On-Policy Reinforcement Learning for Learning to Drive in Urban Settings

Tanmay Agarwal

CMU-RI-TR-20-32

July 28, 2020

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Jeff Schneider, Chair
David Held
Benjamin Eysenbach

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Keywords: Reinforcement Learning, Representation Learning, Robot Planning and Control, Autonomous Driving

To my family, for their endless love and support, and friends and teachers who have made this journey a truly memorable one.

Abstract

Traditional autonomous vehicle pipelines that follow a modular approach have been very successful in the past both in academia and industry, which has led to autonomy deployed on road. Though this approach provides ease of interpretation, its generalizability to unseen environments is limited and hand-engineering of numerous parameters is required, especially in the prediction and planning systems. Recently, deep reinforcement learning has been shown to learn complex strategic games and perform challenging robotic tasks, which provides an appealing framework for learning to drive.

In this thesis, we propose two works that formulate the urban driving tasks using reinforcement learning and learn optimal control policies primarily using waypoints and low-dimensional representations, also known as affordances. We demonstrate that our agents when trained from scratch learn the tasks of lane-following and driving around intersections as well as learn to stop in front of other actors or traffic lights even in the dense traffic setting. Further, we also propose an algorithm which we term as Exploratory Policy Search (EPS) that combines forward search with model-free reinforcement learning algorithms to find the optimal policy with increased exploration.

Acknowledgments

I am glad to use this opportunity to thank numerous people for their guidance and support without whom this thesis would not have been possible.

Firstly, I would like to thank my advisor, Prof. Jeff Schneider for being an incredible advisor and extending extraordinary support in my last two years here at Carnegie Mellon University. He has not only shaped my research passion and skills but also guided me throughout my journey with fruitful advice and exciting discussions. I believe without his guidance, I would not have grown to the position where I am now. Thus, I sincerely thank him for his unconditional support and belief in my potential.

Next, I am highly indebted to my parents for their boundless love and support throughout this journey. I am also thankful to them for believing in me and giving me this opportunity to explore beyond my potential. Besides this, I am grateful to them for everything else they have provided.

I would also like to thank my colleague Hitesh Arora who not only has been a great individual to work with but also a true friend that I have found during my journey. I am glad that our numerous discussions have often lasted for hours which has led us to discover better solutions as well as thought-provoking problems to work on during the research. Besides him, I would also like to thank Theophile, Shuby, Tanvir, Audrey, Adam, Christoph, Vinay, and Mayank for being amazing lab-mates.

I am also grateful to my other committee members Prof. David Held and Benjamin Eysenbach for their concrete research discussions and constructive feedback that has helped me shape this research.

A special thanks to Darshi for helping me choose this research project and supporting me through the journey. Also, I am grateful to my other friends Gautham, Tithi, Aditya, Sarthak, Talha, and many others who have made this expedition fun and exciting.

Also, this work could not have been possible had it not been for all the Auton Lab members and the resources that have led to the success of this thesis. I would also like to extend my gratitude to Predrag Punosevac for being an amazing lab administrator and graciously helping us with all the

system resource needs.

Lastly, I am grateful to the entire student community at CMU that has given me an amazing set of friends and made my stay a fantastic experience with innumerous life learnings and memorable experiences.

Funding

This work was supported by the CMU Argo AI Center for Autonomous Vehicle Research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

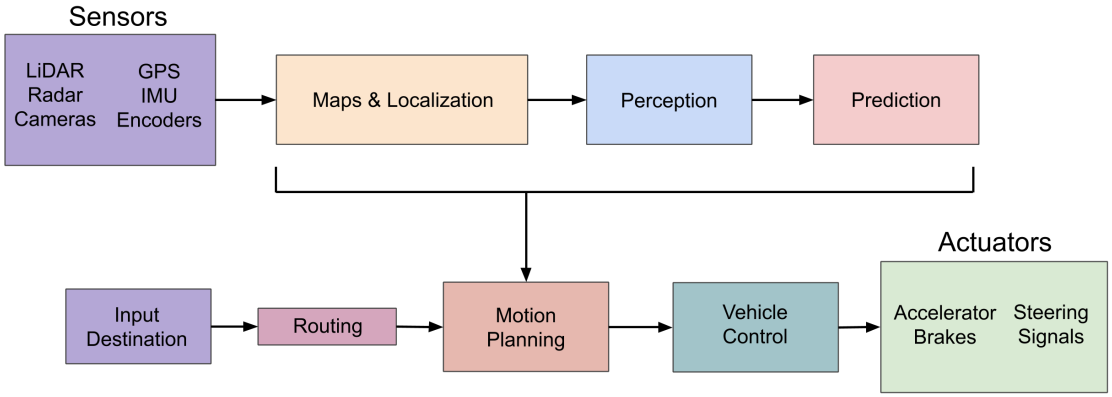

Figure 1.1: Traditional autonomous driving stack that uses a suite of sensors to predict the end control. The overall pipeline can be modularized into different subsystems that include mapping and localization, perception, actor prediction, motion planning and vehicle control.

A recent survey conducted by the National Highway Traffic Safety Administration suggested that more than 94% of road accidents are caused by human errors [94]. This has led to the rapid evolution of the autonomous driving systems over the last several decades with the promise to prevent such accidents and improve the driving experience. But despite numerous research efforts in academia and industry, the autonomous driving problem remains a long-standing problem in the domain of

artificial intelligence and machine learning. This is because the current systems still face numerous real-world challenges for which it is not ready yet. These challenges include ensuring the accuracy and reliability of the prediction systems, maintaining the reasonability and optimality of the decision-making systems, to determining the safety and scalability of the entire system.

Another major factor that impedes success is the complexity of the problem that ranges from learning to navigate in constrained industrial settings, to learning to drive on highways, to navigation in dense urban environments. Navigation in dense urban environments requires understanding complex multi-agent dynamics including tracking multiple actors across scenes, predicting intent, and adjusting agent behavior conditioned on historical states. These factors provide a strong impetus for the need of general learning paradigms that are 'complex' enough to take these factors into account.

The most common practice of solving the massive task of driving is to divide the system into subcategories and employ an array of sensors and algorithms to each of the different modules [19, 48, 60, 69, 72, 100, 107]. The overall system consists of a few core blocks that include localization and mapping, perception, prediction, planning and decision making, and vehicle control (Fig. 1.1). Developing each of these individual modules makes the overall task much easier as each of the sub-tasks can independently be solved by popular approaches in the literature of computer vision [32, 87], robotics [14, 57] and vehicle dynamics [30, 81]. But with the advent of deep learning [58], most of the current state of the art systems use a variant of supervised learning over large datasets [33] of collected logs to learn individual components' tasks. The major disadvantage of these heavily engineered modular systems is that it is extremely hard to tune these subsystems and replicate the intended behavior which leads to its poor performance in new environments. Moreover, these systems are also prone to error propagation [64] as an erroneous perception or prediction subsystem may impact the performance of planning and control subsystems.

Another approach that has recently become popular is exploiting Imitation Learning where the aim is to learn a control policy for driving behaviors based on observations collected from expert demonstrations [9, 12, 13, 20, 23, 24, 71, 77, 80, 83, 86, 91, 109]. The advantage of these methods is that the agent can be trained in an end-to-end fashion to learn the desired control behavior which significantly reduces

the effort of tuning each component that is common to more modular systems. The drawback however is that these systems are challenging to scale and generalize to novel situations since it is impractical to obtain expert demonstrations for all the scenarios that we care about or train an agent that can ever outperform the expert. Furthermore, imitation learning assumes that the data is independent and identically distributed (i.i.d.). In contrast, the driving scenarios are non-i.i.d. where each new state is dependent upon the previous history of states and control outputs, thereby limiting the scale of these approaches.

More recently, deep reinforcement learning (DRL) has made large strides towards solving sequential decision-making problems, including learning to play complex strategic games such as Go [92], chess [93] or Atari games [65, 66] , as well as completing complex robotic manipulation tasks [5, 35, 108]. The superhuman performance attained using this learning paradigm motivates the question of whether it could be leveraged to solve the long-standing goal of creating autonomous vehicles. This approach of using reinforcement learning has inspired a few recent works [29, 49, 50, 53, 61] that learn control policies for navigation task using high dimensional observations like images. The previous approach using DRL [29] reports poor performance in navigation tasks, while the imitation learning-based approach [61] that achieves better performance, suffers from poor generalizability.

Although learning policies from high dimensional state spaces remain challenging due to the poor sample complexity of most reinforcement learning (RL) algorithms, the strong theoretical formulation of reinforcement learning and its generality to unseen scenarios make it a useful learning framework to be applied to autonomous driving. Moreover, RL also offers a corrective mechanism to improve the learned policies. These factors make us strongly believe in its potential to learn common urban driving skills for autonomous driving and form the basis for the research questions that we pose in this work.

1.2 Research Contributions

This work motivates reinforcement learning as a learning paradigm to learn common driving behaviors. Building on some of the advancements in continuous control DRL [88, 89], we propose two works that formulate the urban driving tasks using

reinforcement learning and learn optimal control policies primarily using waypoints and low-dimensional representations. Our agents when trained from scratch learn to drive straight, around intersections and stop for other vehicles and traffic lights without colliding with any static and dynamic obstacles. Additionally, our agents also demonstrate a reduced number of infractions, including collisions with static or dynamic actors, which is at least an order of magnitude lesser than the previously reported results in the prior works.

Since we do not have access to the environment model in the real world, we primarily develop our framework based on model-free reinforcement learning algorithms that are more stable than model-based reinforcement learning methods [1, 38, 41, 56, 68]. Also, as it is impractical to train RL algorithms on the self-driving vehicle in the real world, we demonstrate our findings on the CARLA simulator [29] and hope that our work paves the way for real-world deployment in the future.

Specifically, our research contributions can be listed as follows.

- We pose the autonomous driving problem as a reinforcement learning problem, devise its various components to make the problem easier while keeping it general enough to be easily extensible.
- We demonstrate that using an off-the-shelf reinforcement learning algorithm (Proximal policy optimization [89]), our agents are capable of learning optimal driving policies in the CARLA [29] simulator.
- We propose to use waypoints and low-dimensional representations as feature inputs to the policy network, that encode the current state of the world in the near horizon and is crucial to determining the optimal control.
- We design a dense reward function that is fundamentally similar to the cost function used by the state-of-the-art planning systems.
- We also propose a new algorithm known as Exploratory Policy Search (EPS) that combines forward search with model-free Proximal policy optimization to find the best-optimized driving policy that we care about in the real world.
- Lastly, we plan to open-source our reinforcement learning CARLA environment to readily be used by the research community that works in a similar direction.

1.3 Outline

This section outlines the different chapters that are organized in this thesis.

- Chapter 2 discusses the related work in the autonomous driving domain which can broadly be categorized into modular, imitation, or reinforcement learning-based methods.
- Chapter 3 gives a quick overview of reinforcement learning and the different algorithms that exist in this domain.
- Chapter 4 introduces our CARLA environment that we build from scratch to enable reinforcement learning through interactions with the simulator. We also discuss the sensors and affordance algorithms developed by us that form the basis for the next chapters.
- Chapter 5 describes our first approach to autonomous driving using reinforcement learning. This work [4], demonstrates learning common driving skills like lane-following and driving around intersections.
- Chapter 6 discusses our next approach that formulates the urban driving task in the presence of dynamic actors. We demonstrate that our approach of learning to drive using low-dimensional representations outperforms all our other chosen representations as well as gives us significant improvements when compared with the prior works.
- Chapter 7 introduces our Exploratory Policy Search algorithm that aims to combine forward search with model-free reinforcement learning algorithms.
- Chapter 8 describes the conclusions and key takeaways from this thesis and directs readers to some open problems for future work.

Chapter 2

Related Work

Autonomous driving is a field that has received a great deal of attention from both the research community and the industry. It shares a long history of approaches that can broadly be clustered into three common approaches: modular, imitation learning, and reinforcement learning-based approaches. In this chapter, we review each of the different variety of methods that are popular in literature and that have been used towards solving the task of autonomous driving.

2.1 Modular Approaches

Most of the state-of-the-art autonomous driving systems use approaches that can commonly be clustered under this category. These approaches aim to divide the entire task into different sub-tasks and sub-modules that include the core functional blocks of localization and mapping, perception, prediction, planning, and decision making, and vehicle control [19, 48, 60, 69, 72, 100, 107].

The localization and mapping subsystem senses the state of the world and locates the ego-vehicle with respect to the environment [8, 15, 55, 74, 107]. This is followed by the perception sub-system that detects and tracks all surrounding static and dynamics objects [6, 8, 27, 74, 78, 107]. The intermediate representation produced by the perception subsystem then feeds into the prediction and planning subsystems that outputs an optimal plan of action [8, 74, 76, 90, 107] based on the future trajectories of all the agents in the environment [28, 59, 70] and the planning cost function. Finally, the plan of action is then mapped into low-level control actions that are responsible for the motor actuation. For more details on these approaches, we direct

readers to [8, 60, 107].

Although these systems offer high interpretability, they employ a heavily engineered approach that requires cumbersome parameter tuning and large amounts of annotated data to capture the diverse set of scenarios that the autonomous driving vehicle may face. Tuning these subsystems for each new town and capturing the labeled data is a tedious process that questions the scalability of these approaches. Furthermore, the driving policy does not differ a lot from one town to another which raises the question to learn driving policies that are independent of the driving conditions using more sophisticated approaches.

2.2 Imitation Learning

Another branch of approaches that are common in literature and practice is the imitation learning-based approaches that learn control policies based on expert demonstrations. Dating back to one of the earliest successful works on imitation learning is the ALVINN [80] which uses a simple and shallow feedforward network to learn the task of the road following. This method first demonstrated an end-to-end learning approach that outputs a target direction based on raw images and a laser range finder. Other works, well studied in the literature, include [2, 82, 85, 96] that either leverage expert demonstrations using supervised learning or formulate online imitation learning using game-theory or maximum margin classifiers.

With the development of the deep learning [54, 58], the trend has shifted towards using Convolutional Neural Networks (CNNs) for the perception tasks. The first architecture that used CNN based end-to-end method for steering angle prediction was proposed by Nvidia [12] that learned directly from raw pixels. A follow-up work [13] explained that the architecture does learn to encode salient objects on the road and more subtle features that are hard to anticipate and programmed by engineers. These features correlate with the features that are essential for the driving task and thus learn to ignore structures from the images that are not relevant. Similar works have been proposed in the past that map the raw images to steering angles [71] or learn an additional visual attention model [51] that highlights image regions that contribute to the action predictions . One limitation in the assumption of the above models is that the optimal action can be inferred solely from a single perception

input. To that end, [21, 105, 106] propose to combine spatial and temporal cues using recurrent units (LSTMs / Conv-LSTMs) to learn actions conditioned on the historical states and instantaneous camera observations.

Although the imitation learning based deep networks have learned to follow the roads and avoid obstacles, these policies suffer at test time as they cannot be controlled by human experts. This is because a vehicle trained to imitate the expert cannot be guided when it hits an intersection. [23] proposes to condition the imitation learned policies based on the high-level navigational command input which disambiguates the perceptuomotor mapping and allows the model to still respond to high-level navigational commands provided by humans or a mapping application. Another work [20] decouples the sensorimotor learning task into two, learning to see and learning to act. In the first step, a privileged agent is learned that has access to simulator states and learns to act. The second step involves learning to see where a sensorimotor agent is learned based on supervision provided by the privileged agent. For a more in-depth review of these methods, we direct readers to check out [20, 23, 46, 75, 97].

Although these methods have low model complexity and can be robust with enough training data, their generalization ability to complicated environments is still questionable. This is because none of the above approaches reliably handle the dense traffic scenes and are prone to suffer from inherent dataset biases and lack of causality [24]. Moreover, collecting a large amount of expert data that imitates every potential scenario which the autonomous agent may encounter, remains an expensive process and is difficult to scale. On the other hand, these agents can never outperform the human expert as the training data is annotated solely from expert demonstrations. Additionally, imitation learning assumes the data to be independent and identically distributed (i.i.d.) which differs from the usual driving setup where the data is sequential. These limitations restrict the extent to which imitation learning-based methods can be used for large-scale learning of common urban driving behaviors and suggest the use of reinforcement learning-based approaches for optimizing policies for sequential decision-making tasks.

2.3 Reinforcement Learning

A recently proposed work [49], posed the autonomous driving problem as a Markov decision process (Sec. 3.1) and demonstrated the use of deep reinforcement learning

(RL) to learn the driving task in both simulated and real-world environments. It shows that using a canonical RL algorithm, Deep Deterministic Policy Gradients (DDPG) [62], the model can learn a continuous-valued policy for the lane following task, using a single monocular image as input.

Another line of work that has attracted RL researchers is using the TORCS simulator [104], which is an open-source and easy to use the racing simulator. The first work that demonstrated learning stable driving policies on TORCS was the asynchronous advantage actor-critic (A3C) [67] that learned discrete action policy only using a visual RGB image. [62] extends the prior work to continuous action space by proposing DDPG, which also learns policies in an end-to-end fashion directly from raw pixels.

Recently, many works have focused on using the new CARLA simulator [29] that is an open-source urban driving simulator that includes pedestrians, traffic lights, and other dynamic actors. The original CARLA work [29] released a new driving benchmark along with three baselines that used a modular, imitation learning, and reinforcement learning-based approach respectively. The RL baseline used the A3C algorithm [67] but reported poor results than the imitation learning one. These results were improved by [61] that finetune the imitation learned agent using DDPG for continuous action space. Although the results are better than the original CARLA RL baseline [29], this method relies heavily on the pre-trained imitation learned agent and hence it is unclear whether the improvement comes from the RL fine-tuning. Moreover, they also do not model the traffic light behaviors.

A more recent work [99] combines the ideas of Rainbow, IQN, and Ape-X [26, 40, 44] to propose an end-to-end trainable RL algorithm. The agent proposed by this work divides the task into two phases. The first phase trains a Resnet-18 encoder [39] that predict affordances like traffic light state or distance to the center of the lane. The second phase uses the output features of the encoder, also termed as implicit affordances, as the RL state, with the advantage that the RL optimizer only trains the last past of the network.

The works discussed here highlight the rising trend in the research community to explore deep reinforcement learning algorithms for autonomous driving tasks.

Chapter 3

The Reinforcement Learning Problem

Reinforcement learning (RL) is the branch of machine learning that deals with solving sequential decision-making processes. In the usual setup, an RL problem consists of an agent interacting with an environment, where at each timestep, the agent takes an action and receives an observation and a scalar reward back from the environment (Fig. 3.1). The primary objective that the RL problem seeks, is to maximize the agent's cumulative reward through this interactive process of learning through trial-and-error.

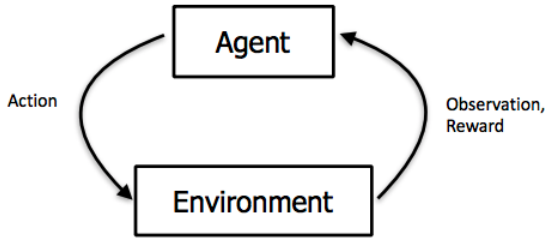

Figure 3.1: RL Setup: An agent interacting with the environment.

This chapter describes some of the preliminaries and the related work that are required for understanding the context of this thesis.

3.1 Markov Decision Processes

The Markov decision processes (MDPs) [11, 45] is the formal framework that describes any RL environment, where the environment is assumed to be fully observable. This implies that the set of states that characterizes the MDP follows the Markovian property, i.e., each of the states captures all the relevant information from the past. Hence the future outcomes are independent of the past given the present state.

Formally, an MDP can be described as (S, A, P, R, γ) whose key components are the following.

- A set of states, $s \in S$, where S denotes the state space.
- A set of actions, $a \in A$, where A denotes the action space.
- A transition probability function, $P(s'|s, a)$, where $s, s' \in S$ and $a \in A$.
- A reward function, $R(s, a, s')$, where $R : S \times S \times A \to \mathbb{R}$ and $R(s, a, s') = \mathbb{E}[r_t|S_t = s, A_t = a, S_{t+1} = s']$.
- Discount factor, denoted by γ.

We now describe the common RL terminology that is widely used across literature and this thesis.

3.1.1 Policy

The policy function, denoted by $\pi(a|s)$, is defined as a probability distribution over actions given the states. It governs the behavior of the agent in the environment and could be either deterministic or stochastic based on the probability distribution function.

3.1.2 Return

The return, denoted by G_t, mathematically describes the cumulative discounted reward from time step t that the RL agent seeks to maximize.

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{3.1}$$

3.1.3 Value Functions

To choose the optimal action from any state (or state-action pairs), almost every RL algorithm estimates a "goodness" score for the agent to be in a given state (or how good it is to perform a given action in a given state). This "goodness" score is defined in terms of the expected return and is defined for every policy π as the future rewards depend on the actions the policy π chooses.

Thus, the state-value function (Eq. 3.2), denoted by $V_\pi(s)$, is defined as the expected return that an agent collects starting from state s and following a policy π thereon.

