfailing support and continuous encouragement throughout my studies. This accomplishment would not have been possible without them.

Contents

Chapter

1 Introduction **1**

1.1 Problem statements 1

1.2 Vehicular Communication 2

1.3 Related Work 5

2 Modeling the system **8**

2.1 Planning Problem 9

2.2 Background 9

2.3 Autonomous Vehicles and Traffic 13

3 Modeling Framework for Autonomous Vehicle Communication **15**

3.1 Platooning 15

3.2 π-calculus and concurrency 18

4 Learning and Autonomous Vehicle Control **23**

4.1 Reinforcement Learning for Control of Autonomous Vehicles 25

4.2 Related Work Using Game Theory 30

5 Conclusion and Future Work **32**

5.1 Results 32

5.2 Conclusion 33

5.3 Future work 34

Bibliography **36**

Appendix

A Platooning Protocol Implemented in Promela **38**

A.1 Promela Code 38

A.2 SPIN Simulation log 45

B Platooning Expressed in π-calculus for The Mobility Workbench **51**

B.1 Mobility Workbench code 51

Tables

Table

2.1 Cost matrix for congestion game I 10

2.2 Cost matrix for congestion game II 11

3.1 Comparing available protocol implementation choices 20

4.1 Neural network summary 29

Figures

Figure

2.1 Tree diagram of protocol 8

2.2 Congestion game 11

2.3 Illustration of a network graph 13

3.1 Communication among concurrent processes 17

3.2 Components of platooning 18

3.3 Joiner attempts to join the platoon 19

3.4 Joined merges into the platoon 19

3.5 Model checking setup using SPIN 21

5.1 Simulation of the autonomous control agent 33

Chapter 1

Introduction

1.1 Problem statements

There are two decision-making problems that we model and analyze in this research. The first problem is at the level of planning routing decisions for ego vehicles in a congested traffic network. The second problem models high-level control decisions and verifies the safety of interactions in autonomous vehicles.

1.1.1 Planning of routes

Given a real-time traffic density map of the city, N vehicles within a particular vicinity of an intersection of roads, for each vehicle, a set of available choices of routes to reach their respective destination, we design a protocol to find the optimal assignment of routes to vehicles that maximizes their payoffs by reducing the collective commute time. There are two principal objectives that we would like to achieve through mechanism design. (1) To model and solve the problem of optimal vehicle routing in a traffic network considering the selfish nature of the vehicles (2) Improve the overall safety of drives.

We will look at the game-theoretic mechanism design in chapter II. It aims to minimize the traffic congestion and thereby reduces the cumulative time taken to reach the destination for the vehicles in consideration. When there is an intersection manager also referred to as Road Side Equipment (RSE), they function as servers in the exchange of traffic information. They help exchange traffic information with the vehicles about various road segments of the network. The

protocol of assigning roads to autonomous vehicles is based on the requests sent by the vehicles according to the given real-time traffic scenario. This involves the exchange of real-time road traffic information between an Intersection Manager and vehicles that lie within its range.

1.1.2 High-Level Control Decisions in Autonomous Driving

In [1] J. Campbell et al. have presented a protocol and a modeling framework for autonomous vehicle communication and reconfiguration using π-calculus. They used this framework to express platooning behavior through π-calculus expressions. π-calculus is a process calculus for describing and analyzing properties of concurrent computations. In chapter III, we discuss the proposed framework in detail and its relevance to this research.

The idea is to bring together the pi-calculus based modeling framework and learning based decision processes into the control protocol we design. Thus, this protocol is easier to model check for temporal safety properties. It's performance metrics are shown to improve by learning from experiences to make better decisions. The ideal properties we expect in this protocol are to avoid collisions, dangerous driving behavior, and take more correct decisions quantified using a reward function.

1.2 Vehicular Communication

Advances in technologies like Vehicular Adhoc Networks (VANETs) that support Vehicle-to-Vehicle (V2V), Vehicle-to-Infrastructure (V2I) and Vehicle-to-Pedestrian (V2P) communications have brought forth opportunities for further advancing the intersection management protocols. We aim to improve the efficiency and functional correctness of the protocol while taking routing decisions or taking high-level control decisions.

1.2.1 Vehicle-to-infrastructure Interaction

According to the United States Department of Transportation, Vehicle-to-Infrastructure (V2I) is the next generation of Intelligent Transportation Systems (ITS) [2]. V2I technologies

capture vehicle-generated traffic data, wirelessly providing information such as advisories from the infrastructure to the vehicle that inform the driver of safety, mobility, or environment-related conditions. The US Department of Transportation foresees installing V2I infrastructure alongside the ITS deployments in the near future. Some potential applications of V2I communication are as follows.

- *Red light violation warning*: Alerts about potential violations of upcoming red lights.
- *Curve speed warning*: Alerts if the driver's current speed is unsafe for upcoming road curve.
- *Stop sign gap assist*: Assists drivers at stop sign intersections, alerting them when it is unsafe to enter the intersection.
- *Reduced speed zone warning*: Alerts drivers to reduce speed in work zones, etc.
- *Spot weather information warning*: Warns about real-time weather events.

1.2.2 Vehicle-to-vehicle Interaction

Vehicle-to-vehicle Interaction (V2V) technology is being researched for developing state of the art Intelligent Transportation Systems. In V2V technology, vehicles communicate with each other by exchanging information such as vehicle size, position, speed, heading, acceleration, brake status, steering angle, turn signal status, etc. thus enabling safety and mobility applications. V2V communication could have the following potential applications concerning the safety of the drives [3].

- *Forward Collision Warning*: To detect stopped vehicles at a long range.
- *Blind spot & Lane Change Warning*: Warning system that alerts the vehicle during a lane change attempt if the blind spot zone into which the vehicle intends to switch to, is, or will soon be occupied by another vehicle.

- *Left Turn Assist*: Alerts the driver when they attempt an unprotected left turn across traffic, to help avoid collision with opposite direction traffic.

- *Intersection Movement Assist*: Warning system that alerts the driver when it is not safe to enter an intersection. For example, when something is blocking the driver's view of the opposing or crossing traffic.

- *Vehicle Turn Right in Front of Bus Warning*: Alerts the bus operators of the presence of vehicles attempting to go around the bus to make a right turn as the bus departs from a bus stop.

V2V devices require two DSRC radios, a GPS receiver with a processor at the minimum to infer information like the vehicle speed and the predicted path from the device's GPS data. V2V communications consist of two types of messages. (1) Safety messages are used for safety applications (2) Certificate exchange messages ensure that safety message is from a trusted source.

1.2.2.1 DSRC

Dedicated Short Range Communications (DSRC) is an open-source protocol for wireless communication, similar in some respects to WiFi. While WiFi is used mainly for wireless Local Area Networks, DSRC is intended for highly secure, high-speed wireless communication between vehicles and the infrastructure. The key features of DSRC are:

- Low Latency: The delays involved in opening and closing a connection are very short, on the order of 0.02 seconds.

- Limited Interference: DSRC is very robust in the face of radio interference. Furthermore, its short range ($\sim$1000 m) limits the chance of interference from distant sources. Besides, the Federal Communications Commission (FCC) protects DSRC for transportation applications. Although it is intended for transportation safety applications, it does not restrict itself, and rather one can also develop purely commercial convenience applications.

- Strong performance during adverse weather conditions.

1.2.3 Human-Robot Interaction

Active research is being carried out in this area, as there are safety concerns while autonomous vehicles enter the humans' world, and continue their interaction with humans [4] [5]. While Vehicle to Pedestrian (V2P) interactions are necessary for facilitating the deployment of autonomous vehicles to coexist with humans, human-in-the-loop control interactions and driver modeling are also topics of interest, to realize shared control and to guarantee the safety of these deployments [5]. Such communication between the two parties is essential to enable pedestrians to coexist safely on roads with autonomous vehicles performing their missions. However, while understanding pedestrian behavior is not intuitive, it is challenging, as not all humans behave according to a generalizable model. It seems that no one model of human works perfectly in all scenarios. Current research aims to integrate computational cognitive models of humans into planning and control protocols for robots [20]. Reliable prediction of driver behavior is essential not only for safe interaction with other vehicles on the road but also in developing active safety systems and interfaces between the driver and the autonomous vehicle. Research is also being carried out in designing human-robot interfaces.

1.3 Related Work

In 1975, L. J. LeBlanc et al., have published their work [6] on a solution to the road network equilibrium-traffic assignment problem. In their words, the equilibrium traffic assignment problem has been stated as follows. We are given a set of roads and zones representing a particular urban area, and we have estimates of the total demand for travel via the road between each pair of zones. We wish to determine how this traffic will distribute itself over the roads of the area. Typically, we have projections of the traffic between each zone pair in some future period, and we wish to determine if the existing or various proposed road networks can handle the increased traffic, what the associated user costs or travel time will be, etc.

A system of roads and intersections has been modeled by a network in which the nodes are used to represent intersections and edges represent the roads. Nodes are connected by directed arcs so that a two-way street is modeled by two arcs in opposite directions. The network was used to represent only the major streets of an urban area, ignoring minor roads. The assumption that each driver takes the path of least cost between his origin and destination gives rise to the concept of network equilibrium. Traffic Equilibrium is an aggregate result of individual decisions taken by vehicles. Also, at equilibrium, no single driver can reduce his own cost by choosing an alternative route in the network.

S. Kim et al., in [7] have combined real-time traffic information with historical traffic data to develop better routing strategies. They also showed that made the services of freight companies more efficient. They modeled the routing problem as a discrete-time, finite horizon Markov decision process (MDP). However, the problem they tried to solve is different from the problem we choose for this research. While they developed decision-making procedures to determine the optimal driver attendance time, optimal departure times, and optimal routing policies, we design a protocol whose objective is to improve safety and performance. We are considering autonomous vehicles, and we aim to reduce traffic congestion and improve vehicle drive performance by routing vehicles to reduce travel time.

