

Safe Reinforcement Learning in Automotive



Ayush Shah
St Cross College, University of Oxford

A thesis submitted for the degree of

MSc in Computer Science

2020

Dedicated to Lord Krishna

Abstract

This work applies Logically Constrained Reinforcement Learning (LCRL) framework [20] for synthesizing policies for active monitoring and management of sensor modules in autonomous vehicles. LCRL allows synthesizing policies for unknown and continuous-state Markov Decision Processes (MDPs) such that a given linear time property is satisfied. We frame the problem of dynamic sensor module selection in automotive as an MDP and define the constraints as a Linear Temporal Logic (LTL) property. The LTL properties are used to guard and guide the MDP in finding an optimal policy. Defining a reward function over the state-action pairs of MDP based on the LTL property results in learning in a constrained environment. This approach leads to an improvement in performance and scalability for finding the right monitoring policies, as the safety properties enforces bounds on the search space.

Keywords— Reinforcement Learning (RL), Cautious RL [21], Deep RL, Safe RL, Autonomous driving, Adaptive active monitoring, Resource management with Deep RL [31], Execution Monitoring [26,32].

Acknowledgements

I am deeply indebted to **Prof. Alessandro Abate** for providing me with this wonderful opportunity to apply the theoretical knowledge gained during the Master's course to solve a challenging real-world problem. His guidance and vision enabled me to muster the courage to keep working hard and stay optimistic despite several hardships. His technical expertise and words of encouragement acted as a catalyst for the successful completion of this dissertation. I felt fortunate to work under the supervision of **Mohammadhosein Hasanbeig**. The field of LCRL, was established by him, Prof. Abate and Prof. Daniel Kroening during his Ph.D. here at Oxford. Given that he is a pioneer in this field, he provided expert advice to adapt the LCRL framework to our problem of active monitoring.

Dr. Shashank Pathak (ZF Group), spent countless hours in mentoring me and introducing me to the practical aspects of autonomous driving. When I was facing extreme technical difficulties, he generously offered his systems to build the entire project and run all the experiments. He also made sure that I understood all the theoretical background and provided detailed explanations on several underlying topics. Dr. Jagmal Singh and Michael Grunwald (ZF Group) have also been instrumental in our collaboration.

I am forever thankful to my mother, father, brother, grandparents and friends for their constant love and support which has brought me to this point in my life.

Contents

List of Figures

List of Tables

1	Introduction	1
1.1	Motivation	1
1.2	Problem Definition	4
1.3	Organization	5
2	Background and Literature	6
2.1	Overview of the Domain	6
2.2	Formal Methods and RL Algorithms	7
2.2.1	Markov Decision Process (MDP)	7
2.2.2	Policy Gradient Methods	9
2.2.3	Logically Constrained RL (LCRL)	12
2.3	Overview of Autonomous Driving	16
2.3.1	SAE Levels	16
2.3.2	Operational Design Domains (ODD)	17
2.3.3	Object and Event Detection and Response (OEDR)	17
2.3.4	CARLA (Car Learning to Act)	20
3	Agent and Environment	21
3.1	Overview	21
3.2	Terminology	23

CONTENTS

3.3	Bird Eye View (BEV)	24
3.4	Sensors	26
3.4.1	Vision Sensors	26
3.5	Scene Builder	30
3.5.1	Actors in Vision	31
3.5.2	Actors on the Path	31
3.5.3	Separation Distance	32
3.6	Planner	33
4	Reinforcement Learning	35
4.1	Overview	35
4.2	RL Modelling	36
4.2.1	States	37
4.2.2	Actions	37
4.2.3	Rewards	38
4.3	Safety Properties and LCRL	39
4.3.1	Guard Property: Collision Prevention	39
4.3.2	Guide Property: Controlled Switching	40
5	Experiments and Results	44
5.1	System Setup	44
5.2	Simulation Scenario	45
5.2.1	Ego Vehicle Path	46
5.2.2	Simulation of Experiments	47
5.3	Experiments	49
5.3.1	Training	49
5.3.2	Testing	51
5.3.3	Experiment 1	54
5.3.4	Experiment 2	55

CONTENTS

5.3.5	Experiment 3	57
5.3.6	Experiment 4	58
5.3.7	Experiment 5	59
5.3.8	Experiment 6	60
5.3.9	Experiment 7	61
5.3.10	Experiment 8	63
5.3.11	Experiment 9	64
5.3.12	Experiment 10	65
5.3.13	Experiment 11	66
5.3.14	Experiment 12	67
5.3.15	Experiment 13	68
5.3.16	Experiment 14	69
6	Conclusions	71
6.1	Contributions	71
6.2	Future Work	73
	Bibliography	75

List of Figures

2.1	Typical OEDR process based on “Sense, Plan, Act”	17
3.1	Agent and environment	21
3.2	Bird Eye View with reference to the agent (marked by a red circle), the horizontal arrow shows the front direction and the vertical arrows point in the left (L) and right (R) directions	24
3.3	Grid position calculation for the location of an actor A with reference to the location of the ego car E	25
3.4	Coverage range of Ultrasonic sensor shown by the shaded region in brown on top of the BEV	27
3.5	Coverage range of Lidar shown by the shaded region in brown on top of the BEV	27
3.6	Coverage range of Short Range Camera (Camera-1) shown by the shaded region in brown on top of the BEV	28
3.7	Coverage range of Far Range Camera (Camera-2) shown by the shaded region in brown on top of the BEV	28
3.8	Coverage range of Radar shown by the shaded region in brown on top of the BEV	28
3.9	Combined coverage range for vision sensors. Each grid position in the BEV marks the group of sensors covering that grid position. The group of sensors are indicated by colour coding as mentioned in the legend below the figure.	29
3.10	Scene Builder and Planner	30

LIST OF FIGURES

3.11	Actors which lie in the path of the ego car at a given instant shown in violet, other road users which are not at present in the path are shown in grey. The Vehicle 3 qualifies as the front actor in the figure above.	31
3.12	Separation distance calculation between the ego car and the front road user which can either be a vehicle (X) or a pedestrian (Y) shown in violet colour	32
4.1	Interactions of the RL Agent with the environment and the LDBA . .	36
4.2	LDBA-1	40
4.3	LDBA-2	41
4.4	LDBA-3	42
4.5	LDBA-4	43
5.1	Map of CARLA's Town 5 which was used for running the simulations	45
5.2	Sample image from the CARLA simulator, showing the agent as a white BMW Grandtourer and other road users in Town 5	46
5.3	Recorded path of the ego vehicle in Town-5 (shown in blue), during it's run in the autopilot mode. The red circle shows the starting position of the ego vehicle and the initial direction of motion is marked by the green arrow.	46
5.4	Simulation output showing the ego car and the environment during one of the training runs. The green boxes mark the road users near the ego car. The console output for this instance is shown in Listing 5.1	48
5.5	The relative positioning of the Maximal Selection Reward and Optimal Selection Reward. High reward region marked by the red arrow. .	51
5.6	Training reward values for Experiment 1	54
5.7	Training episode times for Experiment 1	55

LIST OF FIGURES

5.8	Training reward values for Experiment 2	56
5.9	Training episode times for Experiment 2	56
5.10	Training reward values for Experiment 3	57
5.11	Training reward values for Experiment 4	58
5.12	Training reward values for Experiment 5	59
5.13	Training reward values for Experiment 6	60
5.14	Training reward values for Experiment 7	61
5.15	Training reward values for Experiment 8	63
5.16	Training reward values for Experiment 9	64
5.17	Training reward values for Experiment 10	65
5.18	Training reward values for Experiment 11	67
5.19	Training reward values for Experiment 12	68
5.20	Training reward values for Experiment 13	69
5.21	Training reward values for Experiment 14	70
6.1	(Left) Current framework, green boxes represent the components which can be evolved for advance simulation. (Right) Evolving the current setup for advanced simulation to learn policies for real-world deploy- ment.	73

List of Tables

3.1	Detection failure rates of vision sensors depending upon the specific weather condition at noon	29
3.2	Detection failure rates of vision sensors depending upon the specific weather condition at sunset	30
5.1	List of 14 experiments, each row represents the settings of an experiment, the columns indicate the parameters which are varied across experiments	49
5.2	Testing results for settings S-1 and S-2 reported for Experiments 1-14. Test results with 100% high reward and 0% collision are marked in green. The test with 90-100% high reward and less than 10% collision is marked in yellow.	52
5.3	Testing results	54
5.4	Test results for Experiment 2	56
5.5	Test results for Experiment 3	57
5.6	Test results for Experiment 4	58
5.7	Test results for Experiment 5	60
5.8	Test results for Experiment 6	61
5.9	Test results for Experiment 7	62
5.10	Test results for Experiment 8	63
5.11	Test results for Experiment 9	65
5.12	Test results for Experiment 10	66
5.13	Test results for Experiment 11	67

LIST OF TABLES

5.14	Test results for Experiment 12	68
5.15	Test results for Experiment 13	69
5.16	Test results for Experiment 14	70

Chapter 1

Introduction

1.1 Motivation

As in any intelligent-agent scenario, the actions of an autonomous driving agent are based on the perception of the environment via sensors. In the case of self-driving cars, multiple sensor modules are designed for processing input from each sensor. The bandwidth for monitoring and managing inputs from these modules is limited and needs optimization.

The Society of Automotive Engineers (SAE) defines the level of automation for autonomous vehicles [29]. The level of automation increase from level 0 to 5. These levels are described in Section 2.3.1. We aim to have optimal sensor module selection for SAE level 4+, which corresponds to achieving high/full automation where no involvement of a human driver is required to ensure safety. In such a scenario, deciding on which modules to enable is an extremely challenging problem due to two main reasons:

- Firstly, the environment is very *dynamic*. Following are a few aspects: 1) Vulnerable road users (e.g, pedestrians) can appear in the scene at any time
- 2) The behaviour of road users can not be perfectly modelled, vehicles can take

sudden turns and might change lanes 3) There may be blind spots and merging lanes 4) Weather conditions may change dynamically, impacting motion and perception 5) The surroundings change rapidly as both the car and other road users are in motion.

- Secondly, the system is very *complicated*; it has a large number of sensors, many of which are of different modalities like radar, lidar, and camera. The performance of each sensor varies with changing weather conditions. On top of each sensor there are several modules which have varying computational requirements. The vehicle is controlled by a combination of steering, throttle and brake. The exact impact of these on motion of the vehicle, depends on the components of the vehicle itself.

Sensor module selection in such a diverse and complicated environment can not be ensured by simple rules set a priori as it is not possible to exhaustively enumerate every possible scenario which might be faced during autonomous driving. Optimizing sensor selection for a given utility has been shown to be an NP-hard problem by Bian et al. in [4]. However, even simple changes to this setup, such as considering the minimization of errors via combining more than one sensors, makes even the analysis of the problem itself quite hard. Joshi et al. [24] resorted to empirical approaches in such cases.

More recently, it was shown that assuming linear dynamics of the system, sensor selection even for stable systems, when minimizing error covariance in *posterior* state estimate is NP-hard [56]. On the other hand, though some early approaches such as [22] have attempted looking into offline and online settings for optimal sensor selection for autonomous vehicles, traditionally the industry has opted for deploying as many sensors as possible with a given resource-budget called as *Maximal Sensor Selection* (MSS). Placement of sensor in such cases then becomes a problem to solve such as [55] who use reinforcement learning to solve *sensor placement* problem.

We assume sensor placement is solved and for sensor selection we take a simulation based learning approach. Instead of simplifying complex dynamics of the environment, we assume a simulator exists that is capable of enough *realism* and *variance*. Our objective then is to learn the optimal sensor selection without the detailed information of underlying dynamics of the system. In that sense, we are trying to implicitly emulate the concept of *attention*, in the sense of the approaches such as [5] and [6], where the underlying assumption is that the controller is resource-constrained while the major point of difference is that we neither make any assumption of the system dynamics nor restrict our setup to certain simplified interactions.¹

In case of Maximal Sensor Selection (MSS) there are instances when multiple modules provide redundant information. Therefore, allocating resources to such sensor modules is a waste of resources. If we have an intelligent selector which can predict which sensor modules to enable, it can lead to significantly saving computational resources. ZF Friedrichshafen², amongst other self driving car companies, faces this challenge. Through this project work, we endeavour to come up with a novel solution for this problem in collaboration with ZF. Currently ZF solves the MSS problem for all ODDs (Operation Design Domain: namely driving context, e.g., day/night). To account for the limited bandwidth of the signals, we need automated decision making to actively decide which sensors to be enabled. Automated decision making is essential as manual decision trees do not scale for any real Autonomous Driving (AD) scenario. Hence, in this work we employ Reinforcement Learning (RL) to tackle this challenge.

¹Though in some experiments, we simplify the interactions such as presence of only pedestrians, our approach is general enough as shown in other experiments.

²<https://www.zf.com/mobile/en/homepage/homepage.html>

1.2 Problem Definition

The ego vehicle, i.e. the subject car, has multiple vision sensors and a sensor module corresponding to each one of them. The role of the **Sensor Module Selector** will be to dynamically decide which sensor modules to enable/disable. Whenever a sensor module is enabled, the corresponding sensor information will be processed to perceive the world. Based on this perception, the ego vehicle will accordingly adjust its speed for Adaptive Cruise Control (ACC).

Problem (sensor module selection). *Our goal is to build an “Intelligent Sensor Module Selector”, which ensures optimality, feasibility and scalability.*

Optimality: A fixed cost is associated with enabling/disabling each sensor module, another for keeping them enabled. For optimality, the selector should perform dynamic selection such that it incurs the least cost and the ego vehicle drives safely.

Feasibility: It should be possible to learn an optimal policy within a limited period of time with constrained resources and there must exist at least one optimal policy.

Scalability: The selector should be able to find an optimal policy in diverse ODDs.

Remark 1.2.1. *The problem of sensor module selection implicitly models an underlying problem of active monitoring. We define active monitoring as the problem of actively choosing which regions in the environment should be monitored by the agent. Each sensor module corresponds to a vision sensor with a predefined coverage range. Actively choosing which sensor modules to enable therefore decides on which areas are monitored by the agent.*

1.3 Organization

To systematically approach the problem, first we model the ego vehicle as an autonomous agent acting in the world (the environment). This modelling, enables us to methodologically approach the problem as an AI problem. In Chapter 3, we explain in detail the components of the agent, the environment and the interactions amongst them. Then to solve the problem, we make use of RL by modelling the Intelligent Sensor Module Selector as an RL agent. Chapter 4, describes the complete model and further explains how Logical Constraints can be employed to make the problem tractable in practice. Logical Constraints not only help in reducing the search space by acting as a *guide* for the agent but also allow to introduce safety property in RL to enforce strict *guards*.

We then take the developed Logically Constrained Reinforcement Learning (LCRL) framework, and with the help of the CARLA simulator deploy our approach. In Chapter 5, we elaborate on the specific implementation details and present fourteen experiments. These experiments compare and contrast multiple LTL properties, variations in RL modelling and finally demonstrate the efficacy of the LCRL approach. It shows that our approach is scalable and building an optimal sensor module selector is feasible. We finally conclude in Chapter-6 with clear directions for future work to evolve our setup for adapting to real-world scenarios.

Chapter 2

Background and Literature

2.1 Overview of the Domain

Intel envisions that for enabling safety for self-driving cars, the autonomous system will need to process more than 1 GB of sensor data every second [23]. They also list the need for greater computing power as the topmost requirement to enable safety. The fusion of sensor data from sensors of multiple modalities (e.g., lidar, radar, advanced camera technology) lead to enormous amounts of data which is to be processed quickly enough to control the motion of the ego vehicle in a timely manner. If the load of processing irrelevant or redundant sensor data can be reduced then other critical aspects for ensuring safety can be allocated a higher bandwidth. Hence, we focus on building an intelligent sensor module selector which can significantly reduce the amount of sensor data which needs to be processed every second.

The field of automated driving systems is inspired from mobile robotics and borrows research findings from it. An autonomous vehicle (ego vehicle) can be conceptualised as a robot acting in the environment. The environment for the vehicle is dynamic, partially observable and the stochastic. Hence, the actions of the ego vehicle can not fully determine the updated state of environment and there

are inherent failures. Execution monitoring system is a system that allows the robot to detect and classify these failures [35]. Moreover, it also aims to find the causes of these failures. When considering execution monitoring, in automated driving systems we do not want to make any assumptions about the environment to account for unexpected and unseen behaviours. Using RL enables us to not have any predefined model of the dynamics of the environment and build a framework for learning a general policy. The field of applying RL in execution monitoring is relatively recent and has several open questions. The following are some of those we explore in the course of this work:

1. Active monitoring on perception module is often constrained with limited computational resources. It also must respond safely while doing minimal monitoring. How can an optimal balance between monitoring and performance be achieved?
2. Is maximal sensor set selection the best policy to adopt?
3. Can RL be utilised to obtain a general method to solve (1)?

There have been several interesting works to provide formal guarantees for safety in robotics [10,30] but relatively fewer in self-driving cars [46]. Deep RL frameworks have been explored for trajectory planning in automotive [42] but the combined applications of RL and verification for active monitoring (sensor module selection) is unexplored. In the next section we detail upon the theoretical background for our work.

2.2 Formal Methods and RL Algorithms

2.2.1 Markov Decision Process (MDP)

Markov Decision Process (MDP) are used to model transition systems which have probabilistic and non-deterministic behaviour [17]. The interactions between an

agent and the environment can be modelled using an MDP. The assumption of non-determinism, helps in accounting for the unknown behaviour of the environment and underspecification of the system model.

Definition 2.2.1 (MDP). *An MDP is defined by a tuple $\mathbf{M} = (\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$ over a set of states $\mathcal{S}$, where $\mathcal{A}$ is a finite set of actions, s_0 is the initial state and $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a transition probability function that determines the probability of moving from a given state to another by taking an action [3]. $\mathcal{AP}$ is a finite set of atomic propositions and a labelling function $L : \mathcal{S} \rightarrow 2^{\mathcal{AP}}$ assigns to each state $s \in \mathcal{S}$ a set of atomic propositions $L(s) \subseteq 2^{\mathcal{AP}}$ [14].*

Definition 2.2.2 (Path). *The sequence of actions taken from the initial state (s_0) in an MDP ($\mathbf{M}$) form a path ρ . The i -th state of ρ is denoted by $\rho[i]$, where $\rho[i] = s_i$. Furthermore, the i -th suffix of ρ is $\rho[i..]$ where $\rho[i..] = s_i \xrightarrow{a_i} s_{i+1} \xrightarrow{a_{i+1}} s_{i+2} \xrightarrow{a_{i+2}} s_{i+3} \xrightarrow{a_{i+3}} \dots$. For any path ρ the $P(s_t, a_t, s_{t+1}) > 0$ for all $t \in \mathbb{N}$.*

In an MDP $\mathbf{M}$, we define a function $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ that denotes the immediate scalar bounded reward received by the agent from the environment after performing action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$.