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\big[\sum_{k=0}^{\infty} \gamma^k r_{t+k}|S_t = s\big] \tag{3.2}$$

Similarly, we also define the action-value function (Eq. 3.3), denoted by $Q_\pi(s, a)$, as the expected return that an agent collects starting from state s, taking the action a and following a policy π thereon.

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\big[\sum_{k=0}^{\infty} \gamma^k r_{t+k}|S_t = s, A_t = a\big] \tag{3.3}$$

The above value functions follow a nice recursive property where the return can be broken down into the sum of immediate reward and the value of its possible successor states (or state-action pairs). These equations (Eq. 3.4) are commonly referred to as the Bellman Expectation equations.

$$\begin{aligned} V_\pi(s) &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V_\pi(s')\big] \\ Q_\pi(s, a) &= \sum_{s',r} p(s', r|s, a)\big[r + \gamma \sum_{a'} \pi(a'|s') Q_\pi(s', a')\big] \end{aligned} \tag{3.4}$$

3.1.4 Optimal Value Functions

In order to solve any RL problem, we seek to maximize the above defined value functions. Thus, the optimal value functions are those that maximize the value function over the policy space and in turn yield the optimal policy, denoted by π^*.

Mathematically, the optimal state-value (or optimal action-value) function, denoted as $V_*(s)$ (or $Q*(s,a)$), can be defined as,

$$\begin{aligned} V_*(s) &= \max_\pi V_\pi(s), \forall s \in S, \\ Q_*(s,a) &= \max_\pi Q_\pi(s,a), \forall s \in S, \forall a \in A. \end{aligned} \tag{3.5}$$

Also, for an optimal policy, the following equation holds true.

$$V_*(s) = \max_{a \in A} Q_{\pi^*}(s,a) \tag{3.6}$$

Using the above optimal set of equations, it follows that these too can be expressed in the recursive form yielding the Bellman Optimality Equations (Eq. 3.7).

$$\begin{aligned} V_*(s) &= \max_{a \in A} \sum_{s',r} p(s',r|s,a)\big[r + \gamma V_*(s')\big] \\ Q_*(s,a) &= \sum_{s',r} p(s',r|s,a)\big[r + \gamma \max_{a' \in A} Q_*(s',a')\big] \end{aligned} \tag{3.7}$$

The Bellman Expectation Equations (Eq. 3.4) and the Bellman Optimality Equations (Eq. 3.7) form the basis for most reinforcement learning algorithms and are essential in order to have a concrete understanding.

For a finite MDP, the Bellman Optimality Equation yields a unique solution independent of the policy. This is because, for a N state system, the Bellman Optimality Equations can be written as N equations in N unknowns. If the dynamics (i.e, $p(s',r|s,a)$) of the environment are known, we can determine v_* using any one of the varieties of methods used to solve systems of nonlinear equations.

In practice, the dynamics (i.e, $p(s',r|s,a)$) of the environment is usually unknown, which brings us to a wide family of reinforcement learning algorithms as detailed in Sec. 3.2.

3.2 RL Algorithms Family

The RL family of algorithms is bifurcated based on whether the agent has access to (or learns) a model of the environment, thereby giving rise to model-free and model-based reinforcement learning methods as shown in Fig. 3.2.

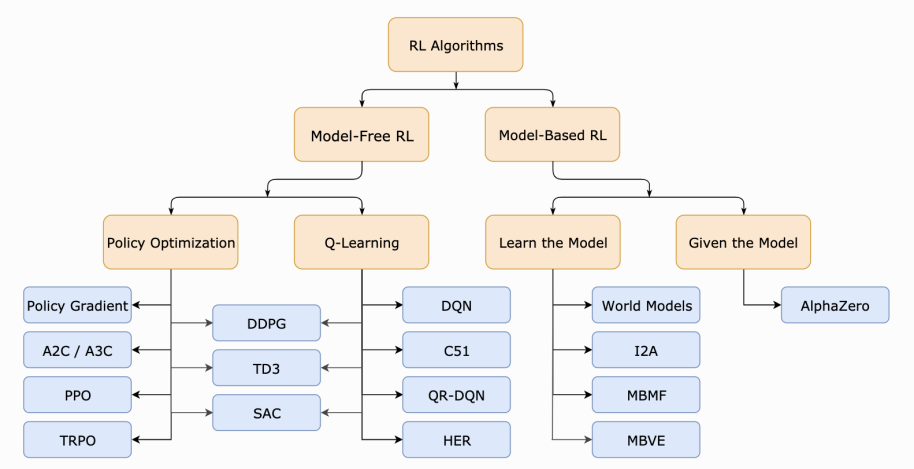

Figure 3.2: Overview of different algorithms in RL. *Credits:* Spinning Up, OpenAI. [3]

- *Model-free methods*: These methods seek to learn the optimal value functions solely through interactions with the environment and explicitly not considering the environment dynamics.

  The model-free methods are further divided into policy optimization and Q-learning based methods. The former parameterizes the policy function and directly optimizes its parameters by doing gradient ascent on the performance objective, or indirectly, by maximizing local approximations of the objective (e.g. policy gradient). The latter methods parameterizes the optimal action-value function, $Q_*(s, a)$, and optimize the objective based on the Bellman equation (e.g. DQN [65], C51 [10], QR-DQN [25]). The intersection of these two methods form the actor-critic methods, that are more common in literature and form the basis for most state-of-the-art model-free approaches (e.g. A2C/A3C [67], DDPG [62], TRPO [88], PPO [89]).

- *Model-based methods*: These methods explicitly seek to learn (or have access to) the model of the environment which allows the agent to plan by thinking ahead, and choose the best actions from a given state. Approaches include pure planning based methods (e.g. MPC [18], MBMF [73]), tree-search algorithms (e.g. MCTS [16]), or joint methods that integrate planning into the policy / value function learning step (e.g. World Models [36], MBVE [31]).

The major upside of having a model of the environment is the substantial improvement in sample efficiency of the learning algorithms whereas the model-free methods are relatively less sample efficient but do provide the ease of implementation and tuning [1, 17, 38, 41, 56, 68, 73].

Another terminology that is common in the RL literature, is the difference between on-policy and off-policy algorithms.

- *On-Policy*: An on-policy algorithm improves or evaluates the same policy that is carried out by the agent including the exploration steps.
- *Off-Policy*: An off-policy algorithm learns the optimal value function independently of the agent's actions, or, improves or evaluates a policy that is different from that used to generate the data.

As discussed in Section 3.1.4, a common way of solving the MDP is to solve the system of equations using dynamic programming for cases where the MDP is finite and we have access to the dynamics of the environment. But this method is too costly owing to the high cost of the matrix inversion ($O(N^3)$) while visiting all the states. One way to solve this is to selectively back up the value function estimates for an MDP or choosing state-actions that the agent visits often. For environments where we do not have access to the dynamics, the agent learns solely through interactions by collecting experiences and estimating the value function based on Monte-Carlo and Temporal-Difference based methods. The backup diagram and equations for these methods can be summarized from Fig. 3.3.

Next, we briefly review Q-Learning based methods in Section 3.3 and dive deeper into the policy optimization-based methods in Section 3.4, that forms the primary focus area of this thesis.

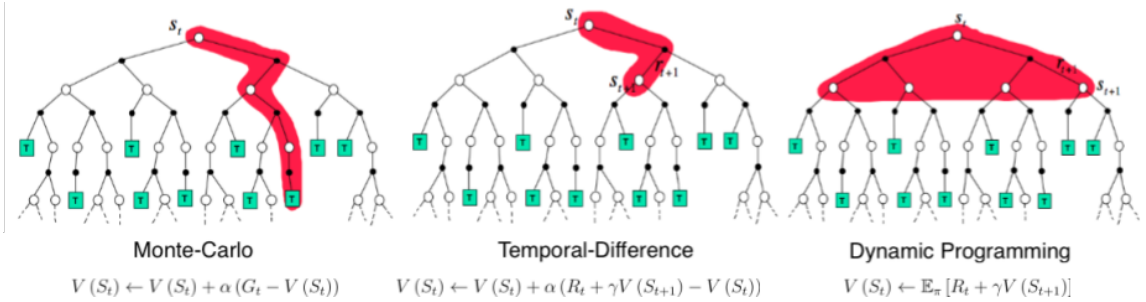

Figure 3.3: Tree backup diagram for different class of RL algorithms. *Left:* Monte-Carlo methods use empirical average of returns to estimate the value function. *Centre:* Temporal-Difference methods estimate the value function based on estimate of the next state (or state-action). *Right:* Dynamic Programming methods solve the entire backup tree to estimate the expected return.

3.3 Q-Learning

Q-Learning [102], one of the foremost breakthroughs in reinforcement learning, was the first algorithm that proposed the development of an off-policy Temporal-Difference control algorithm. In this method, an agent learns the optimal action-value function $Q_*(s, a)$ based on historical experiences that are collected by the old policy. This facilitates the agent to estimate the optimal action-value function under the assumption that the agent can explore all actions from all states infinitely many times.

Deep Q-Learning [65], extends the above idea to learn action-value function approximator in the form of deep networks and reports significant improvement in performance as compared to all previous methods. This work was further extended to develop Deep Deterministic policy gradients (DDPG) [62] that concurrently learn an action-value and policy function for environments with continuous action spaces. Another work that is currently state-of-the-art in off-policy reinforcement learning is the Soft-Actor Critic [37] that does stochastic policy optimization in a maximum entropy RL framework, thus optimizing the trade-off between the expected reward and entropy.

3.4 Policy Optimization

Recalling the primary objective of RL problems, the goal is to maximize the expected reward following a policy π. The policy gradient methods aim to model and optimize the policy parameters directly with the key goal to increase the probabilities of actions that lead to higher returns and decrease the probabilities of actions that lead to lower returns. The policy, defined as $\pi_\theta(a|s)$ where θ denote the set of policy parameters, governs the value of the objective function, which is denoted as $J(\theta)$.

This objective function can be maximized either through gradient-free approaches or gradient ascent algorithms that find the best θ that produces the highest return. Mathematically, the performance objective or the reward function is defined by Eq. 3.8 where $d_\pi(s)$ is the stationary distribution of the Markov chain for π_θ.

$$J(\theta) = \sum_{s \in S} d_\pi(s) V_\pi(s) = \sum_{s \in S} d_\pi(s) \sum_{a \in A} \pi_\theta(a|s) Q_\pi(s,a) \tag{3.8}$$

3.4.1 Policy Gradient

Computing the gradient of the performance objective, $J(\pi_\theta)$, is tricky as the performance depends on both the action selections and the distribution of states in which those selections are made. Given that the environment is usually unknown it becomes difficult to estimate the effect of the policy update on the state distribution. Fortunately, the policy gradient theorem (Eq. 3.9) provides a solution to this problem by giving an analytic expression for the gradient of performance with respect to the policy parameter θ.

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \sum_{s \in S} d_\pi(s) \sum_{a \in A} \pi_\theta(a|s) Q_\pi(s,a) \\ &\propto \sum_{s \in S} d_\pi(s) \sum_{a \in A} \nabla_\theta(\pi_\theta(a|s)) Q_\pi(s,a) \end{aligned} \tag{3.9}$$

In the episodic case, the proportionality constant is equal to the average length of the episode and for the continuing case, it is considered to be equal to 1. Thus the above policy gradient equation (Eq. 3.9) can further be written as (Eq. 3.10),

$$\begin{aligned}\nabla_\theta J(\theta) &\propto \sum_{s\in S} d_\pi(s) \sum_{a\in A} \nabla_\theta(\pi_\theta(a|s)) Q_\pi(s,a) \\ &= \sum_{s\in S} d_\pi(s) \sum_{a\in A} \frac{\nabla_\theta(\pi_\theta(a|s))}{\pi_\theta(a|s)} \pi_\theta(a|s) Q_\pi(s,a) \\ &= \mathbb{E}_{s\sim d_\pi, a\sim\pi_\theta}[\nabla_\theta(\ln \pi_\theta(a|s)) Q_\pi(s,a)].\end{aligned} \tag{3.10}$$

3.4.2 REINFORCE

REINFORCE [95, 103], also known as Monte Carlo policy gradient, uses Monte-Carlo methods to estimate returns of sampled episodes that in turn update the policy parameter θ. The key idea behind this algorithm is that the expectation of the sample gradient is equal to the actual policy gradient.

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{s\sim d_\pi, a\sim\pi_\theta}[\nabla_\theta(\ln \pi_\theta(a|s)) Q_\pi(s,a)] \\ &= \mathbb{E}_\pi[\nabla_\theta(\ln \pi_\theta(A_t|S_t)) G_t],\end{aligned} \tag{3.11}$$

since $Q_\pi(S_t, A_t) = \mathbb{E}_\pi[G_t|S_t, A_t]$.

A commonly used variation of REINFORCE [95, 103] (Eq. 3.12) is to subtract a suitable baseline from the return G_t to reduce the variance of the gradient estimates while keeping the bias unchanged. Often, the state-value function is used as a baseline that is subtracted from the action-value function, yielding the advantage function, denoted by $A_\pi(s,a)$ (Eq. 3.13).

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[\nabla_\theta(\ln \pi_\theta(A_t|S_t)) A_\pi(S_t, A_t)] \tag{3.12}$$

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{3.13}$$

3.4.3 Trust Region Policy Optimization

The vanilla policy gradient methods, keep the updated and old policies close in the parameter space. But this optimization is difficult as even a small change in the

parameter space may yield very large differences in the performance of the policies. Thus, determining the right step size, without hurting the policy performance is cumbersome, which is where the beauty of Trust Region policy optimization (TRPO) [88] lies.

TRPO [88] avoids the policy performance collapse by constraining the updated and old policies in terms of KL-divergence and monotonically improving the performance objective.

Although TRPO [88] is considered to be an on-policy algorithm, it considers the subtle difference in the behavior policy and the policy we seek to optimize for. Thus, the performance objective equation (Eq. 3.8) includes an importance sampling term that compensates for the mismatch between the training data distribution and the true policy state distribution, yielding Eq. 3.14.

$$\begin{aligned} J(\theta) &= \sum_{s \in S} \rho_{\pi_{\theta_{old}}}(s) \sum_{a \in A} \pi_{\theta_{old}}(a|s) \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A_{\pi_{\theta_{old}}}(s,a) \\ &= \mathbb{E}_{s \sim \rho_{\pi_{\theta_{old}}}, a \sim \pi_{\theta_{old}}} \big[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A_{\pi_{\theta_{old}}}(s,a) \big] \end{aligned} \tag{3.14}$$

TRPO [88] aims to maximize the objective function (Eq. 3.14), subject to the following trust region constraint (Eq. 3.15) that enforces the old and updated policies to not diverge much.

$$\mathbb{E}_{s \sim \rho_{\pi_{\theta_{old}}}} \Big[D_{\mathrm{KL}} \big(\pi_{\theta_{old}}(.|s) || \pi_\theta(.|s) \big) \Big] \leq \delta \tag{3.15}$$

3.4.4 Proximal Policy Optimization

Although TRPO [88] proposes an algorithm that offers a monotonic improvement to the policy function, it solves a second-order optimization problem that is complicated and difficult to implement. On the other hand, Proximal policy optimization (PPO) [89] proposes a first-order optimization that aims to optimize a "surrogate" objective function and in practice performs equivalent to current on-policy based algorithms. This method is significantly simpler to implement and offers a reasonable balance between sample complexity, simplicity, and wall-time.

Algorithm 1 Vanilla PPO-Clip Algorithm

Input: initial policy parameters θ_0, initial value function parameters ϕ_0, trajectory length T, total training steps S
for step $= 1, 2, \ldots, S//T$ **do**
    Collect set of trajectories $\mathcal{D}$ of length T by running policy $\pi = \pi\left(\theta_{old}\right)$ in the environment.
    Compute rewards-to-go $\hat{R}_1 \ldots, \hat{R}_T$.
    Compute advantage estimates, $\hat{A}_1, \ldots, \hat{A}_T$ based on the current value function $V_{\phi_{old}}$.
    Update the policy by maximizing the PPO-Clip objective (Eq. 3.16):

$$\theta = \arg\max_{\theta} \frac{1}{|\mathcal{D}|\,T} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \min\left(r(\theta) A_{\pi_{\theta_{old}}}(s_t, a_t), \text{clip}\Big(r(\theta), 1-\epsilon, 1+\epsilon\Big) A_{\pi_{\theta_{old}}}(s_t, a_t) \right)$$

    typically via stochastic gradient ascent with Adam.
    Fit value function by regression on mean-squared error:

$$\phi = \arg\min_{\phi} \frac{1}{|\mathcal{D}|\,T} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \left(V_{\phi}\left(s_t\right) - \hat{R}_t \right)^2$$

    typically via stochastic gradient descent with Adam.
end for

There are two primary variants of PPO [89] that are each discussed below.

- PPO-Penalty: This variant solves a KL-constrained update like TRPO, but penalizes the KL-divergence in the objective function rather than imposing a hard constraint and automatically adjusts the penalty coefficient over the course of training.
- PPO-Clip: This variant neither has an explicit KL-divergence term in the objective function nor has any constraints. Rather, it uses a surrogate objective function that clips the objective function and keeps the new policy close to the old policy.

Empirically, the second version, PPO-Clip [89] (Algo. 1), performs better and is the one that is more widely used. The clipped objective function used by this variant can be defined by Eq. 3.16,

$$J(\theta) = \mathbb{E}_{s,a\sim\pi_{\theta_{old}}}\Big[\min\Big(r(\theta)A_{\pi_{\theta_{old}}}(s,a), \text{clip}\big(r(\theta), 1-\epsilon, 1+\epsilon\big)A_{\pi_{\theta_{old}}}(s,a)\Big)\Big] \quad (3.16)$$

where $r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$ is the probability ratio. The motivation for this modified surrogate objective function is as follows. The first term inside the min is the same as that defined in 3.14, whereas the second term, $\text{clip}\big(r(\theta), 1-\epsilon, 1+\epsilon\big)A_{\pi_{\theta_{old}}}(s,a)$, clips the range of the probability ratio to $[1-\epsilon, 1+\epsilon]$ and the minimum is chosen depending on the sign of the advantage function $A_{\pi_{\theta_{old}}}(s,a)$. The modified objective function forms a lower bound to the original performance objective function (Eq. 3.14) and the clipping is triggered only when the objective is impacted. This clipping parameter also acts as a regularizer by removing incentives for the policy to change abruptly by pessimistically constraining the new policy with respect to the old policy, while still maximizing the performance objective function. The overall algorithm can be outlined in Algo. 1.

In this thesis, we primarily build upon the PPO-Clip variant of Proximal policy optimization [89] and present two works (Chap. 5, 6) that demonstrate the effectiveness of using reinforcement learning methods to learn even the complex tasks of learning to drive in urban settings.

Chapter 4

CARLA Environment

Figure 4.1: CARLA Simulator [29]: A realistic urban-driving simulator this is open-sourced for autonomous driving research. The simulator supports multiple actors, pedestrians, weather conditions and maps that are each configurable in the simulator. *Credits:* CARLA [29]

In this chapter, we describe and detail the CARLA environment (Fig. 4.1) that enables our agent to interact with the simulator [29] and learn optimal driving policies using reinforcement learning approaches presented by us in Chap. 5, 6. We first describe the server-client communication framework that we set up using the simulator, then dive into the array of sensors used by us. Further, we discuss our waypoint computation logic and algorithms developed by us to detect obstacles and traffic light

state. These components (Sec 4.1 - 4.4) combined together form the core building blocks of our environment. Finally, we describe the original CARLA benchmark [29] and the NoCrash benchmark [24] that defines the task for our reinforcement learning agent and forms the basis for comparison with our baseline methods (Sec. 5.4.1, 6.4.1).

4.1 Server-Client Interface

CARLA [29] provides a simple server-client interface where the server is responsible for running the simulation and rendering the dynamic world which enables the client to interact with it. The client sends control and environment commands to the server and receives the sensor readings in return. The entire communication is managed via TCP sockets with the client API's exposed in Python.

We build our environment on top of the server-client interface provided by CARLA [29] and use the synchronous mode of communication that runs at a frequency of $10 fps$.

4.2 Sensors

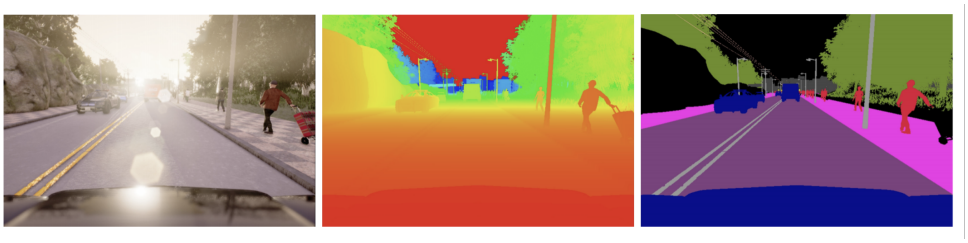

Figure 4.2: The CARLA simulator [29] supports a wide variety of sensors that include the LIDAR, RGB camera, semantic camera, depth sensors and GPS units. The above figure three shows such sensors. *Left:* normal vision camera, *Centre:* ground-truth depth camera and *Right:* ground-truth semantic segmentation camera. *Credits:* CARLA [29]

CARLA [29] offers a variety of sensor suite (Fig. 4.2) that is configurable on the agent's end. To develop our environment and train our reinforcement learning approaches, we equip our agents with the following sensors to determine their state with respect to the simulated world.