S. Coogan et al., in [8] propose a framework for generating a control policy for traffic networks of signalized intersections to accomplish control objectives that are expressed using linear temporal logic. They also worked out a probabilistic extension by modeling the traffic dynamics as Markov Decision Processes. However, the main contribution of their paper is to synthesize a signal control strategy, such that the resultant traffic dynamics and signal sequences satisfy a control objective expressed using *linear temporal logic (LTL)*. In summary, they followed the approach as outlined by them in [6]. They obtain a deterministic Rabin automaton that accepts all and only trajectories that satisfy the LTL specifications. They then construct the synchronous product of the Rabin automaton and T_{prod} that gives a nondeterministic Rabin automaton from which the control strategy is found by playing a Rabin game.

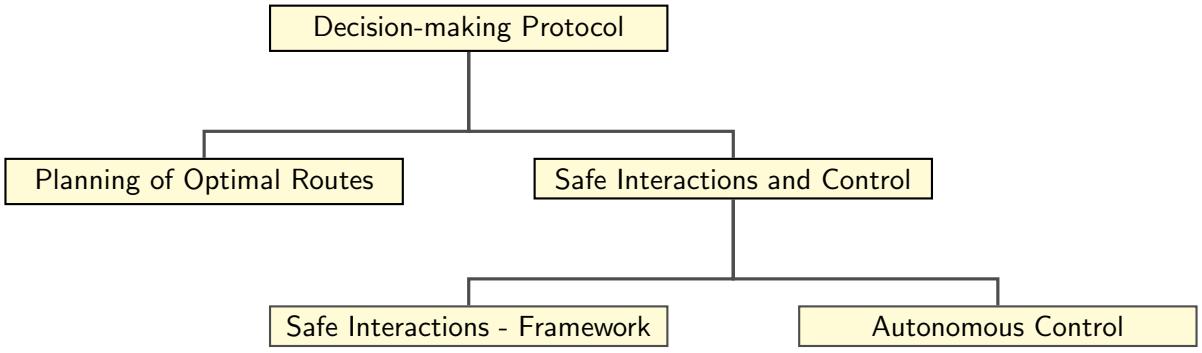
Jerome Thai's work in [9] is on learning game-theoretic models with applications to urban mobility. They approximate real transportation systems using game-theoretic models. They also propose a new framework for estimating highway traffic. Their work is a more rigorous study of some of the topics discussed in chapter 2 of this work.

Chapter 2

Modeling the system

As we are attempting solutions to planning and control, we decouple them as two decision-making problems and solve them separately [17]. These can be seen as two layers of decision-making, each solving for a different objective. In the first layer, we look at the problem from a helicopter view, where we focus on the high-level planning of vehicles for optimal routing, we analyze reducing the congestion on the roads and find an equilibrium traffic flow. In the second layer, we focus on high-level control of an autonomous vehicle and see how learning and control can go hand in hand to both improve the safety and achieve better control of autonomous vehicles while also facilitating vehicle-to-vehicle interactions.

Figure 2.1: Tree diagram of the decision-making protocol



2.1 Planning Problem

There are n vehicles that lie in close vicinity of an intersection or a fork. For each autonomous vehicle, there is a set of available choices of routes to reach their respective destination. These choices are called strategies or actions in game theory. Each vehicle selfishly wants to minimize its own time to destination. The cost of choosing a route is inversely proportional to the number of vehicles using that route. The problem of optimal assignment of autonomous vehicles to routes by leveraging the vehicle-to-infrastructure (V2I) communication can be modeled as a congestion game problem, and we call it as *selfish load balancing*.

2.2 Background

We use game theory in the following analysis. This section on game theory should be enough background to follow the analysis we perform in the following sections to model the problem using game theoretic analysis.

Definition 2.1. A strategic game $\langle$ N, (A_i), $\succeq_i$ $\rangle$ consists of:

- a finite set N (the set of players),
- for each player i $\in$ N a nonempty set A_i (set of actions available to player i),
- for each player i $\in$ N a preference relation $\succeq_i$ on A = $x_{j \in N} A_j$ (the preference relation of player i).

Definition 2.2. Nash equilibrium is considered as a central solution concept for a game. The concept of Nash equilibrium is that if each player chooses their part of the Nash equilibrium strategy, then no other player has a reason to deviate to another strategy [10]. Such a deviation would only result in sub-optimal payoffs for the player.

2.2.1 Congestion Games

Congestion games are a class of games in Game Theory [10]. In a congestion game, we define players and resources, where the payoff of each player depends on the resources it chooses and the number of players choosing the same resource.

Let us consider a directed graph illustrated in Fig. 2.1 where each player has two available strategies, either going through A or going through B. This leads to a total of four possibilities. Table 2.1 represents the costs of the players in terms of delays depending on their choices. Players prefer these costs to be small.

Table 2.1: Cost matrix I

S1 \ S2	A	B
A	(6,6)	(2,3)
B	(3,2)	(7,7)

If player one chooses A, then player two incurs cost 6 for choosing A and cost 3 for choosing B. Hence player two would choose B to minimize its cost given player 1's choice. Then both (A, B) and (B, A) are pure Nash equilibria in this game. In this example, a player's deviation from the Nash equilibrium would increase the cost of the player. No player would be able to gain an advantage over others by changing only their strategy. Now let us consider a cost matrix as in Table 2.2. The cost function is given to be $C(s_1, s_2) = u_1(s_1, s_2) + u_2(s_1, s_2)$. The minimum cost in this example would be when both players choose route A, and the resulting cost is 2 + 2 = 4. However, given the selfish nature of the players, each player would switch from route A to B to improve their situation, given knowledge that the other player's best decision is to choose route B. Therefore, the only Nash equilibrium occurs when both choose route B, in which case the cost is 5 + 5 = 10.

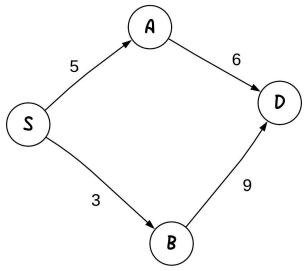
Numerical solution to find a Nash equilibrium on a cost matrix: If the first cost number in

the cost pair of the cell is the minimum of the column of the cell, and if the second number is the minimum number of the row of the cell, then the cell represents a Nash equilibrium. It is based on this idea. Each player has the knowledge that naturally an opponent's best decision is to choose the strategy that minimizes its cost. For each strategy of player one, we assume the opponent's strategy is the minimum cost given *a priori* knowledge of the strategy of player one. Same with player two. If a cost pair contains both such identified costs, then that is the Nash equilibrium.

Table 2.2: Cost matrix II

S1 \ S2	A	B
A	(2,2)	(9,0)
B	(0,9)	(5,5)

Figure 2.2: A directed graph for congestion games



Definition 2.3. The Price of Anarchy (PoA) is a concept in game theory how the efficiency of the system degrades due to selfish behavior of its agents.

In a centralized solution, a central authority can tell each agent (vehicle) which route to take in order to minimize the average travel time. In the decentralized selfish version, each agent selfishly chooses its path. The price of anarchy measures the ratio between average travel time in two cases. Thus, the price of anarchy in the above game (Table 2.2) will be $\frac{10}{4} = 2.5$.

2.2.2 Load Balancing Game

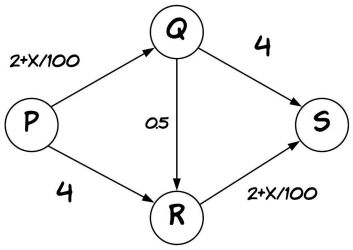
Definition 2.4. A load balancing game [11] is a congestion game based on a congestion model with:

- a finite set N of tasks (each task i has a weight w_i),
- for each player i ∈ N, there is a nonempty set A_i with $A_i \subset$ M. The elements of A_i are the possible machines on which on which task i can be executed.
- the preference relation $\succeq_i$ for each client i is defined by a payoff function u_i : A → $\mathbb{R}$. For any a ∈ A, and for any j ∈ M, let $l_j(a)$ be the expected load on machine j, assuming a is the current pure strategy profile ($l_j(a) = \sum_{\substack{i\in[n]\\ j=a_i}} w_i$). Then the payoff function for task i becomes: $u_i(a) = c_{a_i}(l_{a_i}(a))$.

Load balancing problems are native to networks and distributed systems. There is often a resource and a cost associated with using this resource. Players need to make decisions that allocate themselves to use the available resources. This involves decentralized decision making. By definition, In a load-balancing game, a fixed number m of machines are given and the cost of each player is caused by congestion, which is defined as the load of the machine, i.e., the sum of lengths of the jobs assigned to it. This is the classical load-balancing game [11] and has been extensively studied in the literature. In the following game-theoretic setting, we assume that the vehicles are selfish players, i.e. each vehicle is a player that aims at placing itself on a route that with the smallest load.

In this research we borrow the idea of load balancing from routing in networks, and extend it to solve the problem as explained in chapter I. Whenever a set of tasks are set to be executed on a set of resources, one needs to balance the load among the resources in order to achieve efficient use of the available resources [4]. The problem in this setting is *makespan scheduling on uniformly related routes*. The problem is defined by *m routes* with congestion measured in terms of delay times d_1,..., d_m and *n vehicles* with maximum speed capacities s_1,..., s_n. Let [n] = 1,..., n denote

Figure 2.3: A network graph illustration



the set of vehicles and [m] = 1,..., m the set of routes.

