Reinforcement Learning for Safe and Efficient Planning in Autonomous Driving

Anirban Santara

REINFORCEMENT LEARNING FOR SAFE AND EFFICIENT PLANNING IN AUTONOMOUS DRIVING

Thesis submitted to
Indian Institute of Technology Kharagpur
for the award of the degree

of

Doctor of Philosophy

by

Anirban Santara

under the guidance of

Professor Pabitra Mitra
(Department of Computer Science and Engineering, IIT Kharagpur)

and

Professor Balaraman Ravindran
(Robert Bosch Centre for Data Science and Artificial Intelligence,
and Department of Computer Science and Engineering, IIT Madras)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

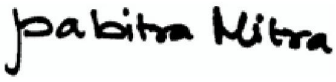
December 2020

– For my parents, Indira Santara and Anit Kumar Santara –

CERTIFICATE OF APPROVAL

Certified that the thesis entitled **Reinforcement Learning for Safe and Efficient Planning in Autonomous Driving**, submitted by **Anirban Santara** to the Indian Institute of Technology Kharagpur, for the award of the degree of Doctor of Philosophy has been accepted by the external examiners and that the student has successfully defended the thesis in the viva-voce examination held today.

Prof. Sudeshna Sarkar
(Chairperson)



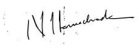
Prof. Pabitra Mitra
(Supervisor)

Prof. Soumyajit Dey
(Member of the DSC)

Prof. Pawan Goyal
(Member of the DSC)

Prof. Sourav Mukhopadhyay
(Member of the DSC)

Prof. Balaraman Ravindran
(External Supervisor)



Prof. N. Hemachandra
(External Examiner)

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur, India-721302

Certificate

This is to certify that this thesis entitled **Reinforcement Learning for Safe and Efficient Planning in Autonomous Driving** submitted by **Anirban Santara**, to the Indian Institute of Technology Kharagpur, is a record of bona fide research work carried out under our supervision and is worthy of consideration for award of Ph.D of the Institute.

Prof. Pabitra Mitra
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
India - 721302.

I.I.T. Kharagpur
December 2020

Prof. Balaraman Ravindran
Robert Bosch Center for Data Science and Artificial Intelligence
Indian Institute of Technology Madras
India - 600036.

I.I.T. Madras
December 2020

Declaration

I certify that

a. the work contained in the thesis is original and has been done by me under the guidance of my supervisors;

b. the work has not been submitted to any other institute for any other degree or diploma;

c. I have followed the guidelines provided by the Institute in preparing the thesis;

d. I have conformed to ethical norms and guidelines while writing the thesis;

e. whenever I have used materials (data, models, figures and text) from other sources, I have given due credit to them by citing them in the text of the thesis, and giving their details in the references, and taken permission from the copyright owners of the sources, whenever necessary.

Anirban Santara

Acknowledgment

I would like to take this opportunity to thank all the people and organizations that played key roles in my life towards making this thesis a reality. First of all, I would like to express my heartfelt thanks and gratitude to my Institute and Alma Mater, IIT Kharagpur. During the nine years I spent there, it has nurtured and crafted me as a professional.

I am indebted to Prof. Pabitra Mitra, my supervisor from IIT Kharagpur. Back in 2015, when I started my PhD, very few people in the Institute believed in the potential of Deep Learning. Prof. Mitra was the only person who supported my aspirations of doing cutting-edge research in this field and always made sure that I had what I needed to be successful. This included access to research funds, opportunity to collaborate with industry experts, permission to do long internships and of course, technical guidance.

I would like to express my deepest gratitude to Prof. Balaraman Ravindran, my supervisor from IIT Madras. My journey in Reinforcement Learning would not have been possible without his constant help and guidance ever since we started working together in 2017. He has always been a source of interesting ideas and precise leads whenever I got stuck in my research.

I would like to thank Bharat Kaul, Leena M., Kunal Banerjee and the entire team at the Parallel Computing Lab, Intel Labs India, for hosting me for a year long internship in 2017. I am indebted to them for introducing me to the world of autonomous vehicles. Two of the three projects that comprise my PhD thesis work were conceived during my internship with the team. The Intel Student Ambassador Program also supported my research by giving me a platform to showcase my work at top-tier conferences around the world and on the official Intel Blog.

I want to sincerely thank Google for supporting me in ways that PhD students in India can only imagine in their wildest dreams. The Google India PhD Fellowship program has kept me financially secure throughout my PhD tenure and has fulfilled my dream of traveling the world. I would like to thank Ashwani Sharma and Divy Thakkar for inducting me into the program and supporting me throughout. I also got an opportunity to do *two* consecutive internships with the incredible Robotics team at Google Brain in Mountain View, California. I would like to thank my recruiter Allison Kemmerling for getting me this opportunity. This was, undoubtedly, the most productive phase of my career and I learned the best practices for doing game-changing research in the field of Machine Learning there. I want to express my heartfelt appreciation and gratitude for Navdeep Jaitly, my manager in Google and a stalwart in the field of learning from sequential data, for his invaluable lessons on the training of recurrent neural network architectures and variational autoencoders. I would also like to appreciate the help, support and guidance of Krzysztof Choromanski and Pannag Sanketi during my internships. The wonderful work that I did with them helped me bag my dream job.

I can not express in words, my gratitude towards my parents Indira Santara and Anit Kumar Santara. Ever since my birth, they have sacrificed their individual pleasures and aspirations and devoted every single moment of their lives towards making me happy and successful. They have happily compromised on their lifestyles to make sure that I did not have to compromise on my education and well being. They have supported me through thick and thin and have always encouraged me to follow my passion. This thesis is worth five years of my work and twenty seven years of sweat, blood, tears and prayers of my parents.

My friends at IIT Kharagpur and around the world have been my support system throughout my PhD tenure. Ashutosh, Sudeshna, Sumitro, Swechcha and Mani deserve special mention. I will badly miss our chit-chats over tea every evening. I would like to convey my sincere thanks to Malaikannan Sankarasubbu for being a wonderful senior and my go-to for career advice. I am indebted to my dear friend Manorama for standing by me and helping me wade through some of the toughest days of my life. Without her support, it would have been practically impossible for me to finish my tenure as a PhD student on a high note. Extra thanks to her for being a wonderful subject for testing the generalizability of policies that I learn by exploring the world.

Abstract

The success of autonomous driving is contingent on the development of safe and efficient motion planning algorithms capable of working in multi-agent environments under stochasticity and partial observability. Reinforcement Learning (RL) provides a powerful framework for learning to behave in such environments. The focus of this thesis is to advance the state-of-the-art in reinforcement learning based motion planning for autonomous driving. We present MADRaS, an open-source Multi-Agent DRiving Simulator that is capable of simulating challenging driving tasks of high variance. We demonstrate how MADRaS can be used as a curriculum learning platform for training RL agents to drive a wide range of cars in different road tracks, navigate through traffic in narrow roads and prevent congestion and deadlocks through multi-agent cooperation. In reinforcement learning, the agent's objective is specified in terms of a scalar reward function making its accurate description crucial for success in achieving the desired behavior. In order to bypass the need to hand-engineer reward functions, Imitation learning algorithms like Generative Adversarial Imitation Learning (GAIL) estimate an expert's reward function from its demonstrations and then maximize it using RL. We observe that although GAIL is effective in matching (and often, exceeding) the expert at mean performance, high-cost trajectories, corresponding to tail-end events of catastrophic failure, are more likely to be encountered by GAIL agents than the expert. To address this issue, we develop Risk-Averse Imitation Learning (RAIL) as an alternative to GAIL in risk-sensitive applications that achieves up to 89% reduction in tail-risk at benchmark continuous control tasks of OpenAI Gym. The sample efficiency and convergence time of an RL algorithm heavily depend on the exploration method used. While human beings use knowledge from prior experiences at related tasks while exploring a new task, most exploration algorithms for RL use the information only from the current task-environment. We develop ExTra, a framework for Transfer-guided Exploration in which we leverage

a known optimal policy of a related task for efficient exploration in a new task. We demonstrate that ExTra is capable of exceeding and complementing the performance of traditional exploration algorithms.

Keywords: Autonomous Driving, Reinforcement Learning, Imitation Learning, Efficient Exploration

Contents

List of Abbreviations

RL	Reinforcement Learning
IL	Imitation Learning
GAIL	Generative Adversarial Imitation Learning
RAIL	Risk-Averse Imitation Learning
CVaR	Conditional Value at Risk
MADRaS	Multi-Agent Driving Simulator
ExTra	Transfer-guided Exploration

List of Figures

List of Tables

CHAPTER 1

Introduction

The evolution of transportation systems has been one of the primary factors that fueled the growth and prosperity of human civilization. As transportation became faster, safer and more accessible, the world became a smaller place. Communication, trade and different kinds of exchange between people flourished and globalized cultures emerged. World economy and average quality of life improved. In our relentless march for the advancement of human civilization, we must identify the challenges that face the transportation systems of today and develop solutions for ameliorating them.