- *Semantic Segmentation Camera*: The semantic segmentation camera in CARLA classifies every object in its view based on 13 predetermined classes. We use the output of this sensor and map it further to 5 important classes that include *Pedestrian, Road Line, Road, Car,* and *Everything Else*. This forms as one of the inputs to our architecture (Fig. 5.2, 6.2) that define our state representation (Sec. 5.1, 6.1).
- *Collision sensor*: This sensor enables our agent to detect collisions with any kind of objects that are present in the simulated world. Specifically, we consider all types of collisions with static and other vehicle actors as infractions and we terminate our agent when this sensor triggers a collision event.
- *Lane-invasion sensor*: This sensor enables us to detect infractions like invading the opposite lane or getting onto the sidewalk. It detects the crossing of any lane markings which helps us determine the episode termination condition for such infractions.

4.3 Waypoint Planner

In the next two chapters (Chap. 5, 6), we present our formulation of learning to drive as goal-directed navigation. Since the visual input is not sufficient to disambiguate the navigation direction, we propose to use waypoints that direct our learning agent to a future world location that lies on the optimal trajectory. These waypoints are computed using a heuristic-based A* search algorithm that determines the optimal trajectory between a pair of start and goal destination points at a predefined resolution of w_d m. This process is repeated for every goal-directed training episode. Additionally, we use the planner implementation that is provided within the CARLA release[1].

4.4 Intermediate Affordances

For the task of learning to drive with dynamic actors (Chap. 6), we formulate the RL problem by defining a pair of affordances: the front *Dynamic Obstacle Affordance*

[1]CARLA v0.9.6 - https://carla.org/2019/07/12/release-0.9.6/

and the *Traffic Light Affordance*. The goal behind proposing these low-dimensional affordances is to simplify the policy learning problem by predicting these intermediate affordances through a separate perception system.

- *Dynamic Obstacle Affordance* : The dynamic obstacle affordance encodes the front actor's state in a lower-dimensional intermediate representation. Particularly, we aim to encode the distance between our agent and the front dynamic actor as well as its corresponding speed. Since the CARLA simulator [29] gives us direct access to the speed and position of any actor, the task reduces to determining if a target dynamic actor exists in front of our agent. This is done by iteratively checking if a target actor's bounding box lies in front of our agent that is within a threshold distance of d_{prox}.
- *Traffic Light Affordance* : Similarly, we also propose the traffic light affordance that encodes the front traffic light's state in a lower-dimensional intermediate representation. This affordance aims to encode the distance to the nearest traffic light in front of our agent. This is done in a similar way like the dynamic obstacle affordance with an extension that the target light's road and lane identifier matches our agent's road and lane identifier. The proximity threshold distance is given by t_{prox}.

4.5 Benchmarks

4.5.1 Original CARLA Benchmark

The original CARLA benchmark [29] proposed four driving tasks. Each task is set up as goal-directed navigation where the agent is initialized randomly at some location in the town and the goal is to reach a destination point. The benchmark includes two towns out of which *Town 01* is used for training whereas the *Town 02* is used for testing. The tasks are organized in the increasing order of their difficulty (Table 4.1) that includes driving on straight roads, around intersections, and navigating through the town with and without dynamic actors. The first three tasks (*Straight*, *One Turn*, *Navigation*) do not include any dynamic actors whereas the fourth task (*Dynamic Navigation*) includes dynamic actors with navigation from start point to goal point.

Task	Drive Straight	Drive w. one turn	Drive w. multiple turns	Drive w. dynamic actors
Straight	✓			
One Turn	✓	✓		
Navigation	✓	✓	✓	
Dynamic Navigation	✓	✓	✓	✓

Table 4.1: Original CARLA Benchmark [29] that describes the different driving tasks: *Straight*, *One Turn*, *Navigation* & *Dynamic Navigation*.

The benchmark [29] proposed evaluating each of these tasks based on the success rate. It considers an episode as a success if the agent reaches the goal regardless of any collisions or other infractions that it may encounter during the episode. Hence, this benchmark mainly focuses on evaluating skills such as lane-keeping and turning around intersections while ignoring the urban driving setting that commonly has complex interactions due to dynamic agents or changing traffic light behavior [24].

4.5.2 NoCrash Benchmark

Owing to the limitations of the original CARLA benchmark [29], a new NoCrash [24] benchmark was proposed that evaluates the agent behaviour on three different driving tasks: *Empty Town*, *Regular Traffic* and *Dense Traffic*. Each task consists of similar goal-directed episodes as the original benchmark [29] (Sec. 4.5.1), while increasing the difficulty in terms of the number of cars and pedestrians. The training is performed in *Town 01* whereas the testing occurs in *Town 02*.

The original benchmark considered a goal conditioned success rate that allows the episode to continue even after the infraction. This metric was improved in the NoCrash benchmark that considers an episode as a failure if the agent collides with any object. For traffic lights, the benchmark suggests counting the traffic light violations separately while not terminating the episode due to this infraction.

Chapter 5

Learning to Drive using Waypoints

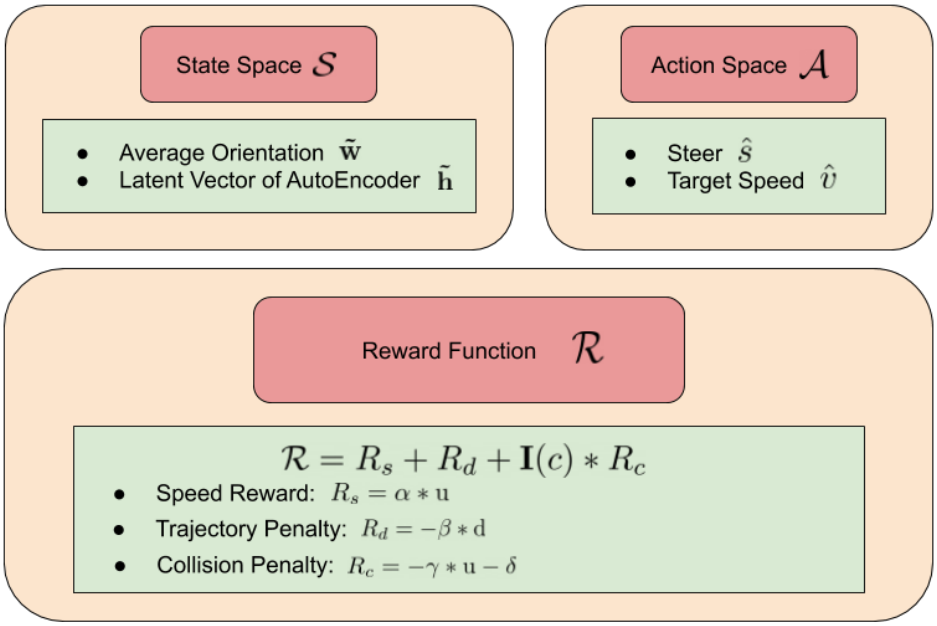

Figure 5.1: Our proposed RL setup that defines our Markov decision process with the state space $\mathcal{S}$, action space $\mathcal{A}$ and reward function $\mathcal{R}$.

We see in Chap. 2 that most of the literature in the autonomous driving domain is predominantly clustered into the modular or imitation learning-based approaches. These approaches often do not scale well to urban driving scenarios where it is common to encounter a variety of interactions between our autonomous agent and traffic lights, pedestrians, and other actors at intersections. The major challenge with the urban driving task is that complexity of the classical rule-based approaches

increases as it becomes difficult to handle such high variability. On the other hand, end-to-end approaches that use imitation learning often suffer from a distribution mismatch between the expert labeled data and the data seen at test-time.

In contrast, reinforcement learning algorithms do not suffer from distribution mismatch as the agent continues to explore new state while still exploiting states that lead to higher cumulative rewards. This motivates us to pose the driving problem as a sequential decision making task and use the classical formulation of reinforcement learning (Chap. 3). Further, we believe that the paradigm of reinforcement learning has a strong potential to learn autonomous driving behaviors towards which we present our first approach in this chapter.

Specifically, we formulate the urban driving problem as a Markov decision process (Sec. 3.1) where the goal of our autonomous agent is to learn optimal driving policy using a reinforcement learning algorithm [4]. The agent seeks to maximize the sum of cumulative rewards that it can collect from the environment. Since the dynamics of the environment is usually difficult to model or unknown, we take a model-free approach to reinforcement learning (Sec. 3.2) where the agent iteratively evaluates the current policy and updates it solely through interaction with the environment.

As RL relies on learning through interactions via trial and error, we develop our framework using the CARLA[1] [29] simulator due to safety and data efficiency constraints. We now describe the RL setup in Sec. 5.1, discuss our model-free reinforcement learning algorithm in Sec. 5.2 and the training methodology Sec. 5.3. Further, we describe our experimental setup in Sec. 5.4 followed by results and discussion in Sec. 5.5 and Sec. 5.6 respectively.

5.1 RL Setup

The major contribution of this work [4] is setting up the autonomous driving task in an RL setup. We develop our own CARLA RL environment (Chap. 4), define our own state space $\mathcal{S}$, action space $\mathcal{A}$ and reward function $\mathcal{R}$ (Fig. 5.1) that facilitates our agent to interact with the simulator and learn solely from the interactions.

[1]CARLA v0.9.6 - https://carla.org/2019/07/12/release-0.9.6/

5.1.1 State Space

The key to the design of our state-space $\mathcal{S}$ is defining the observation O_t that feeds into the algorithm at each time step. Theoretically, state s_t should follow the Markov property that encodes all previous observations which can be done using a recurrent neural network [22, 43]. However, for our task, we consider the observation to itself approximate the state.

In the autonomous driving literature, many sensors have been proposed that provide sophisticated observations for driving algorithms, not limited to cameras, LiDARs, IMUs, GPS units, and IR depth sensors. Although these sensors are equipped with advanced sensing capabilities, they are often not cost-efficient. Conversely, in this work, we propose to use only cameras and GPS units that readily provide the visual and navigational information that is required for an agent to navigate a complex urban scene.

Specifically, we choose the bird's-eye view (BEV) semantically segmented image ($\mathbf{SS}_{\text{image}}$) as one of our state-input that is easily obtained from the CARLA's semantic segmentation camera sensor [4]. Given the current state-of-the-art architectures in perception, we believe segmentation as a task can be trained in isolation and hence we focus on the more complex task of learning to drive using deep reinforcement learning directly from SS images. To that end, we consider feeding in the semantic segmented images through a stack of convolutions [34, 54] or learning a smaller representation of the image, using a Convolutional or Variational Autoencoder (CAE or VAE) [52, 63]. We notice that learning a deterministic latent representation as in the case of an autoencoder, has a stabilizing effect on policy learning. The AE bottleneck embedding forms as one of the state space inputs to our agent's policy network (Fig. 5.2). We refer to the AE bottleneck embedding as $\tilde{\mathbf{h}}$, and define it in Eq 5.1 where e denotes the encoder function of our AE. The reconstructed semantic segmented image ($\mathbf{SS}_{\text{rimage}}$) is decoded using the decoder function d of our AE (Eq. 5.1).

$$\begin{aligned} \tilde{\mathbf{h}} &= e(\mathbf{SS}_{\text{image}}) \\ \mathbf{SS}_{\text{rimage}} &= d(\tilde{\mathbf{h}}) \end{aligned} \tag{5.1}$$

Besides the visual state input, the agent also requires an input to guide its

navigation. Past approaches [23, 29, 61, 86] have used a higher level planner that directs the agent using high-level commands on intersections. Instead of this, we propose to use trajectory waypoints [4] that are readily available from the GPS units and used to guide the agent's navigation. Given a source and destination location, waypoints are intermediate locations pre-computed at a fixed resolution using standard pathfinding algorithms and can be fetched easily from the CARLA simulator (Sec. 4.3). We believe the features computed from waypoints can provide a richer signal to the learning agent for navigation. The waypoint features $\tilde{\mathbf{w}}$ are computed using some generic function f defined by the next n waypoints $(\mathbf{w_1}, \mathbf{w_2}, ..., \mathbf{w_n})$ and agent's current pose $\mathbf{p}$. These features $\tilde{\mathbf{w}}$ form the second input to our agent policy network [4] as defined in Equation (5.2).

$$\tilde{\mathbf{w}} = f(\mathbf{p}, \mathbf{w_1}, \mathbf{w_2}, ..., \mathbf{w_n}) \tag{5.2}$$

For simplicity, we define the function f as the average angle between the agent's current pose $\mathbf{p}$ and the next n waypoints $(\mathbf{w_1}, \mathbf{w_2}, ..., \mathbf{w_n})$ but this can extended to work with any possible functional form. Mathematically, the waypoint feature function f can be defined by Eq. 5.3.

$$\tilde{\mathbf{w}}_\theta = \frac{1}{n} \sum_{i=1}^{n} \left(\theta_{\mathbf{p}} - \theta_{\mathbf{w_i}}\right) \tag{5.3}$$

This leads to our final state representation [4], denoted by $[\tilde{\mathbf{h}}.\tilde{\mathbf{w}}]$, that concatenates the latent features of the autoencoder $\tilde{\mathbf{h}}$ with the waypoint features $\tilde{\mathbf{w}}$.

5.1.2 Action Space

For our autonomous driving agent, it is natural to include the steer, throttle, and break actions, denoted by (s, t, b) respectively, in the action space $\mathcal{A}$, which also form the control input to the CARLA simulator. But these actions could either be discretized or be applied in the continuous domain. In this work [4], we primarily choose to focus on learning continuous action policies.

As the end control predictions may be noisier, we also reparameterize the throttle and brake actions in terms of a target speed set-point. Thus the predicted throttle and brake actions, denoted by $\hat{t}$ and $\hat{b}$ respectively, form the outputs of a classical PID

controller [7, 84] that attempts to match the set-point. This smoothens the control response as well as simplifies learning the continuous actions to just two actions, steer, and target speed. The predicted steer action, denoted by $\hat{s}$ lies in the range of $[-0.5, 0.5]$ that approximately maps to $[-40°, 40°]$ of steering angle whereas the predicted target speed, denoted by $\hat{v}$, lies in $[-1, 1]$ range and linearly maps to $[0, 20]$ km/h.

5.1.3 Reward Function

We know that the reward signal in RL is the sole feedback that guides the agent towards optimal actions conditioned on the state. Hence, we propose a simple and dense reward function [4] to make the optimization easier and which also correlates with how humans learn the driving task. The reward function $\mathcal{R}$ (Eq. 5.4) incentivizes our agent to continue moving towards the goal destination until an infraction occurs that dictates if the episode is to be terminated. Overall, our reward function [4] can be decomposed into three basic components as follows and mathematically be defined by Eq. 5.4.

$$\mathcal{R} = R_s + R_d + \mathbf{I}(c) * R_c \tag{5.4}$$

- *Speed-based Reward (R_s)*: This reward incentivizes the agent to learn the throttle action as it receives a reward that is directly proportional to the current speed u of the agent. Over time, the cumulative reward equals the total distance moved by it from the start point.

$$R_s = \alpha * \mathrm{u} \tag{5.5}$$

- *Distance-based Penalty from Optimal Trajectory (R_d)*: This reward penalizes the agent based on the lateral distance d between the centre of our agent and the optimal trajectory that is precomputed by the planner. This incentivizes the agent to stay close to the optimal planned trajectory.

$$R_d = -\beta * \mathrm{d}; \tag{5.6}$$

- *Collision Penalty (R_c)*: This reward penalizes the agent when it collides (denoted by the indicator function $\mathbf{I}(c)$) with other static or dynamic objects. The penalty incentives the agent to drive safely without any collision infractions.

$$R_c = -\gamma * \mathrm{u} - \delta \tag{5.7}$$

The design of our reward function $\mathcal{R}$ aligns with our intuition that this objective is similar to the objective utilized by most classical planners that determine the best action to be executed from a given state.

5.2 Model-Free RL with Learned Representations

Since we formulate the RL problem for a continuous action domain, we select a state-of-the-art on-policy model-free reinforcement learning algorithm: Proximal policy optimization (PPO) [89] (Sec. 3.4.4), and combine it with our proposed Autoencoder (AE) setup (Sec. 5.1.1). Although the on-policy reinforcement learning algorithms are sample inefficient when compared to off-policy methods, we believe that their performance is much more stable which makes it an easier choice to study the impact of reinforcement learning for autonomous driving. We show that using an off-the-shelf RL algorithm combined with the autoencoder solves the task of learning optimal driving policies, by solving the Markov decision process as defined by us in Sec. 5.1.

We propose a stable learning algorithm (Algo. 2) [4] that uses a pre-trained autoencoder and finetunes it simultaneously with the policy learning step. We conjecture that the pre-training step helps in generating better state representations during the initial phase of the training whereas the finetuning step facilitates the encoder-decoder to improve its representation as the RL algorithm explores diverse states. The autoencoder function approximator $d(e(\mathbf{SS}_{\text{image}}))$, parameterized by β, optimizes for the multi-class cross-entropy loss summed over all the pixels (Eq. 5.8). For a fixed number of time steps (n), the autoencoder is frozen and is updated only using the n semantically segmented images that it collects over that interval. PPO,

Algorithm 2 Learning to Drive via Model-Free RL on Learned Representations

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0, pretrained auto-encoder parameters β_0,
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy π_{θ_k} in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective (Eq. 3.16).

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|\,T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(r(\theta) A_{\pi_{\theta_k}}(s_t, a_t), \text{clip}\Big(r(\theta), 1-\epsilon, 1+\epsilon\Big) A_{\pi_{\theta_k}}(s_t, a_t) \right)$$

7: Fit value function by regression on mean-squared error.

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|\,T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \Big(V_\phi\left(s_t\right) - \hat{R}_t \Big)^2$$

8: Every fixed n steps, finetune the autoencoder based on cross-entropy loss.

$$\beta_{k+1} = \arg\min_{\beta} \sum_{p \in \mathbf{SS}_{\text{image}}} \sum_{c \in \mathcal{C}} \left(-t_c(p) \log o_c(p)\right)$$

9: **end for**

on the other hand, trains an on-policy RL algorithm (Sec. 3.4.4) that uses a surrogate objective function to optimize for the policy performance (Eq. 3.8). It consists of training two function approximators. The first being a critic function approximator $V_\phi : \mathcal{S} \to \mathbb{R}$ that estimates the expected cumulative discounted reward starting from state s and executing policy π thereon. The second approximator is the policy function $\pi_\theta : \mathcal{S} \to \mathcal{A}$ that does policy gradient update based on the clipped objective (Eq. 3.16).

$$\begin{aligned} L(\beta) &= \sum_{p \in \mathbf{SS}_{\text{image}}} \sum_{c \in \mathcal{C}} \left(-t_c(p) \log o_c(p)\right) \\ t_c(p) &= \text{One-hot}(p) \\ o_c(p) &= \text{Softmax}(d(e(p))) \end{aligned} \tag{5.8}$$

The autoencoder encodes the $\mathbf{SS}_{\text{image}}$ to give its latent representation $\tilde{\mathbf{h}}$ which is fused with the waypoint features $\tilde{\mathbf{w}}$ to form the state input $\pi(\hat{s}, \hat{v}|[\tilde{\mathbf{h}}.\tilde{\mathbf{w}}])$, as defined in Sec. 5.1.1. This state representation feeds into both the policy and critic networks, denoted by $\pi(\hat{s}, \hat{v}|[\tilde{\mathbf{h}}.\tilde{\mathbf{w}}])$ and $V([\tilde{\mathbf{h}}.\tilde{\mathbf{w}}])$ respectively (Figure 5.2). Since PPO can be used for continuous action domain, the policy network outputs the mean of the normal distribution through which the actions are sampled during the training time. The variance is added as a separate parameter to aid the exploration process and is optimized by adding in a entropy bonus term [89] to the clipped objective. Over the training time, the randomness in the actions decreases, as the algorithm encourages it to exploit rewards that it has already found, while still maintaining some entropy in its actions.

5.3 Training

To train our proposed algorithm [4], we first pre-train the autoencoder based on the data collected by a mixture of expert and RL training policies. Next, we iteratively sample a random episode as defined by the benchmark task (Sec. 4.5) that determines start and goal location. A static planner then precomputes a list of waypoints that map to the optimal trajectory of the agent (Sec. 4.3) and rewards are calculated as the agent steps through the environment at each time step. Subsequently, we train the policy and critic networks along with finetuning the autoencoder after every n steps based on our Algorithm 2 as discussed in Sec. 5.2, until convergence. The episode is terminated as a success if the agent reaches within d m of the destination, while it is terminated as a failure if the agent faces a collision, or fails to reach near destination within m number of maximum time steps. The episode is not terminated for a traffic light or opposite lane-invasion or sidewalk infractions.