We seek for an assignment A:[n] → [m] of the vehicles to routes that is as balanced as possible. The *load* of route j ∈ [m] under assignment A is defined as

$$l_j = \alpha_j + \sum_{\substack{i \in n \\ j = A(i)}} k \frac{d_j}{s_i}$$

α_j is the distance of the route from source to destination, which is constant. The makespan is defined to be the maximum load over all routes. The objective is to minimize the makespan. If all routes have the same congestion delays, then the problem is known as *makespan scheduling on identical routes.*

2.3 Autonomous Vehicles and Traffic

Let us consider four autonomous vehicles v_1, v_2, v_3 and v_4 within a close vicinity at the fork Q as illustrated in Fig. 2.3. All four vehicles need to reach their destination S, as quickly as possible by avoiding traffic. They need to make a decision at the fork and choose to travel either on the road QR or on the road QS. This situation can be modeled as a game where each vehicle has a choice of 3 strategies. Each strategy is a route from P to S (either PQS, PQRS, or PRS). The payoff of each strategy is the travel time of each route. In the graph illustrated, a vehicle traveling on the route PQS incurs a travel time of $(2+\frac{x}{100})+4$, where x is the number of vehicles traveling on the road PQ. Therefore, payoffs for any given strategy depend on the choices of other players. Our goal is to minimize travel time. Equilibrium will occur when the time on all paths is the same.

When that happens, no vehicle has any incentive to switch routes, since it can only add to their travel time. For the graph illustrated in Fig. 2.3, if, for example, 500 cars are traveling from P to S, then equilibrium will occur when 125 drivers travel via PQS, 250 via PQRS, and 125 via PRS.

As we may notice, this distribution is not socially optimal. If the 500 cars agreed that 250 travel via PQS and the other 250 through PRS, then travel time for any single car would be lesser than it is currently. This is also the Nash equilibrium if the path between Q and R is removed, which means that adding another possible route can decrease the efficiency of the system, a phenomenon known as Braess's paradox [12].

Let us consider four vehicles, v_1, v_2, v_3, and v_4. The problem of choosing an optimal route involves some complexity. We see that vehicle v_1's decision influences the decision of vehicle v_2, $v_3$and v_4. Given there are two routes available, if v_1 and v_2 choose to go on the same route, then it would maximize the payoff's if v_3 and v_4 choose to go on a different route, thus balancing the load and easing the congestion in the road traffic network.

This scenario illustrates a core problem in Muti-agent decision making [13]. v_1 and v_2's decisions influence v_3 and v_4's decisions and v_3 and v_4 have to reason strategically.

Chapter 3

Modeling Framework for Autonomous Vehicle Communication

In chapter II we have seen routing-level planning where autonomous vehicles took the most optimal route. In this chapter, we discuss the modeling framework proposed by J. Campbell, et al. in [1] to improve the safety of the drives while incorporating V2V communication in the control protocols we design. We discuss how we use this framework for the design of our control protocol. The main contribution of their work [1] is to use π-calculus in describing the concurrent protocol. π-calculus is a process calculus used to describe concurrent systems and processes. Let us look at the platooning protocol modeled using this framework. Although here we specifically look at platooning in great detail, the framework and implementation paradigms discussed in the following sections are generalizable to any communication protocol that we may want to design without encountering many problems.

3.1 Platooning

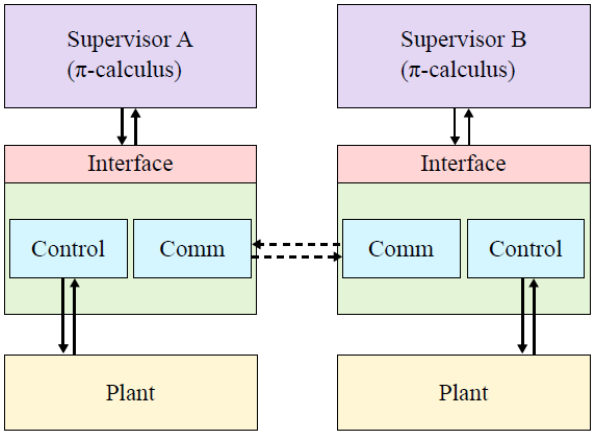
A platoon is a group of vehicles that can travel very closely together, safely at high speed. Platooning makes use of cameras, sensors, and vehicle-to-vehicle communication to establish a virtual link among the vehicles in the platoon, and the vehicles wishing to join the platoon. There is a lead vehicle that controls the speed and direction, and the following vehicles respond to the lead vehicle's movement. This allows the vehicles to automatically accelerate and brake together. This technology detects and reacts to stopped or slow vehicles ahead of the platoon, and adjusts as needed when a foreign vehicle cuts in between the vehicles in the platoon. Vehicle platooning

has been shown to have several advantages as follows.

- It improves the aerodynamic effectiveness and performances, increases the capacity of roads and provides a more steady traffic flow.
- It improves energy efficiency by reducing fuel consumption because the vehicles are drafting one another.
- It improves safety and reduces the likelihood of accidents.

The modeling framework in [1] models concurrency and reconfiguration in vehicular networks. High-level communication protocols in the framework are described using π-calculus and the closed-loop control dynamics are modeled using hybrid automata. Their main contribution is to bring both of these together into the same unified framework. It enables for us to model different communication protocols in a hierarchy, where at the highest level of the hierarchy we can verify the correctness of the protocols. This sort of hierarchical modeling separates concerns into different layers of the framework. In Fig. 3.1, we can see the organization of these layers in two concurrent vehicle processes communicating with each other. Interaction between the two layers of a vehicular process occurs using an interface provided by the low-level layer. By making use of the interface, the high-level layer can send signals to the low-level layer. The communication channels are provided in the interface layer such as get_id for getting the identification information of the vehicle, get_ldr for getting the leader of the vehicle. set_ldr is for setting the leader of the vehicle, drive is for instructing the vehicle to drive forward, keep_dist is for maintaining a safe distance d from the current leader, check_join is for checking whether the vehicle with the given identification occupies the position where this vehicle wants to join, align_start, align_done and merge_start, merge_done are for aligning and merging. Leader of a vehicle is not necessarily the leader the platoon, but rather it is the vehicle that is directly ahead of it in the platoon.

Figure 3.1: Communication among concurrent processes



3.1.1 Platooning protocol description

There are three components involved in platooning, i.e., Leader, follower, and Joiner as shown in Fig. 3.2. Each component is supervised by its corresponding supervisor as illustrated above.

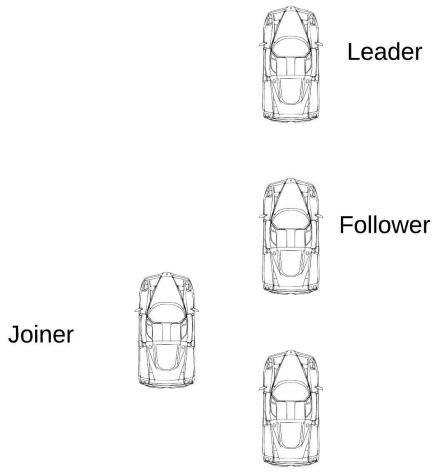
- ◇ When a vehicle wishes to join a platoon, it creates a communication channel 'x' and broadcasts it to every vehicle within range. This establishes a common communication channel, virtually connecting the joiner with every follower. However, this is not a unique channel. Therefore, unique channels are created in the *cooperate* routine of the follower process, resulting in a reconfiguration of the network.

- ◇ The follower transmits its identifier to the joining vehicle using the ident routine of the follower process so that joining vehicle can decide whether to join at the follower's position.

- ◇ The decision of where to join in a platoon is handled by communication with the interface layer's check_join channel in the check routine of the joiner process.

- ◇ If joiner decides to join at the follower's position, then the respond expression will execute the send_ldr channel and transmits an identifier of the vehicle preceding the follower (its current leader) to the joiner. This enables the joining vehicle to position itself next to the

Figure 3.2: Components of platooning



follower so that it is ready to merge.

◇ When the joining vehicle is in position, the follower sets the joiner as its new leader in the rcv_ldr channel of the follower. This causes the follower to increase its distance from the joining vehicle to a distance *d*, as a result of the concurrently executing the *follow* routine as illustrated in Fig. 3.3.

◇ Lastly, the joining vehicle is signaled to merge.

◇ Once the joiner has merged into the platoon, it transitions to a follower as the last action in the merge expression of the joiner process as illustrated in Fig. 3.4.

3.2 π-calculus and concurrency

The π-calculus is a model of concurrent computation based upon the notion of *naming*. It is a way of describing and analysing systems consisting of agents which interact among each other, and whose configuration or neighborhood is continually changing. Efforts in the direction of requiring a unified theory of concurrency led to the development of π-calculus by Robert Milner [22].

Figure 3.3: Follower creates room for joiner

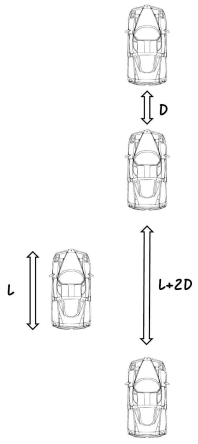
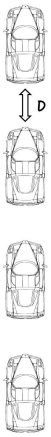


Figure 3.4: Joiner merges into the platoon



3.2.1 Preliminaries

The most primitive entity in π-calculus is a *name*. It can be a variable, unobservable action, communication channel or simply a data value. It can interchangeably take any of these forms.

In its basic form, π-calculus defines P $\in$ Q as a sequence of atomic actions π. When we declare P ::= π.P | 0 we define a process P that is empty (0) and terminating.

We use communication channels all throughout this work. Given an infinite set of names N, we have communication channels w, x, $y \in N$. Outgoing communication happens when a name is sent out on a channel. The expression P = $\bar{x}$ $\langle y \rangle$ declares a process P that sending a name y over a channel x. Incoming communication happens when a name is received over a channel. The expression P = *x(z)* declares a process P that is receiving a name z over a channel x. The ability to communicate by transmitting and receiving communication channels is known as *mobility*. It is a salient feature of π-calculus.

Further π-calculus is extended to form polyadic π-calculus [22]. We have more complex constructs in polyadic π-calculus. Following are these constructs from polyadic π-calculus.

- P = P || P' declares a process P that is concurrently executing P and P'.
- P = P + P' declares a process P that is non-deterministically executing either P or P'.

- ◇ *(vx)P* declares that the name x is unique to P.
- ◇ P = !P' declares that there many concurrently executing copies of P'.
- ◇ $P = x : [y \Longrightarrow P, z \Longrightarrow P']$ declares that process P executes P when x is equal to y, or executes process P' when x is equal to z.