Definition 2.2.3 (Stochastic Policy). *A stochastic policy π_θ , defines the probability distribution from any state $s \in \mathcal{S}$ to the set of actions $a \in \mathcal{A}$. $\pi_\theta(a_t | s_t)$ denotes the probability of taking an action a_t in state s_t at time step t . A deterministic policy is a degenerate case of a stochastic policy in which for each state there exists exactly one action which gives probability 1 (and for all other actions the probability is 0). Formally, $\forall s \in \mathcal{S}, \exists a \in \mathcal{A}, \pi_\theta(a | s) = 1$.*

Definition 2.2.4 (Expected Discounted Reward). *For a deterministic policy π_θ defined on an MDP $\mathbf{M}$, and given a reward function R , the expected discounted reward at state s is defined as [49]:*

$$U^{\pi_\theta}(s) = \mathbb{E}^{\pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid \pi_\theta(a_t | s_t) = 1, s_0 = s \right] \quad (2.1)$$

where $\mathbb{E}^{\pi_\theta}[\cdot]$ denotes the expected value when the agent follows policy π_θ from state s . γ is the discount factor which lies in the range $[0, 1]$, when the episodes have a finite number of steps. $s_0, \dots, s_n$ is the sequence of states generated by policy π_θ up to time step n , initialised at $s_0 = s$.

Definition 2.2.5 (Optimal Policy). An optimal policy π_θ^* is defined as follows:

$$\pi_\theta^*(s) = \arg \sup_{\pi_\theta \in \pi_{\mathcal{D}}} U^{\pi_\theta}(s),$$

where $\pi_{\mathcal{D}}$ is the set of stationary deterministic policies over the state space $\mathcal{S}$.

Theorem 2.2.1 (From [7, 37]). In any MDP $\mathbf{M}$ with a bounded reward function and a finite action space, if there exists an optimal policy, then that policy is stationary and deterministic.

2.2.2 Policy Gradient Methods

To synthesize an optimal policy for an MDP two approaches are popular in practice; 1) Value Iteration-based methods such as Q Learning, and 2) Policy Gradient methods. Q Learning approximates a Q function based on value iteration which in turn helps in finding an optimal policy. On the other hand, policy gradients help in optimizing directly in the search space. Specifically, it uses a gradient ascent algorithm with a gradient estimator which typically has the following form:

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \hat{A}_t \right]$$

In software implementations however the estimate for $\hat{g}$ is computed by differentiating the following objective:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_{\theta}(a_t | s_t) \hat{A}_t \right]$$

Here $\hat{\mathbb{E}}_t$ represents expectation, averaged over a batch of samples from an algo-

rithm that alternates between sampling and optimization. π_θ represents a stochastic policy, $\hat{A}_t$ is an estimator of the advantage function at time-step t .

Trust Region Policy Optimization (TRPO)

TRPO [44] is a trust region based policy gradient method in which a “surrogate” objective is maximized where the policy size is subject to a constraint as follows:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t [\text{KL}[\pi_{\theta_{\text{old}}}(\cdot | s_t), \pi_\theta(\cdot | s_t)]] \leq \delta \end{aligned}$$

If in the above equation, we use $r_t(\theta) = \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right]$ then an equivalent representation of the objective function is maximizing:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t [r_t(\theta) \hat{A}_t]$$

where CPI stands for conservative policy iteration and was introduced by [25]. Without constraining the above objective the updates to the policy become extremely large. Hence, PPO resorts to modify the objective for penalising actions which lead $r_t(\theta)$ away from 1.

Proximal Policy Optimization (PPO)

PPO [45] is one of the recent policy gradient methods for reinforcement learning. As shown in [45] it outperforms most of the previous algorithms on continuous control tasks. Moreover it is more general, simpler to implement than TRPO (even though PPO offers similar benefits as TRPO) and has empirically better sample complexity. Hence, we use PPO for most of our experiments¹. Implementing PPO requires using

¹We use Tensorforce for off-the-shelf implementation of PPO and TRPO but here we explain the main objectives of each for completeness.

a Clipped Surrogate Objective defined as follows:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip} \left(r_t(\theta), 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \right) \right]$$

In the above equation, ϵ is a hyperparameter. The first term inside the minimum function (i.e., $r_t(\theta) \hat{A}_t$), essentially corresponds to $L^{CPI}(\theta)$ as in the surrogate objective of TRPO. The second term bounds the incentive of the surrogate objective within ϵ from 1 in either direction. The minimum of these two terms is used to update the objective, as PPO takes a pessimistic bound on the surrogate objective.

Then using the L^{CLIP} a combined objective L is computed by [45]. We restrict our discussion to L^{CLIP} , as that is the main differentiating factor of PPO when compared to other policy gradient approaches. For completeness we present the following algorithm as used by Schulman et al., to compute the optimal policy.

Algorithm 1: PPO, Actor Critic Style

```

for  $iteration = 1, 2, \dots$  do
    | for  $actor = 1, 2, \dots, N$  do
    | | Run policy  $\pi_{\theta_{old}}$  in environment for  $T$  timesteps
    | | Compute advantage estimates  $\hat{A}_1, \dots, \hat{A}_T$ 
    | end
    | Optimize surrogate  $L$  wrt  $\theta$ , with  $K$  epochs and minibatch size  $M \leq NT$ 
    |  $\theta_{old} \leftarrow \theta$ 
end

```

In the above algorithm, N is the number of parallel actors. The surrogate loss L is optimized over the set of data gathered by the N actors running for T timesteps using minibatch stochastic gradient descent for K epochs. In the end, the updated θ value determines the policy π_θ .

2.2.3 Logically Constrained RL (LCRL)

LCRL [20], is a framework to constrain synthesising the optimal policy for an unknown MDP via RL such that the obtained structure satisfies a given Linear Temporal Logic (LTL) property. We use the above described policy gradient RL algorithms (PPO and TRPO) for LCRL. The LTL property is encoded as a Limit-Deterministic Büchi automaton (LDBA). The autonomous agent is modelled as an MDP. To ensure that the agent while interacting in the environment abides by a given LTL property, LCRL proposes to construct a product MDP using the LTL property and the agent MDP. Next, we explain the necessary background for LCRL.

Linear Temporal Logic (LTL) Properties

Linear Temporal Logic (LTL) enables us to specify long-run properties of transition systems. Specifically, LTL can be used to express temporal properties over the action labels of the MDP. Formally, LTL formulae over a given set of atomic propositions $\mathcal{AP}$ are syntactically defined as [36]

$$\varphi ::= \text{true} \mid \alpha \in \mathcal{AP} \mid \varphi \wedge \varphi \mid \neg \varphi \mid \bigcirc \varphi \mid \varphi \text{ U } \varphi, \quad (2.2)$$

where the operators $\bigcirc$ and U are called “next” and “until”, respectively. To apply LTL properties over MDP we model the set of actions $\mathcal{A}$ of the MDP as the atomic propositions $\mathcal{AP}$ for LTL. The semantics of LTL formulae, when interpreted over MDPs is defined as follows:

Definition 2.2.6 (LTL Semantics). *For an LTL formula φ and for a path ρ , the satisfaction relation $\rho \models \varphi$ is defined as*

$$\rho \models \alpha \in \mathcal{AP} \Leftrightarrow \alpha \in L(\rho[0]),$$

$$\rho \models \varphi_1 \wedge \varphi_2 \Leftrightarrow \rho \models \varphi_1 \wedge \rho \models \varphi_2,$$

$$\rho \models \neg\varphi \Leftrightarrow \rho \not\models \varphi,$$

$$\rho \models \bigcirc\varphi \Leftrightarrow \rho[1..] \models \varphi,$$

$$\rho \models \varphi_1 \mathbf{U} \varphi_2 \Leftrightarrow \exists j \in \mathbb{N}_0 \text{ s.t. } \rho[j..] \models \varphi_2 \text{ and } \forall i, 0 \leq i < j, \rho[i..] \models \varphi_1.$$

The “next” operator $\bigcirc$ requires φ to be satisfied starting from the next-state suffix of the path ρ . The “until” operator $\mathbf{U}$, is satisfied over ρ if φ_1 continuously holds until φ_2 becomes *true*. The until operator can be leveraged to define two more temporal modalities: (1) eventually, $\Diamond\varphi = \text{true} \mathbf{U} \varphi$; and (2) always, $\Box\varphi = \neg\Diamond\neg\varphi$. The intuition for $\Diamond\varphi$ is that φ has to become *true* at some finite point in the future, whereas $\Box\varphi$ means that φ has to remain *true* forever. An LTL formula φ over $\mathcal{AP}$ specifies the following set of words:

$$\text{Words}(\varphi) = \{\sigma \in (2^{\mathcal{AP}})^\omega \text{ s.t. } \sigma \models \varphi\}. \quad (2.3)$$

A set of words can be encoded by the language of an automaton. To encode the $\text{Words}(\varphi)$ we use an LDBA which is a specific form of GBA. Hence, we next discuss GBA and LDBA.

Generalised Büchi Automaton (GBA)

A GBA $\mathbf{N} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ is a structure where $\mathcal{Q}$ is a finite set of states, $q_0 \in \mathcal{Q}$ is the initial state, $\Sigma = 2^{\mathcal{AP}}$ is a finite alphabet, $\mathcal{F} = \{F_1, \dots, F_f\}$ is the set of accepting conditions, where $F_j \subset \mathcal{Q}, 1 \leq j \leq f$, and $\Delta : \mathcal{Q} \times \Sigma \rightarrow 2^{\mathcal{Q}}$ is a transition relation. Let Σ^ω be the set of all infinite words over Σ . An infinite word $\sigma \in \Sigma^\omega$ is accepted by a GBA $\mathbf{N}$ if there exists an infinite run $\theta \in \mathcal{Q}^\omega$ starting from q_0 where $\theta[i+1] \in \Delta(\theta[i], \sigma[i]), i \geq 0$ and for each $F_j \in \mathcal{F}$

$$\inf(\theta) \cap F_j \neq \emptyset, \quad (2.4)$$

where $\text{inf}(\theta)$ is the set of states that are visited infinitely often by the run θ .

Limit-Deterministic Büchi automaton (LDBA)

A GBA $\mathbf{N} = (\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ is limit-deterministic if $\mathcal{Q}$ can be partitioned into two disjoint sets $\mathcal{Q} = \mathcal{Q}_N \cup \mathcal{Q}_D$, such that [47]:

- $\Delta(q, \alpha) \subset \mathcal{Q}_D$ and $|\Delta(q, \alpha)| = 1$ for every state $q \in \mathcal{Q}_D$ and for every $\alpha \in \Sigma$,
- for every $F_j \in \mathcal{F}$, $F_j \subset \mathcal{Q}_D$,
- $q_0 \in \mathcal{Q}_N$, and all the transitions from $\mathcal{Q}_N$ to $\mathcal{Q}_D$ are non-deterministic ε -transitions²

Essentially, an LDBA is a GBA consisting of two partitions: initial ($\mathcal{Q}_N$) and accepting ($\mathcal{Q}_D$). The accepting part includes all the accepting states and has deterministic transitions.

LTL to LDBA Construction

The LTL-to-LDBA algorithm proposed in [47] builds an automaton with two parts (initial $\mathcal{Q}_N$ and accepting $\mathcal{Q}_D$). These two parts have deterministic transitions, and across these the transitions are non-deterministic (ε). This construction of LDBA is stronger than the required conditions for an LDBA as there is determinism in $\mathcal{Q}_N$. During an episode of the RL algorithm, the ε -transitions between $\mathcal{Q}_N$ and $\mathcal{Q}_D$ represents the agent’s “guess” on reaching $\mathcal{Q}_D$. If after an ε -transition the associated labels in the accepting set of the automaton cannot be read, or the accepting states cannot be visited, then the guess is deemed to be wrong, and the trace is disregarded.

Product MDP

Given an MDP $\mathbf{M}(\mathcal{S}, \mathcal{A}, s_0, P, \mathcal{AP}, L)$ and an LDBA $\mathbf{N}(\mathcal{Q}, q_0, \Sigma, \mathcal{F}, \Delta)$ with $\Sigma = 2^{\mathcal{AP}}$, the product MDP is defined as $(\mathbf{M} \otimes \mathbf{N}) = \mathbf{M}_N = (\mathcal{S}^\otimes, \mathcal{A}, s_0^\otimes, P^\otimes, \mathcal{AP}^\otimes, L^\otimes, \mathcal{F}^\otimes)$,

²An ε -transition allows an automaton to change its state without reading any atomic proposition.

where $\mathcal{S}^\otimes = \mathcal{S} \times \mathcal{Q}$, $s_0^\otimes = (s_0, q_0)$, $\mathcal{AP}^\otimes = \mathcal{Q}$, $L^\otimes = \mathcal{S} \times \mathcal{Q} \rightarrow 2^\mathcal{Q}$ such that $L^\otimes(s, q) = q$ and $\mathcal{F}^\otimes \subseteq \mathcal{S}^\otimes$ is the set of accepting states $\mathcal{F}^\otimes = \{F_1^\otimes, \dots, F_f^\otimes\}$, where $F_j^\otimes = \mathcal{S} \times F_j$. The transition kernel $P^\otimes$ is such that given the current state (s_i, q_i) and action a , the new state is (s_j, q_j) , where $s_j \sim P(\cdot | s_i, a)$ and $q_j \in \Delta(q_i, L(s_j))$. Additionally, to handle ε -transitions the following modifications has to be added to the above definition of product MDP:

- (1) for every potential ε -transition to some state $q \in \mathcal{Q}$ we add a corresponding action ε_q in the product:

$$\mathcal{A}^\otimes = \mathcal{A} \cup \{\varepsilon_q, q \in \mathcal{Q}\}.$$

- (2) the transition probabilities corresponding to ε - transitions are given by

$$P^\otimes((s_i, q_i), \varepsilon_q, (s_j, q_j)) = \begin{cases} 1 & \text{if } s_i = s_j, q_i \xrightarrow{\varepsilon_q} q_j = q, \\ 0 & \text{otherwise.} \end{cases}$$

Intuitively, the product MDP does the following: As the actions are performed by the agent (the MDP) the labels corresponding to the actions update the state of the LDBA. If the LDBA reaches a sink state (a state from which it is impossible to reach an accepting state) the agent terminates it's execution. Essentially, we use this concept of early termination to make the RL efficient and effective.

2.3 Overview of Autonomous Driving

We briefly introduce some of the commonly used terms in autonomous driving which are relevant to our work.

2.3.1 SAE Levels

The Society of Automotive Engineers (SAE) defines the levels of automation for autonomous vehicles [29]. The level of automation increase from level 0 to 5. The high level description of these levels are listed below from [8]:

- Human driver is in control of the vehicle for Level 0,1,2. The features are meant only for supporting the driver and need constant supervision.
 - Level 0: Warnings and momentary assistance (e.g., automatic emergency braking, lane departure and blind spot warning)
 - Level 1: Basic steering, brake and acceleration support (e.g., either Adaptive Cruise Control (ACC) or lane centring at a time)
 - Level 2: Advanced steering, brake and acceleration support (e.g., ACC and lane centring at the same time)
- The system controls the motion when automatic features are engaged in Level 3,4,5. These features are meant for automated driving.
 - Level 3: Allows for automated driving in limited environments and can issue a take-over request. On such a request the driver must take over the control. (e.g., traffic jam chauffeur)
 - Level 4: The system works given that certain conditions are satisfied. In no scenario is a driver expected to take over the control. (e.g., local driverless taxi, pedal/steering may or may not be present)
 - Level 5: Automated driving which works under all conditions and never requests for taking over the control. (e.g., driverless cars which can work in all environments)

In our work, we have focused on SAE Level 4+ i.e., we aim at tackling scenario's where the system is completely controlling the ego vehicle and have tried to scale it to different conditions to reach closer to Level 5.

Remark 2.3.1. *We assume that there already exists a self-driving vehicle operating at Level 4+. Our objective is only to enable intelligent sensor module selection (as mentioned in Section 1.2) for such a vehicle.*

2.3.2 Operational Design Domains (ODD)

Operational Design Domains (ODD) are defined by various institutions for enforcing standards (like The British Standards Institution [54]) and are widely used in autonomous driving industry (e.g., BMW [39]). ODD is used to refer to the specific conditions for which an automated driving system is designed to operate within. The conditions typically include geographical conditions (e.g., highways, city roads), environmental conditions (e.g, time of the day, weather conditions, traffic rules), communication infrastructure (e.g., ethernet connection).

2.3.3 Object and Event Detection and Response (OEDR)

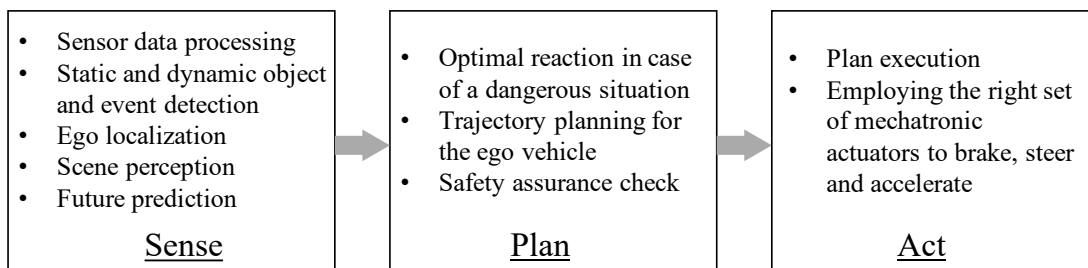


Figure 2.1: Typical OEDR process based on “Sense, Plan, Act”

In robotics, “Sense, Plan, Act” has been a classical approach since 1970s. The same three high levels steps are used across several autonomous driving companies. In the industry there are separate teams working on each of these three aspects. Object and event detection happens at the *Sense* step. The system then decides the

trajectory and control which is the responsibility of *Plan* step. The plan is then executed at the *Act* step.

ZF uses a slight variation, “See, Think, Act” [1, 43] which essentially refers to this triad. ZF uses numerous sensors like highly modern cameras, lidar and radar to *see* the environment. It then builds a complete 360° image of the environment and then *thinks* how best to *act* by controlling numerous mechatronic actuators.

In our work, the “Sense Plan Act” consists of the following (Fig. 3.1):³

- *Sense*: The sensor module selector, determines the active sensor modules that enable the scene builder to *see* the environment.
- *Plan*: ACC using Intelligent Driver Model (IDM) provides control speed and the target location is set based on a predefined path.
- *Act*: A proportional–integral–derivative controller (PID Controller) determines the control values based on target location and control speed.