The actor and critic networks consist of a 2-layer feedforward network whereas the autoencoder consists of 4 Conv and 4 Deconv layers. All the networks are trained with a ReLU non-linearity and optimized using stochastic gradient descent with the Adam optimizer. The overall architecture of our approach can be depicted in Fig. 5.2.

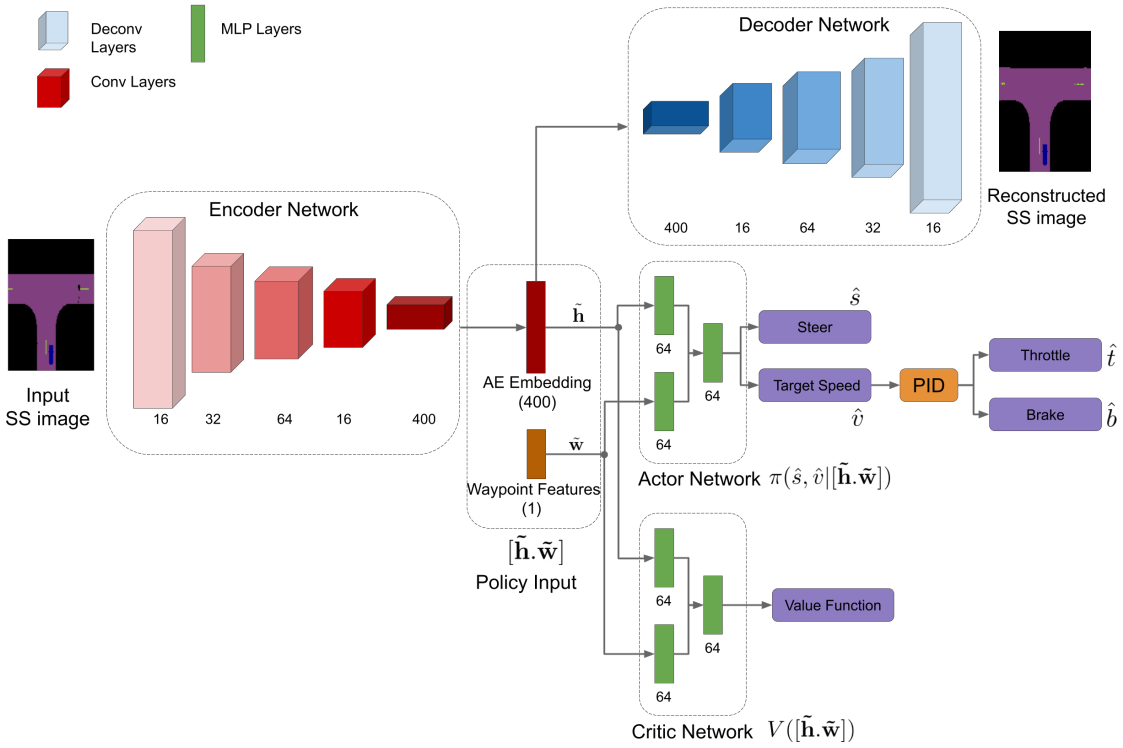

Figure 5.2: Our proposed architecture [4]: The inputs to our architecture are semantically segmented ($\mathbf{SS}_{\text{image}}$) image and intermediate waypoints that are directly fetched from the CARLA simulator. The $\mathbf{SS}_{\text{image}}$ is encoded using a pre-trained auto-encoder whose bottleneck encoding alongwith waypoint features form as inputs to the policy network. The policy network outputs the control actions $(\hat{s}, \hat{v})$ where $\hat{s}$ is the predicted steer and $\hat{v}$ is the predicted target speed which is then mapped to predicted throttle and brake $(\hat{t}, \hat{b})$ using a PID controller.

5.4 Experiments

To evaluate our proposed RL algorithm (Algo. 2) [4], we build on top of the open-source stable-baselines[2] implementation [42]. We train our agents in the CARLA simulator [29] by formulating the task of learning to drive using the Markov decision process and environment as defined in Sec. 5.1 and Chap. 4 respectively. We use the original CARLA benchmark tasks [29] (Sec. 4.5.1) in *Town 01* for training and *Town 02* for evaluating our agent. Since the original CARLA benchmark does not

[2] https://github.com/hill-a/stable-baselines

provide accurate evaluation of our agents on the more complex urban driving tasks, we extend our evaluation to the newer NoCrash benchmark [24] (Sec. 4.5.2) as well. Our evaluation analysis uses the same agent on all the tasks and we do not explicitly fine-tune separately for any scenario.

5.4.1 Baselines

We compare our work [4] with the following baselines that solve the goal-directed navigation task using an either modular approach, end-to-end imitation learning, or reinforcement learning. Since most of the works do not have open-source implementations available or report results on the older versions of CARLA, we report the numbers directly from their work.

- CARLA *MP*, *IL* & *RL* [29]: These baselines, proposed in the original CARLA work [29] suggest three different approaches to the autonomous driving task. The modular pipeline (MP) uses a vision-based module, a rule-based planner, and a classical controller. The imitation learning (IL) one learns a deep network that maps sensory input to driving commands whereas the reinforcement learning (RL) baseline does end-to-end RL using the A3C algorithm [67].
- *AT*: This baseline refers to the CARLA built-in autopilot control that uses a hand-engineered approach to determine optimal control.
- *CIL* [23]: This work proposes a conditional imitation learning pipeline that learns a driving policy from expert demonstrations of low-level control inputs, conditioned on the high-level navigational command.
- *CIRL* [61]: This work proposes to use a pre-trained imitation learned policy to carry-out off-policy reinforcement learning using the DDPG algorithm [62].
- *CAL* [86]: This baseline proposes to learn a separate visual encoder that predicts low-dimensional representations, also known as affordances, that are essential for urban driving. These representations are then fused with classical controllers.
- *CILRS* [24]: This work builds on top of CIL [23] to propose a robust behavior cloning pipeline that generalizes well to complex driving scenarios. The method suggests learning a ResNet architecture [39] that predicts the desired control as well as the agent's speed to learn speed-related features from visual cues.

- *LBC* [20]: This work decouples the sensorimotor learning task into two, learning to see and learning to act. In the first step, a privileged agent is learned that has access to the simulator states and learns to act. The second step involves learning to see that learns an agent based on supervision provided by the privileged agent.
- *IA* [99]: This work proposes to learn a ResNet encoder [39] that predicts the implicit affordances and uses its output features to learn a separate policy network optimized using DQN algorithm [65].

Although all of the above baselines use RGB image and high-level navigational command as inputs, we acknowledge the differences in our inputs and show that our results are comparable or even better with our simplified representation. The results align with our belief that with an ideal perception system our approach using reinforcement learning can beat the current decision making and control systems put together.

We note that since the CARLA 0.9.6 version supports pedestrians that only move along the sidewalks unlike the earlier versions, we limit our baselines and benchmark comparison to not include pedestrian actors. Additionally, all the prior methods except *LBC* [20] and *IA* [99] report results on CARLA versions prior to 0.9.6. Although the newer CARLA 0.9.6 version included a significant rendering engine and pedestrian actor change, our choice of input representation makes us affirm that the difference is not significant across CARLA 0.9.X versions. We also note that the waypoint feature did not exist in CARLA 0.8.X versions. Thus, we report our results on CARLA 0.9.6 and compare our method with all the prior work without any reservations.

5.4.2 Training Stability

To demonstrate the training stability of our proposed RL algorithm (Algo. 2), we plot the mean cumulative reward and success metric which indicates that our RL agent can successfully learn the navigation task as described on the original CARLA benchmark [29]. The plots show that our agent learns the navigation task in just 1.5M time steps of training, where we see that the total reward and total success episode values converge to the maximum possible on that task. Similar training plots

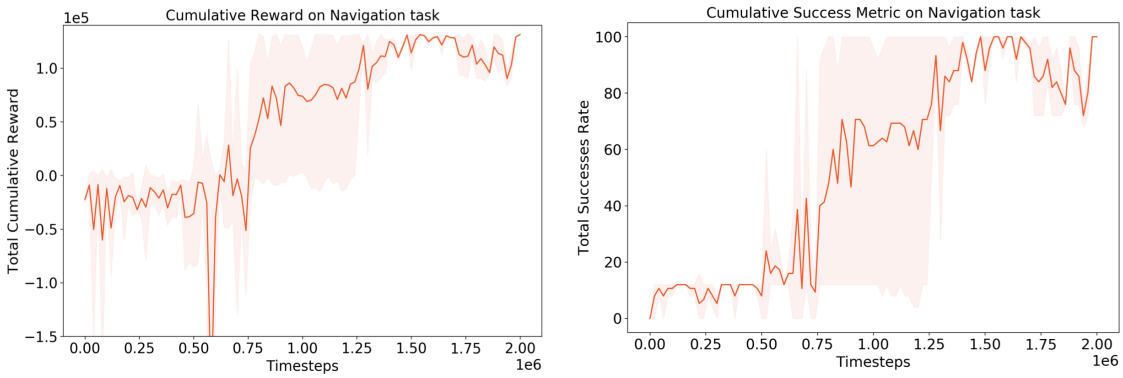

Figure 5.3: The figure reports the mean cumulative reward and success rate on the *Navigation* task defined by the original CARLA benchmark [29]. The plots indicate that the RL agent can successfully learn the task within 2M time steps of training. The shaded region corresponds to the minimum and maximum values showing variation across 3 runs.

are observed in other tasks as well.

We note that our method achieves the optimal driving performance within 2M time steps, equivalent to a single day of training, that is a significant improvement over the standards of deep reinforcement learning where it is common to train for hundreds of millions of steps [67], corresponding to months of subjective experience.

5.5 Results

Given that our method is built on the semantically segmented image as an input, we compare and report results only on the training weather conditions as described in the original CARLA [29] and NoCrash [24] benchmarks. Further, we report our results averaged over 3 seeds and 5 different evaluations of the benchmarks. For all evaluations, we pick the best performing model from each seed based on the cumulative reward it collects at the validation time.

5.5.1 Evaluation on the Original CARLA Benchmark

Since the original CARLA benchmark (Sec. 4.5.1) [29] focuses on skills like lane-following and driving around intersections, we observe from our results (Table. 5.1)

Original CARLA Benchmark (% Success Episodes)										
Task	**Training Conditions (*Town 01*)**									
	MP	*IL*	*RL*	*CIL*	*CIRL*	*CAL*	*CILRS*	*LBC*	*IA*	*Ours*
Straight	98	95	89	98	98	**100**	96	**100**	**100**	**100** $\pm$ 0
One Turn	82	89	34	89	97	97	92	**100**	**100**	**100** $\pm$ 0
Navigation	80	86	14	86	93	92	95	**100**	**100**	**100** $\pm$ 0
Dyn. Navigation	77	83	7	83	82	83	92	**100**	**100**	**100** $\pm$ 0
Task	**Testing Conditions (*Town 02*)**									
	MP	*IL*	*RL*	*CIL*	*CIRL*	*CAL*	*CILRS*	*LBC*	*IA*	*Ours*
Straight	92	97	74	97	**100**	93	96	**100**	**100**	**100** $\pm$ 0
One Turn	61	59	12	59	71	82	84	**100**	**100**	**100** $\pm$ 0
Navigation	24	40	3	40	53	70	69	98	**100**	**100** $\pm$ 0
Dyn. Navigation	24	38	2	38	41	64	66	99	98	**100** $\pm$ 0

Table 5.1: Quantitative comparison with the baselines that solve the four goal-directed navigation tasks using modular, imitation learning or reinforcement learning approaches on the original CARLA benchmark [29]. The table reports the percentage (%) of successfully completed episodes for each task in the training (*Town 01*) and testing town (*Town 02*). **Higher** is better. The baselines include *MP* [29], *IL* [29], *RL* [29], *CIL* [23], *CIRL* [61], *CAL* [86], *CILRS* [24], *LBC* [20] and *IA* [99] compared with our model-free RL method [4]. The reported results are the average over 3 seeds that are evaluated on 5 different runs of the benchmark. **Bold** values correspond to the best mean success rate.

[4] that our agent perfectly masters these skills as it can solve all the driving tasks of *Straight*, *One Turn*, *Navigation* & *Dynamic Navigation* both in *Town 01* and *Town 02*. For a qualitative analysis, we also publish a video[3] of our agent driving on the *Navigation* task.

Further, on comparing our method with the baselines, particularly the *CARLA RL* [29] baseline, we observe that our method achieves a significant improvement in the success rate performance on all the driving tasks. This baseline uses a similar approach to ours that trains reinforcement learning from scratch but instead focuses to learn from high-dimensional image observations. Further, we observe that our method achieves better performance than the *MP* [29], *IL* [29], *CIL* [23], *CIRL* [61], *CAL* [86] and *CILRS* [24] baselines and comparable to the recent baselines of *LBC*

[3] https://www.youtube.com/watch?v=UoEdZqEejL8

NoCrash Benchmark (% Success Episodes)							
Task	**Training Conditions (*Town 01*)**						
	CIL	*CAL*	*CILRS*	*LBC*	*IA*	*AT*	*Ours*
Empty	79 ± 1	81 ± 1	87 ± 1	97 ± 1	**100**	**100** ± 0	**100** ± 0
Regular	60 ± 1	73 ± 2	83 ± 0	93 ± 1	96	**99** ± 1	52 ± 6
Dense	21 ± 2	42 ± 1	42 ± 2	71 ± 5	70	**86** ± 3	19 ± 2
Task	**Testing Conditions (*Town 02*)**						
	CIL	*CAL*	*CILRS*	*LBC*	*IA*	*AT*	*Ours*
Empty	48 ± 3	36 ± 6	51 ± 1	**100** ± 0	99	**100** ± 0	**100** ± 0
Regular	27 ± 1	26 ± 2	44 ± 5	94 ± 3	87	**99** ± 1	45 ± 5
Dense	10 ± 2	9 ± 1	38 ± 2	51 ± 3	42	60 ± 3	12 ± 1

Table 5.2: Quantitative comparison with the baselines that solve the three goal-directed navigation tasks using modular, imitation learning or reinforcement learning approaches on the NoCrash benchmark [24]. The table reports the percentage (%) of successfully completed episodes for each task in the training (*Town 01*) and testing town (*Town 02*). **Higher** is better. The baselines include *CIL* [23], *CAL* [86], *CILRS* [24], *LBC* [20], *IA* [99] and CARLA built-in autopilot control (*AT*) compared with our model-free RL method [4]. The reported results are the average over 3 seeds that are evaluated on 5 different runs of the benchmark. **Bold** values correspond to the best mean success rate.

[20] and *IA* [99]. We conjecture that our perfect results can be attributed to our simpler choice of input representation. However, a notable difference between the latter baselines and our method is that they leverage expert demonstrations or learn a separate lower-dimensional affordance that receives strong supervision, unlike our method that learns the optimal policy from scratch based on trial-and-error learning. This indicates the strength of off-the-shelf reinforcement learning algorithms and their immense potential to solve some of the complex problems in autonomous driving.

5.5.2 Evaluation on the NoCrash Benchmark

Considering the fact that the original benchmark does not account for collision infractions, we extend our evaluation to the recent NoCrash benchmark (Sec. 4.5.2)

[24] that registers a success if the agent reaches the goal destination without colliding with any objects. We observe from our results (Table. 5.2) that our agent perfectly learns to navigate in the *Empty Town 01* and *Town 02* tasks. For the task with *Regular* and *Dense* traffic, we notice that our agent reports poor performance when compared to other baseline methods. This is because our agent fails to recognize the behavior of the brake action in the dynamic actor scenario as it has partial access to the environment state. For a qualitative analysis, we also publish a video[4] that demonstrates the failure case on the *Dense* traffic task.

This points us to the limitation of our Markov decision process formulation as the agent's state (Sec. 5.1) does not explicitly encode the front actor or the nearest traffic light state. While it may seem reasonable to argue that the latent representation of the autoencoder should encode the front actor state, the limitation of this representation is that our agent does not have sufficient information to decode the temporal change in the position of the target actor with respect to its position. Therefore, we improve on this limitation and discuss our new formulation in our next work (Chap. 6) that deals with the dynamic actor and traffic lights interactions.

5.6 Discussion

In this chapter, we present our first approach to autonomous driving that learns optimal driving policies using reinforcement learning. We believe that the key to solving any reinforcement learning task is to deliberately define the Markov decision process (Sec. 3.1) which we detail in Sec. 5.1. Further, using a variant of the off-the-shelf on-policy reinforcement learning algorithm and combining it with an autoencoder objective, we show that the driving agents learned from our proposed model-free RL algorithm learn to master common driving skills like lane-keeping and driving around intersections.

We also note that our approach when learned from scratch achieves comparable or near-perfect performance on the original CARLA [29] and NoCrash [24] benchmarks without any dynamic actors but fails to report a good performance in the complex dynamic actor scenario. We speculate that this performance gap is owing to our

[4] https://www.youtube.com/watch?v=JnRn59mHlPE

formulation of the Markov decision process that is limited in its capability to encode other dynamic actors state. We address this shortcoming in our next work (Chap. 6) that reduces the complexity of the problem to low-dimensional representations that are crucial to navigating the dynamic actors' scenario.

Chapter 6

Learning to Drive with Dynamic Actors

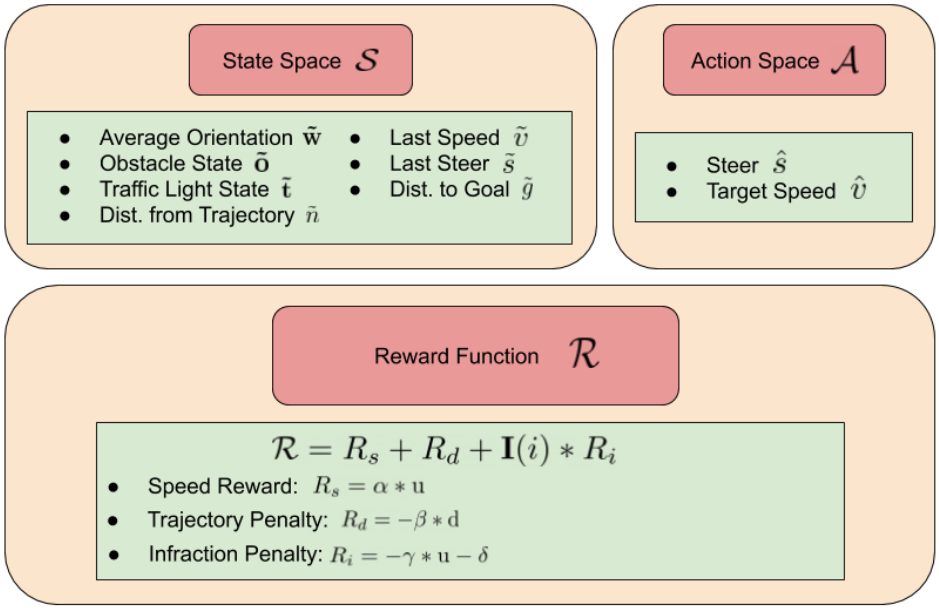

Figure 6.1: Our proposed RL setup for the dynamic actor setting that defines our Markov decision process with the state space $\mathcal{S}$, action space $\mathcal{A}$ and reward function $\mathcal{R}$.

With the recent advancements in deep reinforcement learning, there has been a growing trend towards applying RL for learning the autonomous driving task. Towards that end, Chap 5 discusses one such approach that shows exemplary performance on

learning common driving skills like lane following and driving around intersections using reinforcement learning from scratch. Although the proposed agent drives perfectly around both the train and test towns, it reports poor performance on the more critical urban driving tasks like stopping at traffic lights, stop sign intersections, in front of dynamic actors or pedestrians which are abundantly seen in urban driving scenarios.

The limitation with our earlier proposed approach (Sec. 5.1) is that the agent does not observe states that encode the nearest traffic light and the front actor state. These states are imperative to solving the urban driving tasks that are often complex and stochastic in nature. A recent technique termed as affordances [86, 99] aims to predict these low-dimensional representations directly from high-dimensional visual data. Attributes such as distance to the vehicle ahead, distance from the center-line, nearest traffic light state are predicted from a separate visual encoder. These attributes or intermediate representations are then fed into the control algorithm that determines the optimal policy for the complicated urban driving tasks.

Another approach to deal with these complex urban driving scenarios is to train a convolutional encoder (CNN) [54] that inherently learn the intermediate representations that are crucial to determining the optimal control. The advantage of learning such representations is that the hand-engineering effort is no longer needed which is often difficult to scale. Therefore to address the dynamic actor scenario, we first begin with low-dimensional affordance representations and then move towards the learning these representations using convolutional encoders.

We build on top of our prior work [4] (Chap. 5) and incorporate ideas from the affordance related works [86, 99] to propose learning optimal policies through reinforcement learning from these low-dimensional representations. By dividing the urban driving task into the affordance prediction, and planning and control blocks, we believe that the overall complexity of the problem is reduced. This is because affordance prediction is an easy supervised learning task that reduces the difficulty of the planning and control block which is now responsible for learning the optimal control policy based on affordances that encode the current state of the world.