When two processes are executing concurrently using names x and $\bar{x}$ respectively, then the names x and $\bar{x}$ are considered complementary, and the actions are linked. An example would be when a name is sent over a communication channel linked to another action, both the processes share the communication channel.

In [1], platooning is expressed in π-calculus. Table 3.1 compares some of the available choices for us to implement the protocol described in the modeling framework.

Table 3.1: π-calculus implementation comparison

Language	Concurrency	Message Passing	Ease	Model Checker
Erlang	Concurrent	Yes	Moderate	N/A
GoLang	Concurrent	Yes	Moderate	N/A
SML based lang (for MWB)	Concurrent	N/A	Easy	Yes(Mobility Workbench)
SML/NJ	No	N/A	Moderate	A few (But Isabelle removed support)
Promela	Concurrent	Yes	Easy-Mod	SPIN, Isabelle
Concurrent ML	Concurrent	Yes	Don't know	N/A
HOL	No	N/A	Don't know	Isabelle

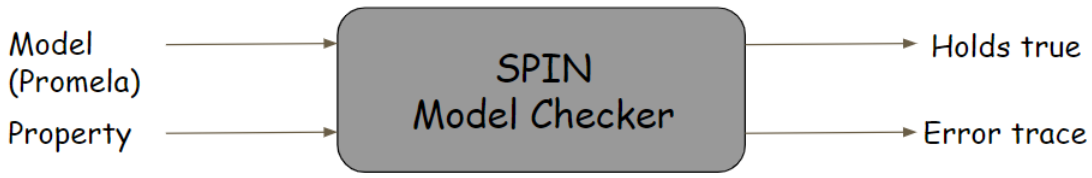
Based on the comparison of implementation choices against the factors taken into consideration in Table 3.1, we decided that it would better suit to express the platooning protocol first using

SML based language for use with Mobility Workbench (MWB) and then implement it in Promela (Process Meta Language) for use to simulate and model check using SPIN. Appendix A describes the protocol implementation for the Mobility Workbench. Appendix B contains the code for the protocol implementation in Promela.

By following the modeling framework discussed, we were able to undertake the verification and validation of the protocols expressed using this framework. Such efforts are of paramount importance to assure safety of the system and the protocol being implemented. There are however challenges due to its concurrent nature since multiple processes are running parallelly. This is a general characteristic feature of most of the protocols involving V2V communication. Therefore by choosing Promela to implement the protocol, we were able to verify safety properties of the protocol by using SPIN model checker [23].

We illustrate the Promela-SPIN verification setup in Fig 3.5. The reason for choosing SPIN is that it is a formal verification tool that is quite suitable for multi-threaded and concurrent applications [23]. Hence, for a concurrent and distributed application such as platooning, SPIN model checker is a suitable choice for undertaking the verification efforts.

Figure 3.5: Model checking setup using SPIN



3.2.2 Verification Plan

(1) At all times there is no more than one and no less than one platoon leader in the entire platooning execution.

(2) After the merge is fully complete, the joiner becomes a follower.

(3) At all times at least a safe distance of 'd' is maintained between each of the vehicles involved in platooning.

(4) Interface layer is utilized by the behaviors to control platooning. We assume the behavior of the subprocesses in the interface layer (using assume keyword) and model check the remaining system. This will restrict the state space.

We are able to verify property *1* by iterating over all process ids (*_pid* in promela) that are executing at any point and counting the number of leader processes. We then assert that there is only one platoon leader at any time. We are able to verify property *2* by asserting that eventually, a joiner property transforms into a follower process and continues as a follower. The property *3* involves physical properties of the vehicle and verifying it would require the model to incorporate physical distances into it. Currently, the interface layer routine *keep_dist* is ensuring this property.

Chapter 4

Learning and Autonomous Vehicle Control

In the previous chapter, we have seen that process algebra based on π-calculus can be used as a modeling framework capturing the essence of communications in the control protocol. It has been shown that there are two benefits to using this approach. Firstly, we can capture the concurrency in the process executions. Secondly, we used message passing to let the vehicular processes communicate, by creating some channels and using a broadcast/receive technique. On the actual hardware, this is realized using Dedicated Short Range Communication to establish a communication link, connecting the vehicles. We expect to run our protocol using this setup. This thesis contributes to this work by modeling the framework in a language that is used to verify that the interactions are safe. In this chapter, we will look at how we use learning for high-level control of the autonomous vehicle. The vehicle makes a *sequence of control decisions*. Training of learning models to make this sort of sequential decisions is very well known as *Reinforcement Learning*. The challenging problem here is that we do not know which action is right or which action is wrong. The autonomous vehicle's control protocol learns from experience. We will model this using Markov Decision Processes (MDPs) as a Reinforcement Learning Problem. In this setting, actions are sampled by trial and error and rewards influence the decision-making process of the agent that is controlling the vehicle. The contribution of this chapter is to develop a control protocol based on RL techniques using the reward function of an environment suitable for simulating the agent's control algorithm. When the agent gets a positive or negative reward from an environment, it then figures out what went right or wrong that which prompted it to get this reward. By learning this,

the protocol tries to take more such right actions and less of the wrong actions.

4.0.1 Challenges in learning optimal action

Learning control actions is not straight forward to the agent, as the agent doesn't know the model of the environment. The agent is tasked with learning a control policy that tells which action to choose in a given state of the MDP. This setting drives us to model-free reinforcement learning. The state-space we are dealing with is continuous and we are looking at a larger problem. The action space too is continuous.

Reinforcement Learning problems model the world using Markov Decision Processes (MDP). We assume that an agent gets some observations regarding the state. At each time step t, an agent observes a state s_t and performs an action a_t that leads to a new state s_{t+1} upon interaction with the environment and the agent observes a corresponding reward $r_t = R(s_t, a_t)$. In our analysis, we don't consider the evolution of low-level control of the vehicle. It is out of the scope of this research. However, an algorithm to control the dynamics at this level is required when developing an end-to-end solution to autonomous vehicle control. We will consider working with the low-level control dynamics in the future.

4.0.2 Action Set

We have a continuous action space. The action set Γ contains the following variables:

1) "Steering Position": *steer*: [-1 1] through 0

2) "Gas Pedal": *acc*: [1 0]

3) "Brake Pedal": *brake*: [1 0]

By controlling these three variables, the protocol controls the manuever of the vehicle. It is possible to take the following actions, similar to the action space in [14].

1) "Do Nothing": (*steer, acc, brake*) = (0, 0, 0);

2) "Accelerate": (*steer, acc, brake*) = (0, [1 0), 0);

3) "Hard Brake": (*steer, acc, brake*) = (0, 0, [1 0));

4) "Turn Left at intersection": (*steer, acc, brake*) = ([-1, 0), [1, 0), 0);

5) "Turn Right at intersection": (*steer, acc, brake*) = ((0, 1], [1, 0), 0);

4.0.3 Reward structure

We use the OpenAI gym's racing car [21] simulation environment in our work. We build an agent that trains a model based on the rewards generated by the OpenAI environment. The state space consists of 96 x 96 pixels. The reward is -0.1 for every frame and $+1000/N$ for every track tile visited, where N is the total number of tiles in the track [21]. By setting a reward structure as described the reward earned is higher when more track tiles are visited within fewer frames. Indirectly this structure favors agents that are able to operate the racing car at higher speeds to accumulate more reward points quickly in this environment. The reward structure also discourages the racing car from venturing into the off-road portions as the agent would then not earn any positive rewards but rather accumulate negative rewards.

4.1 Reinforcement Learning for Control of Autonomous Vehicles

A Reinforcement Learning problem is typically modeled as a Markov Decision Process (MDP). A Reinforcement Learning (RL) agent is assumed to learn the optimal or near-optimal policies from its experiences. To solve difficult decision and control problems in uncertain dynamic systems, data-driven learning control methods, especially Reinforcement Learning have garnered a great deal of attention recently.

We model the problem using MDPs. At each time step t, an agent observes a state s_t and performs an action a_t that leads to a new state s_{t+1} and observes a corresponding reward $r_t = R(s_t, a_t)$. In this research we make use of model-free RL techniques and thus do not consider a transition model $T(s_t, a_t, s_{t+1})$ between the current state s_t and the next states s_{t+1}. By using Reinforcement Learning, autonomous vehicles can learn driving behavior in simulation, as it is

possible to observe failure cases in a safe environment.

4.1.1 Markov Decision Process

An MDP can be expressed by a tuple, $\langle S, A, T, R, \alpha \rangle$. S is the state space, A is the action space, T is the state transition matrix, the reward function is R: S $\times$ A $\rightarrow$ R, and α is the decision making objective. $R(s_t, a_t, s_{t+1})$ represents the reward obtained when taking action a_t in state s_t and ending up in state s_{t+1}. An agent's state space is 96 x 96 pixel image I_t of the environment at each time step t,

An agent's action a_t in state s_t causes it to make a state transition to s_{t+1}. For example, if a vehicle in state s_t makes a decision to accelerate, the vehicle's speed is going to change to v_{t+1}, thus causing the agent to transition to state s_{t+1}. We don't have a complete knowledge of the MDP model in the problem, i.e. we don't have the state transition matrix. We therefore have chosen a model-free learning approach. This approach doesn't require us to have knowledge of the complete MDP model.

4.1.1.1 Learning Rate

The learning rate, $\alpha \in [0, 1)$ determines to what degree newly acquired knowledge overrides old knowledge. A learning rate of 0 means Q-values are never updated, whereas a learning rate close to 1 means the agent learns quickly. We have chosen a learning rate of 3e$-$4 for this work.

4.1.1.2 Discount Factor

The discount factor, $\gamma \in [0, 1)$ establishes the factor of influence future rewards should have on decision-making. A discount factor of 0 will make an agent short-sighted, i.e., it only considers current rewards. A discount factor close to 1 will encourage the agent to strive for long-term high reward. We have chosen a γ of 0.97 for this work.