Adaptive Cruise Control (ACC)

Cruise control is a feature that enables a system to control the speed of a vehicle. In case of ACC, the control speed of the ego vehicle is adjusted with respect to the road user in the front. The control speed ensures that a minimum separation distance is maintained between the ego vehicle and the front road user at all times. We use the **Intelligent Driver Model (IDM)** [50] to implement ACC.

Remark 2.3.2. *We use the IDM to compute the acceleration a_α for the ego vehicle at a frequency of 100 Hz and assume that the acceleration remains constant between two execution steps. This allows us to use the equation of motion for constant acceleration to determine the control speed $v_\alpha = \sqrt{2a_\alpha \times s + u_\alpha^2}$. s is taken as a*

³We elaborate on these in Chapter 3. The purpose of mentioning it here is to relate with “Sense, Plan, Act”

fixed distance on a predefined path. u_α is assumed to be the current speed of the ego vehicle and $u_{\alpha-1}$ denotes the current speed of the front road user. a_α is calculated using the equations of IDM as follows [33].

$$a_\alpha = a \left(1 - \left(\frac{u_\alpha}{v_0} \right)^\delta - \left(\frac{s^*(u_\alpha, \Delta u_\alpha)}{u_\alpha} \right)^2 \right) \quad (2.5)$$

$$s^*(u_\alpha, \Delta u_\alpha) = s_0 + u_\alpha T + \frac{u_\alpha \Delta u_\alpha}{2\sqrt{ab}} \quad (2.6)$$

$$\Delta u_\alpha = u_\alpha - u_{\alpha-1} \quad (2.7)$$

- s_0 : Minimum separation distance required between from the front road user
- v_0 : Set speed, the desired velocity for the ego vehicle
- T : Desired time headway from the front road user
- a : Maximum acceleration possible for the ego vehicle
- b : Comfortable braking deceleration
- δ : Acceleration exponent

Proportional Integral Derivative (PID) Controller

PID Controller [9, 19] is a common control algorithm adopted in several domains which require continuously modulated control. It adjusts a process variable's value to reach a desired setpoint by controlling actuators in a closed loop setting. The response of the actuator is based on the error $e(t)$, calculated as the difference between the setpoint and the measured value of the process variable. The overall response is determined by a combination of proportional response, integral response and derivative response. Mathematically, the control function is defined as follows:

$$u(t) = K_p e(t) + K_i \int_0^t e(t') dt' + K_d \frac{de(t)}{dt}$$

In the above equation K_p , K_i and K_d are constants corresponding to the proportional, integral and derivative terms. For our work, we used the default PID controller provided with CARLA APIs [2].

2.3.4 CARLA (Car Learning to Act)

We used CARLA (0.9.9) [2, 12] which is an open source simulator. CARLA was built to democratize the research and development in autonomous driving. It is built on top of Unreal Engine [16] and the road networks are defined using OpenDRIVE [34]. CARLA provides a scalable client-server architecture along with Python APIs for the client side.

The CARLA server takes care of the rendering of the entire simulation, controlling the physics of all simulated objects (pedestrians, vehicles) and updating the simulation based on messages from the CARLA clients. The server requires a dedicated machine with a powerful GPU.

The CARLA client provides modules for controlling the actors (e.g., pedestrians, vehicles, sensors, traffic lights, sensors) and customizing the world settings (e.g., setting the weather conditions, town). CARLA provides Python APIs [2] for the client side to control these settings.

Chapter 3

Agent and Environment

3.1 Overview

The goal of this chapter is to explain the problem setup for optimal autonomous vehicle perception. We explain in detail - (1) the components of our autonomous agent, (2) the environment and (3) the interactions amongst them. Fig. 3.1 is used as a reference to explain these aspects.

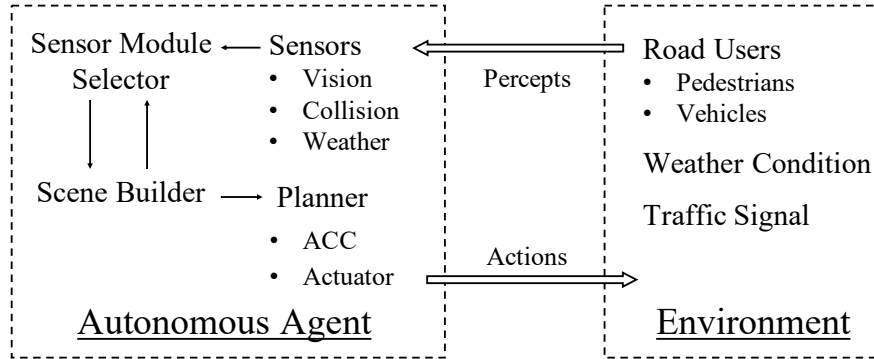


Figure 3.1: Agent and environment

The **autonomous agent** (also referred to as the agent) is the ego vehicle that travels on a predefined path. It uses a wide range of **vision sensors** for perceiving road users. Each vision sensor has a sensor module which can either be turned on or off by the **sensor module selector**. If a sensor module is activated (turned on) then it will detect road users present in its vision range (otherwise no detection will

happen for that sensor). The set of vision sensors for which the sensor modules are turned on is referred as the **active sensor set**. The sensor modules are assumed to be perfect apart from the failure rates based on environmental weather conditions. The **scene builder** takes the output of sensor modules of the active sensor set and builds a perception of the environment. Based on this perception, the ACC in the **planner** computes a set speed which is then used by the actuator to determine the appropriate actions for the ego vehicle.

We use the PEAS (Performance measure, Environment, Actuators, Sensors) approach as mentioned in [41] to model the **task environment**.

The **performance** is measured by ensuring collision-free driving with minimal sensor selection cost. The **environment** comprises of several road users, traffic signals and different weather conditions. **Road users (or actors)** are physical objects that have implicit or explicit motion models. We classify the road users as vulnerable road users [51] (pedestrians) and motorised road users (vehicles). The motorised road users obey traffic rules but the pedestrians may or may not strictly follow the rules. For the agent, the PID controller is the **actuator** which determines the actions i.e, steering, throttle and brakes. The available **sensors** for the agent include Collision Sensor, Weather Sensor, and Vision Sensors: {Ultrasonic, Lidar, Short Range Camera and Far Range Camera}.

The approach followed by autonomous vehicles is generally to have maximal sensor selection, which means that all vision sensors are used to detect the presence of road users. In such a case the environment becomes close to fully observable in ideal scenarios with respect to the detection of road users. However, we believe that the sensor selection cost can be optimized in a way that even with a **partially observable** environment the agent can achieve collision-free driving. The road users also

try to avoid collisions but may not always alter their motion to prevent the collision. Hence, the environment can be treated as **multi-agent** and **partially cooperative**. The overall problem is hard because the environment is **dynamic** and **stochastic** in nature. Further, the actuator values are **continuous** and so is the motion of the agents. To build an Intelligent Sensor Module Selector which is able to perform in such a challenging environment we use LCRL, the approach is elaborated in Chapter 4. This chapter gives the necessary background for building such a selector.

We give the details about modelling the environment, the sensors and the agent's perception. First, we introducing the terms in Section 3.2. Then we show how using a Bird Eye View (BEV) representation (Section 3.3) we model the perceived environment, particularly the locations of the road users. Section 3.4 covers the modelling of different sensors, their coverage ranges and failure rates. The sensor and environment model are then used in Section 3.5 to describe the scene builder. We conclude the chapter by mentioning how the perception influences the control of the actuators in Section 3.6.

3.2 Terminology

Since we are dealing with SAE Level 4+ driving as mentioned in Chapter 1, we can treat our autonomous agent as an intelligent robot and leverage some terms that are also used in robotics.

- Pose: The pose consists of 6 Degrees of Freedom - 3 for location coordinates (x, y, z) and 3 for rotation (roll, pitch, yaw).
- Path: The agent follows a predefined path. This path is defined by a sequence of Vehicle Poses.
- Set Speed: A constant speed at which the agent will travel if there are no other actors present on the path.

- Control Speed: The speed at which the agent aims to travel based on the dynamics of road users.
- Target Location: The location at which the agent aims to reach with a Control Speed.
- Agent Control: The control which determines the motion of the agent. It defines values for steering, throttle and brake.

3.3 Bird Eye View (BEV)

The bird eye view discretizes the world coordinates (and the location of actors) with respect to the agent. It covers 100m in the front direction and 25m on the left and right of the ego vehicle. Fig. 3.2 shows this view with the agent marked by a brown circle and its orientation specified by the arrows pointing in left, right and front direction.

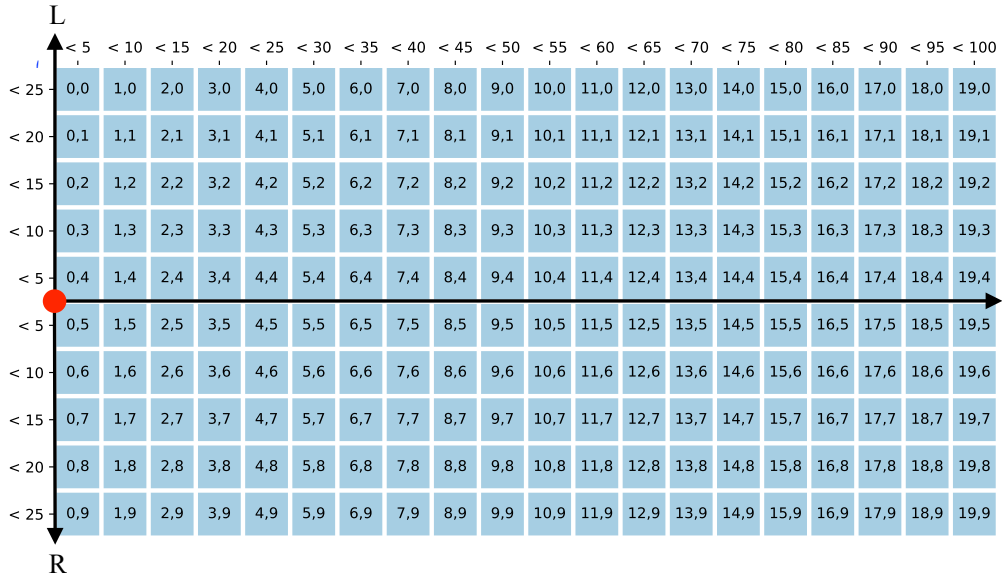


Figure 3.2: Bird Eye View with reference to the agent (marked by a red circle), the horizontal arrow shows the front direction and the vertical arrows point in the left (L) and right (R) directions

The positions are discretized by squares of length 5m. The horizontal and vertical labels indicate the upper limit of the distance in metres for each of the squares. For

example any point which is less than 5m in the front and 20-25m on the left is represented by the grid position (0,0).

Grid position calculation

To find the positions of the actors in the bird eye view, we consider the actors as point objects and find the grid index which corresponds to that point. If at a given instance, the agent is centred at (Ex, Ey) with an orientation of θ radians from the X-Axis and another Actor is centred at (Ax, Ay) then its grid position is calculated as follows:

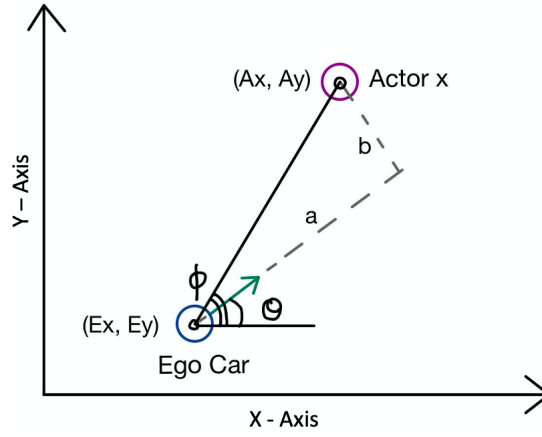


Figure 3.3: Grid position calculation for the location of an actor A with reference to the location of the ego car E

If $(Ax - Ex) > 0$

$$\phi = \tan^{-1}((Ay - Ey)/(Ax - Ex))$$

If $(Ax - Ex) < 0$ and $(Ay - Ey) < 0$

$$\phi = \tan^{-1}((Ay - Ey)/(Ax - Ex)) + \pi$$

If $(Ax - Ex) < 0$ and $(Ay - Ey) > 0$

$$\phi = \tan^{-1}((Ay - Ey)/(Ax - Ex)) - \pi$$

After finding the angle ϕ , we can find the distances a and b as:

$$a = \cos(\theta - \phi) \quad b = \sin(\theta - \phi)$$

The grid position of the actor x is then computed by dividing a and b by the discretization step of the grid-view (5m) which then comes out to be $(\lfloor a/5 \rfloor, \lfloor b/5 \rfloor)$.

3.4 Sensors

We assume that the agent has a **weather sensor**, a **collision sensor** and a **set of vision sensors**. The weather sensor and collision sensor are assumed to be always active and perfect in detection i.e., the collision sensor always detects a collision and the weather sensor always correctly classifies between different weather conditions. We consider a total of 14 weather conditions: 7 conditions (clear, cloudy, wet, wet-cloudy, soft-rain, mid-rain, hard-rain) at two times of the day (noon and sunset).

3.4.1 Vision Sensors

The Sensor Module Selector selects which vision sensors to enable or disable. The vision sensors are used to detect the presence of road users. For our experiments we focus on front facing sensors and aim at avoiding collisions which can be prevented using these sensors. Unavoidable collisions include scenarios where a road user collides with the agent from behind or from the sides.

The detection of the vision sensors is believed to be impacted by the weather conditions i.e, each vision sensor has a failure rate based on the weather condition. The probability of detecting an actor which is in the coverage range of a vision sensor can then be computed as (1-failure rate). The failure rates for each of the vision sensors are shown in Table 3.1 and Table 3.2.

We now list the coverage ranges in metres and coverage angle in degrees of all our vision sensors. The coverage angle is measured from the agent's location.

- Ultrasonic: Less than 10m distance with 180 degrees coverage (Fig. 3.4).
- Lidar: 5-20m in front with a coverage angle of 60 degrees (Fig. 3.5).
- Camera-1: 15-35m in front with a coverage angle of 120 degrees (Fig. 3.6).
- Camera-2: 25-65m in front with a coverage angle of 40 degrees (Fig. 3.7).
- Radar: 25-100m in front with a coverage angle of 28 degrees (Fig. 3.8).

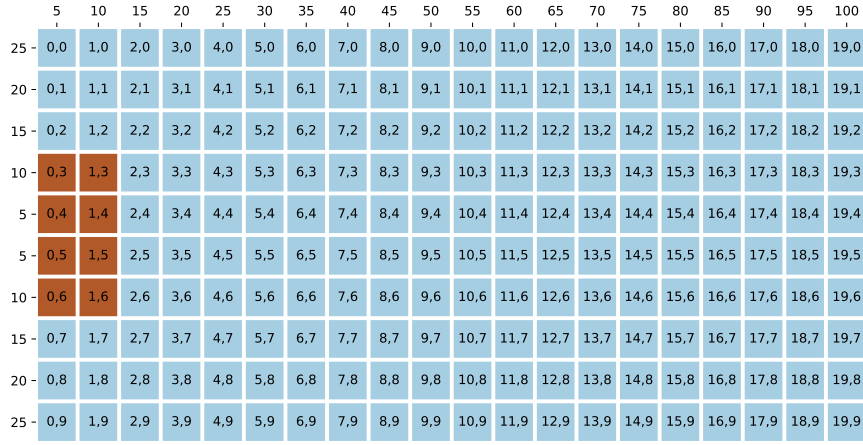


Figure 3.4: Coverage range of Ultrasonic sensor shown by the shaded region in brown on top of the BEV

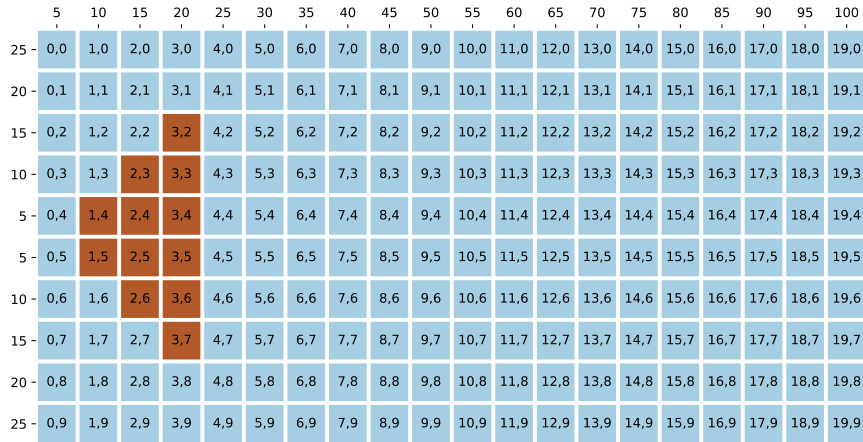


Figure 3.5: Coverage range of Lidar shown by the shaded region in brown on top of the BEV

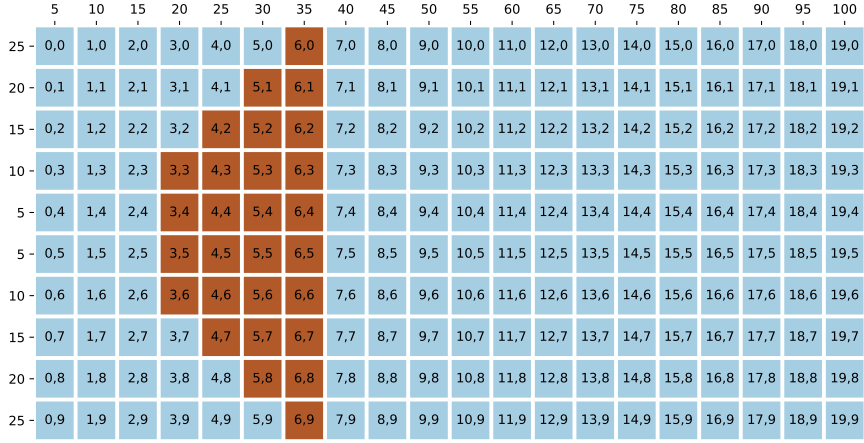


Figure 3.6: Coverage range of Short Range Camera (Camera-1) shown by the shaded region in brown on top of the BEV