In this work, we primarily focus on the planning and control block that learns the optimal action conditioned on the state of the dynamic world. We formulate this task again using the Markov decision process (Sec. 3.1) and use the same model-

free approach to reinforcement learning (Sec. 3.2) as our earlier work [4] (Chap. 5). Further, we extend our previous RL setup (Sec. 5.1) and the CARLA environment (Chap. 4) for this new task of learning to drive with dynamic actors.

The overall work is divided into the following sections. Sec. 6.1 describes our reinforcement learning setup, followed by our model-free reinforcement learning algorithm in Sec. 6.2. Next, we detail the training procedure in Sec. 6.3, and discuss the experimentation in Sec. 6.4. Finally, we present our results and discussion in Sec. 6.5 and 6.6 respectively.

6.1 RL Setup

In this section, we define our state space $\mathcal{S}$, action space $\mathcal{A}$ and reward function $\mathcal{R}$ (Fig. 6.1) that characterizes the formulation of our Markov decision process (Sec. 3.1) that we use for the dynamic actor scenario.

6.1.1 State Space

In RL, defining the state space plays a key role as it determines what observations are available to the agent at the learning time. Since we motivate our formulation using the low-dimensional affordances, these representations should be fully capable of encoding the current state of the agent with respect to the dynamic world. In order to learn the urban driving tasks, these representations should have access to information such as the dynamic actors' state, and the nearest traffic light state that majorly affect our agent's throttle and brake action. Besides these, our agent must also have access to pedestrian behavior that also affects the behavior of our agent.

Since these affordances are pivotal to the performance of our agent in the dynamic actor scenario, we propose determining two affordances: *dynamic obstacle affordance* and the *traffic light affordance*. We do not model a separate pedestrian affordance due to the limitation of the CARLA simulator in version 0.9.6[1] that supports pedestrians actors that only move along the sidewalks.

The *dynamic obstacle affordance* (Sec. 4.4), denoted by $\tilde{\mathbf{o}}$, is responsible for encoding the distance between our agent and the front dynamic actor, as well as its

[1]CARLA v0.9.6 - https://carla.org/2019/07/12/release-0.9.6/

corresponding speed. This low-dimensional representation helps our agent discern whether the target actor in front is accelerating, decelerating, or moving at a constant speed with respect to our ego-agent. The second affordance, *traffic light affordance* (Sec. 4.4), denoted by $\tilde{\mathbf{t}}$, represents the state of the nearest traffic light that affects our agent. Particularly, this encodes the distance of the nearest traffic light from our agent when it is in the red state.

Both the aforementioned affordances are useful representations that assist in determining the optimal actions of our agent. This is because, to avoid collision or traffic light infractions, our agent must learn to stop when approaching a dynamic actor or traffic light in the red state.

Besides the two affordances, another useful state representation is the previous time step action that the policy outputs. This state information is useful in determining the change in the actions that the policy should be able to predict relative to the last action. Since the action space consists of steer and target speed (Sec. 6.1.2), denoted by $\tilde{s}$ and $\tilde{v}$ respectively, we augment our affordance representation with these previous step actions.

The representation proposed above is still deficient in its capability to predict the optimal value function for our learning agent. We believe this is because it does not yet have access to goal destination location which is crucial to determining the correct optimal value function estimate. To avoid this pitfall, we propose adding the distance to goal destination, denoted by $\tilde{g}$. Additionally, we also propose adding in the signed distance from the optimal trajectory, denoted by $\tilde{n}$ as the additional state-input since it aids in disambiguating the position of our agent relative to the optimal trajectory.

Lastly, we also add the waypoint features ($\tilde{\mathbf{w}}$) [4], defined in Eq. 5.2, that helps in directing our agent to the target destination based on the agent's current pose $\mathbf{p}$ and next n waypoints ($\mathbf{w_1}, \mathbf{w_2}, ..., \mathbf{w_n}$). This representation is exactly incorporated from our earlier work (Chap. 5) [4].

In summary, the first state representation (A) that feeds into our reinforcement learning algorithm can be denoted by $A = [\tilde{\mathbf{w}}.\tilde{\mathbf{o}}.\tilde{\mathbf{t}}.\tilde{n}.\tilde{v}.\tilde{s}.\tilde{g}]$ where the (.) indicates the concatenation operator. We believe that this low-dimensional representation encodes all the sufficient information to solve the complex navigation tasks.

Further, as our next goal is to learn the affordance representations directly from high dimensional images, we propose two additional state representations that move

towards that direction. Inspired from our earlier work (Chap. 5), we believe using the latent features $\tilde{\mathbf{h}}$ of an autoencoder that takes in the last k top-down semantically segmented frames helps in determining the obstacle state affordance. This forms our second state representation, denoted by $A + I = [\tilde{\mathbf{h}}.\tilde{\mathbf{w}}.\tilde{\mathbf{o}}.\tilde{\mathbf{t}}.\tilde{n}.\tilde{v}.\tilde{s}.\tilde{g}]$, that concatenates the latent features with our earlier representation (A). The third representation (I) aims to explicitly remove the low-dimensional obstacle affordance $\tilde{\mathbf{o}}$ to give $I = [\tilde{\mathbf{h}}.\tilde{\mathbf{w}}.\tilde{\mathbf{t}}.\tilde{n}.\tilde{v}.\tilde{s}.\tilde{g}]$. Note that $A + I$ and I still use the low-dimensional traffic light affordance explicitly.

6.1.2 Action Space

We use the same action space as defined in our earlier work (Sec. 5.1.2) to learn continuous action policies that predict the target steer and speed actions, denoted by $\hat{s}$ and $\hat{v}$ respectively. The target speed action is then fed into a classical PID controller [7, 84] that outputs the throttle and the brake actions, denoted by $\hat{t}$ and $\hat{b}$.

6.1.3 Reward Function

As this work focuses on both collisions and traffic light infractions, we modify our earlier formulation of the reward function $\mathcal{R}$ (Sec. 5.1.3) and relabel the penalty term as the infraction penalty. Thus, the reward function (Eq. 6.1) again has 3 components to it. For completeness, we present our dense reward scheme again which can be mathematically defined by Eq. 6.1.

$$\mathcal{R} = R_s + R_d + \mathbf{I}(i) * R_i \tag{6.1}$$

- *Speed-based Reward (R_s)*: This reward, that is directly proportional to the current speed u of the agent, incentivizes it to move towards the goal destination in the shortest possible time.

$$R_s = \alpha * \mathrm{u} \tag{6.2}$$

- *Distance-based Penalty from Optimal Trajectory (R_d)*: This penalty, that is directly proportional to the lateral distance d between the centre of our agent and the optimal trajectory, incentivizes the agent to stay close to the planned

optimal trajectory.

$$R_d = -\beta * \mathrm{d}; \tag{6.3}$$

- *Infraction Penalty (R_i)*: This penalty, activated by a collision or traffic light violation ($\mathbf{I}(i)$), penalizes our agent to drive safely without causing any infractions.

$$R_i = -\gamma * \mathrm{u} - \delta \tag{6.4}$$

6.2 Model-Free RL on Low-Dimensional Affordances

To train and test our RL formulation (Sec. 6.1), we once again select a state-of-the-art on-policy model-free reinforcement learning algorithm: Proximal policy optimization (PPO) [89] (Sec. 3.4.4) and show its performance on learning complex urban driving tasks. The algorithm (Algo. 2) follows a similar methodology as described by us in our earlier work (Sec 5.2) except that the autoencoder optimizations are now skipped.

The clipped objective function (Eq. 3.16) of PPO offers a surrogate optimization that constrains the policy updates by keeping the new and old policy close together in the policy vector space. Empirically, this objective function works great in practice with complex continuous-space environments which we also demonstrate in our prior work (Chap. 5). This motivates us to extend it again in order to study the impact of reinforcement learning in learning complex urban driving behaviors.

The algorithm consists of training a policy and a value function approximator that takes in the state representation $\mathcal{S}$ (Sec. 6.1.1) as an input. These networks symbolized by $\pi(\hat{s}, \hat{v}|\mathcal{S})$ and $V(\mathcal{S})$ learn the state to optimal action mapping and the expected return from that state respectively.

6.3 Training

We train the PPO algorithm by iteratively sampling a random episode with a start and goal destination. We then run an A* planner that computes a list of waypoints that trace the optimal trajectory. The state representation we described earlier (Sec. 6.1.1) is then queried at each time step from the simulator, which in

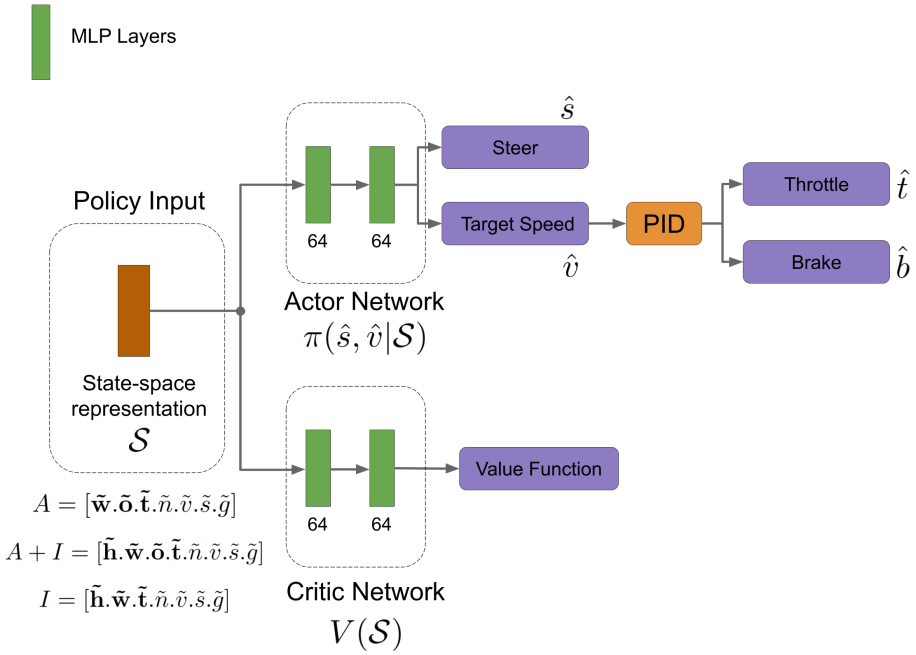

Figure 6.2: Our proposed architecture for state representation ($\mathcal{S} = A$, $A + I$ or I): The state representation is a combination of waypoint features $\tilde{\mathbf{w}}$, dynamic obstacle affordance $\tilde{\mathbf{o}}$, traffic light affordance $\tilde{\mathbf{t}}$, previous time step steer and target speed actions $(\hat{s}, \hat{v})$, distance to goal destination $\hat{g}$, signed distance from the optimal trajectory $\hat{n}$ or latent features of the autoencoder $\tilde{\mathbf{h}}$ depending on the state representation $\mathcal{S}$. The policy network outputs the control actions $(\hat{s}, \hat{v})$ where $\hat{s}$ is the predicted steer and $\hat{v}$ is the predicted target speed which is then mapped to predicted throttle and brake $(\hat{t}, \hat{b})$ using a PID controller.

real-world is accessible from the perception and GPS subsystems. The agent steps through the simulator at every time step collecting on-policy rollouts that determine the updates to the policy and critic networks. The episode is terminated upon a collision, lane invasion, or traffic light infraction, or is deemed as a success if the agent reaches within distance d m of the destination. Finally, this procedure repeats until the policy function approximator converges to the optimal policy.

The actor and critic networks consist of a standard 2-layer feedforward network. All the networks are trained with a ReLU non-linearity and optimized using stochastic gradient descent with the Adam optimizer. The overall architecture of our approach can be depicted in Fig. 6.2.

6.4 Experiments

For experimentation, we build on top of our modified version of stable-baselines[2] implementation and train the algorithm in the CARLA simulator [29]. The Markov decision process is formulated using our RL setup and CARLA environment defined by us in Sec. 6.1 and Chap. 4 respectively.

The training procedure described in the earlier section (Sec. 6.3) trains our agents on the *Dynamic Navigation* task (Sec. 4.5.1). This trained agent is then evaluated across all tasks on the original benchmark [29] (Sec. 4.5.1) as well as on the new dynamic actor tasks proposed by the recent NoCrash benchmark [24] (Sec. 4.5.2). The NoCrash benchmark gives us an accurate analysis of the number of infractions the agent makes as it drives through the towns. Further, to understand the behavior and robustness of the policy learned by our agent, we also conduct a separate infraction analysis that helps us analyze the different types of episode terminations and their relative proportions.

6.4.1 Baselines

For comparing our method with the prior work, we choose modular, imitation learning and reinforcement learning baselines that solve the goal-directed navigation task and report results on the original CARLA [29] and NoCrash [24] benchmarks. As this task is the same as described in our earlier work (Chap. 5) [4], we use the same set of baselines as defined in Sec. 5.4.1. To summarize, we use the following set of baselines.

- Modular Baselines: *MP* [29], *CAL* [86], *AT*.
- Imitation Learning Baselines: *IL* [29], *CIL* [23], *CILRS* [24], *LBC* [20].
- Reinforcement Learning: *RL* [29], *CIRL* [61], *IA* [99].

Although all of the above baselines use forward-facing RGB image and high-level navigational command as inputs, we recognize the differences in our inputs and presume that the current state-of-the-art perception systems are capable of predicting the low-dimensional representations with reasonable accuracy. Further, as the goal of

[2] https://github.com/hill-a/stable-baselines

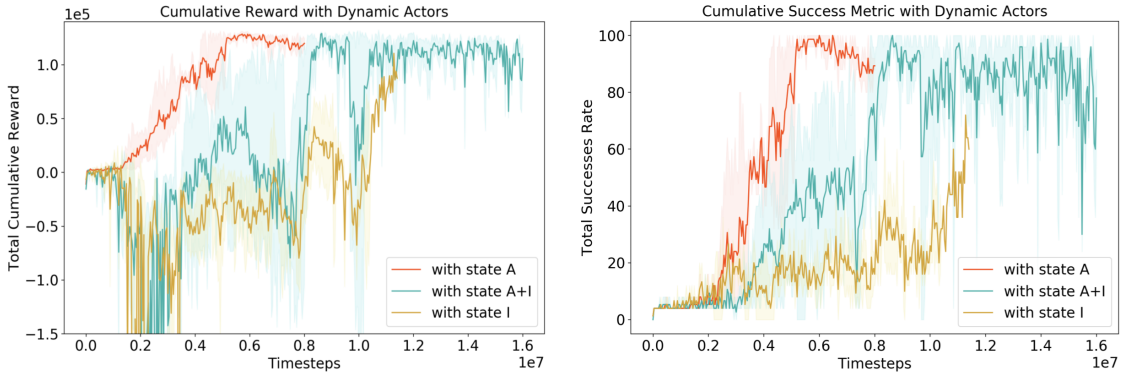

Figure 6.3: The figure reports the mean cumulative reward and success rate for our three choices of state representation: $\mathcal{S}{=}A$, $A+I$ or I, on the *Dynamic Navigation* task [29]. The plots indicate that the state-representation A and $A+I$ learn the navigation task successfully whereas the state-representation I learns the task slowly as observed by the performance improvement after 10M time steps of training. The shaded region in the plot corresponds to the minimum and maximum values showing variation across 3 different seeds.

this work is to solve the planning and control side of the driving problem, we believe that a perception system, trained to predict such low-dimensional representations, when combined with our approach can beat the current modular or imitation learned approaches to autonomous driving.

Additionally, we do not consider the pedestrian actors in our approach owing to the limitations of the CARLA 0.9.6 version. We also note that since our input representation does not differ across CARLA versions, we choose to report our results on CARLA 0.9.6 and compare our method with the prior work that reports results on other versions of CARLA.

6.4.2 Training Stability

We now compare the training stability across our three choices of the state representations ($\mathcal{S}{=}A$, $A+I$, or I) as defined in Sec. 6.1.1. We observe from Fig. 6.3 that the low-dimensional representation A that explicitly use affordances, performs the best as it is stably able to learn the dynamic navigation task on the original CARLA benchmark [29]. We also see that the low-dimensional representation along with the latent features of the autoencoder, represented by $A+I$, also learns the

navigation task after the performance dip around 10M time steps. We note that this representation requires twice the number of time steps that are required with the low-dimensional representation A.

Further, we also note that the state representation I, which does not explicitly contain the low-dimensional obstacle affordance, shows improvement on the dynamic navigation task after 10M time steps of training. This is slower when compared with the other two representations, which makes us empirically believe that it requires more samples to learn the dynamic obstacle affordance when compared with the latter that explicitly encode it. Further, considering the training stability and the policy performance, we choose to evaluate and report results only on the first two state representations A and $A + I$.

We also ask readers to refer Appendix A, Sec. A.1 for an additional set of experiments that aim to run our proposed approach with the forward-facing semantic or RGB camera both with state representation $A+I$ and I. This ablation experiment helps us analyze and compare our proposed approach with the baseline methods that use forward-facing RGB images as input.

6.5 Results

Considering our input representation includes low-dimensional representations or latent features of semantically segmented images, we do not consider the weather differences at the train and test time as proposed in the original CARLA [29] and NoCrash [24] benchmarks. Thus, for comparison with the baselines, we compare and present results only on training weather conditions from all the baselines. Further, we report our results that are averaged over 3 seeds and 5 different evaluations of the benchmarks. We pick the best performing model from each seed based on the cumulative reward it collects at the time of validation.

6.5.1 Evaluation on the Original CARLA Benchmark

On the original CARLA benchmark (Sec. 4.5.1) [29], we note that our method using both state representations A and $A + I$ (Table. 6.1) achieve a perfect success percentage on all the driving tasks in both *Town 01* and *Town 02*. The driving policy learned by our agent demonstrates stopping in front of other actors or traffic lights in

Original CARLA Benchmark (% Success Episodes)											
Task	**Training Conditions (*Town 01*)**										
	MP	*IL*	*RL*	*CIL*	*CIRL*	*CAL*	*CILRS*	*LBC*	*IA*	*Ours (A)*	*Ours (A+I)*
Straight	98	95	89	98	98	**100**	96	**100**	**100**	**100** $\pm$ 0	**100** $\pm$ 0
One Turn	82	89	34	89	97	97	92	**100**	**100**	**100** $\pm$ 0	**100** $\pm$ 0
Navigation	80	86	14	86	93	92	95	**100**	**100**	**100** $\pm$ 0	**100** $\pm$ 0
Dyn. Navigation	77	83	7	83	82	83	92	**100**	**100**	**100** $\pm$ 0	**100** $\pm$ 0
Task	**Testing Conditions (*Town 02*)**										
	MP	*IL*	*RL*	*CIL*	*CIRL*	*CAL*	*CILRS*	*LBC*	*IA*	*Ours (A)*	*Ours (A+I)*
Straight	92	97	74	97	**100**	93	96	**100**	**100**	**100** $\pm$ 0	**100** $\pm$ 0
One Turn	61	59	12	59	71	82	84	**100**	**100**	**100** $\pm$ 0	**100** $\pm$ 0
Navigation	24	40	3	40	53	70	69	98	**100**	**100** $\pm$ 0	99 $\pm$ 1
Dyn. Navigation	24	38	2	38	41	64	66	99	98	**100** $\pm$ 0	**100** $\pm$ 0

Table 6.1: Quantitative comparison with the baselines that solve the four goal-directed navigation tasks using modular, imitation learning or reinforcement learning approaches on the original CARLA benchmark [29]. The table reports the percentage (%) of successfully completed episodes for each task in the training (*Town 01*) and testing town (*Town 02*). **Higher** is better. The baselines include *MP* [29], *IL* [29], *RL* [29], *CIL* [23], *CIRL* [61], *CAL* [86], *CILRS* [24], *LBC* [20] and *IA* [99] compared with our PPO method. The reported results are the average over 3 seeds that are evaluated on 5 different runs of the benchmark. **Bold** values correspond to the best mean success rate.

red-state while perfectly navigating through the town and around intersections.

Further, our results also demonstrate that on the most difficult driving task of *Dynamic Navigation*, we achieve a significant improvement in the success rate performance on both towns when compared with our baselines such as *MP* [29], *IL* [29], *RL* [29], *CIL* [23], *CIRL* [61] and *CAL* [86]. For *CILRS* [24], the improvement is significant for *Town 02* and moderate for *Town 01*. We also note that our method reports comparable performance to the recently works of *LBC* [20] and *IA* [99].