4.1.1.3 Epsilon

The ϵ parameter establishes the trade-off between exploration and exploitation. When we set ϵ to 0, we employ a pure greedy approach where we always select the action with the highest estimated reward. When we set ϵ to 1, we employ a random sampling approach, where an action is randomly sampled. By choosing an appropriate value for ϵ, every once in a while we choose a random action with a small probability of ϵ.

4.1.1.4 Optimal control policy

There are several commonly used ways of finding an optimal policy, and central to all of them is the concept of a *value function* of a policy, $v_\pi : S \to R$, where $v_\pi(s)$ is the expected cumulative value of the reward the agent would receive if it started in state s and behaved according to policy π. A policy π gives an action for each state for each time.

For every MDP, there exists an optimal policy, π^* which maximizes the expected sum of rewards. The idea behind *Optimal Control* is to find the optimal policy π^* for a given MDP.

In Policy Iteration approach, we repeat policy evaluation and policy improvement steps until policy converges. In policy evaluation step we find values with simple bellman updates for some fixed policy until values converge. In policy improvement step we update the policy using one-step look-ahead with resulting converged utilities as future values according to equation 4.1. Here $Q^{\pi(t)}(s_t, a_t)$ is the state-action value function given a state s_t, action a_t, and policy π.

$$\pi_{t+1}(s_t) = argmax_{a_t} Q^{\pi(t)}(s_t, a_t) \tag{4.1}$$

4.1.1.5 Function Approximation

The temporal difference RL algorithm can be combined with function approximation. In practice, this technique is useful in applying the algorithm to larger problems, even when the state space is continuous [18]. We choose to use an artificial neural network as a function approximator.

Neural networks have the property of universality, i.e. they are are an excellent choice for function approximation.

Another choice is to use a Linear Least Squares function approximator in the control protocol. Their performance is far behind what neural networks can accomplish in terms of universality, but they are easy to implement and use, and are easy to be formally verified. Performance issues are possible with linear least squares approximators [19]. This makes them a poor choice for a continuous state space. Therefore we choose to use Convolutional Neural Networks (CNNs) for the purpose of function approximation in our control protocol.

4.1.2 Q-Learning

Unlike policy gradient methods which attempt to learn functions which directly map an observation to an action, Q-Learning attempts to learn the value of being in a given state, and taking a specific action there. After evaluating a simple lookup-table version of the algorithm, we implement a neural network equivalent using TensorFlow. We train the neural network using Temporal Difference (TD) learning. TD Learning is a model-free learning approach which estimates the value function and uses it while training the network. To improve the training, ϵ starts at 1 and anneals to 0.1 over the first million frames. This is known as ϵ-greedy approach. We use weight regularization to reduce overfitting. The agent also holds a memory buffer with past experiences. Every once in a while, we sample a batch of previous experiences and feed them to the neural network. This technique is known as *Experience Replay*. This helps us achieve better convergence rates as the training data is independent and identically distributed i.e, there is less correlation in the data. We use Adam Optimizer for updating the neural network weights iteratively while training the network.

4.1.3 Neural-Network architecture

We build a neural network using convolution layers, pooling layers, and dense layers. Neural-networks are used for function approximation. The inputs to the neural network are preprocessed

images from the observed state space. In the neural-network, the convolutional layers apply convolution filters to the image. Pooling layers reduce the dimensionality of the feature map. In this work, we use Max Pooling algorithm for this purpose. Also, Rectified Linear Units (ReLU) are used as activation function to introduce non-linearity into this network.

Table 4.1: Neural network summary

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(64, 30, 30, 8)	1184
maxPooling2d	(64, 15, 15, 8)	0
conv2d (Conv2D)	(64, 13, 13, 16)	1168
maxPooling2d	(64, 6, 6, 16)	0
flatten1 (Flatten)	(64, 576)	0
dense (Dense)	(64, 256)	147712
dense (Dense)	(64, 12)	3084

4.1.4 Pseudocode for the Learning-based Control Protocol

The following is the pseudocode for the learning based control based on the Q-learning approach described above. We use TensorFlow to implement the RL agent to navigate in the simulation environment. We define the neural network as described in the architecture above, train the network, iteratively by updating the weights of neurons and simulate the agent in the Open AI Box 2d racing car environment. The algorithm is based on deep reinforcement learning where a convolutional neural network is used as a function approximator and reinforcement learning is used to learn the optimal control policy.

Algorithm 1 Deep RL based control

```
1: procedure MakeOpenAIEnv
2:
   Create a race car agent
3: procedure ComputeLoss:
4: InitStateActionTgtTensor()
5: InitStateActionEstTensor()
6:
7:     procedure CalculateTargetQ:
8:         targetQ = r + γ max(Q)
9:
10:    procedure CalculateEstimatedQ:
11:        TotalLoss = regularizationLoss + trainingLoss
12:        InitializeAdamOptimizer()
13:        estimateQ = UseOptimizerToMinimize(TotalLoss)
14:
       UpdateNeuralNetWtsIteratively()
15:
   PreProcessImage()
16: ComputeLoss()
17: TrainNeuralNetwork()
18: PlayOnNN()
19:
```

4.2 Related Work Using Game Theory

N.Li et al in [14] [15] formulated the problem of taking high-level control decisions using a game-theory based model. The action space is discrete in their work.

4.2.1 Action Selection

The approach to selecting actions is based on receding horizon optimization. At each time step t, the vehicle selects a sequence of actions, $\gamma := \{\gamma_0, \gamma_1, ..., \gamma_{n-1}\}$, $\gamma_i \in \Gamma$, i = 0,..., n-1, to maximize the cumulative reward, R, over a horizon of length n, assuming that actions are applied sequentially at each step within the horizon. After the action sequence that achieves the highest cumulative reward, $\hat{\gamma}$, has been found, the vehicle applies the first action $\hat{\gamma_0}$ of $\hat{\gamma}$ at the current step. After the motion proceeds with $\hat{\gamma_0}$ applied over one time step, the process is repeated at the next time step *t+1*. The cumulative reward is given by $R(\gamma) = \Sigma_{i=0}^{n-1} \lambda^i R_i(\gamma)$

4.2.2 Game-theoretic decision making

They use the formulation of level k game-theoretic decision making in their work. A level-0 vehicle treats other vehicles as stationary obstacles and selects the action sequence $\hat{\gamma_0}$. In this case, a level *0* vehicle may represent an aggressive driver in real traffic, who usually assumes that other drivers will yield the right of way. After level *0* is defined, the action selection procedure for level k for k ≥ 1, in the case of 2-agent interactions has the following form, $\hat{\gamma_k}^{ego} \in \arg\, max_{\gamma_i \in \Gamma}\ R_k(\gamma)$.

They make a simplification here. When a level k ego agent computes $\hat{\gamma_k}^{ego}$, it assumes that the other agent applies $\hat{\gamma_{k-1}}^{other}$ which is computed by assuming that the ego agent was at level *k-2*. Thus, $\hat{\gamma_{k-1}}^{other}$ is independent of $\hat{\gamma_k}^{ego}$ and can be determined first.

4.2.3 Comparison

Although in our work, we have not simulated this control protocol in a multi-agent environment as in [14] [15], we expect to conceive a way of coordinating independent learners by using an Interac-DEC-MDP formalism [16]. Interac-DEC-MDP stands for Interaction between decentralized MDP. This is an extension of DEC-MDP model where each agent has complete observability and where the global reward is only partially perceived by each agent. By adding an interaction module, we enable each agent to contribute partially to the global reward function. In this setting, our interaction module is expected to make use of the vehicle-to-vehicle (V2V) Interaction technology described in chapter 1.

As we see it, the action module focuses on individual interests, and the interaction module focuses more on collective interests. This RL-based learning and collective behavior form the core idea to coordinate independent learners in a multi-agent environment. We haven't implemented this in this work, and it is still an open problem for future work.

Chapter 5

Conclusion and Future Work

In this chapter, we conclude the thesis and present future work. We present the results from this thesis and compare them with the research work in [1] and [14].

5.1 Results

5.1.1 Model Checking Platooning execution

In this work, we verify the high-level communication layer of platooning execution using Promela implementation for SPIN. We compared some of the available implementation choices according to the criterion laid out in chapter 3. We found Promela to be suitable for analyzing the protocol and verifying its properties. J. Campbell et al. in [1] simulated both the high-level communications layer and low-level control dynamics of their protocol on Webots 8.3 robotics simulation environment. Although we limited ourselves to only the high-level communications layer, we implemented it in Promela so that we could verify its properties using the SPIN Model Checker. The Promela source code, logs of model checker run are presented in Appendix A and the mobility workbench code is presented in Appendix B.

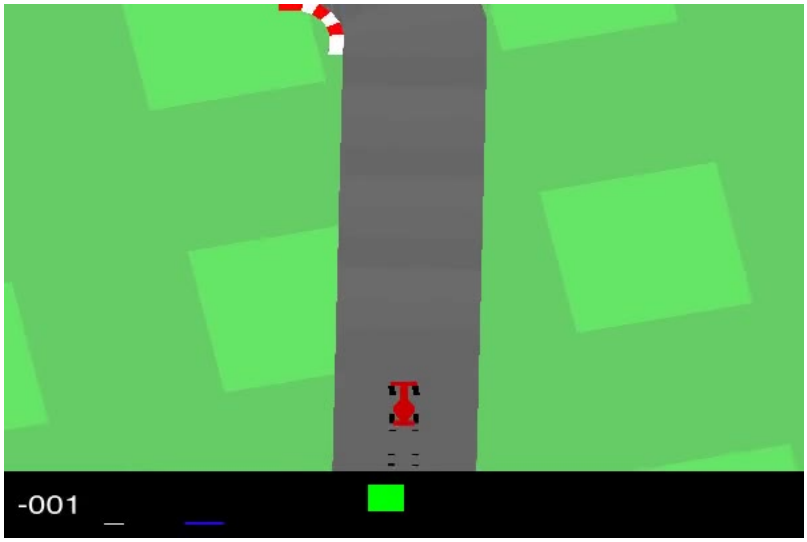
5.1.2 Simulation runs for OpenAI Racing Car

We considered the Racing car simulation environment provided by OpenAI gym [21]. We used their Box2D simulator. The observation states are 96 x 96 pixels with three color channels for each pixel and 256 magnitude (8 bit) values for each color, representing the view from the top.