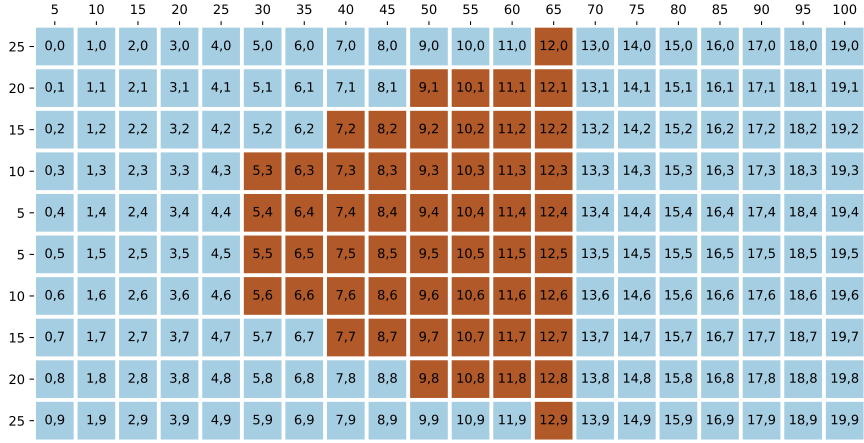


Figure 3.7: Coverage range of Far Range Camera (Camera-2) shown by the shaded region in brown on top of the BEV

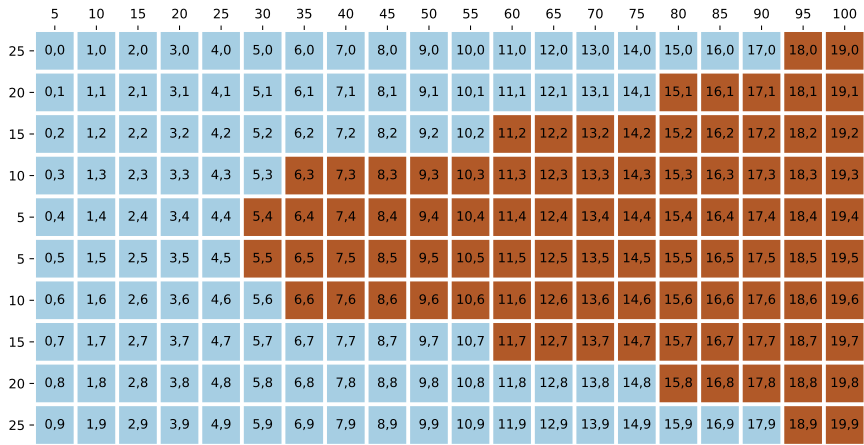


Figure 3.8: Coverage range of Radar shown by the shaded region in brown on top of the BEV

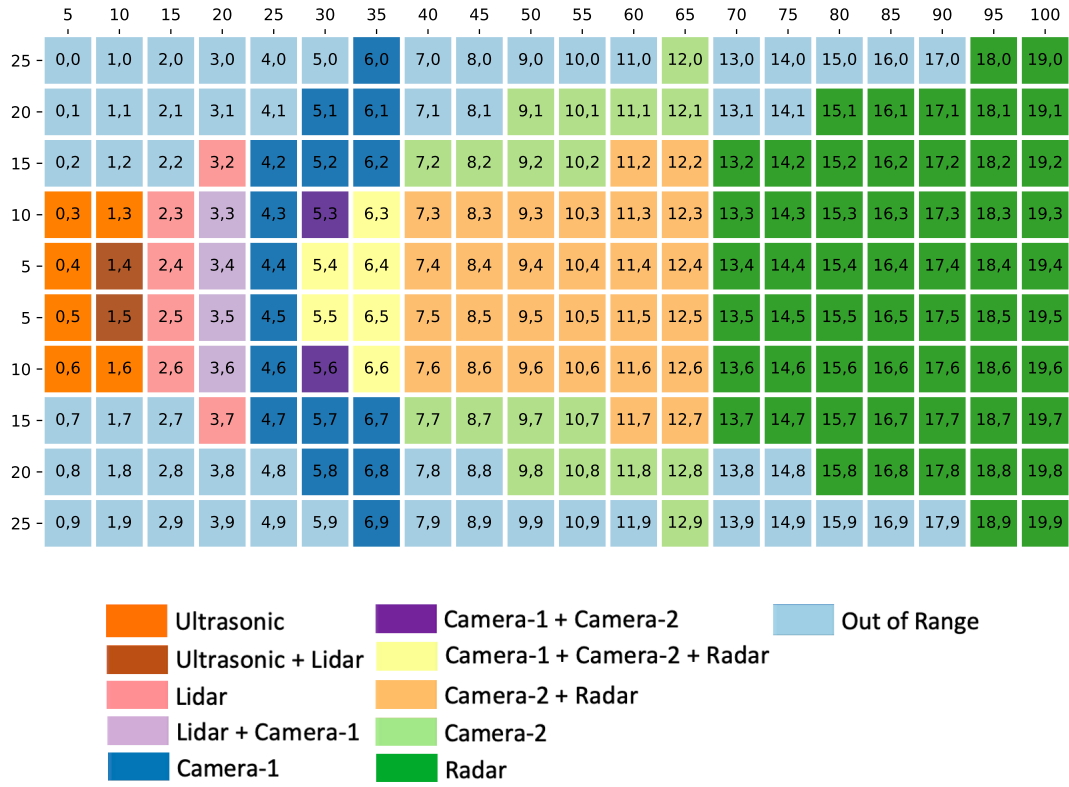


Figure 3.9: Combined coverage range for vision sensors. Each grid position in the BEV marks the group of sensors covering that grid position. The group of sensors are indicated by colour coding as mentioned in the legend below the figure.

Sensor	Noon Weather Conditions						
	Clear	Cloudy	Wet	Wet Cloudy	Mid Rain	Hard Rain	Soft Rain
Ultrasonic	0.02	0.02	0.02	0.02	0.02	0.02	0.02
Lidar	0.02	0.02	0.05	0.05	0.15	0.25	0.1
SR Camera	0.03	0.02	0.03	0.03	0.04	0.075	0.02
LR Camera	0.03	0.02	0.03	0.03	0.04	0.075	0.02
Radar	0.02	0.02	0.02	0.02	0.02	0.02	0.02

Table 3.1: Detection failure rates of vision sensors depending upon the specific weather condition at noon

Sensor	Sunset Weather Conditions						
	Clear	Cloudy	Wet	Wet Cloudy	Mid Rain	Hard Rain	Soft Rain
Ultrasonic	0.02	0.02	0.02	0.02	0.02	0.02	0.02
Lidar	0.02	0.02	0.05	0.05	0.15	0.25	0.1
SR Camera	0.04	0.03	0.04	0.03	0.04	0.05	0.03
LR Camera	0.04	0.03	0.04	0.03	0.04	0.05	0.03
Radar	0.02	0.02	0.02	0.02	0.02	0.02	0.02

Table 3.2: Detection failure rates of vision sensors depending upon the specific weather condition at sunset

3.5 Scene Builder

The components of the Scene Builder and Planner are shown in Fig. 3.10. The **active sensor modules** determine the **actors in vision** (Section 3.5.1) of the agent. Out of these, the Scene Builder first identifies the **actors on the path** (Section 3.5.2) ahead. The BEV positions and velocities of these is shared with the Sensor Module Selector (this is further detailed in Chapter 4). The first actor on the path is termed as the **front road user**. The **separation distance** (Section 3.5.3) and velocity of the front road user is then used by the planner to set control values.

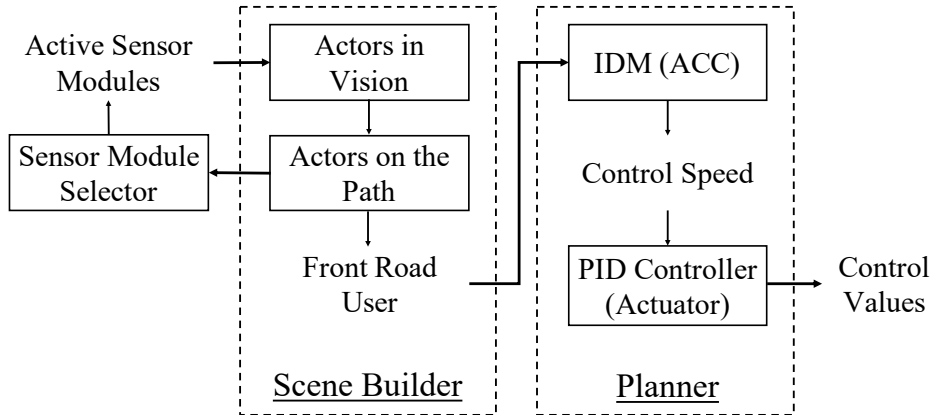


Figure 3.10: Scene Builder and Planner

3.5.1 Actors in Vision

To compute the actors in Vision of the agent we take the union of actors in vision for each active sensor. First, the grid positions for all actors located within the bird eye view are computed (Section 3.3). For a sensor to detect the presence of an actor, we check if the actor lies in the coverage range based on the grid values. If the actor is present then we assume that with (1-failure rate) the actor is detected (Section 3.4).

3.5.2 Actors on the Path

Once we have the actors in vision of the agent we then find the actors which lie on its path. The path is defined by the sequence of locations it plans to traverse and a fixed margin on either side of those locations (Fig. 3.11). For our setup we consider a distance of 0.25m on left and right of the agent and observe a distance of at most 150m in front. The locations planned for the agent are shown by a green line. The path is shown by the orange lines. The ‘Actors in Vision’ which share some overlap with the path are the ‘Actors on the Path’. For example in the Fig. 3.11 Vehicle-3 and Pedestrian-2 are on the path of the agent. As the actors are in motion, at every time step we compute which actors are on the path and generate a ranked list of actors based on which one lies on the path first. The first actor on the path is termed as the **front actor**.

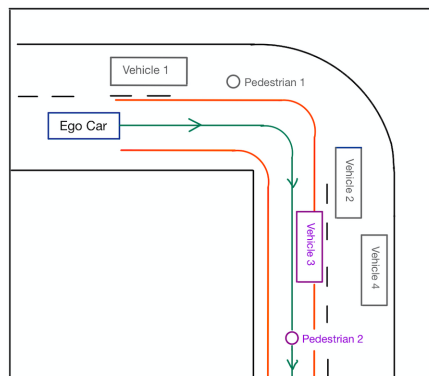


Figure 3.11: Actors which lie in the path of the ego car at a given instant shown in violet, other road users which are not at present in the path are shown in grey.

The Vehicle 3 qualifies as the front actor in the figure above.

3.5.3 Separation Distance

Separation distance is the minimum distance between the agent and the front actor. We consider bounding boxes for modelling vehicles and for pedestrians we use a cylindrical region of 1m radius centred at the pedestrian's position. To find the separation we consider the projection of these on the horizontal plane and consider a vehicle as a rectangle and a pedestrian as a circle.

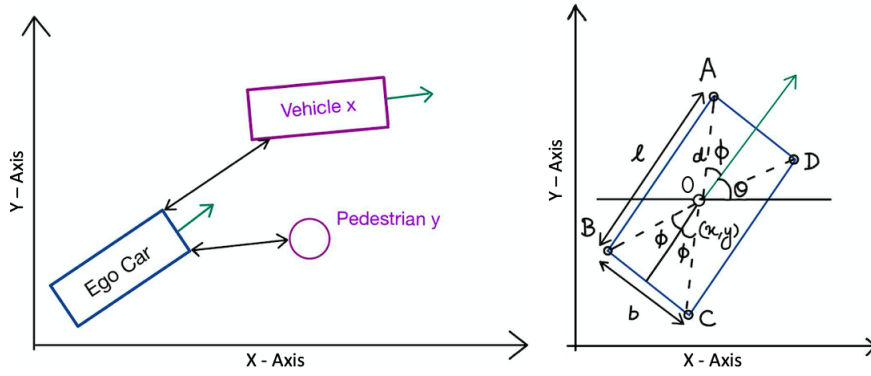


Figure 3.12: Separation distance calculation between the ego car and the front road user which can either be a vehicle (X) or a pedestrian (Y) shown in violet colour

Front Actor is a Vehicle

We can find the vertices of the rectangle modelling a vehicle as shown in Fig. 3.12. Consider a vehicle centred at point O with coordinates (x,y), having length l, breadth b and making an angle of θ with the X-Axis.

$$\phi = \tan^{-1}(b/l), \quad d = \sqrt{l^2 + b^2}$$

$$A = (x + (d/2) \times \cos(\theta + \phi), y + (d/2) \times \sin(\theta + \phi))$$

$$B = (x - (d/2) \times \cos(\theta - \phi), y - (d/2) \times \sin(\theta - \phi))$$

$$C = (x - (d/2) \times \cos(\theta + \phi), y - (d/2) \times \sin(\theta + \phi))$$

$$D = (x + (d/2) \times \cos(\theta - \phi), y + (d/2) \times \sin(\theta - \phi))$$

Once we have the vertices, we also know the edges of the rectangles representing

the ego vehicle and the front vehicle. The distances between these two rectangles cant then be found by taking the minimum distance between every edge of the rectangle for the agent with every edge of the rectangle for the front vehicle.

Front Actor is a Pedestrian

As shown in the previous part we first find the rectangle for the agent. Then we find the separation distance of the rectangle from the circumference of the circle representing the pedestrian. If the circle has radius r and is centred at (px, py) then the separation distance equals r less than the minimum of all distances from every edge of the rectangle to (px, py) .

3.6 Planner

The agent follows a predefined path, as discussed in Section 5.2.1. However, the control speed is computed using Intelligent Driver Model (IDM) for Adaptive Cruise Control (ACC, Section 2.3.3). The computed control speed is then achieved through the actions determined by the PID Controller (Section 2.3.3).

The control speed of the agent is updated with an update frequency. To ensure that the agent moves smoothly along it's predefined path we compute the control speed based on a target location in the path ahead. This target location is set at a fixed distance ' s ' farther than the location of the agent. ' s ' is chosen such that it is just a little greater than the distance which can be covered at the set speed. The lower limit ensures that the agent is provided the right direction with every update and for preventing the car to halt and wait for the subsequent updates. Having a larger ' s ' on the other hand would lead to deviating from the predefined path in case of curved paths.

The initial speed of the agent ‘ u ’ is already known and the target location is at a distance ‘ s ’. The IDM determines the acceleration ‘ a ’ for the agent by using the velocity of the front actor and it’s separation distance (as computed in Section 3.5.3). If no front actor was found by the Active Sensors, the IDM assumes a stationary front actor located infinite distance away and then computes the acceleration. We then use the formulas of motion for constant acceleration to compute the control speed ‘ v ’ as:

$$v = \sqrt{2as + u^2}$$

The target location and control speed are then supplied to the PID control to find the appropriate control ego vehicle control.

Chapter 4

Reinforcement Learning

4.1 Overview

To achieve our goal of intelligent sensor module selection (Section 1.2) the autonomous agent must be able to navigate **safely** and **efficiently**:

1. Safely: Avoid collisions with road users
2. Efficiently: Optimal sensor module selection

In the case of maximal sensor selection, all the sensor modules are enabled to ensure maximum possible safety with the available sensors. However, it is not an efficient approach, as indicated in Chapter 1. To have an ‘optimal’ sensor selection we use RL to find an optimal policy. LCRL helps in achieving this policy in an efficient and effective manner as shall be demonstrated by experiments in Chapter 5. This chapter focuses on two objectives: 1) Modelling the problem as an RL problem (Section 4.2) and 2) Defining the Logical Constraints which enhances the process of learning an optimal policy (Section 4.3).

4.2 RL Modelling

To achieve an optimal sensor module selection, we place an RL agent inside the sensor module selector of the autonomous agent (Section 3.1). As described in Section 2.2.1, we can model our RL setup as a finite MDP. To specify the MDP, we define the state space, action space and reward structure.

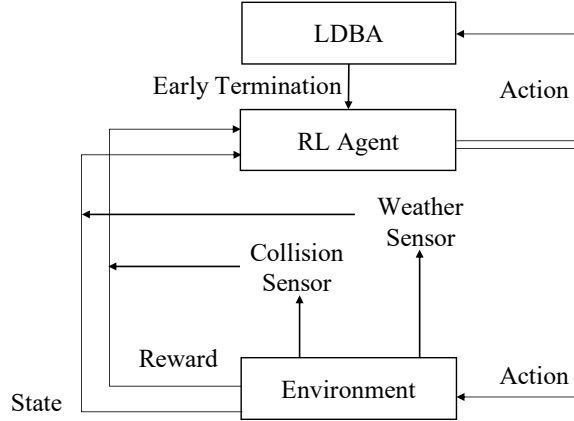


Figure 4.1: Interactions of the RL Agent with the environment and the LDBA

The **action** of the RL Agent is to determine the active sensor module set (on behalf of the sensor module selector). As was explained in Section 3.1, this selection affects the scene builder and consequently the perception of the environment. The *perceived state* of the environment is used by the RL Agent to make decisions. The state includes weather information, kinematics of the ego vehicle, the perceived location and velocity of near-by road users (as determined via Section 3.5.2). The actions also update the current state of the LDBA, where reaching a sink state in an LDBA leads to an early termination of the RL learning episode. If the ego vehicle detects a collision (via the collision sensor) then the RL agent is given a high negative **reward** and the learning episode is terminated. Sensor selection also yields certain rewards explained in Section 4.2.3. Fig. 4.1 shows the interactions of the RL agent with the environment and the LDBA.

4.2.1 States

Based on our understanding of the factors that are relevant for the RL agent in sensor module selection we came up with the following discrete state space $\mathcal{S}$:

$$\mathcal{S} = \langle w, p, v_0, a_0, x_1, y_1, v_1, x_2, y_2, v_2, x_3, y_3, v_3 \rangle$$

Weather index: $w \in \{1, \dots, 14\}$

Previous action: $p \in \{0, \dots, 31\}$

Longitudinal grid position: $x_i \in \{0, \dots, 19\}$ for $i \in \{1, 2, 3\}$

Lateral grid position: $y_i \in \{0, \dots, 9\}$ for $i \in \{1, 2, 3\}$

Discrete velocities: $v_i \in \{0, 1, 2, 3, 4\}$ for $i \in \{0, 1, 2, 3\}$

Discrete accelerations: $a_0 \in \{0, 1, 2, 3, 4\}$

In the above notation the variables corresponding to $i = 0$ refer to the ego vehicle. The first three actors on the path (computation demonstrated in Section 3.5.2) are denoted by $i = 1, i = 2, i = 3$.

Remark 4.2.1. *The size of the state space comes out to be $14 \times 32 \times 5 \times 5 \times (20 \times 10 \times 5)^3 = 1.12 \times 10^{13}$ which already makes the problem hard for simple Q Learning based approaches. If we increase the number of actors on the path, in the state space or introduce the acceleration values of the actors, then the size of the state space will increase further. Hence, to adopt a general approach of tackling enormous sized state spaces we use Deep RL based approaches.*

4.2.2 Actions

The selection of the active sensor modules performed by the RL Agent are the **actions** for our MDP. The action space consists of 32 sensor module configurations consisting of 5 sensor modules where each can be enabled or disabled. In our setup, the agent performs actions at a frequency of 2Hz.