The biggest challenge that faces humanity at the current time is climate change. According to World Health Organization (WHO) [1], the transport sector is the fastest growing contributor to emissions responsible for climate change. It contributed 23% of global Carbon Dioxide (CO_2) emissions in the year 2008 with the share of road transport alone being 16.5%. While climate change jeopardizes the future of human civilization [2], burgeoning traffic also poses a constant threat of fatal injuries. According to a report published by WHO [3], traffic accidents account for 1.35 million deaths worldwide each year. Additionally, $20 - 50$ million people are injured or disabled [4]. More than 90% of these incidents happen in the low and middle income countries of the world [3]. In 2017, 16 people died and

53 people suffered injuries every hour due to traffic accidents in India [5]. 94% of these accidents happen due to avoidable human error [6].

Autonomous driving refers to the task of making a car drive by itself with minimum interference from the human user. Autonomous driving technology along with electric and shared mobility has the potential to cut emissions responsible for climate change by 50% by the year 2050 [7]. Fully autonomous driving can also potentially eliminate all chances of accident due to human error, thus bringing down traffic fatality by over 90% [6]. Traffic congestion is a major contributor to the loss of productivity and environmental pollution. Multi-agent coordination between self-driving cars and route optimization provides a viable solution to traffic congestion [8]. The motivation of this thesis is to develop algorithms for improving the safety and optimality of self-driving cars and build open-source software tools for the democratization of technology.

1.1 Autonomous Driving

Autonomous driving has gained unprecedented attention of late due to its promise of increasing road safety by eliminating accidents due to human error, reducing traffic congestion, boosting shared mobility and reducing environmental pollution. Given a route in the form of a sequence of *waypoints* or markers on the map, autonomous cars drive by performing perception, planning and action in a loop [9].

In the *perception* stage, the car scans its environment and localizes itself in the global map. It identifies static and dynamic obstacles and creates an object oriented semantic representation of its surroundings. Common static obstacles are kerbs, signboards and houses. Dynamic obstacles can be other moving cars in the vicinity of the self-driving car, pedestrians and animals. The final task in the perception stage is to predict the behaviour and trajectory of each dynamic obstacle over a number of time steps into the future.

The second stage in the autonomous driving pipeline is *trajectory planning*. A

trajectory is a sequence of pose and velocity values that takes the car from one waypoint to the next. The task of motion planning involves calculation of a set of collision-free trajectories and selection of the most optimum one that respects the mechanical constraints of the system.

The third and last step involves vehicular platform implementation of the planned trajectory. The components of the vehicle that are directly responsible for its motion must be controlled for executing the planned sequence of poses and velocities. Engine dynamics, platform stability and passive safety are some of the issues that must be addressed at this stage.

Although realization of fully autonomous driving seems far flung, some specific low level tasks pertaining to driving such as adaptive cruise control, lane keep assistance and parking assistance have already been automated at a production scale in the form of Advanced Driver-Assistance Systems (ADAS) [10, 11]. Safe, optimal and fast motion planning in complex, multi-modal, multi-agent, and partially observed environments is the foremost technological challenge towards achieving full autonomy. Achieving these goals tractably using traditional motion planning algorithms – like Model Predictive Control, RRT, A*, and Dijkstra – is only possible under certain simplifying assumptions on the complexity the environment [12]. On the other hand, Machine Learning based approaches including Reinforcement Learning (RL) [13] and Learning from Demonstration (LfD) [14] are capable of fast and reactive control under lesser assumptions [15, 16]. However their training phase is often data-hungry and requires trial and error that entails risk.

In this thesis, we investigate the safety and optimality of reinforcement learning and related algorithms from a standpoint of autonomous driving. We also develop an open-source multi-agent driving simulator that is capable of simulating rich highway and track driving conditions for use in the training and evaluation of motion-planning agents.

1.2 Research Challenges

This section summarizes the challenges that are addressed in this thesis.

1.2.1 Developing a Simulator for Multi-agent Trajectory Planning in Autonomous Driving

Simulators are crucial for Artificial Intelligence (AI) research in autonomous driving because of their role as a sandbox for the training and evaluation of AI agents. Popular open-source driving simulators like CARLA [17], Microsoft AirSim [18], DeepDrive.io [19] and Udacity's Self Driving Car Simulator [20] focus on the task of perception. Apart from robust perception, an agent learning to face real world driving scenarios must learn invariances to road geometries, traffic patterns and vehicular dynamics during motion planning. These simulators do not offer enough variability along these dimensions that is necessary to learn the invariances. In a typical driving scene, multiple entities (cars, buses, bikes, and pedestrians) try to achieve their objectives of getting from one place to another fast, yet safely and reliably. A simulator for such an environment should provide an easy way to create arbitrary traffic configurations. The task of negotiating in traffic is akin to finding the winning strategy in a multi-agent game. Hence, an autonomous driving simulator should be able to simulate different varieties of traffic and support multiple agents learning to drive through cooperation and competition. Among the aforementioned simulators, AirSim, DeepDrive.io and Udacity provide some preset driving conditions mostly without traffic. They do not provide any straightforward way to create custom traffic or train multiple agents. CARLA does provide an API for independent control of cars that can be used to create traffic and multi-agent training. However, most of the variability presented by CARLA is in the perceived inputs and not in the behavioral dynamics of the ego-vehicle or the traffic agents. Our first challenge is to develop a dedicated multi-agent simulator for learning to plan in autonomous driving with a focus on learning invariances to road geometries, traffic patterns and vehicular dynamics.

1.2.2 Minimizing Tail-Risk of Imitation Learning Agents

Reinforcement Learning (RL) algorithms, along with efficient function approximators like deep neural networks, have achieved human-level (and sometimes, super-human) performance at many challenging planning tasks like continuous-control ([21, 22]) and game-playing ([23, 24]). In classical RL, the cost function is handcrafted based on heuristic assumptions about the goal and the environment. Not only is this challenging in most real-world applications, it is also prone to subjective bias [25]. Imitation learning or Learning from Demonstration (LfD) ([26, 27, 28]) addresses this challenge by providing methods of learning policies through imitation of an expert's behavior without the need of a handcrafted cost function. However we observe that, despite matching expert performance on an average, agents trained with state-of-the-art imitation learning algorithms like Generative Adversarial Imitation Learning (GAIL) [29] have a tendency to encounter high-cost trajectories more often than the experts. Since high trajectory-costs may correspond to events of catastrophic failure, GAIL agents are not reliable in risk-sensitive applications. Our second challenge is to formulate a direct approach to minimizing the tail risk of GAIL-learned policies.

1.2.3 Leveraging Prior Knowledge from Related Tasks for Efficient Exploration

Reinforcement Learning algorithms suffer from high sample complexity when used to learn complex tasks of high variance [30]. Traditional approaches to sample efficient exploration in RL only use domain specific knowledge. On the other hand, while attempting to solve a new task, human beings tend to take actions motivated by similar situations faced in the past. Our third challenge is to formulate a mathematical framework for transfer-guided exploration that will allow the use of prior experiences for improving the sample complexity of RL algorithms.

1.3 Research Contributions

The main contributions of this thesis can be summarized as follows:

1.3.1 MADRaS: A Multi-Agent Driving Simulator

We develop MADRaS, an open-source *M*ulti-*A*gent *DR*iving *S*imulator for autonomous driving. MADRaS builds on TORCS, a popular car racing platform [31], and adds a suite of features like hierarchical control modes, domain randomization, custom traffic, partial observability, stochastic outcomes of actions and support for multi-agent training. We train reinforcement learning agents to accomplish challenging tasks like generalizing across a wide range of track geometries and vehicular dynamics, driving under stochasticity and partial observability, navigating through static and moving obstacles and negotiating with other agents to pass through a traffic bottleneck. These studies demonstrate the viability of MADRaS to simulate rich highway and track driving scenarios of high variance and complexity that are valuable for autonomous driving research.

1.3.2 RAIL: A Risk Averse Imitation Learning Algorithm

We propose RAIL, a *R*isk *A*verse *I*mitation *L*earning algorithm which incorporates Conditional Value at Risk ($CVaR$) optimization [32] within the Generative Adversarial Imitation Learning (GAIL) [29] framework to minimize tail risk and thus improve the reliability of the learned policies. We report significant improvement over GAIL at a number of evaluation metrics on five benchmark continuous-control tasks of OpenAI Gym [33]. Thus the proposed algorithm is a viable step in the direction of learning low-risk policies by imitation learning in complex environments, especially in risk-sensitive applications like autonomous driving.

1.3.3 ExTra: A Transfer-guided Exploration Algorithm for Reinforcement Learning

We investigate the fundamental possibility of using transfer to guide exploration in RL and formulate a novel transfer guided exploration algorithm, ExTra, based

on the theory of bisimulation based policy transfer in MDPs [34]. We demonstrate that our method achieves faster convergence compared to traditional exploration methods that only use local information. Further, ExTra is robust to source task selection and can complement traditional exploration methods by improving their rates of convergence. We also provide theoretical guarantees in the form of a lower bound on the optimal advantage of an action in the target domain in terms of bisimulation distance from the source environment.

1.4 Thesis Organization

The rest of the thesis is structured as follows.