For actions, there are three continuous values that the agent can choose from, i.e., for the steering of wheel, it can choose a real value ∈ [-1 (left), 1 (right)], for acceleration, it can choose a real value ∈ [0 (No acc), 1 (Max acc)], and for brake, it can choose a real value ∈ [0 (No brake), 1 (Max brake)].

After several hours of training the model on a PC with decent hardware specifications, using the trained model, the agent is able to successfully navigate in the simulated environment by following the track laid out in the environment as illustrated below in Fig. 5.1.

Figure 5.1: Racing car control simulation



5.2 Conclusion

While autonomous driving systems are beta-tested on some public roads, methods to validate and verify autonomous driving systems are still evolving, and are active areas of research. In the wake of a gradual transition to autonomy on the roads, the term safe autonomy is getting more attention especially with autonomous vehicle crashes happening once in a while. Road testing does not get us all the way to classifying autonomous vehicles as safe. The onus is on the research community to come up with better behavioral models of autonomous systems. State of the art validation and verification techniques need to be employed. Not to mention, the use of artificial

intelligence technologies like neural networks, makes it even harder to verify the system exhaustively, i.e., it gets challenging to employ formal verification techniques.

Our research provides insights into protocol design methods to consider in the outcome of better vehicular communication technologies like vehicle-to-vehicle and vehicle-to-infrastructure interactions. We show that we get closer to achieving some of the goals we set forth for our research. We can say that by using the platooning protocol framework as described, we improve the safety of interactions and drives of vehicles involved in platooning. We support this claim by showing running the platooning protocol implementation on SPIN model checker. The neural-network based control was not formally verified as it involves challenges like conceiving safety specifications on pixel images. We also compare the techniques with existing work in these areas. There is a scope further to take the ideas evaluated in this research and improve the modeling framework and protocol design techniques for this application to improve the safety of drives. We are still a long way from being comfortable with autonomous vehicles driving on public roads and co-existing with humans. We are hopeful that this gradual transition will happen while placing more emphasis on robust verification and validation techniques involving the deployment of these systems.

5.3 Future work

A challenge in the current traffic routing models is that they lack a real-world understanding of how drivers make routing decisions. Traffic prediction models are based on the assumption that drivers are rational and choose the most optimal route that minimizes travel time. In order for routing models to be more reliable, they need to model how human drivers make routing decisions. We are also interested in investigating incentive design mechanisms to alleviate congestion problems as we transition to make our cities smarter. We would want the incentives to be personalized and context-driven. Thriving cities are adding new traffic every year, while the city's traffic capacities are static. Most of the traffic appears to be concentrated during a specific time window. This includes commuters traveling from home to get to work and back again to home. We see an

increase in travel time and congestions on the roads. We are therefore interested in dynamic incentive generation mechanisms that aim to reduce such traffic congestion problems in smart cities.

Automated driving with humanlike driving behavior requires interactive and cooperative decision-making. Other motorists' intentions need to be inferred and integrated into the protocols that allow for rational, cooperative decision-making and using inter-vehicle communication minimally.

Bibliography

[1] Campbell, J., Tuncali, C. E., Liu, P., Pavlic, T., Ozguner, U., & Fainekos, G. (2016). Modeling concurrency and reconfiguration in vehicular systems : A π-calculus approach. 2016 IEEE International Conference on Automation Science and Engineering, CASE 2016. Vol. 2016-November IEEE Computer Society, 2016. pp. 523-530

[2] United States Department of Transportation, Vehicle-to-Infrastructure (V2I) Resources, 2018, Accessed from https://www.its.dot.gov/v2i/index.htm

[3] Harding, J., Powell, G., R., Yoon, R., Fikentscher, J., Doyle, C., Sade, D., Lukuc, M., Simons, J., & Wang, J. (2014, August. Vehicle-to-vehicle communications: Readiness of V2V technology for application. (Report No. DOT HS 812 014). Washington, DC: National Highway Traffic Safety Administration.

[4] Sadigh, D. (2017). Safe and Interactive Autonomy: Control, Learning, and Verification, EECS Department, University of California, Berkeley, Accessed from http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-143.html

[5] Driggs-Campbell, K. (2017). Tools for Trustworthy Autonomy: Robust Predictions, Intuitive Control, and Optimized Interaction, EECS Department, University of California, Berkeley, Accessed from https://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-41.pdf

[6] LeBlanc, L. J., Morlok, E. K., Pierskalla, W. P., An efficient approach to solving the road network equilibrium traffic assignment problem, Transportation Research, Vol. 9(5), 309-318, 1975, Accessed from https://www.sciencedirect.com/science/article/pii/0041164775900301

[7] Kim, S., Lewis M. E., White & C. C. III, Optimal Vehicle Routing with Real-Time Traffic Information, IEEE Transactions on Intelligent Transportation Systems, Vol. 6(2), 178-188, 2005

[8] Coogan, S., Gol, E. A., Arcak, M., & Belta, C. (2016). Traffic network control from temporal logic specifications. IEEE Transactions on Control of Network Systems, 3(2), 162-172.

[9] Thai, J. (2017). On learning Game-Theoretical models with Application to Urban Mobility. EECS Department, University of California, Berkeley, Accessed from https://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-212.pdf

[10] Vazirani, V., Nisan, N., Roughgarden, T., Tardos, E. (2007). Algorithmic Game Theory, Cambridge University Press, Cambridge, UK.

[11] Caragiannis, I., Kaklamanis, C., Kanellopoulos, P., Improving the Efficiency of Load Balancing Games through Taxes, Proceedings of the 4th International Workshop on Internet and Network Economics, pp 374-385, 2008.

[12] Valiant, G., Roughgarden, T., (2010). Braess's Paradox in Large Random Graphs, Random Structures and Algorithms, Accessed from https://theory.stanford.edu/~tim/papers/rbp.pdf

[13] Bulling, N. Künstl Intell (2014) 28: 147. A Survey of Multi-Agent Decision Making, 2014, Accessed from https://doi.org/10.1007/s13218-014-0314-3

[14] Li N., Kolmanovsky I., Girard A. & Yildiz Y., Game Theoretic Modeling of Vehicle Interactions at Unsignaled Intersections and Application to Autonomous Vehicle Control, American Control Conference, pp 3215-3220, 2018.

[15] Tian R., Li S., Li N., Kolmanovsky I., Girard A. & Yildiz Y., Game Theoretic Decision Making for Autonomous Vehicle Control at Roundabouts, IEEE Conference on Decision and Control (CDC), 2018.

[16] Thomas, V., Bourjot, C., & Chevrier V., Interac-DEC-MDP: Towards the use of interactions in DEC-MDP, Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2004.

[17] Schwarting W., Alonso-Mora J., & Rus D., Planning and decisionmaking for autonomous vehicles, Annual Review of Control, Robotics, and Autonomous Systems, no. 0, 2018.

[18] Abbeel, P., Lecture on Markov Decision Processes and Exact Solution Methods, Lecture Notes in Advanced Robotics, Fall 2012, University of California, Berkeley.

[19] Lagoudakis, M. G., and Parr, R., Least-squares policy iteration, The Journal of Machine Learning Research, Vol. 4 (December 2003), 1107-1149.

[20] Dragan, D. Anca, Robot Planning with Mathematical Models of Human State and Action, 2017, Accessed from https://arxiv.org/pdf/1705.04226.pdf.

[21] OpenAI, Open AI Gym Racing Car Environment, 2018, Accessed from https://gym.openai.com/envs/CarRacing-v0/.

[22] R. Milner, (1993) The Polyadic π-Calculus: a Tutorial

[23] G. Holzmann, Spin Model Checker, the: Primer and Reference Manual (First ed.), Addison-Wesley Professional, 2003.

Appendix A

Platooning Protocol Implemented in Promela

A.1 Promela Code

Listing A.1: Promela Code

```
/*
  Author: Lakhan Shiva Kamireddy
  University of Colorado Boulder
*/
mtype = {drive, merge_done, merging, align_done, aligning, keep_dist};
chan leader = [1] of { mtype };
chan follow = [1] of { mtype };
chan get_id = [1] of {int};
chan set_ldr = [2] of {int};
chan get_ldr = [1] of {int};
chan follower_id = [1] of {int};
chan y = [1] of { mtype }; //This is only used in joiner process
chan check_join = [2] of { mtype, int};
int cur_ldr = 1;
int joiner_val = 4;
int ldr_proc_cnt = 0;
int jnr_proc_cnt = 0;
int flr_proc_cnt = 0;
mtype curact = drive;