4.2.3 Rewards

The RL Agent receives rewards based on sensor selection and occurrence of a collision event. The reward function is defined as follows:

$$\text{Reward} = 1 - (\text{Active Sensor Cost} + \text{Collision Cost} + \text{Switching Cost})$$

The **Active Sensor Cost** is the sum of the costs of all the active sensors modules. If no sensor module is active, then this cost is zero. We classify the sensor modules into two categories, low cost and high cost.

low-cost: {Ultrasonic, Lidar, Camera-2, Radar}, high-cost: {Camera-1}

We assign a cost of 0.1 for every active low-cost sensor module and a high cost of 0.2 for every active high-cost sensor module.

Remark 4.2.2. *The low cost and high cost is not based on empirical observations, it is only meant to showcase how different sensor modules can be weighted in our method. We consider Camera-1 to be high cost as it covers a larger area closer to the agent.*

The **Collision Cost** varies across experiments but is essentially set to a multiple of the episode length for a collision event and set to zero otherwise. Invariably, the episode terminates in case of a collision.

The **Switching Cost** is incurred whenever the Sensor Module Selector changes the set of active sensor modules. We fix this cost to 0.1.

4.3 Safety Properties and LCRL

As explained in Section 2.2.3, logical constraints (LTL properties) can be specified over MDP by conceptualising a product MDP. A product MDP is constructed by synchronizing the states of an LDBA (representing an LTL property) with the execution (sequence of actions) of an MDP. The labels associated with the actions are sufficient to determine the transitions for the LDBA. Whenever the LDBA reaches a sink state (state from where it is impossible to reach an accepting state) the execution of the MDP (RL Agent) is terminated. We next describe the LTL based safety properties we used for ensuring safe and efficient driving.

The safety properties serve a dual role of acting as a Guard and a Guide:

- 1) **Guard Property:** Collision prevention
- 2) **Guide Property:** Faster learning of the optimal policy by reducing the search space.

To achieve these two objectives, we next describe the formulation of LTL properties and the corresponding LDBAs. We then represent the Guard Property by ϕ , the Guide Property by ψ and the combined safety property by $\varphi = \phi \wedge \psi$.

For each safety property φ we construct the LDBA using an online tool Spot 2.0 [13]. We then encoded the output automaton into Graphviz [15] - an open source library for generating the automaton diagrams (Figs. 4.2 to 4.4).

4.3.1 Guard Property: Collision Prevention

For Collision Prevention coming up with an LTL property is straight forward:

$$\phi = \Box \neg \text{Collision} \tag{4.1}$$

4.3.2 Guide Property: Controlled Switching

For ensuring Controlled Switching we had to experiment with multiple properties. We define switching as any change in active sensor configuration. This means that if any sensor module which was previously ‘On’ is turned ‘Off’ or what was earlier ‘Off’ is turned ‘On’ it will be considered as a switch. Formally a **Switch** (S) is defined as:

$$S_i = (\text{On}_i \wedge \bigcirc \text{Off}_i) \vee (\text{Off}_i \wedge \bigcirc \text{On}_i) \quad (4.2)$$

$$S = \bigvee_{i=1}^5 S_i \quad (4.3)$$

Remark 4.3.1. In the above equations, we use $i=\{1,2,3,4,5\}$ to denote the sensor modules corresponding to $\{\text{Ultrasonic}, \text{Lidar}, \text{Camera-1}, \text{Camera-2}, \text{Radar}\}$.

We experimented with four different guide properties (Repeated Switching Prevention 1-4) and constructed an LDBA for each case by conjuncting it with the guard property ϕ (Eq. (4.1)). The experimental results for each are shown in Sections 5.3.4 to 5.3.7. We next list the four attempts at finding a suitable guard property.

Repeated Switching Prevention - 1

Using the definition of switching from Eq. (4.3), we formulated a property (Eq. (4.4)) to have persistent sensor module configuration for two consecutive states.

$$\psi = \Box(S \implies \neg(\bigcirc S \vee \bigcirc \bigcirc S)) \quad (4.4)$$

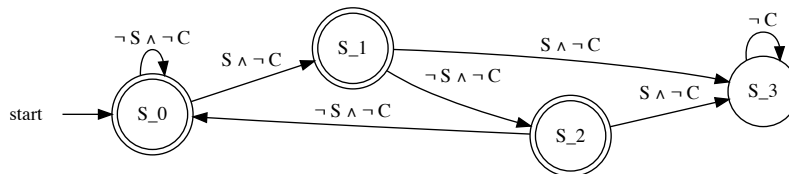


Figure 4.2: LDBA-1

We tested the impact of this LTL property in Experiment 2 (Section 5.3.4), it seemed to expedite the learning process (when compared with not using any guide property). However, it converged to a sub-optimal policy making it efficient but not effective. Hence, we tried relaxing the constraint in the next LTL property.

Repeated Switching Prevention - 2

In the second attempt, again using the definition of switching from Eq. (4.3), we formulated a property Eq. (4.5) to prevent four consecutive switching of sensor module configurations. The aim here was to relax the constraint of the Eq. (4.4) by allowing up to three consecutive switches.

$$\psi = \Box(S \implies \neg(\bigcirc S \wedge \bigcirc \bigcirc S \wedge \bigcirc \bigcirc \bigcirc S)) \quad (4.5)$$

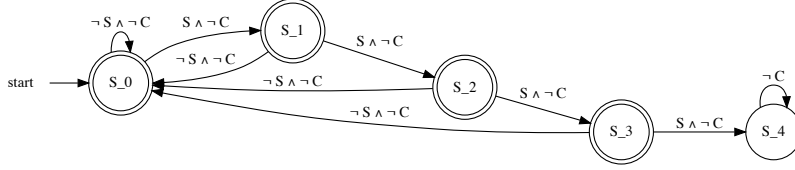


Figure 4.3: LDBA-2

The performance impact of this guide property as measured in Experiment 3 (Section 5.3.5) was still not satisfactory and seemed to prune essential search space. Hence, we made another attempt at relaxing the guide property further.

Repeated Switching Prevention - 3

In the third attempt, again using the definition of switching from Eq. (4.3), we extended the property Eq. (4.5) to relax the constraint further to allow for up to five consecutive switches of sensor module configuration.

$$\psi = \Box(S \implies \neg(\bigcirc(S) \wedge \bigcirc \bigcirc(S) \wedge \bigcirc \bigcirc \bigcirc(S) \wedge \bigcirc \bigcirc \bigcirc \bigcirc(S) \wedge \bigcirc \bigcirc \bigcirc \bigcirc \bigcirc(S))) \quad (4.6)$$

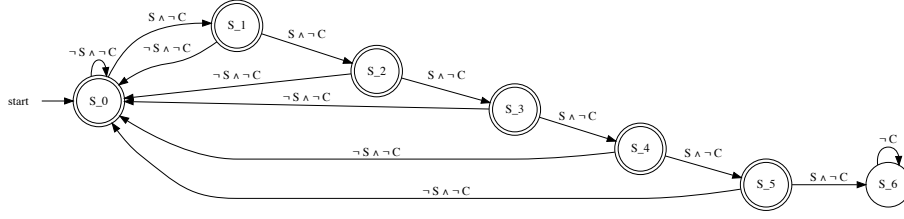


Figure 4.4: LDBA-3

Experiment 4 (Section 5.3.6) was conducted for this guide property. The training took much longer than with Eq. (4.5) but during the training phase, the agent was able to consistently achieve near to optimal maximum reward unlike with Eq. (4.6). This clearly suggested that relaxing the constraint was increasing the search space and allowing the agent to steadily find an optimal policy at the price of prolonged training. Since, relaxing the LTL was diminishing the utility of LCRL based approach, in the next attempt we refined our notion of switching.

Repeated Switching Prevention - 4

As a fourth attempt, rather than designing a constraint based on switching of the entire sensor module configuration, we extend the notion of individual switching as defined in Eq. (4.2) to three consecutive switches for every sensor module as stated formally by Eq. (4.7) and Eq. (4.8).

$$\psi = \bigwedge_{i=1}^5 \psi_i \quad (4.7)$$

$$\psi_i = \Box(S_i \implies \neg(\bigcirc S_i \wedge \bigcirc \bigcirc S_i)) \quad (4.8)$$

To make the environment truly memory-less, rather than storing the previous sensor configurations in the environment we make use of the automaton to store that information. This implies the transitions become based on the current configuration

and not on switch. Then an equivalent representation of Eq. (4.7) is Eq. (4.9).

$$\psi_i = \psi_{i,1} \wedge \psi_{i,2} \quad (4.9)$$

$$\psi_{i,1} = \Box(\text{On}_i \implies \neg(\bigcirc(\text{Off}_i) \wedge \bigcirc \bigcirc (\text{On}_i) \wedge \bigcirc \bigcirc \bigcirc (\text{Off}_i))) \quad (4.10)$$

$$\psi_{i,2} = \Box(\text{Off}_i \implies \neg(\bigcirc(\text{On}_i) \wedge \bigcirc \bigcirc (\text{Off}_i) \wedge \bigcirc \bigcirc \bigcirc (\text{On}_i))) \quad (4.11)$$

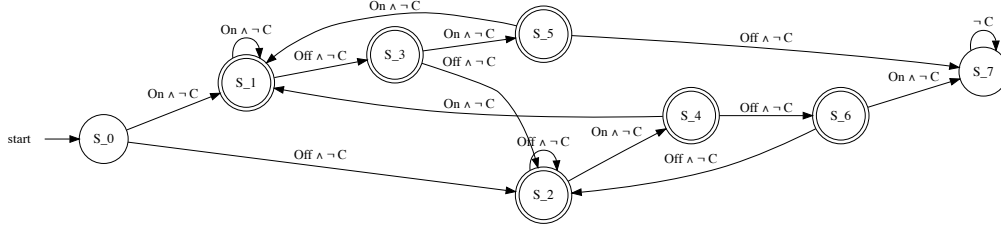


Figure 4.5: LDBA-4

The Experiment 5 (Section 5.3.7) confirmed that the guide property based on Eq. (4.7) was efficient and effective in leading the agent to learn an optimal policy. Experiment 14 (Section 5.3.16) further confirmed that the guide property was also able to adapt to diverse settings. The experiment setup and all the experiments are explained in detail in the next chapter.

Chapter 5

Experiments and Results

5.1 System Setup

We used CARLA (0.9.9) [12] which is an open source simulator for doing research in autonomous driving. The simulator has a client-server architecture. CARLA provides Python APIs [2] which were used to interact with the server. We used Python 3.7 to create client-side simulation scripts. CARLA requires sophisticated hardware (advanced GPU, CPU and RAM) which is hard to get even with high-end systems. Hence we had to resort to using two machines (inter-connected on LAN) with advanced GPUs; the CARLA server was set up on a desktop and a laptop was used to run the CARLA client. The configuration details are as follows:

Desktop: AMD Ryzen 5 3600 6-Core Processor, GeForce RTX 2070 Super GPU (8GB), 16GB RAM

Laptop: Intel Core i7-7700 HQ CPU @ 2.80GHz, GeForce GTX 1050 (4GB), 32 GB RAM

For employing off-the-shelf Reinforcement Learning Algorithms (PPO and TRPO), we used Tensorforce [28]. Tensorforce is an open source deep reinforcement learning framework, built on top of tensorflow [11]. It provides separation of RL algorithm

from application. For using Tensorforce for a specific application, we only need to define the agent and environment in the required format. This then enables to plug-in various deep RL algorithms and fine tune the hyper parameters.

5.2 Simulation Scenario

We simulate the agent and environment (described in Chapter 3) in CARLA. The simulation consists of a town (the environment) in which road users (pedestrians and vehicles) are placed and weather is set. The town includes the map, terrain, lanes and all the objects in the simulation world. CARLA provides multiple towns, out of which we picked Town 5 as it has a large map and a possibility to simulate different ODDs. The top view of the map is shown in Fig. 5.1 with a zoomed in version on the right side of the figure.



Figure 5.1: Map of CARLA's Town 5 which was used for running the simulations

Pedestrians are controlled by CARLA's WalkerAIController [2] and the vehicles are set to autopilot mode. For simulating the autonomous agent (ego vehicle) we use a white BMW Grandtourer (Fig. 5.2) with a set speed of 11m/s.



Figure 5.2: Sample image from the CARLA simulator, showing the agent as a white BMW Grandtourer and other road users in Town 5

5.2.1 Ego Vehicle Path

As explained in Section 3.1, for our setup we require the autonomous agent to follow a predefined path. To define this path, we set the ego car to drive across the entire city in autopilot mode. During this run the vehicle pose is recorded for the entire path. The recorded locations are plotted in Fig. 5.3. This path is then used in all the experiments reported in this chapter.

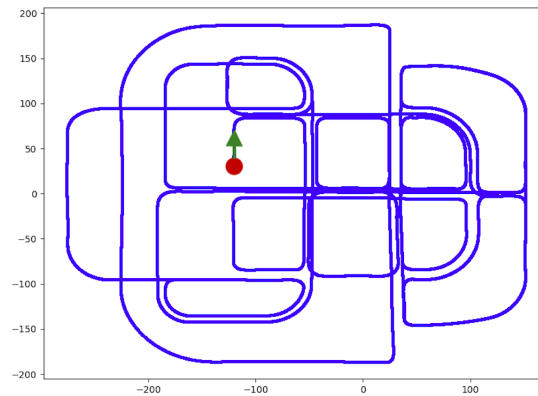


Figure 5.3: Recorded path of the ego vehicle in Town-5 (shown in blue), during it's run in the autopilot mode. The red circle shows the starting position of the ego vehicle and the initial direction of motion is marked by the green arrow.

5.2.2 Simulation of Experiments

Once we have recorded the path for the ego vehicle we proceed with our simulation experiments. Each experiment consists of a number of episodes. Episodes are of fixed maximum lengths. The ego car is placed at a point on the recorded path and other road users are placed around the path that will be followed by the ego car. The ego car runs till the episode completes or it collides with a road user. The sensor module selector operates at 2Hz i.e, at every half a second it determines the active sensor module set. Based on this set the scene builder creates a perceived scene of the environment (Section 3.5) and the ACC then sets the control of the ego car (e.g, line 23, Listing 5.1) as explained in Section 3.6.

The scene builder also shares the perceived scene with the sensor module selector. Based on this scene, the sensor module selector infers the state (e.g, line 3, Listing 5.1) and supplies it to the RL Agent (Section 4.2). The agent determines the action to be taken (of active sensor module selection e.g, lines 5-7, Listing 5.1). The action yields certain reward (e.g, line 4, Listing 5.1) and these rewards are accumulate over the length of the episode. The total reward of the episodes is then used to evaluate the performance of the RL agent during that episode.

Demo Video Recordings

The simulation explained above, is showcased by two training experiment videos:

Experiment 5 training video: [53]

Experiment 10 training video : [52]

The details about the settings for each Experiment (5 and 10 mentioned above) will be explained in Section 5.3. The video links show the simulation and the console output as the agent learns to find a sensor module selection policy. Below we explain the outputs shown at any given instant in the demo video (or the training script):

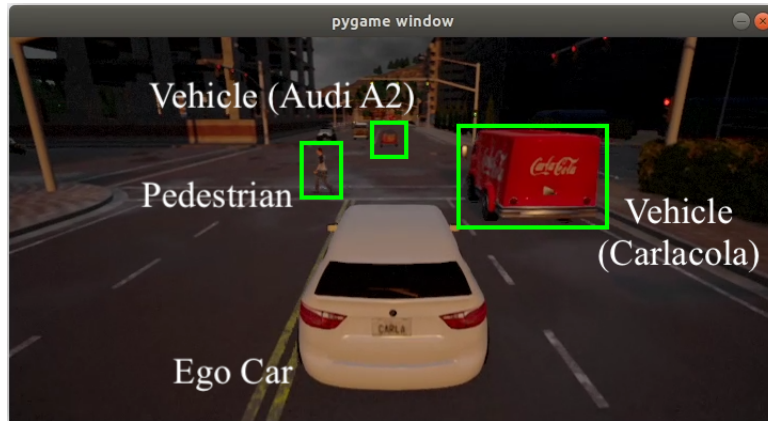


Figure 5.4: Simulation output showing the ego car and the environment during one of the training runs. The green boxes mark the road users near the ego car. The console output for this instance is shown in Listing 5.1

```

1 Episode Number : 34
2 Time (secs)    : 46.5
3 New State      : [[14, 11, 0, 4, 1, 5, 1, 1, 5, 0, 6, 5, 2]]
4 Reward        : 0.7 #For the previous action
5 Action Taken  : 11
6 Binary Action  : 11010
7 #Sensor Modules to Enable: {Ultrasonic, Lidar, Camera-2}
8 #Sensor Modules to Disable: {Camera-1, Radar}
9 Ego Location   : (-148.6, 2.5, 0.1)
10 Ego Rotation  : (-0.6, 1.1, 0.0)
11 Ego Speed     : 0.0 m/s
12 Town Distance : 6320.6 m
13 Weather       : SoftRainSunset
14 Active Sensors : [Ultrasonic, LiDar, Camera-2]
15 Failure Rates  : [0.02, 0.1, 0.03]
16 Front Actor(FA): Actor(id=7882, type=vehicle.carlamotors.
    carlacola)
17 FA Location    : (-142.1, 6.4, -0.0)
18 FA Speed       : 0.0 m/s
19 Approach Speed : 0.0 m/s
20 Separation     : 2.2 m
21 FA Grid        : (5, 1)
22 Detected Actors: {Actor(id=7882, type=vehicle.carlamotors.
    carlacola):[Ultrasonic, LiDar], Actor(id=7902, type=walker
    .pedestrian.0002): [Ultrasonic, LiDar], Actor(id=7849,
    type=vehicle.audi.a2): [Camera-2]}
23 Ego Control    : VehicleControl(throttle=0.000000, steer
    =0.006722, brake=0.052077, hand_brake=False, reverse=False
    , manual_gear_shift=False, gear=0)

```

Listing 5.1: Console output during one of the training runs corresponding to simulation output shown in Fig. 5.4

5.3 Experiments

Our objective is to provide the autonomous agent a useful LTL property, a suitable training setting and appropriate rewarding, so that the sensor module selector learns to perform optimal selection in diverse testing settings.