- Chapter 2 presents the essential mathematical concepts and prior literature that form the backbone of this thesis.
- Chapter 3 describes the construction of MADRaS and presents six case studies on its application as a platform for autonomous driving research.
- Chapter 4 presents RAIL and how it improves the tail risk of GAIL-learned policies.
- Chapter 5 describes ExTra and compares it with traditional exploration methods.
- Finally, Chapter 6 concludes the thesis with a summary of our contributions and scope of future work.

1.5 Summary

In this section we presented a brief introduction to the problem of autonomous driving that is one of the most vibrant areas of research in the current times. We described the motivation of this thesis and the research challenges that we aim to address. We also presented the thesis organization and briefly discussed our contributions.

CHAPTER 2

Background

2.1 Introduction

In this chapter, we discuss the theoretical concepts that form the basis of the research presented in this thesis. Out of the perceive-plan-act pipeline of autonomous driving [9], our focus lies on the task of planning. So, we start with a brief introduction to the theory of trajectory and motion planning in robotics [12] in Section 2.2. Traditional planning algorithms depend on the availability of an accurate model of the world. Decision theoretic planning algorithms, on the other hand, do not make this assumption and are capable of planning under uncertainty in real world conditions. One such class of algorithms is Reinforcement Learning (RL) [13] that has demonstrated remarkable success in the recent years [21, 22, 23, 24]. We present a brief introduction to the foundations of RL in Section 2.3 and discuss the challenges that face their application in the real world. The task of autonomous driving is a multi-agent planning problem that requires multiple actors with potentially different objectives to negotiate and arrive at a solution. In Section 2.4 we introduce the theory of Multi-Agent Reinforcement Learning [35, 36] that is used to train multiple learning agents to interact with one another and accomplish their individual as well as collective goals. RL agents

learn by maximizing a scalar reward function that is usually handcrafted to describe the intended task. As handcrafting is tedious and prone to human error, a family of algorithms called Imitation Learning [26, 27, 28] use expert demonstrations to bypass the reward engineering step of RL. We introduce Imitation Learning algorithms in Section 2.5. The introduction serves as a primer for Chapter 4 of this thesis where we discuss the reliability of imitation learning algorithms applied to risk-sensitive tasks like autonomous driving.

2.2 Motion Planning

Robot motion tasks are usually specified in terms of a pair of initial and final configurations of the robot and a set of constraints that the robot must abide by while executing the desired motion. Motion planning refers to the task of converting such high level specifications of tasks into low level descriptions comprising of translations and rotations of how to accomplish the task optimally [12]. Being a mechanical system, each robot has certain limitations on the kinds of motion it can execute in a given state. These contribute a set of differential constraints to the motion planning problem. Uncertainties in the environment and modeling errors pose additional challenges. In this section we describe the mathematical foundation of motion planning under differential constraints. We also discuss how decision theoretic approaches like machine learning can be used to address the uncertainties arising in real world conditions.

2.2.1 Configuration Space of a Robot

Robot motion planning problems are usually defined in terms of the configuration space of a robot. The configuration space is used to denote the set of all possible positions or configurations that the robot can find itself with respect to the environment while acting in it. Mathematically, the environment that the robot operates in is called the "*world*", $\mathcal{W}$. Usually, $\mathcal{W} = \mathbb{R}^2$ or $\mathbb{R}^3$. The robot is described as a semi-algebraic set $\mathcal{A} \subset \mathcal{W}$. $\mathcal{A}$ could be a rigid body or a collection of m links – $\mathcal{A}_1, \mathcal{A}_2, \ldots \mathcal{A}_m$. The *world* may contain an *obstacle region* represented by a semi-algebraic set $\mathcal{O} \subset \mathcal{W}$ where the robot is not allowed to tread. The *con-*

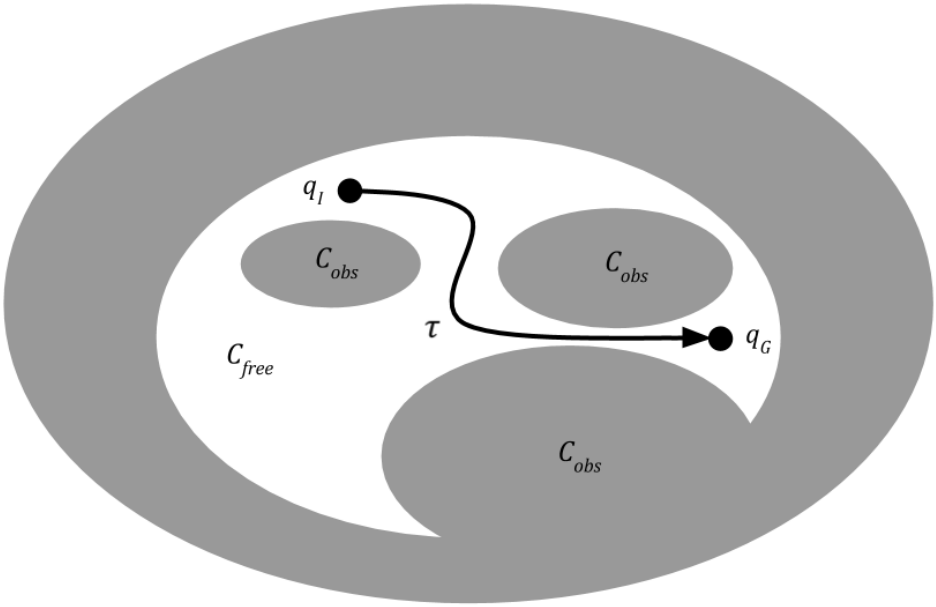


Figure 2.1: Piano Mover's Problem defined in the C-space of a robot. The task is to find a path τ from q_I to q_G in $\mathcal{C}_{free}$. The entire blob represents $\mathcal{C} = \mathcal{C}_{free} \bigcup \mathcal{C}_{obs}$.

figuration space or C-space, $\mathcal{C}$, is determined by specifying the set of all possible transformations that may be applied to the robot. For example, if the robot is a rigid body and $\mathcal{W} = \mathbb{R}^2$, a configuration $q \in \mathcal{C}$ is of the form (x, y, θ) where (x, y) are the coordinates of the centre of mass and θ is the orientation in the global coordinate system. Similarly, if $\mathcal{W} = \mathbb{R}^3$, then $q = (x, y, z, h)$ where (x, y, z) are the 3D coordinates of the centre of mass and h represents the unit quaternion.

The *obstacle space* $\mathcal{C}_{obs} \subset \mathcal{C}$ is defined as the subset of the configuration space that results in overlap of the robot and the obstacle region.

$$\mathcal{C}_{obs} = q \in \mathcal{C} | \mathcal{A}(q) \cap \mathcal{O} \neq \phi \tag{2.1}$$

$\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$ comprises the *free space* in which the robot is allowed to navigate.

2.2.2 Motion Planning: The Piano Mover's Problem

Given an *initial configuration* q_I and *goal configuration* q_G, $q_I, q_G \in \mathcal{C}_{free}$, the task of a motion planning algorithm is to compute a continuous *path*, $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$ such that $\tau(0) = q_I$ and $\tau(1) = q_G$. If a solution does not exist, the algorithm should correctly report that too. This simple version of the motion planning problem is known as the Piano Mover's Problem.

2.2.3 Differential Constraints

A robot operating in an environment must abide by certain constraints that are related to the outcomes of its interactions with the environment. These constraints are included in the differential equation based motion-planning problem definition and are known as differential constraints. Let X be the state space of a robot. The state space is usually the same as the configuration space, $\mathcal{C}$. The sub-spaces of X corresponding to $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$ are denoted by X_{obs} and X_{free} respectively. Let $U(x)$ denote the set of actions that the robot can take in state $x \in X$. Robot motion in continuous time can be described by the *state transition equation* as follows:

$$\dot{x} = f(x, u) \tag{2.2}$$

where, $\dot{x}$ refers to the robot velocity that results from taking action $u \in U(x)$ in state x. Starting from $x(0)$ at time $t = 0$, a *state trajectory* $\tilde{x}$ can be derived from an action trajectory $\tilde{u}$ as:

$$x(t) = x(0) + \int_0^t f(x(t'), u(t'))dt' \tag{2.3}$$

Constraints on robot motion are expressed in the form of differential equations. These equations take the form of:

$$g(\ddot{q}, \dot{q}, q) \bowtie 0 \tag{2.4}$$

where $\dot{q}$ and $\ddot{q}$ represent the first and second order temporal derivatives of the robot configuration variable q and $\bowtie$ could be $=$, $<$, $\leq$, $>$, or $\geq$. Differential

constraints are of three broad categories:

- **Position constraints**: These constraints define $\mathcal{C}_{obs}$. The robot must not enter this region of the configuration space during its operation in the environment.
- **Velocity or Kinematic constraints**: These constraints limit the velocities that the robot may have in a given configuration taking into account the mechanical limitations of the robot and restrictions in the environment.
- **Non-holonomic constraints**: These refer to constraints that can not be integrated over time and are related to the restricted movement abilities of a given robotic system.

These differential constraints are incorporated into the state transition equation of the robot using methods like *Lagrange Multipliers*.