```

```
21 proctype Leader()
22 {
23   curact = drive;
24   ldr_proc_cnt++;
25   /*Let us write 2 to get_ldr (the leader number)*/
26   get_ldr!1;
27   cur_ldr = 1;
28   leader!curact;
29   printf("Executed Leader process\n");
30   /*Making sure there is only one leader process at any time
31     Leader is started with pid 1. If a second leader get's started,
32     following assertion gets violated.*/
33   assert(ldr_proc_cnt == 1);
34 }
35
36 proctype Cooperate(chan lk)
37 {
38   chan y2 = [10] of { int };
39   int j_id;
40   printf("Starting cooperate process\n");
41   /*msg?x -> receive a broadcasted message(by joiner) and binds it
42     to x*/
43   /*(vy)((x!y).Ident(y))*/
44   /*should it be (vy)((x?y).Ident(y)) in the paper ?*/
45   /*x!y*/
46   lk?j_id;
47   int f_id;
48   int temp;
49   /*Test-checking what is in y2*/
50   y2!j_id;
51   y2?j_id;
52   printf("Cooperate process j_id is %d\n", j_id);
53   if
```

```
54   :: (j_id != 0) ->
55   y2!j_id;
56   /*Ident(get_id, y)*/
57   int id;
58   /*Follower's id - let this be 1. we will write it to get_id channel*/
59   /*Added this since, something has to be in get_id*/
60   get_id!2;
61   /*Test-print follower id*/
62   get_id?f_id;
63   printf("Follower id is %d\n", f_id);
64   get_id!f_id;
65   int f;
66   /*flag is set to 1*/
67   /*get_id(id)*/
68   get_id?id;
69   /*y!id*/
70   y2!id;
71   /*y(flag)*/
72   y2?f;
73   /*output y*/
74   y2?temp;
75   /*Test - testing what is in f*/
76   printf("Cooperate process, printing flag: %d\n", f);
77   /*Respond(y, flag)*/
78   if
79   :: (f != 0) ->
80     printf("Passed cooperate process' if condition\n");
81     /*Send_Ldr(get_ldr, y)*/
82     /*Acc to the descr, Send_Ldr transmits the vehicle preceding the
83       Follower to Joiner, which is 1 in this case*/
84     int ldr;
85     /*get_ldr(ldr)*/
86     get_ldr?ldr;
```

```
87      /*get_ldr is giving us the preceding vehicle id*/
88      printf("Current leader in cooperate process is %d\n",ldr);
89      /*The follower communicates the preceding vehicle id to
90        the joiner*/
91      lk!ldr;
92      /*y!ldr*/
93      /*Making this y2!nldr since, our new preceding vehicle is
94        Joiner*/
95      y2!j_id;
96      /*Rcv_Ldr(y, set_ldr);*/
97      /*Acc to descr, when joiner is in position, in Rcv_ldr routine,
98        the Follower sets the Joiner as its new preceding vehicle*/
99      int nldr;
100     /*y(nldr)*/
101     /*nldr is the new ldr*/
102     y2?nldr;
103     printf("Current new leader in cooperate process is %d\n",nldr);
104     /*set_ldr!nldr*/
105     set_ldr!nldr;
106     /*Align(y);*/
107     mtype align_status = aligning;
108     /*Delay*/
109     /*Here, the distance is increased to d*/
110     align_status = align_done;
111     /*align_done*/
112     /*Wait(y)*/
113     mtype merge_status = merging;
114     /*Delay*/
115     /*Here the joiner vehicle is fully merging into the platoon*/
116     merge_status = merge_done;
117     /*merge_done*/
118   fi
119   fi
```

```
120   printf("End of Cooperate process\n");
121 }
122
123 proctype Follow()
124 {
125   follow!keep_dist;
126   flr_proc_cnt++;
127   /*The folowing assertion fails if the joiner fails to become
128     follower after executing merge*/
129   assert(flr_proc_cnt > jnr_proc_cnt);
130 }
131
132 proctype Listen(chan ly)
133 {
134   chan mm = [1] of { int };/*We will use channel mm at the end of
135                              successful join*/
136   chan z = [1] of { int };
137   int id = 4; /*Id of the Joiner vehicle*/
138   int f_id;
139   int v_id = 1; /*Joiner wants to join behind a vehicle of id 1*/
140   /*channel j is considered x in the process
141     x(y)
142     Since x and y channels are of incompatible types, getting
143     directly from channel j
144     y(id)*/
145   /*ly?id;
146     printf("Printing joiner id %d\n", id);
147     Wait until the ly channel has the id communicated by
148     Follower process*/
149   ly?f_id;
150   printf("Print from listen proc preced vehicle id %d\n", f_id);
151   /*Here the joiner process knows before hand that it wants to
152     join behind vehicle with id 1. This is checked in the
```

```
153     check_join routine.
154     Check(y,id)*/
155   bool ok;
156   /*(vz)check_join<z,id>*/
157   if
158   :: (f_id == v_id) ->
159     /*check_join sub process matched id for compatibility
160       and puts a 1
161       on the z which is ok
162       id is compatible and let z have 1*/
163     z!1;
164     printf("The id received from follower is compatible\n");
165   fi
166   /*z(ok)*/
167   z?ok;
168   /*Ans(y,ok)*/
169   /*y!ok*/
170   y!ok;
171   /*if(ok == True) then Rcv_Ldr(y)*/
172   if
173   :: (ok == 1) ->
174     /*Rcv_Ldr(y)*/
175     printf("Passed listen proc if condition\n");
176     int ldr;
177     /*y(ldr)*/
178     /*y chan is of incompatible type */
179     ldr = cur_ldr;
180     printf("Current leader in Listen process is %d\n",ldr);
181     /*set_ldr!ldr*/
182     set_ldr!ldr;
183     /*Align(y);*/
184     mtype align_status = aligning;
185     /*Delay*/
```

```
186     align_status = align_done;
187     /*Wait(y)*/
188     int fid;
189     /*Added this since, something has to be in get_id*/
190     get_id!ldr;
191     /*get_id(id)*/
192     get_id?fid;
193     /*y!id*/
194     /*y is of incompatible type*/
195     /*We write to follower_id channel*/
196     follower_id!fid;
197     /*Test-remove later*/
198     //follower_id?fid;
199     //printf("Test print fid %d\n",fid);
200     /*y.Merge(y)*/
201     /*Merge(y)*/
202     mtype merge_status = merging;
203     /*Delay*/
204     merge_status = merge_done;
205     /*merge_start'.merge_done.y'.Follower*/
206     printf("End of Listen process\n");
207     /*After merge fully complete, the joiner will
208       take the role of the follower*/
209     joiner_val = 0;
210     run Joiner(mm);
211     printf("Joiner becomes follower\n");
212     run Follow();
213     jnr_proc_cnt++;
214   fi
215 }
216
217 proctype Joiner(chan laa)
218 {
```

```
219   laa!joiner_val;
220   //laa!4;
221   /*(vx)(b<x>||!Listen(x)) - broadcasts message x to any
222     vehicle in the range with it's intention to join*/
223   /*In the program we are broadcasting to channel j*/
224
225 }
226
227 init
228 {
229   chan j = [1] of { int };
230   run Leader();
231   run Joiner(j);
232   run Cooperate(j);
233   run Listen(j);
234   run Follow();
235 }
```