5.3.1 Training

To achieve this, we design fourteen experiments with the following objectives:

- Experiments 1-5: To find a ‘useful’ LTL property (i.e, a property which is necessary, efficient and effective in enabling the agent to find an optimal policy)
- Experiments 6-7: To check if the same LTL property can be useful for training in a different setting
- Experiments 8-13: To find a suitable training setting such that the selector is able to perform optimally in diverse testing settings
- Experiment 14: Fine tuning rewarding policy

Exp	#Start	LTL	Coll.	Time	#Vehicle	Vision	Signal	Alg.
1	1	0	60	60	50	50	Green	PPO
2	1	1	60	60	50	50	Green	PPO
3	1	2	60	60	50	50	Green	PPO
4	1	3	60	60	50	50	Green	PPO
5	1	4	60	60	50	50	Green	PPO
6	1	0	60	60	0	50	Green	PPO
7	1	4	60	60	0	50	Green	PPO
8	1	4	60	60	50	100	Green	PPO
9	5	4	60	60	50	50	Dynamic	PPO
10	5	4	30	30	50	50	Dynamic	PPO
11	5	0	30	30	50	50	Dynamic	PPO
12	∞	4	60	30	50	50	Dynamic	PPO
13	1	4	60	60	50	50	Green	TRPO
14	1	4	120	60	50	50	Green	PPO

Table 5.1: List of 14 experiments, each row represents the settings of an experiment, the columns indicate the parameters which are varied across experiments

The parameters which are varied in Table 5.1 are explained in detail below:

- **#Start** indicates the number of starting positions available to the agent across episodes. 1 implies that the agent always starts from a fixed location, 5 implies 5 fixed locations and ∞ implies that it is free to start from any location
- **LTL** refers to the index of the LTL property (as per those listed in Section 4.3.2) used for training. 1, 2, 3, 4 correspond to Eq. (4.4), Eq. (4.5), Eq. (4.6), Eq. (4.7). Simple RL (no LTL) property is denoted by 0
- **Coll.** refers to the collision cost (Section 4.2.3). We experiment with Coll. set to either the max episode length (in seconds) or double that value
- **Time** is the maximum episode length (in seconds)
- **#Vehicle** indicates the maximum number of vehicles which can be found on or near the path followed by the agent
- **Vision** refers to the distance (in metres) up-to which the agent limits it's observation range
- **Signal** is used to denote the state of the traffic lights on the path of the agent. 'Green' represents the setting where the agent encounters only green traffic lights on it's path. However, that does not imply that the agent need not stop. Presence of road users can require the agent to halt from time to time. 'Dynamic' represents the setting in which traffic lights change dynamically as in the real-world
- **Alg.** is the RL algorithm we employ for training. We experiment with Proximal Policy Optimization (PPO, Section 2.2.2) and Trust Region Policy Optimization (TRPO, Section 2.2.2)

The other factors remain the same across all these experiments: the number of pedestrians in the town is set to 70, the weather changes dynamically and the episode terminates whenever a collision happens or the ego vehicle completes the

maximum episode length. For training we report the minimum, the average, and the maximum episode reward calculated in a window size of 10 episodes. We say that the **training converges** when these three metrics become equal.

Since we do not have a ground truth about the optimal values, we estimate an **Optimal (Opt.) selection reward** using the reward that will be achieved by keeping a minimum sensor module selection for the entire episode length. For Experiment 6 and 7 (no vehicles present) we observed that an average of two low cost sensor modules (as defined in Section 4.2.3) should suffice. For all other experiments (where up to 50 vehicles can be present) we observed that an average of three sensor modules would be optimal.

We consider an episode to have a **high reward** if the reward value of the episode is close (within 5 points) to the optimal selection reward or greater than it. The **Maximal Sensor Selection (MSS) reward** is the reward received by the agent in an episode when it keeps all the sensors on.



Figure 5.5: The relative positioning of the Maximal Selection Reward and Optimal Selection Reward. High reward region marked by the red arrow.

5.3.2 Testing

To evaluate the learned policy, we performed tests in two types of settings: **Test-S1** and **Test-S2**. The results of the tests conducted for each experiment are presented in Table 5.2, the notation used is as follows:

- **#Ep.:** Number of testing episodes

- **Time Avg.:** Average test episode time
- **Time Opt.:** Optimal episode time or the maximum possible episode length
- **Reward**
 - **Avg.:** Average test episode reward
 - **MSS:** Reward in case of Maximal Sensor Selection
 - **Opt.:** Optimal reward in the given setting
 - **%High:** Percentage of test episodes with high reward values
- **%Coll.:** Percentage of test episodes in which the agent encountered a collision

Exp.	Test	#Ep.	Time		Reward				%Coll.
			Avg.	Opt.	Avg.	MSS	Opt.	%High	
1	S1	18	60	60	74	48	84	17	0
2	S1	17	60	60	72	48	84	0	0
3	S1	17	52	60	73	48	84	65	35
4	S1	23	50	60	45	48	84	57	43
5	S1	17	60	60	84	48	84	100	0
5	S2	22	56	60	71	48	84	86	14
6	S1	15	57	60	74	48	96	53	20
7	S1	24	60	60	105	48	96	100	0
7	S2	22	53	60	82	48	96	77	23
8	S1	27	49	60	48	48	84	48	52
9	S1	26	57	60	72	48	84	85	15
10	S1	72	29	30	40	24	42	93	7
11	S1	19	29	30	28	24	42	0	5
12	S1	20	24	30	34	24	42	70	30
13	S1	23	53	60	31	48	84	0	26
14	S1	12	60	60	84	48	84	100	0
14	S2	50	60	60	84	48	84	100	0

Table 5.2: Testing results for settings S-1 and S-2 reported for Experiments 1-14. Test results with 100% high reward and 0% collision are marked in green. The test with 90-100% high reward and less than 10% collision is marked in yellow.

In **Test-S1** of an experiment, we place the agent in the same setting as the training phase of that experiment and run multiple test episodes. It is important to note that “same setting” implies that only the parameters mentioned in Table 5.1 remain the same as that of the training phase. The environment however can be arbitrary,

i.e, the position of road users, their trajectories of motion, weather condition are not fixed by the experiment setting. For example, in Test-S1 for Experiment 1 the agent will start from a fixed position, would incur a collision cost of 60, face at most 50 vehicles and 70 pedestrians, the vision will be limited to 50m and all signals will be green (as per Table 5.1). If all the test-episodes return high reward values and there are no collisions then we proceed to Test-2 where we relax the settings.

In **Test-S2**, the agent is placed at any random position in the recorded path (Fig. 5.3) and the episode is run for 60 secs. The traffic signals change dynamically, the number of vehicles and pedestrians are kept the same as the training environment.

Remark 5.3.1. *Running the CARLA simulations even on the client side was GPU and CPU intensive process. It led to severe overheating of the laptop, when simulations were run for long periods. Also, because of the ongoing COVID-19 pandemic, we were unable to arrange for better resources and thus we could run limited experiments. For each experiment, we ran it for varying number of episodes depending on the observed behaviour of the agent. We terminated the learning early when it was clear after enough episodes that the agent was not learning or the learning converged. In cases when the agent seemed to learn slowly, we ran the experiment in intervals and used check-points to save the progress of training and then resumed it later.*

Experiment Overview

While evaluating the results from training and testing of the 14 experiments we reflect upon the impact of the following factors on learning an optimal policy:

- LCRL v/s RL
- Changing LTL properties
- Configuring the environment settings (Vehicles, Start Positions)
- Changing the RL setup (State, Reward, Algorithm, Episode Length)

5.3.3 Experiment 1

For Experiments 1-5, the ego vehicle starts from a fixed location and all the traffic lights on it's way are set to green. There are 30-50 vehicles placed on and around the path of the ego vehicle and there are 70 pedestrians roaming in the town. The road users only within 50m are perceived by the ego vehicle. Collision cost is set equal to the episode length which is 60 seconds and PPO agent is used for learning.

In Experiment 1, we do not use any LTL property to train the agent. As can be seen from Table 5.3, the agent learns to perform better than maximal sensor selection but does not reach the optimal sensor selection. We can say this, as the Avg. Reward of Test-S1 comes out to be greater than MSS Reward but lower than the Opt. Reward. All episodes in Test-S1 completed 60 seconds as the Avg. Time comes out to be same as the Opt and the ego vehicle was able to achieve collision free driving.

Exp.	Test	#Ep.	Time		Reward				%Coll.
			Avg.	Opt.	Avg.	MSS	Opt.	%High	
1	S1	18	60	60	74	48	84	17	0

Table 5.3: Testing results

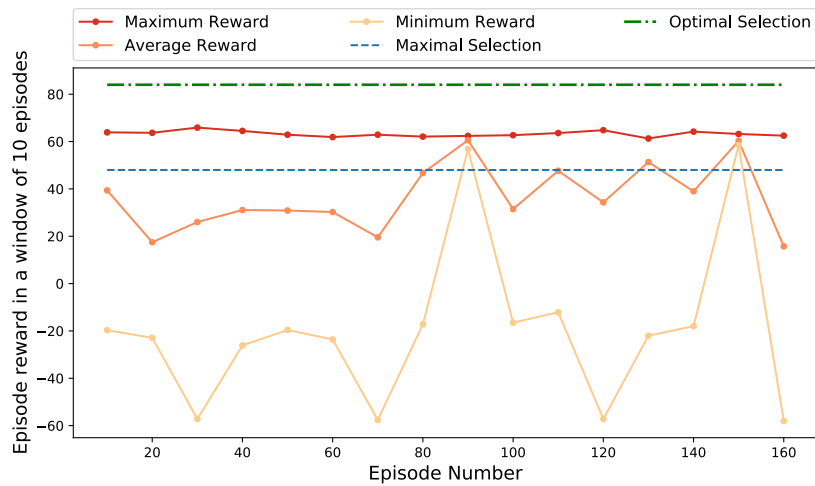


Figure 5.6: Training reward values for Experiment 1

However, as we can see from the average, minimum and maximum plots of

training (Fig. 5.6), the learning did not converge and perhaps it will take a long time for the training values to converge if at all. Fig. 5.7 shows the average, minimum and maximum training time for Experiment 1. The average time remains in the range 40-60 secs. We wish to reduce the time taken for the agent during training and learn a policy with average reward value closer to the optimal value. Hence, we introduce logical constraints to expedite the learning process.

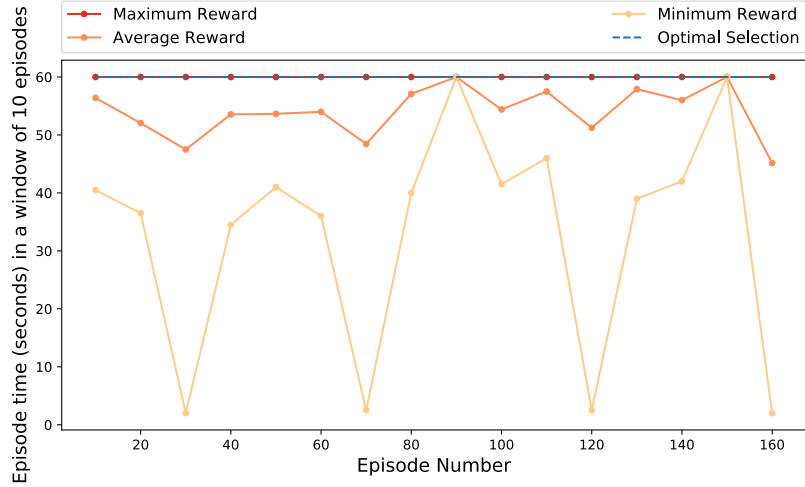


Figure 5.7: Training episode times for Experiment 1

5.3.4 Experiment 2

In Experiment 2, we try out the first LTL property as arrived at in Eq. (4.4).

$$\psi = \Box(S \implies \neg(\bigcirc S \vee \bigcirc \bigcirc S))$$

As we terminate the episode whenever the Automaton-1 (Fig. 4.2) reaches a sink state, initially the training reward values are close to zero. Then there is a steady increase in the average and maximum training reward as can be seen from Fig. 5.8. The maximum training reward saturates at 72.6 within 60 episodes, this is higher than the maximum reward achieved in Experiment 1 (which is close to 65).

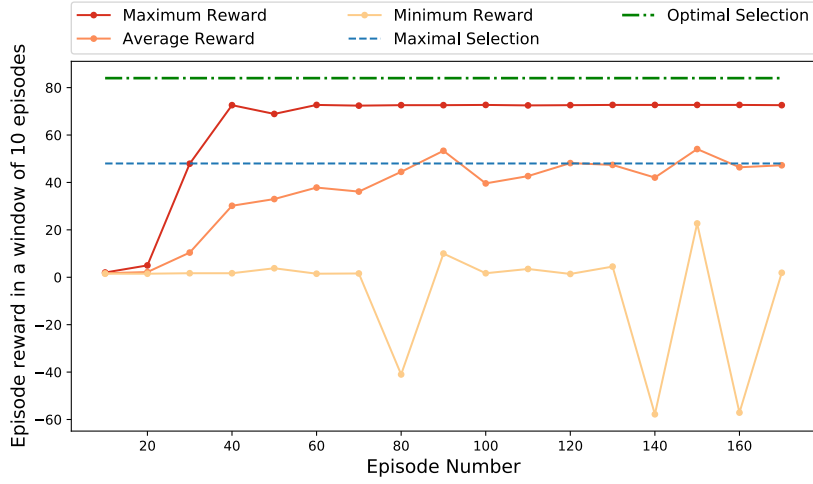


Figure 5.8: Training reward values for Experiment 2

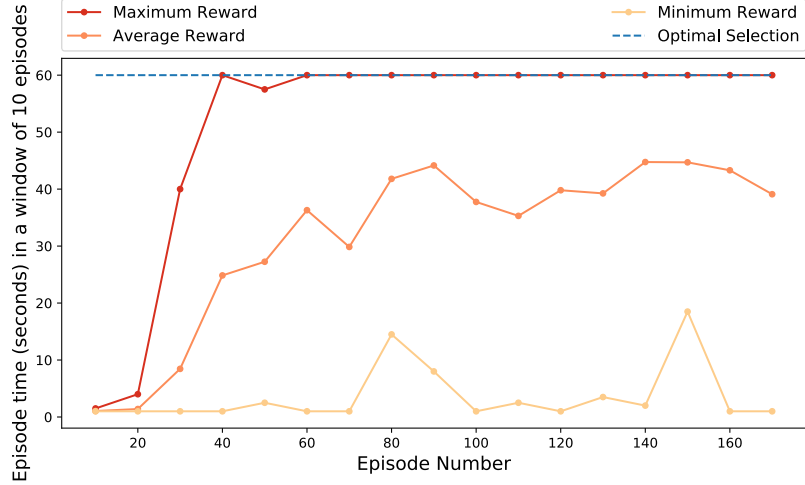


Figure 5.9: Training episode times for Experiment 2

It is also, important to note that the first 40 episodes are of significantly shorter lengths as compared to those in Experiment 1 this can be verified by comparing Fig. 5.7 and Fig. 5.9. This implies that the *learning for the agent using the LCRL constraint is way faster than simple RL based approach in terms of both the total episode length and also the number of episodes.*

Exp.	Test	#Ep.	Time		Reward				%Coll.
			Avg.	Opt.	Avg.	MSS	Opt.	%High	
2	S1	17	60	60	72	48	84	0	0

Table 5.4: Test results for Experiment 2

We observe from Table 5.4 that the test returned average test greater than MSS but still none of the episode rewards return high reward values. So we then try changing our LTL property and proceed with the next experiment.

5.3.5 Experiment 3

In Experiment 3, we try out the second LTL property as arrived at in Eq. (4.5). This aims to relax the constraint from our previous experiment as we believe it did not explore enough and the agent got stuck in a local optimum.

$$\psi = \Box(S \implies \neg(\bigcirc S \wedge \bigcirc \bigcirc S \wedge \bigcirc \bigcirc \bigcirc S))$$



Figure 5.10: Training reward values for Experiment 3

As can be seen from Fig. 5.10 the agent was faced with a larger search space. The agent took more than about 100 episodes to first encounter rewards close to maximal sensor selection. There was no persistent steady increase in training rewards and understandably the performance in testing was also poor (Table 5.5).

Exp.	Test	#Ep.	Time		Reward				%Coll.
			Avg.	Opt.	Avg.	MSS	Opt.	%High	
3	S1	17	52	60	73	48	84	65	35

Table 5.5: Test results for Experiment 3

Although the average testing reward was high but in testing there were several collisions in testing. The variance in rewards was very high as in 65% of the training episode we were able to achieve high reward but also at the same time in 35% other testing episodes the agent encountered a collision. Due to this high variance, we felt that the LTL property could still be pruning search space which might be essential for learning the optimal policy.

5.3.6 Experiment 4

In Experiment 4, we try out the third LTL property as arrived at in Eq. (4.6). This aims to relax the constraint even further from our previous two experiments.

$$\Box(S \Rightarrow \neg(\bigcirc(S) \wedge \bigcirc\bigcirc(S) \wedge \bigcirc\bigcirc\bigcirc(S) \wedge \bigcirc\bigcirc\bigcirc\bigcirc(S) \wedge \bigcirc\bigcirc\bigcirc\bigcirc\bigcirc(S)))$$

As shown in Fig. 5.11 the average and maximum training reward values increased steadily. However, even after 250 episodes it did not converge. Also, while testing the agent encountered collisions in 43% of the episodes (Table 5.6).

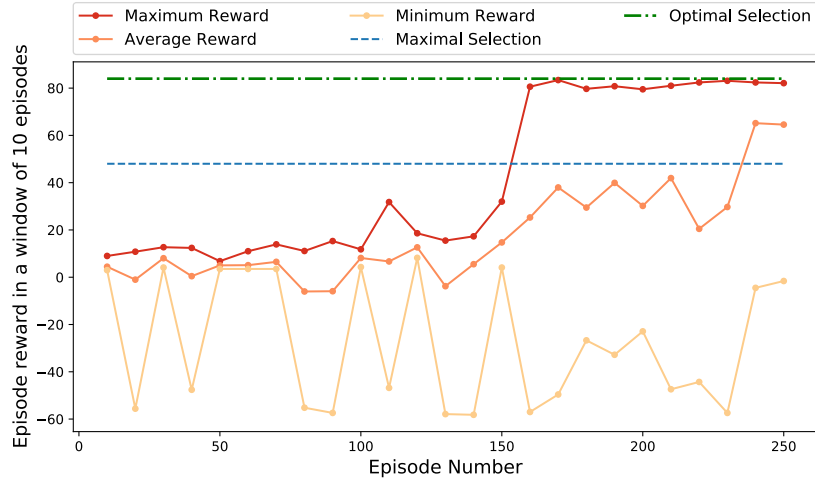


Figure 5.11: Training reward values for Experiment 4

Exp.	Test	#Ep.	Time		Reward				%Coll.
			Avg.	Opt.	Avg.	MSS	Opt.	%High	
4	S1	23	50	60	45	48	84	57	43

Table 5.6: Test results for Experiment 4

Observing Fig. 5.11, it seemed like we would have to run the training for several more episodes for the training to converge and to find an optimal policy. If we keep relaxing the logical constraint then we would essentially be losing on the benefits of LCRL and the problem will become intractable. So then we take a step back to rethink the kind of behaviour we want the agent to avoid and to come up with a better LTL policy in the next experiment.