6.5.2 Evaluation on the NoCrash Benchmark

We also extend our evaluation to the recent NoCrash benchmark (Sec. 4.5.2) [24] that considers collision infractions and episode termination if the agent happens to collide with any objects. Our quantitative results (Table. 6.2) again demonstrate that our driving agent learns the optimal control almost perfectly across different levels of traffic and town conditions. The agent learned with the low-dimensional

NoCrash Benchmark (% Success Episodes)								
Task	**Training Conditions (*Town 01*)**							
	CIL	*CAL*	*CILRS*	*LBC*	*IA*	*AT*	*Ours (A)*	*Ours (A+I)*
Empty	79 ± 1	81 ± 1	87 ± 1	97 ± 1	**100**	**100** ± 0	**100** ± 0	**100** ± 0
Regular	60 ± 1	73 ± 2	83 ± 0	93 ± 1	96	**99** ± 1	98 ± 2	90 ± 2
Dense	21 ± 2	42 ± 1	42 ± 2	71 ± 5	70	86 ± 3	**95** ± 2	94 ± 4
Task	**Testing Conditions (*Town 02*)**							
	CIL	*CAL*	*CILRS*	*LBC*	*IA*	*AT*	*Ours (A)*	*Ours (A+I)*
Empty	48 ± 3	36 ± 6	51 ± 1	**100** ± 0	99	**100** ± 0	**100** ± 0	**100** ± 0
Regular	27 ± 1	26 ± 2	44 ± 5	94 ± 3	87	**99** ± 1	98 ± 1	96 ± 2
Dense	10 ± 2	9 ± 1	38 ± 2	51 ± 3	42	60 ± 3	**91** ± 1	89 ± 2

Table 6.2: Quantitative comparison with the baselines that solve the three goal-directed navigation tasks using modular, imitation learning or reinforcement learning approaches on the NoCrash benchmark [24]. The table reports the percentage (%) of successfully completed episodes for each task in the training (*Town 01*) and testing town (*Town 02*). **Higher** is better. The baselines include *CIL* [23], *CAL* [86], *CILRS* [24], *LBC* [20], *IA* [99] and CARLA built-in autopilot control (*AT*) compared with our PPO method. The reported results are the average over 3 seeds that are evaluated on 5 different runs of the benchmark. **Bold** values correspond to the best mean success rate.

representation A outperform our agent learned with representation $A + I$. This difference can be attributed to the low-dimensionality of the state representation A.

Moreover, we also note that the success rate performance achieved by our agent is significantly higher than all the prior modular and imitation learning baselines such as *CIL* [23], *CAL* [86], *CILRS* [24], *LBC* [20] and *IA* [99]. The *AT* baseline, which refers to the autopilot control that is shipped with the CARLA binaries, uses a hand-engineered approach to determine the optimal control. We observe from our results that even on the hardest task of *Dense* traffic, our method significantly outperforms even the most engineered approach to urban driving. For a qualitative analysis, we publish a video[3] of our agent successfully driving on the dynamic navigation task.

We presume that our near-perfect results can be accounted for our simpler choice of input representation. Thus for a fair comparison, our future direction is to move towards high-dimensional RGB images. Nevertheless, a notable difference between

[3] https://www.youtube.com/watch?v=AwbJSPtKHkY

Infraction Analysis on NoCrash Benchmark (% Episodes)									
Task	**Metric**	**Training Conditions (*Town 01*)**				**Testing Conditions (*Town 02*)**			
		CIL	*CAL*	*CILRS*	*Ours (A)*	*CIL*	*CAL*	*CILRS*	*Ours (A)*
Empty	Success	79.00	84.00	96.33	**100.00**	41.67	48.67	72.33	**100.00**
	Col. Vehicles	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	Col. Other	11.00	9.00	1.33	**0.00**	51.00	45.33	20.00	**0.00**
	Timeout	10.00	7.00	2.33	**0.00**	7.33	6.00	7.67	**0.00**
Regular	Success	61.50	57.00	87.33	**98.40**	22.00	27.67	49.00	**98.16**
	Col. Vehicles	16.00	26.00	4.00	**0.27**	34.67	30.00	12.67	**0.00**
	Col. Other	16.50	14.00	5.67	**0.53**	37.33	36.33	28.00	**0.92**
	Timeout	6.00	3.00	3.00	**0.80**	6.00	6.00	10.33	**0.92**
Dense	Success	22.00	16.00	41.66	**95.38**	7.33	10.67	21.00	**91.20**
	Col. Vehicles	49.50	57.00	20.67	**0.62**	55.67	46.33	35.00	**3.73**
	Col. Other	25.00	24.00	34.67	**1.23**	34.33	35.33	35.00	**1.87**
	Timeout	3.50	3.00	3.00	**2.77**	**2.67**	7.67	9.00	3.20

Table 6.3: Quantitative analysis of episode termination causes and comparison with the baselines that solve that three goal-direction navigation tasks using modular, imitation or reinforcement learning approaches on the NoCrash benchmark [24]. The table reports the percentage (%) of episodes with their termination causes for each task and for the training (*Town 01*) and testing town (*Town 02*). The columns for a single method/task/condition should add up to 1. For each cause of episode termination we **bold** the method with the **best** performance. The baselines include *CIL* [23], *CAL* [86] and *CILRS 100* [24] compared with our PPO method. The reported results are the average over 3 seeds that are evaluated on 5 different runs of the benchmark.

the baseline methods and our method is that they use strong supervision signals, unlike our method that learns the optimal policy from scratch based on trial-and-error learning. These supervised signals, often expensive to collect, do not always capture the distribution of scenarios entirely. In contrast, we show that our results demonstrate the prospects of utilizing reinforcement learning to learn complex urban driving behaviors.

6.5.3 Infraction Analysis

To analyze the failure cases of our agent, we also perform an infraction analysis that reports the percentage of episodes for each termination condition and driving task on the NoCrash benchmark [24]. We then compare this analysis with few of our

baselines like *CIL* [23], *CAL* [86] and *CILRS* [24] that perform end-to-end imitation learning or take a modular approach to predicting low-dimensional affordances like ours. We incorporate the baseline infraction metrics from the *CILRS* work [24].

We observe (Table. 6.3) from this analysis that our agent significantly outperforms all the other baselines across all traffic and town conditions. Further, we note that our agent in the *Empty* town task achieves a perfect success percentage across both *Town 01* and *Town 02* without facing any infractions. Additionally, on the *Regular* and *Dense* traffic tasks, we notice that our approach reduces the number of collision and timeout infractions by at least an order of magnitude across both the towns. Therefore, our results indicate that the policy learned by our agents using reinforcement learning is robust to variability in both traffic and town conditions.

6.6 Discussion

In this chapter, we present our second approach to learn the urban driving tasks that commonly include numerous interactions between dynamic actors, and traffic signs and signals. We formulate a reinforcement learning approach primarily using low-dimensional representations also known as affordances. We demonstrate that using such low-dimensional representations makes the planning and control problem easier as we can learn stable and robust policies demonstrated by our results with state representation A. Further, we also observe that as we move towards learning these representations inherently using convolutional encoders, the performance and robustness of our learned policy decreases which requires more training samples to learn the optimal representations, as evident from Fig. 6.3.

Thus, we empirically believe that learning optimal policies from high-dimensional state representations is still challenging given the current state-of-the-art on-policy reinforcement learning algorithms. Additionally, we conjecture that adding auxiliary objectives to the primary reinforcement learning objective will make the learning task easy both from the representation and policy learning aspects. On the contrary, our perfect results (Sec. 6.5) using the low-dimensional representations is a promising direction for the autonomous driving community that effectively demonstrates dividing the complex driving problem into perception (representation learning), and planning and control (policy learning) subsystems.

Chapter 7

Exploratory Policy Search

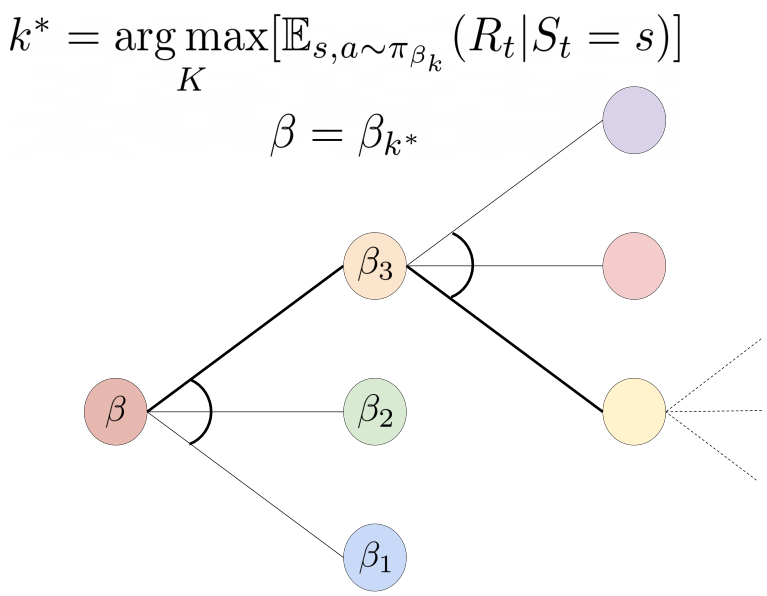

Figure 7.1: Backup diagram of our proposed algorithm, Exploratory Policy Search (Algo. 3) that the finds the optimal policy among the K randomly explored and trained policies.

This chapter introduces a new algorithm that we formulate, with the goal to find the best-optimized driving policy using reinforcement learning that can easily be deployed in the real-world. In recent years, deep reinforcement learning (DRL) has seen great success with the significant breakthroughs both in model-free [65, 66] and model-based [92, 93] reinforcement learning. To that end, model-free reinforcement

learning is the one that is widely been discussed in literature owing to its simplicity and stability in the learning process. And yet among others, policy gradient-based methods are commonly used to search for the optimal policy.

A major challenge with the policy gradient-based method is to estimate the right step size for updating the policy, as an improper step size may lead to severe degradation in the performance of the policy. This degradation may be catastrophic to the optimizing objective as the input data strongly depends on the behavior of the current policy [47, 88]. Consequently, to achieve the optimal performance the algorithms should strike the right balance between the learning stability and speed of convergence.

To address this tradeoff, there exists two representative methods, Trust Region policy optimization (TRPO) [88], and Proximal policy optimization (PPO) [89]. In particular, TRPO (Sec: 3.4.3) [88] adds in a divergence constraint when updating the policy distributions whereas PPO (Sec. 3.4.4) [89] adopts a surrogate objective to avoid imposing the hard constraint completely. The divergence metric, although proved theoretically to guarantee a monotonic performance improvement, is computationally inefficient as it involves solving a complicated second-order optimization. On the other hand, PPO [89] reduces the complexity to a first-order optimization by adopting a clipping mechanism through a surrogate objective that is simple to implement and tune.

However, despite PPO's success, the algorithm still reports poor performance under a bad step size (Sec. 7.1). Moreover, the algorithm is prone to suffer from the risk of lack of exploration [101] especially under a bad initialization that may lead to a sub-optimal convergence. Thus, the main motivation of this work is to build a better algorithm that finds the best performing policy over the span of policy updates to resolve the aforementioned problems. To that end, we propose a new algorithm known as Exploratory Policy Search, that combines exploratory search with reinforcement learning algorithms to find the optimal policy.

In this chapter, we first describe the motivation that led to the formulation of the Exploratory Policy Search Algorithm in Sec. 7.1, describe the algorithm in detail in Sec. 7.2 and then present the experimentation and results in Sec. 7.3 and 7.4 respectively. Lastly, we conclude the chapter by providing few discussion points in Sec. 7.5.

7.1 Motivation

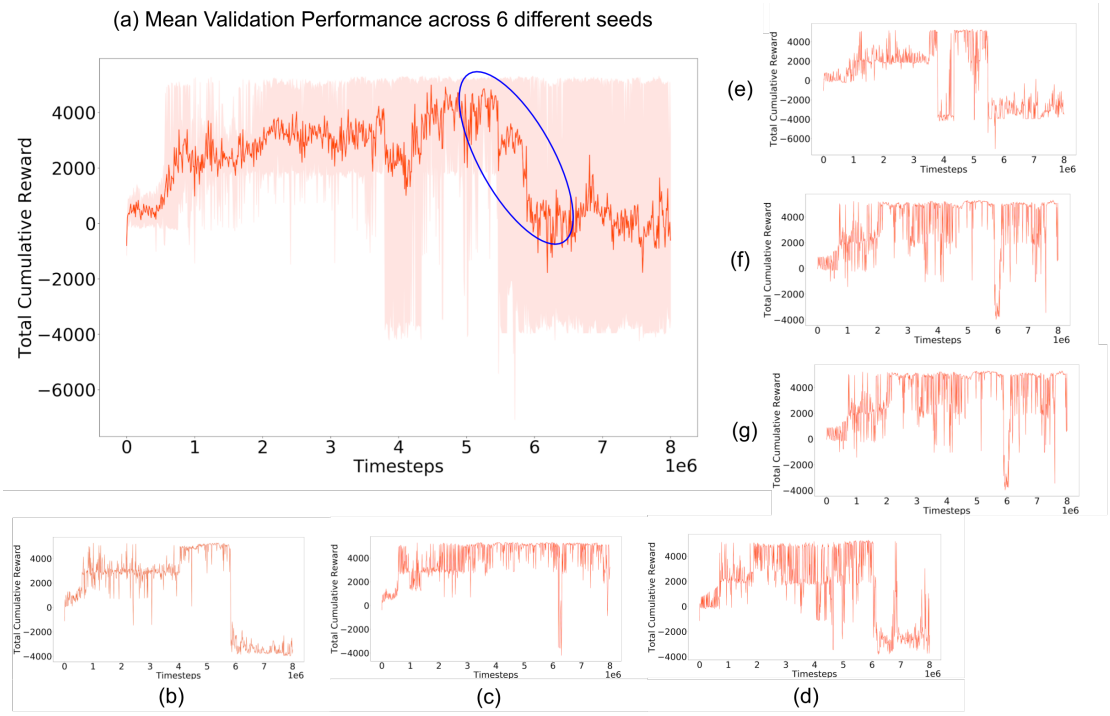

Figure 7.2: Catastrophic performance decay observed during one of our experiments on learning to drive with the dynamic actor scenarios. (a): Mean validation reward across 6 different seeds of the experiments. The **blue** oval represents the performance decay observed after **5.5M** time steps of training. (b) - (g): Individual seed-wise validation reward for the same experiment.

The primary objective of any reinforcement learning algorithm is to overtime find the best performing policy that leads to maximum possible cumulative return. In contrast, policy gradient methods are known to be sensitive to the correct step size where it is empirically common to observe the performance decay catastrophically in a short range of policy updates. This behavior can be observed in Fig. 7.2 that demonstrates the mean performance across 6 different seeds of an experiment that we conduct in the CARLA simulator [29]. As seen from the figure, we observe that the mean validation reward recorded by 6 different seeds of our experiments reaches to a near-maximum after 4.5M time steps of training. But as we observe, the mean validation reward drops drastically after 5.5M time steps which is apparent from

Algorithm 3 Our EPS Algorithm

1: **Input**: initialize model parameters β, population size K, total epochs E
2: **for** epoch= $1, 2, \ldots E$ **do**
3: Clone model parameters β for each individual in population $\{\beta_1, \ldots \beta_K\}$.
4: Train each individual in parallel with any RL algorithm for S steps.

$$\beta_k = \text{Train}(\text{parameters} = \beta_k, \text{steps} = S)$$

5: Evaluate each individual by running the policy, derived from β_k, in the environment to get expected cumulative rewards $\hat{R}_1 \ldots, \hat{R}_K$.

$$\hat{R}_k = \mathbb{E}_{s,a\sim\pi_{\beta_k}}(R_t|S_t = s)$$

6: Choose the best performing policy and its corresponding parameters β_{k^*} to update β.

$$k^* = \arg\max_K(\hat{R}_k)$$

$$\beta = \beta_k^*$$

7: **end for**

Fig. 7.2 (b), (d), and (e) that indicate individual seed reward plots. In contrast, Fig. 7.2 (c) and (f) indicate seeds that reported stable performance throughout the training whereas Fig. 7.2 (g) indicates a seed that failed to explore the action space and reports intermediate performance.

The above observations highlight a limitation of the state-of-the-art policy gradient algorithms in terms of its exploration capabilities and its learning stability with respect to the performance objective (Eq. 3.8). This motivates us to propose a new algorithm, which we term Exploratory Policy Search, that searches for the best policy over time across K different explorations while still individually maximizing the reinforcement learning objective.

7.2 Algorithm

The key notion behind the formulation of our algorithm is to iteratively propagate K explorations and run any reinforcement learning algorithm on them, with the goal to update the current policy to the best-optimized policy so far. At the beginning of

each iteration, the base policy of the explorations is set to the current best policy and is then independently trained and evaluated in parallel. The current policy is then updated to the best amongst those K different policies (Eq. 7.1) by picking the policy that maximizes the performance objective (Eq. 3.8). This process is repeated over several iterations until the final current policy converges to the optimal policy. Note that the exploration in our algorithm comes due to the inherent nature of the random seeds that sample actions either from a distribution or choose a ϵ-greedy action. The overall algorithm can be outlined in Algorithm 3, where population size K denotes the number of random explorations, and the epochs E indicates the total number of iterations.

$$\begin{aligned} k^* &= \arg\max_K [\mathbb{E}_{s,a\sim\pi_{\beta_k}}(R_t|S_t = s)] \\ \beta &= \beta_{k^*} \end{aligned} \tag{7.1}$$

While our proposed algorithm shares similar ideas as the popular asynchronous advantage actor-critic (A3C) algorithm [67], the global parameters (β) of our model are updated only from one of the children that fetches the maximum evaluation reward, rather than updating it asynchronously from each child model as in A3C algorithm. This is because, in our formulation, we only consider the best optimized policy and train it further, without accounting for other policies that may have suffered from bad updates. Another difference between A3C [67] and our algorithm is that A3C accumulates gradients that asynchronously update the master parameters whereas our algorithm maintains K set of model parameters β_k that are each individually updated by the base reinforcement learning algorithm.

7.3 Experiments

We conduct experiments on 4 different continuous control tasks (*Walker2d-v2, HalfCheetah-v2, Swimmer-v2* and *Hopper-v2*) in the MuJoCo simulator [98] as well as on the most difficult dynamic navigation scenarios on the CARLA simulator [29].

Theoretically, our proposed algorithm (Algo. 3) is invariant to the choice of the reinforcement learning algorithm. However, to compare with our earlier work (Chap. 5, 6) we choose the vanilla PPO-Clip variant (Algo. 1) [89] as our primary RL

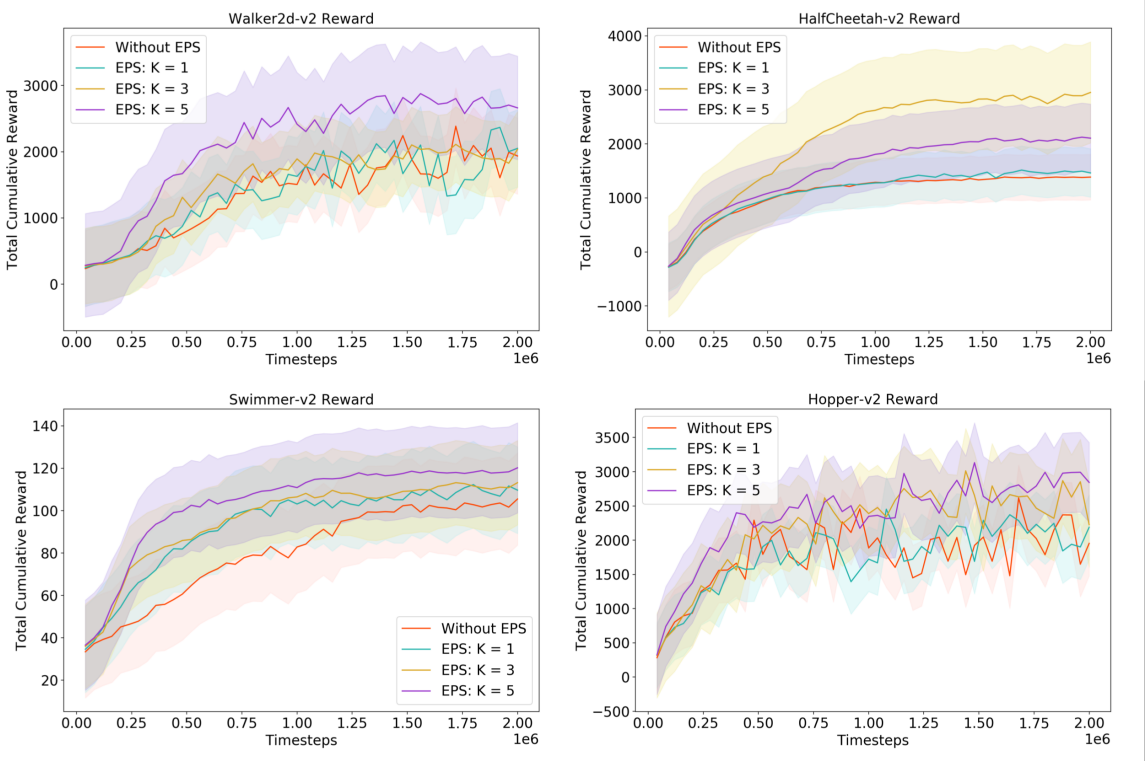

Figure 7.3: Comparison of our EPS algorithm with the the vanilla RL algorithm (*without EPS*) across different population sizes ($K = 1, 3, 5$) on four distinct MuJoCo [98] control tasks. The figure reports the mean reward plots that is averaged over 3 random seeds. We use the PPO-Clip [89] variant as the vanilla RL algorithm and train our proposed algorithm for $E = 50$ epochs and a total of 2M time steps.

algorithm and train it according to our proposed algorithm. We run the experiments starting with the best setting of hyper-parameters and then perform a grid-search on the epochs E and population size K to understand the behavior of our algorithm.