2.2.4 Motion Planning under Differential Constraints

Motion Planning under differential constraints can be formulated as a classical *two-point boundary value problem* (BVP) [37]. Let U denote the set of all permissible action trajectories over an unbounded *time interval*, $T = [0, \infty)$. Let us assume that U contains at least one action trajectory $\tilde{u}$ such that the integrand of Equation 2.3 is integrable over time. Given an initial state $x_I \in X$ and a *goal region* $X_G \subset X$, the task of motion planning is to compute an *action trajectory* $\tilde{u} : T \to U$ for which the state trajectory $\tilde{x}$ resulting from Equation 2.3 satisfies:

1. $x(0) = x_I$
2. $\exists t > 0$ for which $u(t) = u_T$ and $x(t) \in X_G$.

In some cases, additional constraints such as continuity or smoothness over time, may be placed on $\tilde{u}$.

2.2.5 Planning under Uncertainty

In most real world scenarios planning algorithms need to take into consideration uncertainty in the environment. The uncertainty can be in the form of noisy or partial observations that prevent the robot from knowing its current state accurately. Also, the transition dynamics of the environment can be stochastic, which means that the outcome of an action in a given state can be probabilistic. This results in uncertainty in the robot's ability to predict future states. In most real world applications, the transition dynamics are unknown and the robot is required to model it explicitly or implicitly from the data collected during exploration. These challenges call for mathematical formulations that consider uncertainty as an integral part of the planning problem. Decision theoretic planning algorithms become relevant in this case. In the following section we discuss reinforcement learning algorithms that fall within this category.

2.3 Reinforcement Learning

In this section we present a brief introduction to Reinforcement Learning (RL). Reinforcement learning is a computational approach to *learning from interaction* [13]. Learning from interaction comprises a major form of learning from an early age in humans. Even in the absence of an expert supervisor, mere interaction with the environment provides a host of information regarding *cause and effect*, consequences of actions and how to achieve goals. With information on how the environment responds to actions, it is possible to plan behaviors that produce desired outcomes or accomplish useful tasks.

In reinforcement learning, the target task or goal is defined in the form of a scalar *reward signal*. After each action, an RL agent may receive a reward signal that signifies how close it is to achieving the target task. These reward signals are sometimes sparse, delayed and abstract. The agent learns its desired behavior by maximizing the reward signal through trial and error. RL algorithms, along with efficient function approximators like deep neural networks, have achieved human-level or beyond human-level performance at many challenging planning tasks like

continuous-control [21, 22] and game-playing [23, 24].

Reinforcement learning is different from the two most common forms of machine learning algorithms, viz. *supervised learning* and *unsupervised learning*. In supervised learning, the agent is given a set of example situations along with the correct actions that must be chosen in those situations. On the other hand, in an interactive learning setting like reinforcement learning, the agent must learn from its own experience. Feedback received in the form of reward signals only provide direction to the final goal and not examples of correct behavior. The task of unsupervised learning is to find hidden structures in a collection of *unlabelled* data. Reinforcement Learning is different from unsupervised learning because it aims to achieve a given task by maximizing the reward signal rather than just looking for meaningful structures in data.

In this thesis, we explore RL algorithms in the context of planning in autonomous driving. In the remaining part of this section, we introduce the mathematical formulation of reinforcement learning and describe some algorithms that have been studied in the later chapters of this thesis.

2.3.1 Markov Decision Process

A Markov Decision Process (MDP) is a mathematical construct that is used to formalize agent-environment interaction in a reinforcement learning scenario [13]. An MDP can be expressed as a 5-tuple: $\mathcal{M} = \langle S, A, P, R, \gamma \rangle$. It consists of the following elements:

- **State space:** The state space, S denotes the set of all possible states that the agent can find itself in during its interaction with the environment. A state $s \in S$ comprises of a set of state variables some or all of which may be observable to the agent. For an agent learning to drive a car, the state variables may include the current speed of the car, distance from the road boundaries, static and dynamic obstacles in the vicinity and engine parameters.

- **Action space**: The action space, A denotes the set of all actions available to the agent. For a driving agent, the action space can comprise of steering, acceleration, brake and other control variables.

- **Transition function**: The transition function, $P : S \times A \rightarrow S$ describes how the state of the environment changes in response to actions taken by the agent. In stochastic environments, the transition function is probabilistic and is written as $P : S \times A \times S \rightarrow [0, 1]$, where $\forall s \in S, a \in A, \int_{s'} P(s'|s, a) = 1$. The transition function definition makes the assumption that given the current state and action the next state is conditionally independent of all previous states and actions. This memory-less assumption is known as the Markovian assumption and this is what contributes the term "Markov" to the name Markov Decision Process.

- **Reward function**: The reward function, $R : S \times A \rightarrow \mathbb{R}$ characterizes the goal of the agent. It provides scalar feedback signals to the agent that indicate its progress towards the goal. The agent learns its desired behavior by maximizing the cumulative reward function.

- **Temporal discount factor**: The temporal discount factor, γ is a positive scalar that is used to provide different levels of importance to the short and long term outcomes of an agent's actions.

2.3.2 Policy and Trajectory

In an interactive learning environment, an agent chooses actions according to a *policy*. A policy, $\pi : S \rightarrow A$, is a function that maps states to actions. The goal of RL is to learn a policy that maximizes the cumulative reward accrued by the agent while acting in the environment. In many situations, especially when the environment is probabilistic, stochastic policies are used. A stochastic policy, $\pi : S \times A \rightarrow [0, 1]$, is defined as a probability distribution over actions, where $\forall s \in S, \int_a \pi(a|s) = 1$.

A *trajectory*, $\tau = \langle s^0, a^0, r^0, s^1, a^1, r^1, \ldots, s^t, a^t, r^t, \ldots \rangle$, $s^i \in S, a^i \in A, r^i \in \mathbb{R}$, is a sequence of state, action and reward values, that an agent produces while

interacting with the environment.

2.3.3 Returns and Episodes

The goal of an RL agent is to maximize the total reward it receives in the long run. The action chosen at time step t should maximize the sequence of rewards that follows this action, viz. $R(t), R(t+1), R(t+2)\ldots$ The *return* at time t, G_t is defined as a function of this sequence of rewards that consolidates them into the learning objective for time t. Some popular definitions of $G(t)$ are:

- **Sum of rewards:** $G(t) = \sum_{t' \geq 0} R(t + t')$.
- **Discounted sum of rewards:** $G(t) = \sum_{t' \geq 0} \gamma^{t'} R(t + t')$ where $0 \leq \gamma \leq 1$ is a discount factor.

While sum of future reward gives equal importance to all the time steps that follow, the discounted sum provides a way to assign more weightage to the immediate rewards than the ones in distant future. The amount of discounting of future rewards can be controlled by tuning the value of γ. When $\gamma = 1$, the discounted sum of future rewards becomes a simple sum.

There are some tasks in which experience happens in the form of repeated interactions that start in one of a set of *initial states* and terminate in one of a set of *terminal states*. Examples of such tasks are plays of a game, trips through a maze, etc. In such tasks, the agent-environment interactions can be naturally broken into sub-sequences called *episodes*. The branch of reinforcement learning that deals with episodic tasks is known as *episodic reinforcement learning* and will be the prime focus of this thesis. The maximum length of an episode is referred to as the *horizon*, T. The other class of tasks in which the agent's interaction with the environment goes on continually for ever and does not break naturally into episodes are known as *continuing tasks*. Although we do not investigate such tasks in this thesis, the concepts developed can be applied without any major modification to them as well.

2.3.4 Exploration-Exploitation Dilemma

When a reinforcement learning agent interacts with the environment with a goal to maximize its cumulative reward, it has two options regarding how to choose its actions. The first option is to *exploit* its best policy till the current time that is known to give high rewards. The second option is to *explore* by trying to take actions different from the one given by the best known policy as there could potentially be a better strategy capable of producing a higher reward. Barring some trivial cases, following any one of the two strategies exclusively can never produce optimal behavior. The agent must explore actions in each state but progressively favor the ones that appear to be the best. The *sample complexity* of an RL algorithm is defined as the total number of interactions that the agent needs to experience in order to arrive at the final performance. Reduction of sample complexity is crucial for practical application of RL and a large section of RL literature is dedicated to the development of *sample efficient* RL algorithms. One of the most effective approaches to reducing sample complexity is to formulate sample-efficient exploration methods. We study these methods in Chapter 5 and present a novel framework of transfer-guided exploration called "ExTra", that leverages an agent's prior experiences at solving related tasks for guiding exploration in a new task environment.

2.3.5 Value Functions

The value function of a policy gives an estimate of how useful it is for an agent to be in a given state or how useful it is to take a certain action in a given state. Utility is usually quantified in terms of expected return. Almost all reinforcement learning algorithms involve estimating value functions. There are two major types of value functions that are widely used in the reinforcement learning literature.