5.3.7 Experiment 5

In Experiment 5, we try out the fourth LTL property as arrived at in Eq. (4.7).

$$\psi = \bigwedge_{i=1}^5 \psi_i : \quad \psi_i = \Box(S_i \implies \neg(\bigcirc S_i \wedge \bigcirc \bigcirc S_i))$$

This property, aims at placing constraints over repeated switching of individual sensor modules rather than the entire set. It is closer to our desired exploration pruning as we do not want to repeatedly enable/disable a particular sensor module in consecutive steps but still wish to allow switching of other sensor modules. From Fig. 5.12 we observe that this LTL property leads to convergence of training to optimal policy within 140 episodes. The average training reward reaches above the maximal selection within the first 40 episodes. Moreover, the maximum training reward also touches and saturates at the optimal policy within this time frame.

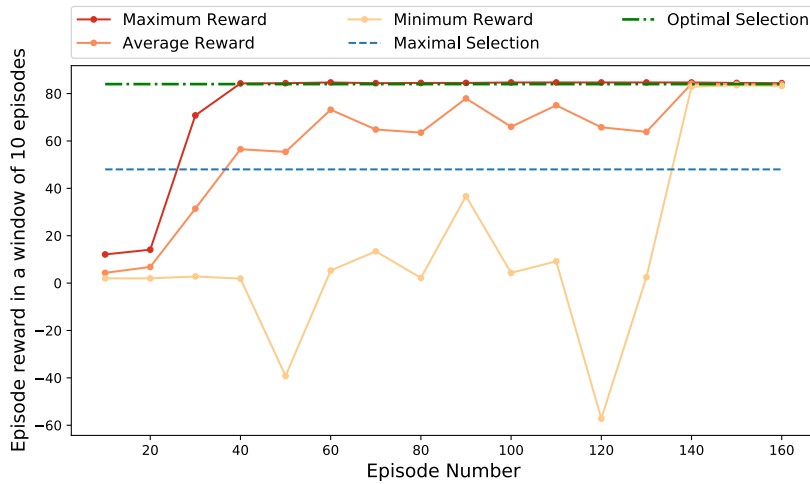


Figure 5.12: Training reward values for Experiment 5

Exp.	Test	#Ep.	Time		Reward				%Coll.
			Avg.	Opt.	Avg.	MSS	Opt.	%High	
5	S1	17	60	60	84	48	84	100	0
5	S2	22	56	60	71	48	84	86	14

Table 5.7: Test results for Experiment 5

The Test-S1 confirms that the agent has successfully learned an optimal policy (Table 5.7) as high reward was reached in 100% testing episodes with 0% collision. As the Test-S1 was successful we then move to Test-S2 which tests for scalability of the learned policy by placing the agent at a random position in the map. However, as seen from the second entry in Table 5.7 the agent faces collision in 14% of the test episodes.

Hence, this training setting is not letting the agent learn a policy which can be scaled to a diverse testing environment. We resume our search of an appropriate experiment setting for training in Experiments 8-13. However, first we wish to check if the LTL-4 property which enabled the agent to succeed with Test-S1 is applicable with different experiment settings in Experiment 6-7.

5.3.8 Experiment 6

In Experiments 6 and 7, we consider a no vehicle scenario (only pedestrians present). All the other settings remain the same as that of the first five experiments.

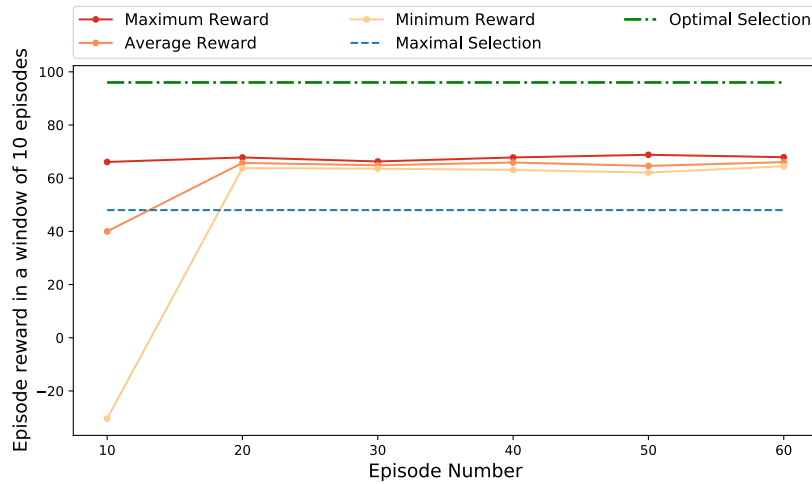


Figure 5.13: Training reward values for Experiment 6

In Experiment 6, we use simple RL based approach and observe from Fig. 5.13 that the agent quickly converges to a sub-optimal policy which is better than maximal sensor selection but still much lower than the optimal policy.

Exp.	Test	#Ep.	Time		Reward				%Coll.
			Avg.	Opt.	Avg.	MSS	Opt.	%High	
6	S1	15	57	60	74	48	96	53	20

Table 5.8: Test results for Experiment 6

The test values show high variance, as there are 20% collision and 53% episodes with high rewards. The high variance and convergence of training to a sub-optimal policy, indicates that simple RL would not allow finding an optimal policy even if the experiment is run for longer periods. Hence an LCRL based approach is essential even in this environment with relatively fewer road users.

5.3.9 Experiment 7

In this experiment we test if the fourth LTL property of Eq. (4.7) as shown to be useful in Experiment 5 is also useful here when applied in the no-vehicle scenario.



Figure 5.14: Training reward values for Experiment 7

As demonstrated by Fig. 5.14 the average training reward increases over time, and the maximum reward goes beyond the assumed optimal selection. Although

the training does not converge in 220 episodes, the average testing value exceeds the optimal selection. This implies that the agent was able to learn a policy which kept less than 2 low-cost sensors active on average and still achieved collision-free driving. Since it was successful with Test-S1 we then move to Test-S2.

Exp.	Test	#Ep.	Time		Reward				%Coll.
			Avg.	Opt.	Avg.	MSS	Opt.	%High	
7	S1	24	60	60	105	48	96	100	0
7	S2	22	53	60	82	48	96	77	23

Table 5.9: Test results for Experiment 7

As an aside, it is interesting to note that the training took longer in terms of the number of episodes and the total time taken to learn close to an optimal policy as compare with Experiment 5. The primary reason being that the environment is relatively sparse and collisions are less likely to occur, which means it takes longer for the ego vehicle to identify bad policies.

For Test-S2 we start the ego vehicle at a random starting position with 300 pedestrians roaming in the town. We consider this as a special case of S-2 where there are more number of pedestrians than the training environment. This is to provide a harder challenge to the agent (as vehicles are not present). It is a harder environment as compared to the 70 pedestrians in the training environment, as collisions are more likely with an increase in the number of vulnerable road users. From the second entry of Table 5.9 we observe that the ego vehicle collides with pedestrians in this challenging environment.

Similar to Experiment 5, in this experiment it learned an optimal policy for the settings of the experiments it was trained in, however it did not scale to a more challenging environment. In the subsequent experiments we try to find a suitable training environment such that the learning scales to diverse testing environments.

5.3.10 Experiment 8

In this experiment we increase the perception range of the ego vehicle from 50 to 100 to test if the hindrance in learning was constraining the coverage area of the front vision sensors. From Fig. 5.15 it is clear that constraining the vision range was only helping the learning process. Even after training the agent for 250 episodes, the test average values do not surpass the maximal sensor selection. Table 5.10 shows that there are collisions in 52% of the testing episodes, which implies that the learned policy is not useful.

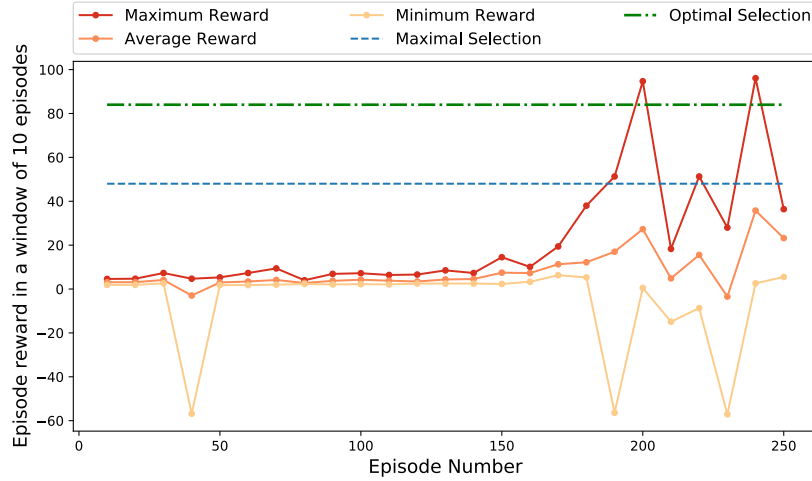


Figure 5.15: Training reward values for Experiment 8

Exp.	Test	#Ep.	Time		Reward				%Coll.
			Avg.	Opt.	Avg.	MSS	Opt.	%High	
8	S1	27	49	60	48	48	84	48	52

Table 5.10: Test results for Experiment 8

When the range is increased from 50 to 100 in this experiment there is a drastic increase in the size of the search space. This is the reason for such a poor performance in this experiment. To solve this issue we would probably need to prune the state space or introduce additional constraints. However, we do not explore that direction and rather continue our search of finding a suitable training environment.

5.3.11 Experiment 9

For Experiments 9, 10 and 11 we select 5 random positions on the map and train the agent starting from these locations. The attempt is to provide a diverse set of training environments. The traffic signals are set to change dynamically and the vision range is reset to 50m. The training is stopped when the agent is waiting at a red light, as in a stationary car changing sensor modules won't benefit the learning.

In Experiment 9, the maximum episode length is maintained at 60 seconds, as in the experiments before. We observe from Fig. 5.16 that the average training reward barely increases from 0 even after training it for more than 250 episodes. A closer look at the minimum and maximum training rewards after 100 episodes, tells that the agent is able to gain higher rewards in several episodes (peaks in maximum reward) but also leads to collisions in several others (troughs in minimum reward). We also found by observation, that the positive reward gets offset by the high negative reward for a collision at a later point within episodes. This problem appears in this experiment as the total time spent in an episode has increased, because of the agent waiting at traffic lights. During such intervals the learning (and the episode time) is paused.

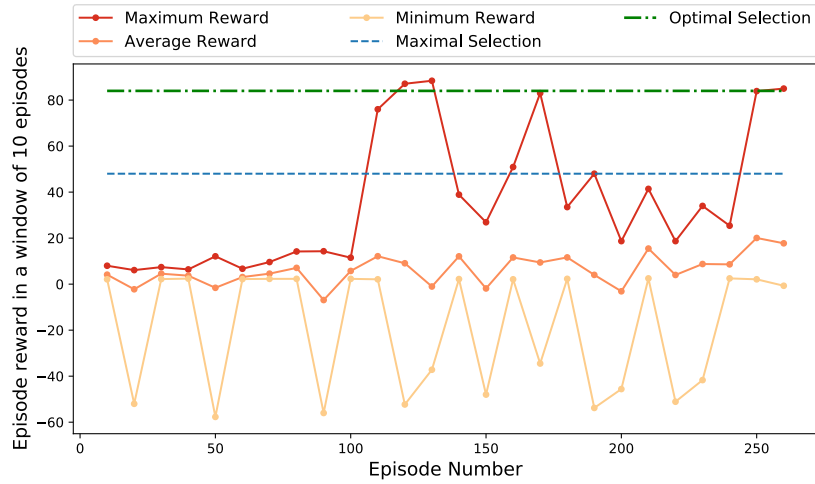


Figure 5.16: Training reward values for Experiment 9

Exp.	Test	#Ep.	Time		Reward				%Coll.
			Avg.	Opt.	Avg.	MSS	Opt.	%High	
9	S1	26	57	60	72	48	84	85	15

Table 5.11: Test results for Experiment 9

During testing we observed that there was high variance in the collision rewards. Collisions happened in 15% of the testing episodes and in all other episodes the agent was able to achieve high rewards (Table 5.11). To reward the agent's behaviour of having a collision-free ride for a shorter time span, in the next experiment, we decrease the maximum episode length and terminate the episode. This shall ensure that the agent gets timely feedback and a collision does not nullify the previously accumulated rewards during training.

5.3.12 Experiment 10

In this experiment, we set the episode length to 30 seconds with all the other settings same as Experiment 9.



Figure 5.17: Training reward values for Experiment 10

As can be seen from Fig. 5.17, the training is close to convergence after 450 episodes and this clearly performs better than Experiment 9 where we had maximum episode length set to 60 seconds. The training average also increases steadily if we

consider a larger averaging window size. Understandably, this takes much longer to converge to an optimal policy when compared with Experiment 5 (where perfect convergence was achieved in 160 episodes) as the environment is more dynamic and diverse.

Exp.	Test	#Ep.	Time		Reward				%Coll.
			Avg.	Opt.	Avg.	MSS	Opt.	%High	
10	S1	72	29	30	40	24	42	93	7

Table 5.12: Test results for Experiment 10

We performed more number of tests as the environment is diverse as compared to those in Experiments 1-8. As can be seen from Table 5.12, the testing average rewards reaches close to the optimal policy. In 93% of the testing episodes the agent received a high reward. However, in the remaining 7% of the testing episodes the agent collided with a road user. Hence the learned policy although close to optimal is still not perfect.

We believe, that we will probably have to train the agent for several more episodes to find a perfect policy. However, since each episode in this environment takes significantly longer to run as the agent waits at the traffic lights and the learning is paused, we could not afford to keep the experiment running. In the next experiment we run the same setup with simple RL to check if the LCRL based approach in this experiment was adding any value.

5.3.13 Experiment 11

In this experiment we keep all the parameters same as in Experiment 10, except that we use Simple RL instead of LCRL to check the benefit of LCRL based approach.

As seen from Fig. 5.18 the training average, minimum or maximum, none of them show any gradual increase till 130 episodes hence we terminate the experiment

constrained by finite computational resources. When compared with Experiment 10 the maximum training reward had reached the optimal value within the first 100 episodes. Moreover the testing average (Table 5.13) remains closer to the maximal selection with collision in 5% of test episodes. *Hence Experiments 10 and 11 clearly demonstrate that LCRL based approach is necessary, efficient and effective.*

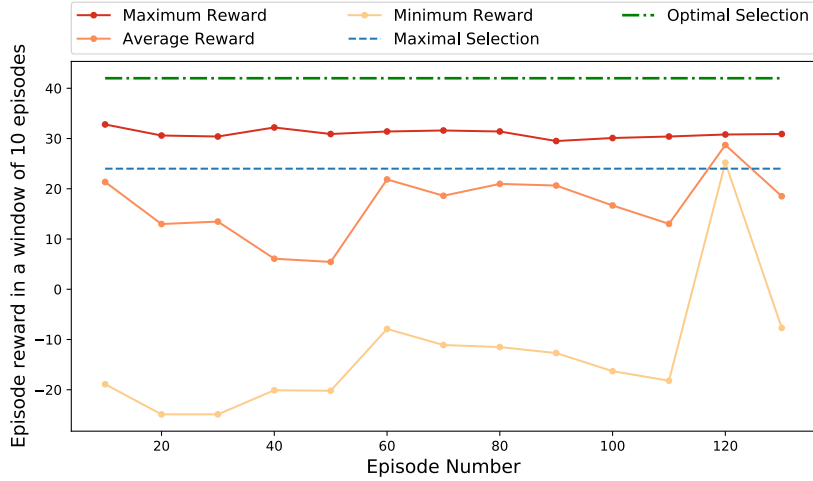


Figure 5.18: Training reward values for Experiment 11

Exp.	Test	#Ep.	Time		Reward				%Coll.
			Avg.	Opt.	Avg.	MSS	Opt.	%High	
11	S1	19	29	30	28	24	42	0	5

Table 5.13: Test results for Experiment 11

In the next three experiments, we continue our search to find a suitable training setting such that the agent learns an optimal policy for diverse environments.

5.3.14 Experiment 12

In this experiment, we start from any random position on the map and the rest of the setup is same as Experiment 10. This is an attempt to generalize the learning to diverse scenarios. We observe from Fig. 5.19 that the learning is again very slow and it takes about 150 episodes for the average training rewards to reach near maximal selection. On comparing it with Fig. 5.17 it appears that we would encounter the

same issue as in Experiment 10 and the training would take too long to converge. The test values (Table 5.14) also indicate that the agent does not learn much over 170 episodes as the agent collides in 30% of the test episodes. Hence, we explore alternative approaches to make the agent learn a better policy.

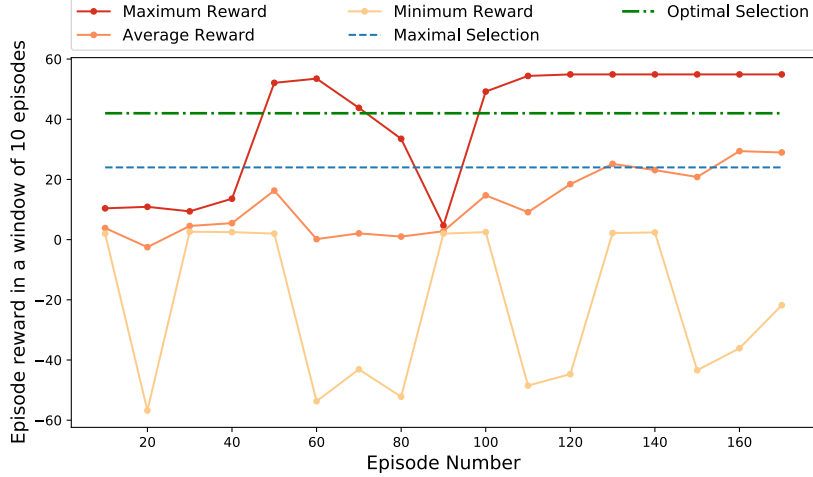


Figure 5.19: Training reward values for Experiment 12

Exp.	Test	#Ep.	Time		Reward				%Coll.
			Avg.	Opt.	Avg.	MSS	Opt.	%High	
12	S1	20	24	30	34	24	42	70	30

Table 5.14: Test results for Experiment 12

In the next two experiments we return to the setup of Experiment 5 where we were able to at least find a policy which led to optimal values with Test-S1 and try changing other parameters which could possibly make the agent learn a scalable policy which would be successful even with Test-S2.