A.2 SPIN Simulation log

Listing A.2: Promela Code

```
1 /*
2   Results when simulated on Ubuntu 16.04
3   SPIN Model checker Simulation Log
4 */
5   0:  proc  - (:root:) creates proc  0 (:init:)
6 spin: platooning.pml:0, warning, proctype Cooperate,
7 `mtype align\_status` variable is never used (other than in print stmnts)
8 spin: platooning.pml:0, warning, proctype Cooperate,
9 `mtype merge\_status` variable is never used (other than in print stmnts)
10 spin: platooning.pml:0, warning, proctype Listen,
```

```
11 'int   id' variable is never used (other than in print stmnts)
12 spin: platooning.pml:0, warning, proctype Listen,
13 'mtype align\_status' variable is never used (other than in print stmnts)
14 spin: platooning.pml:0, warning, proctype Listen,
15 'mtype merge\_status' variable is never used (other than in print stmnts)
16 Starting Leader with pid 1
17   1:  proc  0 (:init::1) creates proc  1 (Leader)
18   1:  proc  0 (:init::1) platooning.pml:260 (state 1) [(run Leader())]
19   2:  proc  1 (Leader:1) platooning.pml:23 (state 1)  [curact = drive]
20 Starting Joiner with pid 2
21   3:  proc  0 (:init::1) creates proc  2 (Joiner)
22   3:  proc  0 (:init::1) platooning.pml:261 (state 2) [(run Joiner(j))]
23   4:  proc  2 (Joiner:1) platooning.pml:244 (state 1) [laa!joiner\_val]
24   5:  proc  1 (Leader:1) platooning.pml:26 (state 2)  [get\_ldr!1]
25 Starting Cooperate with pid 3
26   6:  proc  0 (:init::1) creates proc  3 (Cooperate)
27   6:  proc  0 (:init::1) platooning.pml:262 (state 3)
28   [(run Cooperate(j))]
29   7:  proc  1 (Leader:1) platooning.pml:27 (state 3)  [cur\_ldr = 1]
30 Starting Listen with pid 4
31   8:  proc  0 (:init::1) creates proc  4 (Listen)
32   8:  proc  0 (:init::1) platooning.pml:263 (state 4) [(run Listen(j))]
33 Starting cooperate process
34   9:  proc  3 (Cooperate:1) platooning.pml:37 (state 1)
35   [printf('Starting cooperate process\\n')]
36  10:  proc  3 (Cooperate:1) platooning.pml:43 (state 2) [lk?j\_id]
37  11:  proc  1 (Leader:1) platooning.pml:29 (state 4)  [leader!curact]
38  12:  proc  3 (Cooperate:1) platooning.pml:46 (state 3) [f\_id = 0]
39 Starting Follow with pid 5
40  13:  proc  0 (:init::1) creates proc  5 (Follow)
41  13:  proc  0 (:init::1) platooning.pml:264 (state 5) [(run Follow())]
42 Executed Leader process
43  14:  proc  1 (Leader:1) platooning.pml:30 (state 5)
```

```
44  [printf('Executed Leader process\\n')]
45  15:  proc  3 (Cooperate:1) platooning.pml:49 (state 4) [temp = 0]
46  16:  proc  5 (Follow:1) platooning.pml:143 (state 1)
47  [follow!keep\_dist]
48  16:  proc  5 (Follow:1) terminates
49  17:  proc  3 (Cooperate:1) platooning.pml:49 (state 5) [y2!j\_id]
50  18:  proc  3 (Cooperate:1) platooning.pml:50 (state 6) [y2?j\_id]
51 Cooperate process j\_id is 4
52  19:  proc  3 (Cooperate:1) platooning.pml:51 (state 7)
53  [printf('Cooperate process j\_id is %d\\n',j\_id)]
54  20:  proc  3 (Cooperate:1) platooning.pml:54 (state 8) [((j\_id!=0))]
55  21:  proc  3 (Cooperate:1) platooning.pml:56 (state 9) [y2!j\_id]
56  22:  proc  3 (Cooperate:1) platooning.pml:62 (state 10)  [id = 0]
57  23:  proc  3 (Cooperate:1) platooning.pml:62 (state 11)  [get\_id!2]
58  24:  proc  3 (Cooperate:1) platooning.pml:65 (state 12)  [get\_id?f\_id]
59 Follower id is 2
60  25:  proc  3 (Cooperate:1) platooning.pml:66 (state 13)
61  [printf('Follower id is %d\\n',f\_id)]
62  26:  proc  3 (Cooperate:1) platooning.pml:67 (state 14)  [get\_id!f\_id]
63  27:  proc  3 (Cooperate:1) platooning.pml:73 (state 15)  [f = 0]
64  28:  proc  3 (Cooperate:1) platooning.pml:73 (state 16)  [get\_id?id]
65 ,2 29:  proc  3 (Cooperate:1) platooning.pml:76 (state 17)  [y2!id]
66  30:  proc  3 (Cooperate:1) platooning.pml:79 (state 18)  [y2?f]
67  31:  proc  3 (Cooperate:1) platooning.pml:82 (state 19)  [y2?temp]
68 Cooperate process, printing flag: 4
69  32:  proc  3 (Cooperate:1) platooning.pml:85 (state 20)
70  [printf('Cooperate process, printing flag: %d\\n',f)]
71  33:  proc  3 (Cooperate:1) platooning.pml:89 (state 21)  [((f!=0))]
72 Passed cooperate process' if condition
73  34:  proc  3 (Cooperate:1) platooning.pml:90 (state 22)
74  [printf('Passed cooperate process' if condition\\n')]
75  35:  proc  3 (Cooperate:1) platooning.pml:97 (state 23)  [ldr = 0]
76  36:  proc  3 (Cooperate:1) platooning.pml:97 (state 24)  [get\_ldr?ldr]
```

```
77 Current leader in cooperate process is 1
78  37:  proc  3 (Cooperate:1) platooning.pml:99 (state 25)
79  [printf('Current leader in cooperate process is %d\\n',ldr)]
80  38:  proc  3 (Cooperate:1) platooning.pml:102 (state 26) [lk!ldr]
81  39:  proc  4 (Listen:1) platooning.pml:163 (state 1) [ly?f\_id]
82 Printing from listen process preceding vehicle id 1
83  40:  proc  4 (Listen:1) platooning.pml:164 (state 2)
84  [printf('Printing from listen process preceding vehicle id %d\\n',f_id)]
85  41:  proc  3 (Cooperate:1) platooning.pml:106 (state 27) [y2!j\_id]
86  42:  proc  4 (Listen:1) platooning.pml:172 (state 3) [ok = 0]
87  43:  proc  3 (Cooperate:1) platooning.pml:115 (state 28) [nldr = 0]
88  44:  proc  4 (Listen:1) platooning.pml:173 (state 4) [((f\_id==v\_id))]
89  45:  proc  4 (Listen:1) platooning.pml:176 (state 5) [z!1]
90  46:  proc  3 (Cooperate:1) platooning.pml:115 (state 29) [y2?nldr]
91 Current new leader in cooperate process is 4
92  47:  proc  3 (Cooperate:1) platooning.pml:116 (state 30)
93  [printf('Current new leader in cooperate process is %d\\n',nldr)]
94 The id received from follower is compatible
95  48:  proc  4 (Listen:1) platooning.pml:177 (state 6)
96  [printf('The id received from follower is compatible\\n')]
97  49:  proc  3 (Cooperate:1) platooning.pml:119 (state 31) [set\_ldr!nldr]
98  50:  proc  3 (Cooperate:1) platooning.pml:125 (state 32)
99  [align\_status = aligning]
100  51:  proc  3 (Cooperate:1) platooning.pml:125 (state 33)
101  [align\_status = align\_done]
102  52:  proc  3 (Cooperate:1) platooning.pml:133 (state 34)
103  [merge\_status = merging]
104  54:  proc  4 (Listen:1) platooning.pml:181 (state 9) [z?ok]
105  55:  proc  3 (Cooperate:1) platooning.pml:133 (state 35)
106  [merge\_status = merge\_done]
107  58:  proc  4 (Listen:1) platooning.pml:185 (state 10)  [y!ok]
108 End of Cooperate process
109  59:  proc  3 (Cooperate:1) platooning.pml:138 (state 40)
```

```
110  [printf('End of Cooperate process\\n')]
111  60:  proc  4 (Listen:1) platooning.pml:189 (state 11)  [((ok==1))]
112 Passed listen proc if condition
113  61:  proc  4 (Listen:1) platooning.pml:191 (state 12)
114  [printf('Passed listen proc if condition\\n')]
115  62:  proc  4 (Listen:1) platooning.pml:196 (state 13)  [ldr = 0]
116  63:  proc  4 (Listen:1) platooning.pml:196 (state 14)  [ldr = cur\_ldr]
117 Current leader in Listen process is 1
118  64:  proc  4 (Listen:1) platooning.pml:197 (state 15)
119  [printf('Current leader in Listen process is %d\\n',ldr)]
120 ,1 65:  proc  4 (Listen:1) platooning.pml:200 (state 16)  [set\_ldr!ldr]
121  66:  proc  4 (Listen:1) platooning.pml:205 (state 17)
122  [align\_status = aligning]
123  67:  proc  4 (Listen:1) platooning.pml:205 (state 18)
124  [align\_status = align\_done]
125  68:  proc  4 (Listen:1) platooning.pml:211 (state 19)  [fid = 0]
126  69:  proc  4 (Listen:1) platooning.pml:211 (state 20)  [get\_id!ldr]
127  70:  proc  4 (Listen:1) platooning.pml:214 (state 21)  [get\_id?fid]
128  71:  proc  4 (Listen:1) platooning.pml:219 (state 22)
129  [follower\_id!fid]
130  72:  proc  4 (Listen:1) platooning.pml:229 (state 23)
131  [merge\_status = merging]
132  73:  proc  4 (Listen:1) platooning.pml:229 (state 24)
133  [merge\_status = merge\_done]
134 End of Listen process
135  74:  proc  4 (Listen:1) platooning.pml:232 (state 25)
136  [printf('End of Listen process\\n')]
137  75:  proc  4 (Listen:1) platooning.pml:235 (state 26)  [joiner\_val = 0]
138 Starting Joiner with pid 5
139  76:  proc  4 (Listen:1) creates proc  5 (Joiner)
140  76:  proc  4 (Listen:1) platooning.pml:236 (state 27)  [(run Joiner(mm))]
141 Joiner becomes follower
142  77:  proc  4 (Listen:1) platooning.pml:237 (state 28)
```

```
143  [printf('Joiner becomes follower\\n')]
144  79:  proc  5 (Joiner:1) platooning.pml:244 (state 1) [laa!joiner\_val]
145  79:  proc  5 (Joiner:1) terminates
146  79:  proc  4 (Listen:1) terminates
147  79:  proc  3 (Cooperate:1) terminates
148  79:  proc  2 (Joiner:1) terminates
149  79:  proc  1 (Leader:1) terminates
150  79:  proc  0 (:init::1) terminates
151 7 processes created
```

Appendix B

Platooning Expressed in π-calculus for The Mobility Workbench

B.1 Mobility Workbench code

Listing B.1: Mobility Framework Models

```
/*Leader process in a platoon formation*/
/*Leader just drives forward indefinitely. drive is an
  atomic action*/
/*Author: Lakhan Shiva Kamireddy
 Date: 11/20/18
 University of Colorado Boulder
*/
agent Leader(drive) = 'drive.Leader

/*Follower process in a platoon formation*/
/*The follower must simultaneously follow its leader while
  cooperating with vehicles wishing to join the platoon*/
/*Author: Lakhan Shiva Kamireddy
 Date: 11/20/18
 University of Colorado Boulder
*/
/*Outputing y, then action merge_done*/
agent Wait(y, merge_done) = y.'merge_done.0
agent Align(y, align_start, align_done) = 'align_start.'y<align_done>.Wait
agent Rcv_ldr(y, ldr, set_ldr, nldr) = y(nldr).'set_ldr<nldr>.Align(y)
```

```
21 agent Send_ldr(y, get_ldr, ldr) = get_ldr(ldr).'y<ldr>.Rcv_ldr(y, ldr)
22 agent Respond(y, flag) = [flag]Send_ldr(y)
23 agent Ident(y, get_id) = get_id(id).'y<id>.y(flag).Respond(y, flag)
24 agent Cooperate(r,x) = Cooperate|r(x).(^y)('x<y>.Ident(y))
25 agent Follow(keep_dist) = 'keep_dist.Follow
26 agent Follower = Follow | Cooperate
27
28 /*Joiner process in a platoon formation*/
29 /*Until a decision to join is made by the high-level layer,
30   the joining vehicle drives forward just like leader.
31   Joiner behavior transitions to a follower as the
32   last action in the merge state.*/
33 /*Author: Lakhan Shiva Kamireddy
34  Date: 11/20/18
35  University of Colorado Boulder
36 */
37 agent Merge(y,merge_start,merge_done) = 'merge_start.'y<merge_done>.Follower
38 agent Wait(y,get_id) = get_id(id).'y<id>.y.Merge
39 agent Align(y,align_start,align_done) = 'align_start.align_done.Wait(y)
40 agent Rcv_ldr(y) = y(ldr).'set_ldr<ldr>.Align(y)
41 agent Ans(y, ok) = 'y<ok>.[ok]Rcv_Ldr(y)
42 agent Check(y, id, join_ok) = (^z)('join_ok<z,id>.z(ok).Ans(y, ok))
43 agent Listen(x) = x(y).y(id).Check(y, id)
44 agent Joiner(r,x) = (^x) Listen(x)|Joiner|'r<x>.0
```