- **State Value Function:** For any state $s \in S$, the state value function, $V^{\pi}(s)$, is defined as the expected return that the agent would get if it starts

in state s and follows policy π.

$$\begin{aligned} V^{\pi}(s) =& \mathbb{E}_{\pi}\left[G(t)|s^t = s\right] \\ =& \mathbb{E}_{a^t, a^{t+1}, \cdots \sim \pi}\left[G(t)|s^t = s\right] \end{aligned} \tag{2.5}$$

$\mathbb{E}_{\pi}$ denotes expectation over actions derived from policy π.

- **State-Action Value Function:** Given a state $s \in S$ and action $a \in A$, the state-action value function, $Q^{\pi}(s, a)$ is defined as the expected return that the agent would get if it takes action a in state s and follows the policy π thereafter.

$$\begin{aligned} Q^{\pi}(s, a) =& \mathbb{E}_{\pi}\left[G(t)|s^t = s, a^t = a\right] \\ =& \mathbb{E}_{a^{t+1}, a^{t+2}, \cdots \sim \pi}\left[G(t)|s^t = s, a^t = a\right] \end{aligned} \tag{2.6}$$

2.3.6 Bellman Equations

The value functions have interesting recursive properties that are useful in their estimation from data. These relationships are known as Bellman Equations. For a discounted sum of rewards definition of gain, we have the following expressions of the Bellman equation for state and state-action value functions.

2.3.6.1 Bellman Equation for State Value Function

The Bellman equation for the state value function V^{π} can be stated as follows:

$$V^{\pi}(s) = \mathbb{E}_{a \sim \pi(a^t|s)}\left[R(s, a) + \gamma \mathbb{E}_{s' \sim P(s^{t+1}|s,a)}\left[V^{\pi}(s')\right]\right] \tag{2.7}$$

Proof:

$$\begin{aligned} V^{\pi}(s) =& \mathbb{E}_{\pi}\left[G(t)|s^t = s\right] \\ =& \mathbb{E}_{\pi}\left[R(t) + \gamma R(t+1) + \gamma^2 R(t+2) + \gamma^3 R(t+3) + \ldots |s^t = s\right] \\ =& \mathbb{E}_{\pi}\left[R(t) + \gamma(R(t+1) + \gamma R(t+2) + \gamma^2 R(t+3) + \ldots)|s^t = s\right] \\ =& \mathbb{E}_{\pi}\left[R(t) + \gamma G(t+1)|s^t = s\right] \end{aligned}$$

$$
\begin{aligned}
&= \mathbb{E}_{a\sim\pi(a^t|s)}\left[R(s,a) + \gamma \mathbb{E}_{s'\sim P(s^{t+1}|s,a)}\left[G(t+1)|s^{t+1}=s'\right]\right] \\
&= \mathbb{E}_{a\sim\pi(a^t|s)}\left[R(s,a) + \gamma \mathbb{E}_{s'\sim P(s^{t+1}|s,a)}\left[V^{\pi}(s')\right]\right]
\end{aligned}
$$

2.3.6.2 Bellman Equation for State-Action Value Function

The Bellman equation for the state-action value function Q^{π} is given by:

$$
Q^{\pi}(s,a) = R(s,a) + \gamma \mathbb{E}_{s'\sim P(s^{t+1}|s,a)}\left[\mathbb{E}_{a'\sim\pi(a^{t+1}|s')}\left[Q^{\pi}(s',a')\right]\right] \tag{2.8}
$$

Proof:

$$
\begin{aligned}
Q^{\pi}(s,a) =& \mathbb{E}_{\pi}\left[G(t)|s^t=s, a^t=a\right] \\
=& \mathbb{E}_{\pi}\left[R(t) + \gamma R(t+1) + \gamma^2 R(t+2) + \gamma^3 R(t+3) + \ldots | s^t = s, \right. \\
& \left. a^t = a\right] \\
=& R(s,a) + \mathbb{E}_{\pi}\left[\gamma(R(t+1) + \gamma R(t+2) + \gamma^2 R(t+3) + \ldots)\right] \\
=& R(s,a) + \mathbb{E}_{\pi}\left[\gamma G(t+1)\right] \\
=& R(s,a) + \gamma \mathbb{E}_{s'\sim P(s^{t+1}|s,a)}\left[\mathbb{E}_{a'\sim\pi(a^{t+1}|s')}\left[G(t+1)|s^{t+1}=s', \right.\right. \\
& \left.\left. a^{t+1}=a'\right]\right] \\
=& R(s,a) + \gamma \mathbb{E}_{s'\sim P(s^{t+1}|s,a)}\left[\mathbb{E}_{a'\sim\pi(a^{t+1}|s')}\left[Q^{\pi}(s',a')\right]\right]
\end{aligned}
$$

2.3.7 Optimal Policy and Value Functions

The goal of reinforcement learning is to find a policy for the agent that fetches highest expected return starting from any of the initial states. A policy π is defined to be better than or equivalent to a policy π' if and only if $V^{\pi}(s) \geq V^{\pi'}(s), \forall s \in S$. In this way, the value function imposes a partial order over policies. There always exists at least one policy that is better than or equivalent to all other policies. We call this an *optimal policy*. We represent all these optimal policies by π^*. All optimal policies share the same state and state-action value functions. They are known as the *optimal value functions* and denoted by V^* and Q^*.

$$V^*(s) = \max_\pi V^\pi(s);\ \forall s \in S \tag{2.9}$$

$$Q^*(s,a) = \max_\pi Q^\pi(s,a);\ \forall s \in S,\ a \in A \tag{2.10}$$

The optimal value functions satisfy the *Bellman Optimality Equations*:

$$V^*(s) = \max_a \left[R(s,a) + \gamma \mathbb{E}_{s' \sim P(s'|s,a)} V^*(s')\right] \tag{2.11}$$

$$Q^*(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \sim P(s'|s,a)} \max_{a'} Q^*(s',a') \tag{2.12}$$

The Bellman Optimality Equations are used to derive update rules for estimating the optimal value functions by dynamic programming.

2.3.8 Policy Evaluation

Given an arbitrary policy π, *policy evaluation* refers to the task of finding the value of the policy V^π. Estimating V^π from the Bellman equation (Equation 2.7) amounts to solving $|S|$ simultaneous equations when $P(s'|s,a)$ is known. This could be a tedious process when $|S|$ is large. A more efficient method of estimating is through an iterative process. Let us consider a sequence of approximate value functions $V_0^\pi, V_1^\pi, V_2^\pi, \ldots$ where $V_i^\pi : S \to R$. V_0^π, the initial approximation, is chosen arbitrarily and each successive approximation is obtained from the previous one using the Bellman equation as an update rule:

$$V_{k+1}^\pi(s) = \mathbb{E}_{a \sim \pi(s)}[R(s,a) + \gamma \mathbb{E}_{s' \sim P(s'|s,a)} V_k^\pi(s')] \tag{2.13}$$

$V_k^\pi = V^\pi$ is a *fixed point* for this update rule since Bellman equation gives equality in this case. If V^π exists, the sequence $\{V_k^\pi\}_k$ converges to V^π as $k \to \infty$. This algorithm for determining V^π iteratively is known as *iterative policy evaluation.*

Algorithm 1 Iterative Policy Evaluation

Input: An arbitrary policy $\pi : S \rightarrow A$, A small positive tolerance threshold parameter $\theta > 0$

Output: Value of the policy V^{π}

Initialization:
Assign $V^{\pi}(s) \in \mathbb{R}$ arbitrarily $\forall s \in S \setminus \{s_{terminal}\}$ and $V^{\pi}(s_{terminal}) = 0$
Policy evaluation:
repeat
    $\Delta \leftarrow 0$
    for each $s \in S$ **do**
        $v \leftarrow V(s)$
        $V^{\pi}(s) \leftarrow E_{a \sim \pi(s)} \left[R(s,a) + \gamma E_{s' \sim P(s'|s,a)} V^{\pi}(s') \right]$
        $\Delta \leftarrow \max(\Delta, |v - V^{\pi}(s)|)$
    end for
until $\Delta < \theta$

2.3.9 Dynamic Programming Methods for Policy Improvement

The primary motivation behind the construction of value functions is to improve the policy of a reinforcement learning agent. The *policy improvement theorem* formalizes the idea. Let π and π' be a pair of deterministic policies such that,

$$\forall s \in S, Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s) \tag{2.14}$$

Then,

$$\forall s \in S, V^{\pi'}(s) \geq V^{\pi}(s) \tag{2.15}$$

In other words, π' must be as good as, or better than π, in terms of expected return. We construct a new *greedy* policy π' as:

$$\begin{aligned} \pi'(s) =& arg\max_{a} Q^{\pi}(s,a) \\ =& arg\max_{a} \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a) + \gamma V^{\pi}(s') \right] \end{aligned} \tag{2.16}$$

This greedy policy takes actions that look best in the short term with one step look-ahead in terms of V^{π}. Note that π' satisfies the conditions of the policy

improvement theorem and hence it is at least as good as π in terms of expected return. This process of improving a given policy by making it greedy with respect to its own value function is known as *policy improvement*.