7.4 Results

7.4.1 MuJoCo Control tasks

The Fig. 7.3 reports the mean reward achieved by our algorithm on each environment for 4 different settings of experiments, *without EPS*, *with EPS* $K = 1$, $K = 3$ and $K = 5$. The *without EPS* experiment is the vanilla PPO RL algorithm without our EPS addition. As seen from the reward plots, we observe that the EPS

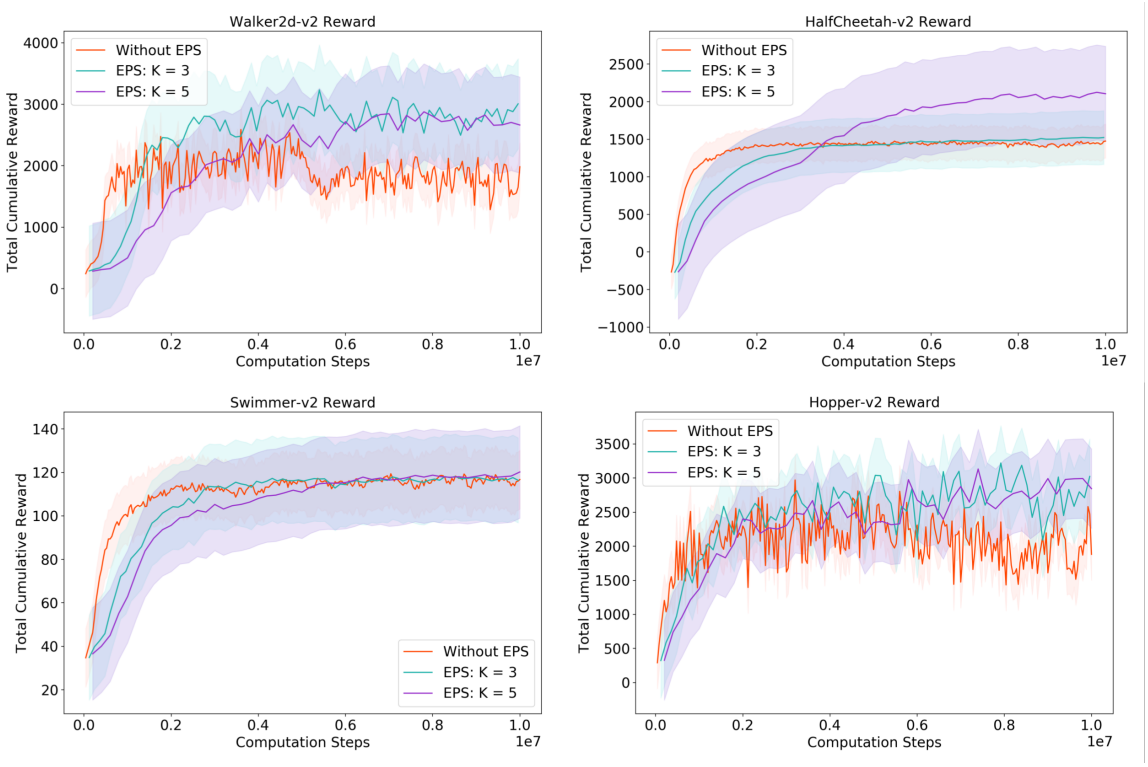

Figure 7.4: Fair comparison, in terms of compute, of our EPS algorithm with the the vanilla RL algorithm (*without EPS*) across different population sizes ($K = 3, 5$) on four distinct MuJoCo [98] control tasks. The figure reports the mean reward plots that is averaged over 3 random seeds. We use the PPO-Clip [89] variant as the vanilla RL algorithm and train our proposed algorithm for $E = 50$ epochs and a total of 10M time steps.

algorithm on average performs better than one *without EPS*. We also observe the performance to improve as the population size K increases. This higher performance can be attributed to the fact that our algorithm exhaustively explores the action space by collecting K sets of trajectories in parallel and optimizing each of them with respect to the performance objective (Eq. 3.8).

Additionally, we observe that the $K = 5$ variant performs better than the $K = 3$ and $K = 1$ for the *Walker2d-v2*, *Swimmer-v2* and *Hopper-v2* environments. The results align with our intuition that higher K should lead to better rewards. On the contrary, we observe that for the *HalfCheetah-v2* environment, this observation does not hold true where we observe that $K = 3$ is the best setting.

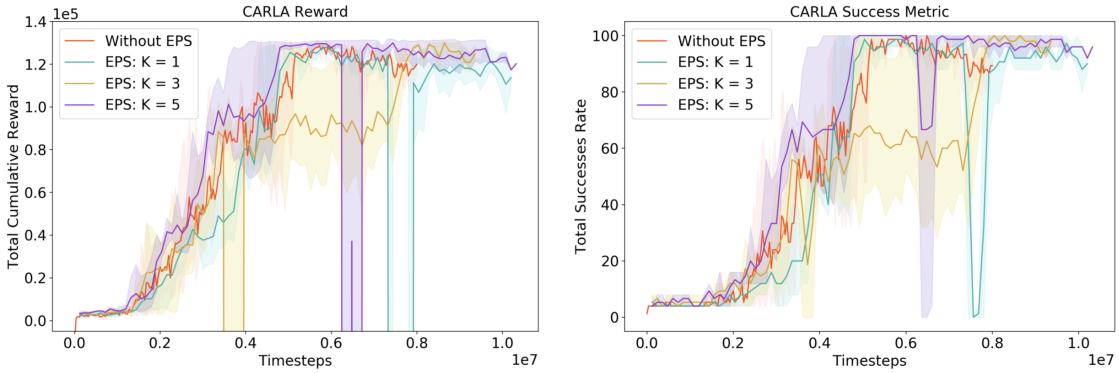

Figure 7.5: Comparison of our EPS algorithm with the the vanilla PPO-Clip variant (*without EPS*) across different population sizes ($K = 1, 3, 5$) on the *Dynamic Navigation* task defined in the CARLA simulator [90]. The figure reports the mean cumulative reward and success rate that is averaged over 3 random seeds.

Further, we also conduct a computational analysis in order to have a fair comparison with the *without EPS* version (Fig. 7.4). This is because our algorithm requires K times more computation as it independently optimizes K policy objectives in parallel. We observe from this analysis that EPS algorithm still works better on the *Walker2d-v2*, *HalfCheetah-v2* and *Hopper-v2* environments. We also note that the results on the *HalfCheetah-v2* environment are significantly better with the $K = 5$ variant when compared with the other two variants. This observation helps us conclude that our proposed algorithm is promising both from the unfair and fair comparison standpoints as it suggests a good way to parallelize training or setup distributed RL algorithms.

Lastly, we ask readers to refer Appendix A, Sec. A.2 for an additional set of experiments that demonstrate the degenerate form of our EPS algorithm.

7.4.2 CARLA Driving task

We also evaluate our EPS algorithm (Algo. 3) on the *Dynamic Navigation* task in the CARLA simulator and compare the results with our prior work as discussed in Chap. 6. Since the CARLA simulator is too slow to run various experiments, we pick our low dimensional affordance model from our earlier work (Chap. 6), referred here as *without EPS*, and combine it with our EPS algorithm to run on 3 different

population sizes, $K = 1, 3, 5$.

The Fig. 7.5 demonstrates the mean reward and success plot achieved by each experiment when trained on the original CARLA dynamic navigation scenario [29] (Sec. 4.5.1). We observe from the figure 7.5 that the *EPS, K=5* version performs marginally better than all the other variants. We also note from the success plot that the validation performance of the optimal policy resulting from $K = 5$ yields a stable and robust policy between 5 to 8M time steps besides the dip in the performance that is evident in the graph. Further, we conjecture that our algorithm is not robust to the performance collapse and thus suffers from determining the optimal step size.

Additionally, when compared with the *without EPS* version, we observe that this improvement in the performance is not significant. The marginal improvement in the performance for $K = 5$ comes at an extra compute cost, which makes us conclude that the EPS algorithm proposed by us does not add any significant benefit of our proposed exploration strategy. Moreover, $K = 3$ performs worse than the *without EPS* experiment, and $K = 1$ yields similar performance to it. Therefore, we believe that to completely validate the hypothesis of our proposed algorithm, additional experimentation needs to be performed.

7.5 Discussion

In summary, our first set of results from the MuJoCo experiments suggest that our hypothesis to find the optimal policy by combining forward search with model-free reinforcement learning algorithms is promising. We believe this is because our method aids in the exploration process by training over K different explorations. The results presented in Sec. 7.4.1 make us believe that our hypothesis is promising both from the unfair and fair comparison standpoints as it suggests a better strategy to parallelize training or set up distributed RL algorithms. Nevertheless, our analysis on the CARLA simulator does not show any added advantage with our EPS algorithm. Thus we believe that to reap the full benefits of search combined with reinforcement learning methods, additional experimentation needs to be done to completely validate our hypothesis.

Chapter 8

Conclusion

In this thesis, we propose two works that aim to solve the urban driving task using reinforcement learning. The first work (Chap. 5) learns to master common urban-driving skills like lane-keeping and driving around intersections, by fusing navigational features, also known as waypoints, with learned latent representations. This work highlights the first step towards achieving our goal of formulating the driving problem through reinforcement learning.

Next, we describe our second work (Chap. 6) that solves the limitations of our earlier proposed method and learns optimal policies for the common urban driving setting that involves numerous complex interactions between multiple actors, and traffic signs and symbols. We show that our method when trained using low-dimensional representations learns the urban navigation task which is comparable to the modular or the imitation learning-based approaches. We also show that our agents report a significantly lower number of infractions that are at least an order of magnitude less than the prior works on the same tasks.

Further, we introduce in Chap. 7 our Exploratory Policy Search (EPS) algorithm (Algo. 3) that combines the ideas of forward search with model-free reinforcement learning methods to determine the best policy. We show that our proposed method showcases promising results on the MuJoCo control tasks but fails to show any improvement on the CARLA dynamic navigation task. Thus, we conclude that our work is a step towards combining efficient search methods with the reinforcement learning objectives and needs additional experimentation to validate our hypothesis.

Moreover, we also describe our CARLA environment (Chap. 4) that forms the core basis of our work as well as discuss our plans to open-source our implementation to push research efforts in this direction.

8.1 Key Takeaways

We now present the important lessons and key takeaways that we have learned from our overall work.

- We demonstrate in our work that reinforcement learning as a learning paradigm has a strong potential to learn complex control tasks. We believe that our work is a step towards exploiting its full potential in learning general driving skills and we hope that our work inspires more research into applying reinforcement learning to the autonomous driving task.
- We believe that the key to the success of any reinforcement learning task is to carefully design the Markov decision process (Sec. 3.1). This includes choosing the optimal state representations, determining the output action spaces as well as designing reward functions based on the horizon of rewards that impact the optimal control policy.
- Since most current state-of-the-art reinforcement learning algorithms often require millions of samples to learn from high-dimensional state representations, we believe adding auxiliary objectives to the primary reinforcement learning objective will reduce the complexity of the overall problem.

8.2 Future Work

Finally, to conclude this work we will highlight some future directions in which this work can be extended.

- The first step should be to incorporate high dimensional RGB images in our proposed work by either training a separate visual encoder that predicts the low-dimensional affordances or training the entire pipeline in an end-to-end manner. We empirically observe that since the latter idea requires millions

of samples to learn the correct representations, we believe adding auxiliary objectives to the reinforcement learning objective is beneficial.

- An interesting direction as part of future work could be to explore Asymmetric Actor-Critic methods [79] that exploit the full state observability of the simulators to train the critic networks while feeding in partial observations (RGBD images) to the policy at the test time. This method has shown promising results when compared with the vanilla actor-critic methods.
- Another direction could be to move towards off-policy reinforcement learning algorithms that are known to be more sample efficient than the on-policy methods. The motivation behind this is to exploit tons of stored experiences in the form of log data in order to bootstrap the policy learning process.
- As the CARLA simulator [29] operates almost in real-time, we propose running multiple environments in parallel to concurrently train on different maps of CARLA as well generate more variability in the training data. Additionally, supporting pedestrians, traffic signs (like stop or yield), and multi-lane towns with roundabouts will be useful in order to move towards more complicated urban driving settings.

In the end, our objective of using reinforcement learning as the learning paradigm for autonomous driving will be a great success if we can see a reinforcement learning-based policy learn to drive a real-world autonomous vehicle.

Appendix A

Additional Experiments

A.1 Learning to Drive with Dynamic Actors

In this section, we show an additional experiment that runs our proposed approach with the dynamic actors, using forward-facing semantic or RGB camera with both state representation *A+I* and *I* as defined in Sec. 6.1.1. This ablation experiment helps us analyze and compare our proposed approach with the baseline methods that use forward-facing RGB images as input.

A.1.1 Ablation with Different Cameras

The Fig. A.1 shows seven different versions of the experiment. The first three state representations *A*, *A+I* and *I* use the top-down semantic segmentation camera. The next two state representation *FS:A+I* and *FS:I* use the forward-facing semantic segmentation camera whereas the last two *FR:A+I* and *FR:I* use the forward-facing RGB camera.

We observe from this figure (Fig. A.1) that the state representation *FR:I* and *FS:A+I* show some promise as we observe the reward and success rate to improve after 2M and 4M time steps respectively. The other ablation experiments (*FS:I* and *FS:A+I*) with the forward-facing camera are inconclusive when trained for the first 3M time steps. We also note that all the image experiments in CARLA require immense computation that is equivalent to more than 3 weeks of real-time training on Nvidia

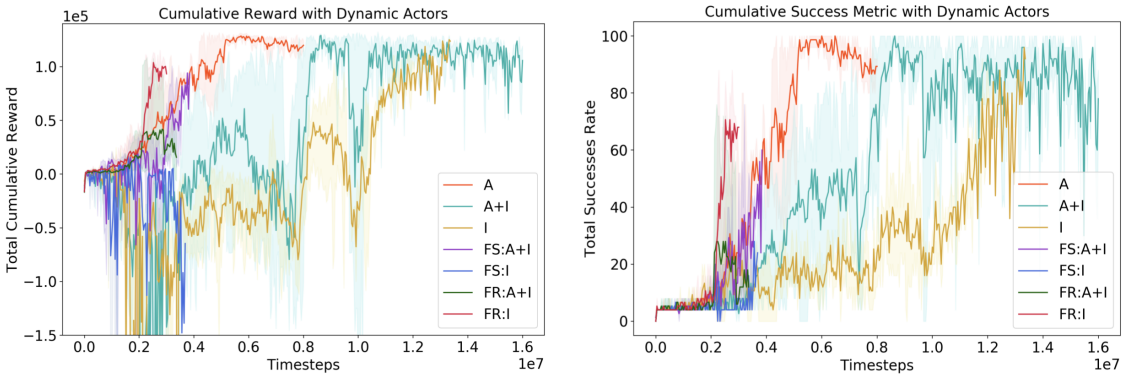

Figure A.1: The figure reports the mean cumulative reward and success rate for seven different choices of state representation on the *Dynamic Navigation* task [29]. The first three representations A, $A + I$, and I use the top-down semantic segmentation camera. Next, *FS:A+I* and *FS:I* use the forward-facing semantic segmentation camera whereas the last two *FR:A+I* and *FR:I* use the forward-facing RGB camera. The plots indicate that the state-representation A and $A + I$ successfully learn the task whereas the state-representation I gradually learns the task after 10M time steps of training. Further, among the forward-facing camera experiments, *FR:I* and *FS:A+I* seem to show a performance improvement while the rest do not run long enough and show similar performance as the top-down camera experiments when trained for the first 3M time steps. The shaded region in the plot corresponds to the minimum and maximum values showing variation across 3 different seeds.

2080Ti GPUs. Since this was infeasible, we report results till the maximum time steps each run ran for and note that the learning speed does not change substantially when compared with the experiments with the top-down camera view.

A.2 Exploratory Policy Search

In this section, we show the results of an additional experiment that runs the degenerate form of our algorithm, which is equivalent to running EPS with $E = 1$.

A.2.1 Degenerate EPS

The Fig. A.2 and Fig. A.3 reports the mean reward achieved by our algorithm on each environment with different settings of experiments, *with* and *without EPS*. We refer to the $E = 1$ variants as the degenerate form of our proposed algorithm. The

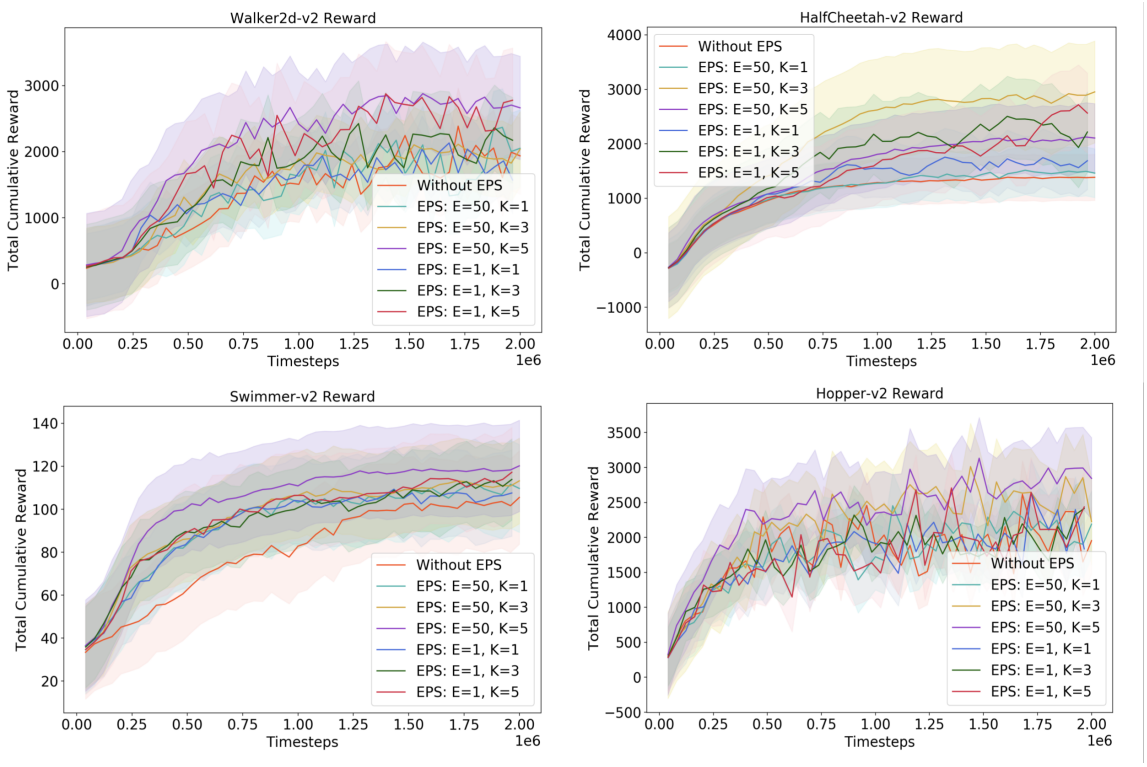

Figure A.2: Comparison of our EPS algorithm with the the vanilla RL algorithm (*without EPS*) across different population sizes ($K = 1, 3, 5$) and two epoch variants ($E = 1, 50$) on four distinct MuJoCo [98] control tasks. The figure reports the mean reward plots that is averaged over 3 random seeds. We use the PPO-Clip [89] variant as the vanilla RL algorithm and train our proposed algorithm for a total of 2M time steps.

without EPS experiment is equivalent to running any vanilla RL algorithm without our EPS addition, which in our case in the Proximal policy optimization algorithm.

As seen from Fig. A.2, we observe that the EPS algorithm on average performs better than ones without EPS, across different settings of the two hyper-parameters, which are population size K and training steps of each epoch that is governed by the total number of epochs E. Next, we observe from Fig. A.3 that the degenerate form of EPS ($E = 1$) still performs better on the *Walker2d-v2* and *HalfCheetah-v2* environments, when performing a fair comparison based on compute. Further, we make a note that *EPS: E=1, K=3* gives a significant improvement on the *HalfCheetah-v2* environment when compared with other variants.