5.3.15 Experiment 13

In Experiment 13, we try changing the RL algorithm to TRPO and keep the other parameters same as that of Experiment 5. Even after training for 250 episodes the training reward values do not show any increase (Fig. 5.20). Surprisingly, the agent was able to reach a testing average of 31 (Table 5.15), which is much higher than the maximum reward achieved during training. A possible explanation to this

anomaly could be that the agent was focusing a lot on exploration but too little on exploitation during its training time. In any case, the PPO is more suitable for learning with our setup and hence we resume the use of PPO. In the next experiment we try changing the rewarding function for the agent.

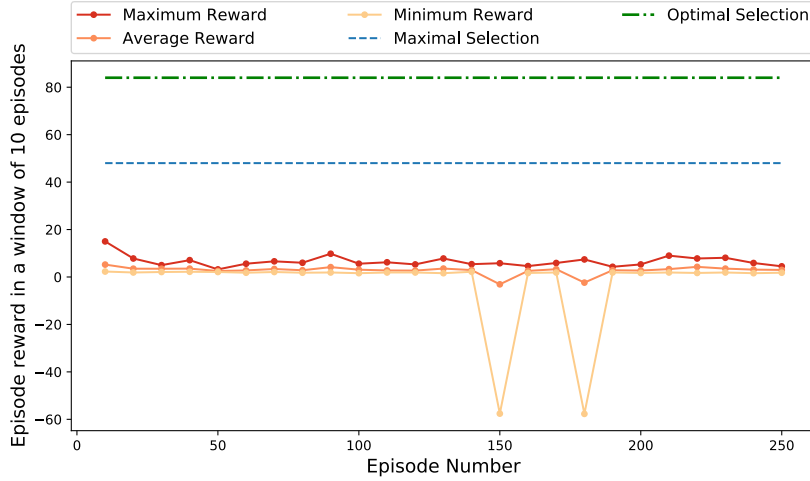


Figure 5.20: Training reward values for Experiment 13

Exp.	Test	#Ep.	Time		Reward				%Coll.
			Avg.	Opt.	Avg.	MSS	Opt.	%High	
13	S1	23	53	60	31	48	84	0	26

Table 5.15: Test results for Experiment 13

5.3.16 Experiment 14

In Experiment 14, we take the settings as in Experiment 5 but double the collision cost. The rationale behind this is the following:

- We wish to learn a policy which works in diverse testing environments. Hence, we want to avoid converging to a local optimum specific to training.
- In Test-S2 of Experiment 5, we observed that the agent was colliding in a number of episodes. We strictly wish to discourage collisions but can still afford to spend more time in training, as Experiment 5 converged and reached optimum quickly.

- Doubling the collision cost would ensure that an episode having a collision event always yields a negative cumulative reward for that episode.

We observe from Fig. 5.21 that the training converged to the optimum value within 120 episodes and the first entry of Table 5.16 confirms that Test-S1 was successful with 100% test episodes having high reward and none leading to a collision. This is similar to the Test-S1 of Experiment 5 (Table 5.7).

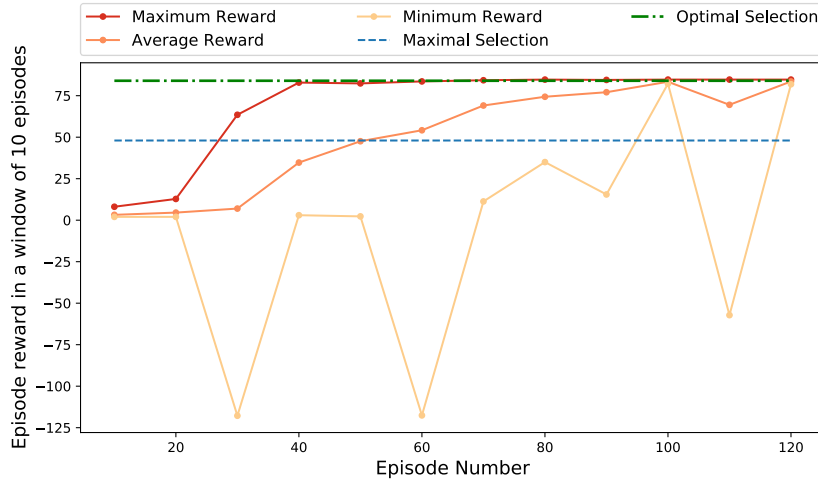


Figure 5.21: Training reward values for Experiment 14

Exp.	Test	#Ep.	Time		Reward				%Coll.
			Avg.	Opt.	Avg.	MSS	Opt.	%High	
14	S1	12	60	60	84	48	84	100	0
14	S2	50	60	60	84	48	84	100	0

Table 5.16: Test results for Experiment 14

We then proceed to Test-S2 which tests for diverse settings. We conduct 50 tests where the ego vehicle starts from any random location on the map, the weather conditions change with every episode, the traffic lights change dynamically, and the ego vehicle needs to achieve 60 seconds of collision free driving with an optimal sensor module selection. In all the test episodes the agent achieved high reward and did not face any collision (Table 5.16). Therefore, the Test-S2 was also successful and we conclude that the settings of Experiment 14 were sufficient to provide the agent with an appropriate training environment to learn a policy which works in diverse testing settings.

Chapter 6

Conclusions

6.1 Contributions

Through our work we demonstrate the following:

- The problem of active monitoring (dynamic sensor module selection) can be framed as an RL problem (Chapter 4). We estimate the size of the state space for a reasonably complex scenario (multiple road users in a city setting) and find that the state space increases exponentially. Hence we must resort to Deep RL based approaches.
- Even with Deep RL based approaches, owing to the enormous state space the training is either extremely slow (Experiments 1,11; Sections 5.3.3 and 5.3.13) or converges to a suboptimal result (Experiment 6; Section 5.3.8).
- Using carefully designed LTL properties, LCRL can expedite the learning process and lead the agent to the optimal policy within a matter of a couple of hours (Experiments 5,7,10,14; Sections 5.3.7, 5.3.9, 5.3.12 and 5.3.16)
 - If the LTL property is not chosen carefully, it can lead to pruning of essential search space (i.e, a search space which was essential for training) and lead the agent to either learn a sub-optimal policy such as Experiment 2 (Section 5.3.4) or lead to non convergence, like in Experiments 3 and 4

(Sections 5.3.5 and 5.3.6).

- With the help of LCRL, the onus on reward shaping is reduced. To some extent, reward shaping is still required to further improve the performance of LCRL based approaches such as in Experiment 5 (Section 5.3.7) and Experiment 14 (Section 5.3.16).
- Changing the ODD impacts the learning process. The agent can take longer to learn an optimal policy in one of the following settings:
 - Environments where there are lesser chances of failure: For example, in Experiment 7 (Section 5.3.9) there are no vehicles and the agent takes more than 200 episodes to find an optimal policy, whereas in Experiment 5, the optimal policy is learnt within 140 episodes.
 - Dynamically changing environments: For example, in Experiments 9-11 (Sections 5.3.11 to 5.3.13) where there are 5 starting positions and dynamically changing traffic lights, it takes around 400 episodes to reach close to optimal (as compared to only 140 episodes needed for convergence in Experiment 5). In Experiment 12 (Section 5.3.14) when the agent can start from any position, the learning seems to be even slower.
- Open source simulators like CARLA can be used to learn meaningful policies, despite some challenges inherent to it: 1) the learning can not be fast forwarded, as each episode takes the same amount of time as the actual simulation time 2) there are often delays in client-server messages which leads to a lag, leading to unexpected collisions. The first challenge was addressed with the help of LCRL as the learning itself was fast so the total running time was reduced for training. To address the second challenge we had to ensure high speed connectivity between the client and the server. In some rare cases, we manually analyzed some of the collision scenarios in case of high client/server load.

6.2 Future Work

Our work lays the foundation for several interesting research directions and motivates the real-world application of LCRL for intelligent sensor module selection.

To learn policies for the real-world, it is possible to build an advanced simulation keeping our sensor module selector fundamentally the same.

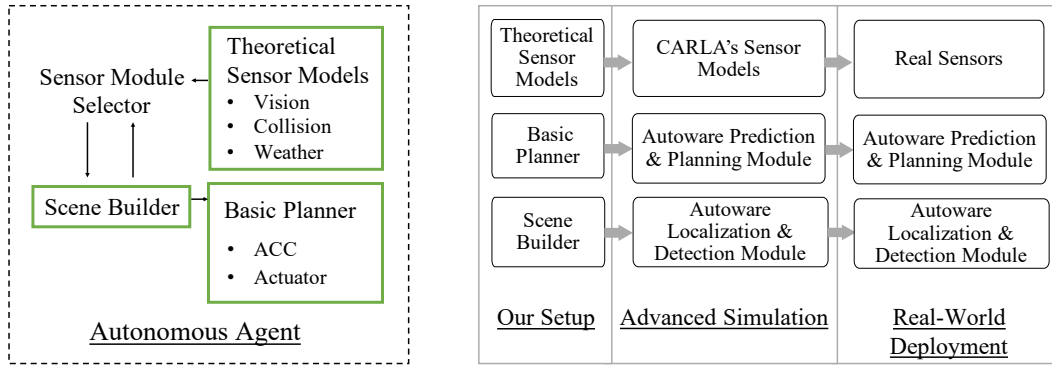


Figure 6.1: (Left) Current framework, green boxes represent the components which can be evolved for advance simulation. (Right) Evolving the current setup for advanced simulation to learn policies for real-world deployment.

In our work we assumed theoretical sensor models (Section 3.4), a scene builder component (Section 3.5) and a basic planner (Section 3.6) in which the agent follows a predefined path. To evolve our setup Autoware [18, 27] can be used for designing advanced simulations. Autoware is an open-source software for autonomous driving based on ROS1 [38]. Autoware¹ provides modules for localization and detection which can replace the scene builder component of our setup. The basic planner can be substituted with Autoware's prediction and planning modules [48]. Also, instead of sensor models Carla's sensor models can be used to get realistic sensor data. This will enable in learning policies which are based on realistic sensor data (with possible noise and errors).

¹the integration of Autoware with CARLA required significantly higher computational resource (than what were available to us) hence we had to limit ourselves to theoretical models

Since Autoware is based on ROS, it works on a publish-subscribe architecture. In the advanced simulation, CARLA’s sensor models would publish sensor data and Autoware sensor modules would subscribe to it. Once the policies are learned over the advance simulation, the same setup can be deployed for learning in the real world. The only change then will be to replace the CARLA’s sensor models with the real sensor data. The Autoware modules would remain the same and they would subscribe to the real sensor data.

Our learning approach can be applied for learning in a real-world (online) setting safely. In case of an online setting we will care for more safety properties in addition to preventing collisions. As demonstrated in Section 4.3 we can conjunct multiple safety properties using LTL. After having the required safety properties we can find an appropriate training setting so that agent learns to abide by the safety properties in diverse settings (as shown through Experiment-14, Section 5.3.16).

Leveraging the LCRL framework only for early termination we were able to significantly expedite the learning process. However, the framework provides a plethora of opportunities to further improve the learning by assigning specific state/transition based rewards on top of LTL properties [40]. These can be utilized to learn sensor module selection in complex real-world settings.

Bibliography

- [1] See Think Act. https://www.zf.com/mobile/en/technologies/see_think_act/see_think_act.html. [Online; accessed 7-Sep-2020].
- [2] Carla Python API. https://carla.readthedocs.io/en/latest/python_api/. [Online; accessed 27-Aug-2020].
- [3] Dimitir P Bertsekas and Steven Shreve. *Stochastic optimal control: the discrete-time case*. Athena Scientific, 2004.
- [4] Fang Bian, David Kempe, and Ramesh Govindan. Utility based sensor selection. In *Proceedings of the 5th international conference on Information processing in sensor networks*, pages 11–18, 2006.
- [5] Stefan Braun, Daniel Neil, Jithendar Anumula, Enea Ceolini, and Shih-Chii Liu. Attention-driven multi-sensor selection. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.
- [6] Luca Carlone and Sertac Karaman. Attention and anticipation in fast visual-inertial navigation. *IEEE Transactions on Robotics*, 35(1):1–20, 2018.
- [7] Rolando Cavazos-Cadena, Eugene A Feinberg, and Raúl Montes-De-Oca. A note on the existence of optimal policies in total reward dynamic programs with compact action sets. *Mathematics of Operations Research*, 25(4):657–666, 2000.

- [8] SAE Levels Chart. <https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-%E2%80%9Clevels-of-driving-automation%E2%80%9D-standard-for-self-driving-vehicles>. [Online; accessed 1-Sep-2020].
- [9] PID Controller. https://en.wikipedia.org/wiki/PID_controller. [Online; accessed 7-September-2020].
- [10] Ankush Desai, Tommaso Dreossi, and Sanjit A Seshia. Combining model checking and runtime verification for safe robotics. In *International Conference on Runtime Verification*, pages 172–189. Springer, 2017.
- [11] Tensorforce docs. <https://tensorforce.readthedocs.io/en/latest/>. [Online; accessed 6-September-2020].
- [12] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.
- [13] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω -automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA’16)*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129. Springer, October 2016.
- [14] Richard Durrett. *Essentials of stochastic processes*, volume 1. Springer, 1999.
- [15] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. Graphviz—open source graph drawing tools. In *International Symposium on Graph Drawing*, pages 483–484. Springer, 2001.

- [16] Unreal Engine. <https://www.unrealengine.com/>. [Online; accessed 7-Sep-2020].
- [17] Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman, and David Parker. Automated verification techniques for probabilistic systems. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 53–113. Springer, 2011.
- [18] Autoware GitHub. <https://github.com/Autoware-AI/autoware.ai>. [Online; accessed 5-September-2020].
- [19] Graham C Goodwin, Stefan F Graebe, Mario E Salgado, et al. *Control system design*. Upper Saddle River, NJ: Prentice Hall,, 2001.
- [20] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Certified reinforcement learning with logic guidance. *arXiv preprint arXiv:1902.00778*, 2019.
- [21] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Cautious reinforcement learning with logical constraints. *arXiv preprint arXiv:2002.12156*, 2020.
- [22] Yi-Chen Hsieh, Feng-Li Lian, and Chih-Ming Hsu. Optimal multi-sensor selection for driver assistance systems under dynamical driving environment. In *2007 IEEE Intelligent Transportation Systems Conference*, pages 696–701. IEEE, 2007.
- [23] Intel. <https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/automotive-autonomous-driving-vision-paper.pdf>, 2020. [Online; accessed 7-September-2020].
- [24] Siddharth Joshi and Stephen Boyd. Sensor selection via convex optimization. *IEEE Transactions on Signal Processing*, 57(2):451–462, 2008.

- [25] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274, 2002.
- [26] Aaron Kane, Omar Chowdhury, Anupam Datta, and Philip Koopman. A case study on runtime monitoring of an autonomous research vehicle (arv) system. In *Runtime Verification*, pages 102–117. Springer, 2015.
- [27] Shinpei Kato, Eijiro Takeuchi, Yoshio Ishiguro, Yoshiki Ninomiya, Kazuya Takeda, and Tsuyoshi Hamada. An open approach to autonomous vehicles. *IEEE Micro*, 35(6):60–68, 2015.
- [28] Alexander Kuhnle, Michael Schaarschmidt, and Kai Fricke. Tensorforce: a tensorflow library for applied reinforcement learning. *Web page*, 2017.
- [29] SAE Levels. https://www.sae.org/standards/content/j3016_201806/. [Online; accessed 1-Sep-2020].
- [30] Matt Luckcuck, Marie Farrell, Louise A Dennis, Clare Dixon, and Michael Fisher. Formal specification and verification of autonomous robotic systems: A survey. *ACM Computing Surveys (CSUR)*, 52(5):1–41, 2019.
- [31] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56, 2016.
- [32] Juan Pablo Mendoza, Reid Simmons, and Manuela Veloso. Detection and correction of subtle context-dependent robot model inaccuracies using parametric regions. *The International Journal of Robotics Research*, 38(8):887–909, 2019.
- [33] Intelligent Driver Model. https://en.wikipedia.org/wiki/Intelligent_driver_model. [Online; accessed 7-September-2020].
- [34] OpenDRIVE. <http://www.opendrive.org/>. [Online; accessed 7-Sep-2020].

- [35] Ola Pettersson. Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems*, 53(2):73–88, 2005.
- [36] Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science*, pages 46–57. IEEE, 1977.
- [37] Martin L Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, 2014.
- [38] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.
- [39] BMW Safety Assessment Report. <https://www.bmwusa.com/content/dam/bmwusa/innovation-campaign/autonomous/BMW-Safety-Assessment-Report.pdf>. [Online; accessed 7-September-2020].
- [40] Martin Riedmiller. Neural fitted Q iteration-first experiences with a data efficient neural reinforcement learning method. In *ECML*, volume 3720, pages 317–328. Springer, 2005.
- [41] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002.
- [42] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.
- [43] Daniel Schubert and Klaus Möller. See, think, act-wie zf globalen trends begegnen. *Controlling*, 30(2):69–71, 2018.

- [44] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.
- [45] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [46] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. On a formal model of safe and scalable self-driving cars. *arXiv preprint arXiv:1708.06374*, 2017.
- [47] Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínský. Limit-deterministic Büchi automata for linear temporal logic. In *CAV*, pages 312–332. Springer, 2016.
- [48] Autoware Webpage: Open source software for urban autonomous driving. <https://www.autoware.ai/>. [Online; accessed 6-September-2020].
- [49] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.
- [50] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 62(2):1805, 2000.
- [51] Vulnerable Road Users. https://ec.europa.eu/transport/themes/its/road/action_plan/its_and_vulnerable_road_users_en. [Online; accessed 24-Aug-2020].
- [52] Experiment 10 Training Video. <https://www.dropbox.com/s/ow9rwe70p0sshrh/Experiment-10-Training.mkv?dl=0>. [Uploaded on 3-Sep-2020].

- [53] Experiment 5 Training Video. <https://www.dropbox.com/s/5b2g6035zr1sjbe/output.mkv?dl=0>. [Uploaded on 3-Sep-2020].
- [54] BSI CAV Vocabulary. <https://www.bsigroup.com/en-GB/CAV/>, 2020. [Online; accessed 7-September-2020].
- [55] Zhi Wang, Han-Xiong Li, and Chunlin Chen. Reinforcement learning-based optimal sensor placement for spatiotemporal modeling. *IEEE transactions on cybernetics*, 50(6):2861–2871, 2019.
- [56] Haotian Zhang, Raid Ayoub, and Shreyas Sundaram. Sensor selection for kalman filtering of linear dynamical systems: Complexity, limitations and greedy algorithms. *Automatica*, 78:202–210, 2017.