From the definition of π^*, we have:

$$V^{\pi'}(s) = \max_a \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a) + \gamma V^{\pi}(s') \right] \tag{2.17}$$

By construction, if π' is not better than π, π' must be as good as π, or in other words, $V^{\pi'}(s) = V^{\pi}(s),\ \forall s \in S$. Hence, we have in this case, from equation 2.17:

$$V^{\pi'}(s) = \max_a \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a) + \gamma V^{\pi'}(s') \right] \tag{2.18}$$

Equation 2.18 is the same as the Bellman optimality equation. This shows that policy improvement must give us a *strictly* better policy except when the original policy is already optimal.

Starting from an initial policy π_0 we can apply policy improvement iteratively and produce a sequence of monotonically improving policies until we arrive at the optimal policy. This algorithm is known as *policy iteration*.

A drawback of policy iteration is the fact that it performs a full policy evaluation in every iteration which poses huge computational burden. We aim to truncate the policy evaluation step without losing the convergence guarantees of policy iteration. *Value Iteration* is one such method in which only a single step of policy evaluation is performed after each policy update. The policy evaluation and policy update steps can be combined into a single update step as follows:

$$V^{\pi}_{k+1}(s) = \max_a [R(s,a) + \gamma \mathbb{E}_{s' \sim P(s'|s,a)} V^{\pi}_k(s')];\ \forall s \in S \tag{2.19}$$

If V^* exists, for arbitrary V^{π}_0, the sequence $\{V^{\pi}_k\}_k$ converges to V^*. Value iteration can also be arrived at by using the Bellman optimality equation as an update rule.

Dynamic programming (DP) based RL algorithms are also called *bootstrapping*

Algorithm 2 Policy Iteration.

1: **Initialization:**
2: Assign $V^{\pi}(s) \in \mathbb{R}$ and $\pi(s) \in A$ arbitrarily $\forall s \in S$
3: **Policy evaluation:**
4: **repeat**
5: $\quad \Delta \leftarrow 0$
6: $\quad$ **for** each $s \in S$ **do**
7: $\quad\quad v \leftarrow V(s)$
8: $\quad\quad V^{\pi}(s) \leftarrow E_{a \sim \pi(s)} \left[R(s,a) + \gamma E_{s' \sim P(s'|s,a)} V^{\pi}(s') \right]$
9: $\quad\quad \Delta \leftarrow \max(\Delta, |v - V^{\pi}(s)|)$
10: $\quad$ **end for**
11: **until** $\Delta < \theta$ $\quad \triangleright\ \theta > 0$ is a small number denoting accuracy of estimation
12: **Policy improvement:**
13: $policy\_stable \leftarrow true$
14: **for** each $s \in S$ **do**
15: $\quad old\_action \leftarrow \pi(s)$
16: $\quad \pi(s) \leftarrow arg\max_a \mathbb{E}_{s' \sim P(s'|s,a)}[R(s,a) + \gamma V^{\pi}(s')]$
17: $\quad$ **if** $old\_action \neq \pi(s)$ **then**
18: $\quad\quad policy\_stable \leftarrow false$
19: $\quad$ **end if**
20: $\quad$ **if** $policy\_stable$ **then**
21: $\quad\quad$ Stop and return $V^{\pi} \approx V^{*}$ and $\pi \approx \pi^{*}$
22: $\quad$ **else**
23: $\quad\quad$ Go to line 3
24: $\quad$ **end if**
25: **end for**

methods [13] because they use the current estimate of the value function to calculate the next estimate. A major drawback of DP methods like Policy Iteration and Value Iteration is their assumption of complete knowledge of the environment. Calculating the expectations over next states $s' \sim P(s'|s,a)$ in Algorithms 2 and 3 require the transition dynamics P to be known. In RL parlance, these methods are *model based*. However, in most practical settings, a model of the transition dynamics is not available. Model-free reinforcement learning methods like Monte Carlo and Temporal Difference (TD) Learning attempt to address this limitation by using sample-based estimates of the expectations. We discuss these methods later in this section.

Algorithm 3 Value Iteration

Input: A small positive tolerance threshold parameter $\theta > 0$

Initialization:
Assign $V^{\pi}(s) \in \mathbb{R}$ arbitrarily $\forall s \in S \{ s_{terminal}$ and $V^{\pi}(s_{terminal}) = 0$
repeat
    $\Delta \leftarrow 0$
    for each $s \in S$ **do**
        $v \leftarrow V(s)$
        $V^{\pi}(s) \leftarrow \max_a \left[R(s,a) + \gamma E_{s' \sim P(s'|s,a)} V^{\pi}(s') \right]$
        $\Delta \leftarrow \max(\Delta, |v - V^{\pi}(s)|)$
    end for
until $\Delta < \theta$
Output a deterministic policy, $\pi \approx \pi^*$, such that:

$$\pi(s) = arg \max_a \left[R(s,a) + \gamma E_{s' \sim P(s'|s,a)} V^{\pi}(s') \right]$$

2.3.10 Generalized Policy Iteration

Both policy and value iteration algorithms involve the interaction of two processes happening simultaneously:

- *Policy evaluation:* Making the value function consistent with the current policy.
- *Policy improvement:* Making the policy greedy with respect to the current value function.

This interaction is known as *Generalized Policy Iteration* (GPI) [13]. Almost all reinforcement learning algorithms can be described as a GPI algorithm.

2.3.11 Monte Carlo Methods for Reinforcement Learning

Monte Carlo methods for reinforcement learning use sample average of returns as model-free estimates of their true expectations [13]. Hence these methods become useful when a model of the transition dynamics is not available. The value estimate and policy are updated only at the end of each episode when all the returns have been observed. This is in contrast with Dynamic Programming based RL

where the updates to the value function are bootstrapped without waiting for the final outcome.

In the absence of a model, Monte Carlo methods attempt to obtain the optimal policy by estimating the optimal state-action value function, Q^*. In order to make sure that every possible state-action pair is visited, we assume that each state-action pair has a finite probability of being chosen at the start of an episode. This assumption is known as *exploring starts* [13]. In Monte Carlo policy iteration, we perform policy evaluation and policy improvement after each episode. At the end of an episode, the observed returns are used to update the Q value function for the state-action pairs visited in the episode (policy evaluation). The updated Q function is then used to update the policy for the states visited in that episode (policy improvement). Algorithm 4 presents the pseudo-code of Monte Carlo policy optimization with exploring starts. A drawback of Monte Carlo methods is their difficulty in applying to continuing tasks (trajectory length $\rightarrow \infty$) as the updates are computed only after a full trajectory has been rolled out.

Algorithm 4 Monte Carlo with Exploring Starts (*first visit*) [13]

Initialization:
Assign $\pi(s) \in A$ arbitrarily, $\forall s \in S$
Assign $Q^\pi(s,a) \in \mathbb{R}$ arbitrarily, $\forall s \in S, a \in A$
Assign an empty list to $Returns(s,a)$, $\forall s \in S, a \in A$
repeat forever:
    Sample s^0, a^0 from an initial state-action distribution that gives non-zero probability to all state-action pairs.
    Roll out an episode starting with s^0, a^0 following π: $\left\langle s^0, a^0, r^1, s^1, a^1, r^2, \ldots, s^{T-1}, a^{T-1}, r^T \right\rangle$.
    $G \leftarrow 0$
    for each step of the episode, $t = T-1, T-2, \ldots, 0$ **do**
        $G \leftarrow \gamma G + r^{t+1}$
        if the pair s^t, a^t does not appear in $\langle s^0, a^0, s^1, a^1, \ldots, s^{t-1}, a^{t-1} \rangle$ **then**
            Append G to $Returns(s^t, a^t)$
            $Q^\pi(s^t, a^t) \leftarrow \text{average}(Returns(s^t, a^t))$
            $\pi(s^t) \leftarrow arg\max_a Q(s^t, a)$
        end if
    end for

2.3.12 Temporal Difference (TD) Learning

Temporal Difference (TD) Learning algorithms leverage the best of both Dynamic Programming and Monte Carlo methods and present an approach that incorporates both bootstrapping and sampling-based estimation [13]. Like Monte Carlo, TD algorithms are model-free, meaning, they can learn directly from experiences without access to a model of the environment's transition dynamics. Unlike Monte Carlo, TD algorithms do not wait till the end of an episode to update the value function and the policy. They just wait till the next *one or few* time steps in order to make an update. The simplest form of TD learning known as TD(0) waits for just a single step and has the following update equation:

$$V(s^t) \leftarrow (1-\alpha)V(s^t) + \alpha(r^{t+1} + \gamma V(s^{t+1})) \tag{2.20}$$

α is known as the *step-size* or the *learning rate*. The second part of the right hand side of Equation 2.20 is the bootstrapped target value obtained by one step look-ahead at time step t. We rewrite this equation as:

$$\begin{aligned} V(s^t) \leftarrow & V(s^t) + \alpha(r^{t+1} + \gamma V(s^{t+1}) - V(s^t))) & (2.21) \\ = & V(s^t) + \alpha\delta^t & (2.22) \end{aligned}$$

$\delta^t = r^{t+1} + \gamma V(s^{t+1}) - V(s^t)$ is called the *TD error*. TD learning algorithms usually converge faster than Monte Carlo and are also applicable to continuing tasks. Popular TD algorithms are SARSA (on-policy) and Q-Learning (off-policy) [13]. Algorithm 5 describes Q-learning. We use this algorithm in Chapter 5 in our study of efficient exploration methods for RL.