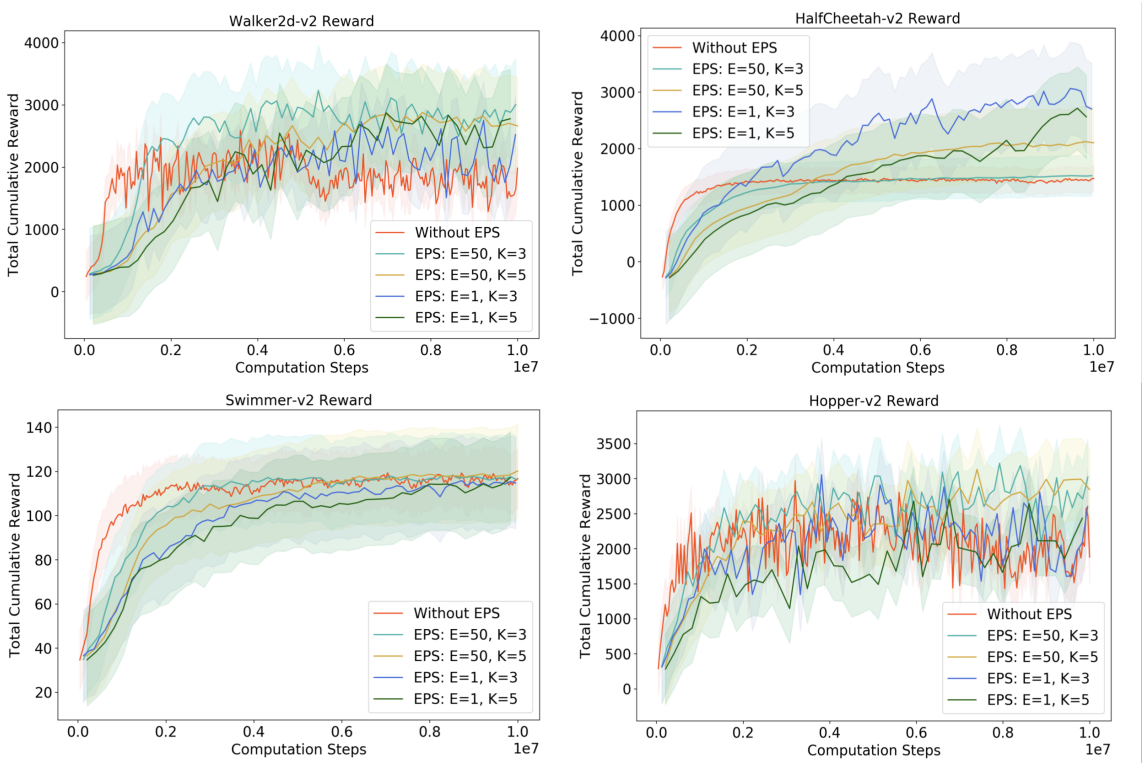

Figure A.3: Fair comparison, in terms of compute, of our EPS algorithm with the vanilla RL algorithm (*without EPS*) across different population sizes ($K = 3, 5$) and two epoch variants ($E = 1, 50$) on four distinct MuJoCo [98] control tasks. The figure reports the mean reward plots that is averaged over 3 random seeds. We use the PPO-Clip [89] variant as the vanilla RL algorithm and train our proposed algorithm for a total of 10M time steps.

Therefore, our analysis from the above experiment makes us conclude that our algorithm is promising in two ways. Firstly, the performance of the algorithm proposed by us is sensitive to the choice of hyper-parameters E and K chosen to run the experiment. Secondly, the algorithm paves a way to ultimately propose a strategy that can automatically tune these hyper-parameters as we observe the performance on average to improve when changing from $E = 1$ to $E = 50$ or $K = 1$ to $K = 5$.

Appendix B

Supplementary Details

B.1 CARLA Environment

B.1.1 Hyperparameters

In this subsection, we detail all the parameters we use for our experiments with the CARLA simulator [29], as discussed in Chap. 5, 6, 7. The list mentioned below covers all the parameters to the best of our knowledge, except those that are set to default values in the simulator.

- Camera top-down co-ordinates: $(x = 13.0, y = 0.0, z = 18.0, pitch = 270°)$
- Camera front-facing co-ordinates: $(x = 2.0, y = 0.0, z = 1.4, pitch = 0°)$
- Camera image resolution: $(x = 128, y = 128)$
- Camera field-of-view = 90
- Server frame-rate = $10 fps$
- Maximum target-speed = $20 km/h$
- PID parameters: $(K_P = 0.1, K_D = 0.0005, K_I = 0.4, dt = 1/10.0)$
- Waypoint resolution $(w_d) = 2m$
- Number of next waypoints $(n) = 5$
- Maximum time steps $(m) = 10000$

- Success distance from goal (d) = $10m$

B.2 Learning to Drive using Waypoints

B.2.1 Hyperparameters

In this subsection, we detail all the hyper-parameters used by us for the experiments and results reported in Chap. 5. The list mentioned below covers all the parameters to the best of our knowledge, except those that are set to default values as defined in the stable baselines documentation [42].

- Total training time steps = 2M
- Speed-based reward coefficient (α) = 1
- Distance-based penalty from optimal trajectory (β) = 1
- Collision penalty speed-based coefficient (γ) = 250
- Collision penalty constant coefficient (δ) = 250
- Learning rate = 0.0002
- Validation interval = 20K
- Maximum static time steps (m) = 1000
- Random seeds = 3
- Number of benchmark evaluations = 5

B.3 Learning to Drive with Dynamic Actors

B.3.1 Hyperparameters

In this subsection, we detail all the hyper-parameters used by us for the experiments and results reported in Chap. 6. The list mentioned below covers all the parameters to the best of our knowledge, except those that are set to default values as defined in the stable baselines documentation [42].

- Total training time steps = 16M

- N-steps = 10000
- Number of epochs = 10
- Number of minibatches = 20
- Clip parameter = 0.1
- Speed-based reward coefficient (α) = 1
- Distance-based penalty from optimal trajectory (β) = 1
- Infraction penalty speed-based coefficient (γ) = 250
- Infraction penalty constant coefficient (δ) = 250
- Learning rate = 0.0002
- Validation interval = 40K
- Number of dynamic actors at training time = $\mathcal{U}(70, 150)$, where $\mathcal{U}$ refers to uniform distribution.
- Image frame-stack fed to autoencoder (k) = 3
- Dynamic obstacle proximity threshold (d_{prox}) = $15m$
- Traffic light proximity threshold (t_{prox}) = $15m$
- Minimum threshold distance for traffic light detection = $6m$
- Random seeds = 3
- Number of benchmark evaluations = 5

B.4 Exploratory Policy Search

B.4.1 Hyperparameters

In this subsection, we detail all the hyper-parameters used by us for the experiments and results reported in Chap. 7. The list mentioned below covers all the parameters to the best of our knowledge, except those that are set to default values as defined in the stable baselines documentation [42].

- Total training time steps = 2M
- N-steps = 2048

- Number of epochs = 10
- Number of minibatches = 32
- Number of environments = 1
- Lam = 0.95
- Discount factor (γ) = 0.99
- Clip parameter = 0.2
- Entropy coefficient = 0.0
- Learning rate = 0.0003
- Epoch validation interval = 40K
- Random seeds = 3

Bibliography

[1] Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 1–8, 2006. 1.2, 3.2

[2] Pieter Abbeel, Adam Coates, and Andrew Y Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010. 2.2

[3] Joshua Achiam. Spinning up in deep rl, 2018. *URL https://spinningup. openai. com*. (document), 3.2

[4] Tanmay Agarwal, Hitesh Arora, Tanvir Parhar, Shuby Deshpande, and Jeff Schneider. Learning to drive using waypoints. In *Workshop on'Machine Learning for Autonomous Driving'at Conference on Neural Information Processing Systems*, 2019. (document), 1.3, 5, 5.1, 5.1.1, 5.1.1, 5.1.1, 5.1.2, 5.1.3, 5.2, 5.3, 5.2, 5.4, 5.4.1, 5.1, 5.5.1, 5.2, 6, 6.1.1, 6.4.1

[5] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019. 1.1

[6] Eduardo Arnold, Omar Y Al-Jarrah, Mehrdad Dianati, Saber Fallah, David Oxtoby, and Alex Mouzakitis. A survey on 3d object detection methods for autonomous driving applications. *IEEE Transactions on Intelligent Transportation Systems*, 20(10):3782–3795, 2019. 2.1

[7] Karl Johan Åström and Tore Hägglund. *PID controllers: theory, design, and tuning*, volume 2. Instrument society of America Research Triangle Park, NC, 1995. 5.1.2, 6.1.2

[8] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius Brito Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago Paixao, Filipe Mutz, et al. Self-driving cars: A survey. *arXiv preprint arXiv:1901.04407*, 2019. 2.1

[9] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning

to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018. 1.1

[10] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*, 2017. 3.2

[11] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957. 3.1

[12] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016. 1.1, 2.2

[13] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, and Urs Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv preprint arXiv:1704.07911*, 2017. 1.1, 2.2

[14] Michael Brady, John M Hollerbach, Timothy L Johnson, Tomás Lozano-Pérez, Matthew T Mason, Daniel G Bobrow, Patrick Henry Winston, and Randall Davis. *Robot motion: Planning and control*. MIT press, 1982. 1.1

[15] Guillaume Bresson, Zayed Alsayed, Li Yu, and Sébastien Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2(3):194–220, 2017. 2.1

[16] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012. 3.2

[17] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, pages 8224–8234, 2018. 3.2

[18] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013. 3.2

[19] Mark Campbell, Magnus Egerstedt, Jonathan P How, and Richard M Murray. Autonomous driving in urban environments: approaches, lessons and challenges. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 368(1928):4649–4672, 2010. 1.1, 2.1

[20] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75, 2020. (document),

1.1, 2.2, 5.4.1, 5.1, 5.2, 5.5.1, 6.4.1, 6.1, 6.5.1, 6.2, 6.5.2

[21] Lu Chi and Yadong Mu. Deep steering: Learning end-to-end driving model from spatial and temporal visual cues. *arXiv preprint arXiv:1708.03798*, 2017. 2.2

[22] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. 5.1.1

[23] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018. (document), 1.1, 2.2, 5.1.1, 5.4.1, 5.1, 5.5.1, 5.2, 6.4.1, 6.1, 6.5.1, 6.2, 6.5.2, 6.3, 6.5.3

[24] Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9329–9338, 2019. (document), 1.1, 2.2, 4, 4.5.1, 4.5.2, 5.4, 5.4.1, 5.5, 5.1, 5.5.1, 5.2, 5.5.2, 5.6, 6.4, 6.4.1, 6.5, 6.1, 6.5.1, 6.5.2, 6.2, 6.5.2, 6.3, 6.5.3

[25] Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. 3.2

[26] Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. *arXiv preprint arXiv:1806.06923*, 2018. 2.3

[27] Guilherme N DeSouza and Avinash C Kak. Vision for mobile robot navigation: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 24 (2):237–267, 2002. 2.1

[28] Nemanja Djuric, Vladan Radosavljevic, Henggang Cui, Thi Nguyen, Fang-Chieh Chou, Tsung-Han Lin, and Jeff Schneider. Motion prediction of traffic actors for autonomous driving using deep convolutional networks. *arXiv preprint arXiv:1808.05819*, 2, 2018. 2.1

[29] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio López, and Vladlen Koltun. Carla: An open urban driving simulator. In *CoRL*, 2017. (document), 1.1, 1.2, 2.3, 4.1, 4, 4.1, 4.2, 4.4, 4.5.1, 4.1, 4.5.2, 5, 5.1.1, 5.4, 5.4.1, 5.4.2, 5.3, 5.5, 5.5.1, 5.1, 5.6, 6.4, 6.4.1, 6.3, 6.4.2, 6.5, 6.5.1, 6.1, 6.5.1, 7.1, 7.3, 7.4.2, 8.2, A.1, B.1.1

[30] Roy Featherstone and David Orin. Robot dynamics: equations and algorithms. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*,

volume 1, pages 826–834. IEEE, 2000. 1.1

[31] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018. 3.2

[32] David A Forsyth and Jean Ponce. *Computer vision: a modern approach*. Prentice Hall Professional Technical Reference, 2002. 1.1

[33] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013. 1.1

[34] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018. 5.1.1

[35] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017. 1.1

[36] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018. 3.2

[37] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018. 3.3

[38] Jessica B Hamrick. Analogues of mental simulation and imagination in deep learning. *Current Opinion in Behavioral Sciences*, 29:8–16, 2019. 1.2, 3.2

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2.3, 5.4.1

[40] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 2.3

[41] Todd Hester and Peter Stone. Learning and using models. In *Reinforcement learning*, pages 111–141. Springer, 2012. 1.2, 3.2

[42] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/`

stable-baselines, 2018. 5.4, B.2.1, B.3.1, B.4.1

[43] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 5.1.1

[44] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018. 2.3

[45] Ronald A Howard. Dynamic programming and markov processes. 1960. 3.1

[46] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017. 2.2

[47] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274, 2002. 7

[48] Takeo Kanade, Chuck Thorpe, and William Whittaker. Autonomous land vehicle project at cmu. In *Proceedings of the 1986 ACM fourteenth annual conference on Computer science*, pages 71–80, 1986. 1.1, 2.1

[49] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254. IEEE, 2019. 1.1, 2.3

[50] Qadeer Khan, Torsten Schön, and Patrick Wenzel. Latent space reinforcement learning for steering angle prediction. *arXiv preprint arXiv:1902.03765*, 2019. 1.1

[51] Jinkyu Kim and John Canny. Interpretable learning for self-driving cars by visualizing causal attention. In *Proceedings of the IEEE international conference on computer vision*, pages 2942–2950, 2017. 2.2

[52] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 5.1.1

[53] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *arXiv preprint arXiv:2002.00444*, 2020. 1.1

[54] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 2.2, 5.1.1, 6

[55] Sampo Kuutti, Saber Fallah, Konstantinos Katsaros, Mehrdad Dianati, Francis Mccullough, and Alexandros Mouzakitis. A survey of the state-of-the-art localization techniques and their potentials for autonomous vehicle applications. *IEEE Internet of Things Journal*, 5(2):829–846, 2018. 2.1

[56] Leonid Kuvayev and Richard S Sutton. Model-based reinforcement learning with an approximate, learned model. In *Proceedings of the ninth Yale workshop on adaptive and learning systems*, pages 101–105. Citeseer, 1996. 1.2, 3.2

[57] Jean-Paul Laumond et al. *Robot motion planning and control*, volume 229. Springer, 1998. 1.1

[58] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436–444, 2015. 1.1, 2.2

[59] Stéphanie Lefèvre, Dizan Vasquez, and Christian Laugier. A survey on motion prediction and risk assessment for intelligent vehicles. *ROBOMECH journal*, 1 (1):1–14, 2014. 2.1

[60] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 163–168. IEEE, 2011. 1.1, 2.1

[61] Xiaodan Liang, Tairui Wang, Luona Yang, and Eric P. Xing. Cirl: Controllable imitative reinforcement learning for vision-based self-driving. In *ECCV*, 2018. (document), 1.1, 2.3, 5.1.1, 5.4.1, 5.1, 5.5.1, 6.4.1, 6.1, 6.5.1

[62] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 2.3, 3.2, 3.3, 5.4.1

[63] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International conference on artificial neural networks*, pages 52–59. Springer, 2011. 5.1.1

[64] Rowan McAllister, Yarin Gal, Alex Kendall, Mark Van Der Wilk, Amar Shah, Roberto Cipolla, and Adrian Weller. Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning. International Joint Conferences on Artificial Intelligence, Inc., 2017. 1.1

[65] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 1.1, 3.2, 3.3, 5.4.1, 7

[66] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. 1.1, 7

[67] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves,

Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016. 2.3, 3.2, 5.4.1, 5.4.2, 7.2

[68] Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2020. 1.2, 3.2

[69] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, et al. Junior: The stanford entry in the urban challenge. *Journal of field Robotics*, 25(9):569–597, 2008. 1.1, 2.1

[70] Sajjad Mozaffari, Omar Y Al-Jarrah, Mehrdad Dianati, Paul Jennings, and Alexandros Mouzakitis. Deep learning-based vehicle behaviour prediction for autonomous driving applications: A review. *arXiv preprint arXiv:1912.11676*, 2019. 2.1

[71] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L Cun. Off-road obstacle avoidance through end-to-end learning. In *Advances in neural information processing systems*, pages 739–746, 2006. 1.1, 2.2

[72] Farzeen Munir, Shoaib Azam, Muhammad Ishfaq Hussain, Ahmed Muqeem Sheri, and Moongu Jeon. Autonomous vehicle: The architecture aspect of self driving car. In *Proceedings of the 2018 International Conference on Sensors, Signal and Image Processing*, pages 1–5, 2018. 1.1, 2.1

[73] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018. 3.2

[74] Ryosuke Okuda, Yuki Kajiwara, and Kazuaki Terashima. A survey of technical trend of adas and autonomous driving. In *Technical Papers of 2014 International Symposium on VLSI Design, Automation and Test*, pages 1–4. IEEE, 2014. 2.1

[75] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, and Jan Peters. An algorithmic perspective on imitation learning. *arXiv preprint arXiv:1811.06711*, 2018. 2.2

[76] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1(1):33–55, 2016. 2.1

[77] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Agile off-road autonomous driving using end-to-end deep imitation learning. *arXiv preprint arXiv:1709.07174*, 2, 2017. 1.1

[78] Anna Petrovskaya and Sebastian Thrun. Model based vehicle detection and tracking for autonomous urban driving. *Autonomous Robots*, 26(2-3):123–139, 2009. 2.1

[79] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017. 8.2

[80] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989. 1.1, 2.2

[81] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011. 1.1

[82] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736, 2006. 2.2

[83] Nicholas Rhinehart, Rowan McAllister, and Sergey Levine. Deep imitative models for flexible inference, planning, and control. *arXiv preprint arXiv:1810.06544*, 2018. 1.1

[84] Daniel E Rivera, Manfred Morari, and Sigurd Skogestad. Internal model control: Pid controller design. *Industrial & engineering chemistry process design and development*, 25(1):252–265, 1986. 5.1.2, 6.1.2

[85] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011. 2.2

[86] Axel Sauer, Nikolay Savinov, and Andreas Geiger. Conditional affordance learning for driving in urban environments. *arXiv preprint arXiv:1806.06498*, 2018. (document), 1.1, 5.1.1, 5.4.1, 5.1, 5.5.1, 5.2, 6, 6.4.1, 6.1, 6.5.1, 6.2, 6.5.2, 6.3, 6.5.3

[87] Robert J Schalkoff. *Digital image processing and computer vision*, volume 286. Wiley New York, 1989. 1.1

[88] 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. 1.2, 3.2, 3.4.3, 3.4.3, 3.4.4, 7

[89] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. (document), 1.2, 3.2, 3.4.4, 3.4.4, 5.2, 5.2, 6.2, 7, 7.3, 7.3, 7.4, A.2, A.3

[90] Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. Planning and decision-

making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 2018. (document), 2.1, 7.5

[91] David Silver, J Andrew Bagnell, and Anthony Stentz. Learning from demonstration for autonomous navigation in complex unstructured terrain. *The International Journal of Robotics Research*, 29(12):1565–1592, 2010. 1.1

[92] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. 1.1, 7

[93] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017. 1.1, 7

[94] Santokh Singh. Critical reasons for crashes investigated in the national motor vehicle crash causation survey. Technical report, 2015. 1.1

[95] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000. 3.4.2, 3.4.2

[96] Umar Syed and Robert E Schapire. A game-theoretic approach to apprenticeship learning. In *Advances in neural information processing systems*, pages 1449–1456, 2008. 2.2

[97] Lei Tai, Jingwei Zhang, Ming Liu, Joschka Boedecker, and Wolfram Burgard. A survey of deep network solutions for learning control in robotics: From reinforcement to imitation. *arXiv preprint arXiv:1612.07139*, 2016. 2.2

[98] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. (document), 7.3, 7.3, 7.4, A.2, A.3

[99] Marin Toromanoff, Emilie Wirbel, and Fabien Moutarde. End-to-end model-free reinforcement learning for urban driving using implicit affordances. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7153–7162, 2020. (document), 2.3, 5.4.1, 5.1, 5.2, 5.5.1, 6, 6.4.1, 6.1, 6.5.1, 6.2, 6.5.2

[100] Richard S Wallace, Anthony Stentz, Charles E Thorpe, Hans P Moravec, William Whittaker, and Takeo Kanade. First results in robot road-following. In *IJCAI*, pages 1089–1095. Citeseer, 1985. 1.1, 2.1

[101] Yuhui Wang, Hao He, Xiaoyang Tan, and Yaozhong Gan. Trust region-guided proximal policy optimization. In *Advances in Neural Information Processing Systems*, pages 626–636, 2019. 7

[102] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8 (3-4):279–292, 1992. 3.3

[103] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. 3.4.2, 3.4.2

[104] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net*, 4(6):2, 2000. 2.3

[105] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2174–2182, 2017. 2.2

[106] Zhengyuan Yang, Yixuan Zhang, Jerry Yu, Junjie Cai, and Jiebo Luo. End-to-end multi-modal multi-task vehicle control for self-driving cars with visual perceptions. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 2289–2294. IEEE, 2018. 2.2

[107] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access*, 8:58443–58469, 2020. 1.1, 2.1

[108] Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft, and Peter Corke. Towards vision-based deep reinforcement learning for robotic motion control. *arXiv preprint arXiv:1511.03791*, 2015. 1.1

[109] Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint arXiv:1605.06450*, 2016. 1.1