2.4 Multi-Agent Reinforcement Learning

Many real world tasks including autonomous driving involve the interaction of multiple agents. In a multi-agent environment, the outcome of an agent's actions depends upon the actions chosen by the other agents. This poses additional challenges that require special treatment beyond the scope of traditional single-agent

Algorithm 5 Q-learning [13]

Input: learning rate $\alpha \in (0, 1]$, small $\epsilon > 0$

Initialization:
Assign $Q^{\pi}(s, a) \in \mathbb{R}$ arbitrarily $\forall s \in S \setminus \{s_{terminal}\}, a \in A$; $Q(s_{terminal}, \cdot) = 0$
repeat for each episode:
    Get initial state, $s \in S$
    repeat for each step of the episode
        Choose action a greedily from $Q(s, \cdot)$ with probability $(1 - \epsilon)$ and randomly with probability ϵ (ϵ-greedy)
        Take action a and observe reward r and next state s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$
    until s is a terminal state

reinforcement learning frameworks. In this section we present a brief introduction to Multi-Agent Reinforcement Learning (MARL) and the associated challenges [35, 36]. We utilize these concepts in Chapter 3, where we teach simulated self-driving cars to cooperate and pass through a traffic bottleneck using MARL.

2.4.1 Markov Games

Markov Games (MG) or Stochastic Games [38, 39] are a generalization of Markov Decision Processes that are popularly used to formalise MARL settings. A Markov Game comprises of a tuple: $\langle S, \{\alpha_i\}_{i=1}^{n}, \{A_i\}_{i=1}^{n}, P, \{R_i\}_{i=1}^{n}, \gamma \rangle$. S denotes the state space of the environment that is observed by all the agents. $\{\alpha_i\}_{i=1}^{n}$ denotes a set of n agents that simultaneously learn to act in the environment. Each agent, α_i, can have a different action space, A_i, and a reward function, R_i, that governs its objective. Let $A := A_1 \times A_2 \times \cdots \times A_n$ be the joint action space of all the agents. The environment dynamics function $P : S \times A \times S \rightarrow [0, 1]$ gives the probability of the environment transitioning from state $s \in S$ to state $s' \in S$ in response to a joint action $a \in A$. γ is the temporal discount factor.

The policy learned by agent α_i is denoted by $\pi_i : S \times A_i \rightarrow [0, 1]$. The joint policy $\pi : S \times A \rightarrow [0, 1]$ is defined as $\pi(a|s) := \Pi_{i=1}^{N} \pi_i(a_i|s)$, $a_i \in A_i, \forall s \in S$. As in a multi-agent environment, the outcome of an agent's action depends upon the

actions of the other agents, the state value function, $V_i : S \to \mathbb{R}$ of agent α_i is defined with respect to the joint policy π as follows:

$$V_i^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t R^i(s^t, a_i^t) \middle| a_i^t \sim \pi_i(\cdot|s^t), s^0 = s\right] \tag{2.23}$$

When we investigate the individual policy, π_i, in the context of the joint policy, π, we write $\pi = \pi_i \pi_{-i}$, where π_{-i} denotes the joint policy of all the agents other than α_i. The solution of the Markov Game depends on the choices of all the agents involved. The most common solution concept is known as Nash Equilibrium (NE) [40] that characterizes an equilibrium point where none of the agents have any incentive to deviate. The Nash Equilibrium of a Markov Game is defined as a joint policy $\pi^* = \pi_1^* \pi_2^* \dots \pi_n^*$ such that, $\forall s \in S$ and $i \in \{1, \dots, n\}$:

$$V_i^{\pi_i^* \pi_{-i}^*}(s) \geq V_i^{\pi_i \pi_{-i}^*}(s), \qquad \forall \pi_i \tag{2.24}$$

π_i^* can be interpreted as the best response of agent α_i to the the joint policy of the other agents π_{-i}^*. It can be shown that there always exists at least one NE for discounted Markov Games [41].

2.4.2 Types of Multi-Agent Reinforcement Learning Algorithms

In MARL, multiple agents operate in a common environment. The objectives of these agents may or may not align with one another. The agents interact with one another and the environment and update their policies in order to maximize their individual long term returns. Depending on how the agents' objectives relate to one another, MARL algorithms fall under three broad categories:

1. ***Fully cooperative:*** In a fully cooperative setting, all the agents have a common reward function $R_1 = R_2 = \dots = R_n = R$. In this special setting, Markov Games are called Multi-Agent Markov Decision Processes (MMDP).

2. ***Fully competitive:*** A fully competitive Markov Game is a *zero-sum* game, meaning:

$$\sum_{i=1}^{n} R_i(s, a_i) = 0, \qquad \forall s \in S, a_i \in A_i \tag{2.25}$$

In a two-agent fully competitive game, the reward of one agent is exactly the loss of the other.

3. ***Mixed:*** This refers to the *general-sum* game setting in which there is no strictly cooperative or competitive relationship among the agents.

2.4.3 Challenges in Multi-Agent Reinforcement Learning

In MARL, each learning agent is allowed to have a different goal. Hence, the objectives of all the agents do not necessarily align with one another [42]. As a result, in many cases, finding a good equilibrium point becomes difficult. This also calls for additional performance criteria beyond reward maximization, such as, *communication efficiency* and *robustness* against adversarial agents. Since all the learning agents update their policies concurrently, the environment becomes *non-stationary* from the point of view of the individual agents [43]. Due to the *combinatorial nature* of MARL, the dimension of the joint action space, A, increases exponentially with the number of learning agents. This affects the scalability of MARL algorithms [44]. The agents in a Markov Game often choose to share their observations and actions. However an agent might want to share different sets of variables with different agents. Thus the *structure of information sharing* among the agents adds a new dimension to the nature and complexity of the learning problem [45].

2.5 Imitation Learning

In Section 2.3, we discussed Reinforcement Learning (RL), a family of algorithms that is used to learn complex behaviors in unknown environments through trial and error by maximizing a reward signal. In the RL paradigm, the reward function completely defines the task of the agent in the environment. While RL can in theory learn from sparse rewards that denote success or failure at achieving the

global objective, oftentimes, learning complex tasks from sparse rewards poses a huge sample complexity that makes it computationally intractable. In such tasks, dense reward shaping can help in learning to behave within a reasonable sample budget [46].

In most reinforcement learning experiments, hand-engineering reward functions is necessary for achieving the required behavior in the agent. Hand engineering is often tedious and introduces subjective bias. Mis-specification of the reward function often poses detrimental as the agent learns to exploit loopholes in the reward function definition and accrue a large amount of reward without executing the desired behavior [25]. Imitation learning or learning from demonstration [26, 27, 47, 48, 28, 49] aims to address this issue by learning a behavior directly from an expert's demonstrations.

2.5.1 Types of Imitation Learning

An expert demonstration is a sequence of state-action pairs denoting an episode of the expert's interaction with the environment. Depending on how the expert data is used for learning a policy, imitation learning algorithms fall in two broad categories:

1. Behavioral Cloning
2. Apprenticeship Learning

2.5.1.1 Behavioral Cloning

Behavioral cloning algorithms [50, 51, 52] use supervised learning to directly learn a policy by fitting to the expert data. The main attraction of Behavioral cloning algorithms is their simplicity and efficacy in short-term planning problems. However, it has some major drawbacks. As the expert's actions are considered to be "gold standard" for the corresponding states and the agent is penalised for deviating from the expert's actions, the trained agent's performance is always

upper-bounded by the expert's performance. If the expert's behavior is not optimal, there is no way for the agent to exceed the expert's performance and learn to behave more optimally. Also, behavioral cloning algorithms suffer from the problem of compounding error due to covariate shift [53, 54], especially in tasks that require long-term planning. Supervised learning treats each state-action example from the expert data as independent and identically distributed (i.i.d.). However, in a sequential decision making problem, the action at the current time step influences the observation at the next time step. Hence the i.i.d. assumption fails. Several methods have been proposed in literature to combat compounding error. DAGGER [14], SEARN [55], and scheduled sampling [56] are some of them.

2.5.1.2 Apprenticeship Learning

Apprenticeship learning [28] takes an indirect approach to policy learning from expert demonstration. It uses the expert data to learn the reward function that the expert maximizes for achieving its behavior. The algorithm alternates between two steps using a minorization maximization [57] approach. In the first step, the agent models the expert's reward function from its demonstration data. This step is known as Inverse Reinforcement Learning [58, 49, 48]. In the second step, it performs reinforcement learning by maximizing the estimated reward function to learn a policy. Apprenticeship learning is often interpreted as Reinforcement Learning over Inverse Reinforcement Learning ($RL \circ IRL$) and is expressed as:

$$RL \circ IRL(\pi_E) = \underset{\pi \in \Pi}{argmin} \max_{c \in \mathcal{C}} \mathbb{E}_{\pi}[c(s,a)] - \mathbb{E}_{\pi_E}[c(s,a)] - H(\pi) \tag{2.26}$$

where, π_E denotes the expert-policy. $c(\cdot,\cdot)$ denotes the *cost function*, the negative of reward function. Π and $\mathcal{C}$ denote the hypothesis classes for policy and cost functions. $H(\pi)$ denotes entropy of policy π. The term $-H(\pi)$ provides causal-entropy regularization [59, 60] which helps in making the policy optimization algorithm unbiased to factors other than the expected reward.

2.6 Summary

In this chapter we discussed some theoretical concepts that form the background of the thesis. We described the mathematical formulation of robot motion planning under differential constraints and Reinforcement Learning algorithms for planning under uncertainty. These ideas comprise the central theme of this thesis. We discussed about the importance of reward shaping for sample-efficient learning of complex tasks using RL. We also described how imitation learning can be used to bypass the tedious task of hand-engineering of reward functions by leveraging expert demonstrations. These discussions form the background of our contributions in Chapter 4 where we develop RAIL, an algorithm for Risk-Averse Imitation Learning that minimizes worst case risk for sensitive applications like autonomous driving. We also discussed the issue of sample complexity of RL algorithms and how exploration can be made efficient to reduce it. This serves as a primer for Chapter 5 where we present ExTra, a framework for Transfer-guided Exploration that uses the optimal policy of a known a task-environment to expedite exploration in an unknown task environment.

CHAPTER 3

MADRaS: Multi-Agent Driving Simulator

3.1 Introduction

In this chapter we present MADRaS, an open-source Multi-Agent DRiving Simulator for design and evaluation of motion planning algorithms for autonomous driving. Machine Learning based approaches to motion planning like Reinforcement Learning (RL) [13] and Learning from Demonstration (LfD) [14] are capable of learning policies for complex reactive control [15, 16]. However the training phase of these algorithms is often data-hungry [61, 62] especially for those using highly expressive and complex models like deep neural networks. RL based methods also require online interaction with the environment that entails risk [63, 64]. Driving simulators attempt to address these problems by rendering realistic driving conditions and traffic patterns in which agents can collect training data many times faster than real time. They also provide a sandbox environment where the agent can run into catastrophic situations while learning to drive without causing physical damage in the real world.

MADRaS provides a framework for constructing a wide variety of highway and track driving scenarios where multiple driving agents can train using reinforcement learning and other machine learning algorithms. Based on the TORCS platform [31], it uses simplified physics simulation and representative graphics to reduce the computational overhead for perception and action. Each driving agent gets a high-level object-oriented representation of the world as observation and an OpenAI gym interface for independent control. TORCS offers a variety of cars with different dynamic properties and driving tracks with different geometries and surface properties. MADRaS inherits these functionalities from TORCS and introduces support for multi-agent training, inter-vehicular communication, noisy observations, stochastic actions, and custom traffic cars whose behaviors can be programmed to simulate challenging traffic conditions encountered in the real world. MADRaS can be used to create driving tasks whose complexities can be tuned along eight axes in well defined steps. This makes MADRaS particularly suited for curriculum and continual learning. MADRaS is lightweight and it provides a convenient OpenAI Gym [33] interface for independent control of each car. Apart from the primitive steering – acceleration – brake control mode of TORCS, MADRaS also offers a higher level track position – speed control mode that can potentially be used to achieve better generalization. MADRaS uses a UDP based server-client model [65] where the simulation engine is the server and each client is a driving agent. MADRaS uses multiprocessing to run each agent as a parallel process for efficiency and integrates well with popular reinforcement learning libraries like Berkeley RLLib [66], RLPyT [67], OpenAI Baselines [68] and Intel RLCoach [69]. MADRaS is open source[1] and aims to contribute to the democratization of artificial intelligence.

3.2 Related Work

Since the early days of autonomous driving research, simulators have been used in the development of different parts of the perceive-plan-act pipeline [70]. Most of these simulators cater to the task of perception. Back in 1989, the creators of

[1] http://github.com/madras-simulator/MADRaS

Autonomous Land Vehicle In a Neural Network (ALVINN), Pomerleau et al. [71], had used a simulator to generate training images for road detection. Thanks to the recent advances in computer graphics, modern driving simulators and games like GTA-V [72] can render photo-realistic driving scenes with accurate depiction of illumination, weather and other physical phenomena. They also simulate real-life sensors that can be used to collect synthetic data from these scenes to augment real-world driving datasets. Recent works [73, 74, 75, 76] have demonstrated that training perception algorithms on these augmented datasets result in better generalization in the real world that is crucial for safe and reliable autonomous driving. Most notable open-source driving simulators in this category are CARLA [17], Microsoft AirSim [18], DeepDrive.io [19] and Udacity's Self Driving Car Simulator [20]. These simulators can, in principle, be also used for planning. However, an agent learning to face real world driving scenarios must learn to be invariant to road geometries, traffic patterns and vehicular dynamics. These simulators do not offer enough variability along these dimensions that is necessary to learn the invariances. In a typical driving scene, multiple entities (cars, buses, bikes, and pedestrians) try to achieve their objectives of getting from one place to another fast, yet safely and reliably. A simulator for such an environment should provide an easy way to create arbitrary traffic configurations. The task of negotiating in traffic is akin to finding the winning strategy in a multi-agent game [77]. Hence, an autonomous driving simulator should be able to simulate different varieties of traffic and support multiple agents learning to negotiate and drive through cooperation and competition. Among the aforementioned simulators, AirSim, DeepDrive.io and Udacity provide some preset driving conditions mostly without traffic. They do not provide any straightforward way to create custom traffic or train multiple agents. CARLA does provide an API for independent control of cars that can be used for multi-agent training and creating custom traffic cars. However, most of the variability presented by CARLA is in the perceived inputs and not in the behavioral dynamics of the ego-vehicle or the traffic agents. This motivated us to develop a dedicated simulator for learning to plan in autonomous driving with a focus on learning invariances to road geometries, traffic patterns and vehicular dynamics in both single and multi-agent learning settings.

3.3 Background

Before describing the structure and organization of the MADRaS simulator, we present a brief overview of the TORCS simulator and associated prior works on which MADRaS has been built. Afterwards, we describe our contributions and discuss their relevance in the context of planning in autonomous driving.

3.3.1 TORCS Simulator

MADRaS is based on TORCS which stands for The Open Racing Car Simulator [31]. It is capable of simulating the essential elements of vehicular dynamics such as mass, rotational inertia, collision, mechanics of suspensions, links and differentials, friction and aerodynamics. Physics simulation is simplified and is carried out through Euler integration of differential equations at a temporal discretization level of 0.002 seconds. The rendering pipeline is lightweight and based on OpenGL [78] that can be turned off for speed. TORCS offers a large variety of tracks and cars as free assets that we discuss later in this section. It also provides a number of programmed robot cars with different levels of performance that can be used to benchmark the performance of human players and software driving agents. TORCS was built with the goal of developing Artificial Intelligence for vehicular control and has been used extensively by the machine learning community ever since its inception [79, 80, 81, 82, 83].

3.3.2 SCR Server-Client Architecture

The Simulated Car Racing (SCR) Championship [84] is an annual car-racing competition where participants submit controllers for racing in the TORCS environment. It provides a software patch for TORCS known as `scr_server` [85] that sets up a UDP based client-server architecture in which the competing cars can operate independent of one another. The server runs the TORCS simulator. Each client represents a car that runs as a separate process and communicates with the server through a dedicated UDP port. The patch also provides a layer of abstraction over TORCS in which each car has access to an egocentric view of the environment and

not the entire game state. The server polls actions from the clients and updates the game-state every 0.02 seconds of simulated time. The official build of TORCS supports up to 10 SCR clients at a time but with modifications like in [86] the number of clients can be increased arbitrarily.

3.3.3 GymTORCS Environment

GymTORCS [87] is an OpenAI Gym wrapper for SCR cars built for use in Reinforcement Learning experiments. It uses a custom library called *Snake Oil* to create a client for communicating with the TORCS server through the `scr_server` interface. Snake Oil also provides plug-ins for automatic-transmission, traction control and throttle control which can be used to provide different control modes to the driving agent. GymTORCS is popular in the reinforcement learning community for experiments on driving tasks [86, 88, 89, 90]. MADRaS builds on GymTORCS by increasing its stability and ease of use and adding features like multi-agent training and custom traffic cars.

3.4 MADRaS: Multi-Agent DRiving Simulator

Having described TORCS and associated prior works that form the foundation of MADRaS, we now present our contributions. As GymTORCS is pre-dominantly designed for single-agent training, the environment is inherently structured as a single-agent Markov Decision Process. This restricts its use for multi-agent training. MADRaS is GymTORCS restructured as a Markov Game (discussed in Section 2.4.1) with some added functionalities. A Markov Game (MG) is a generalization of Markov Decision Process that is used to capture the interplay of multiple agents in a common environment [38, 35, 91, 36, 92]. An MG is a tuple $\langle S, \mathcal{N}, \mathcal{A}, P, \mathcal{R} \rangle$. $\mathcal{N} = \{\alpha_i\}_{i=1}^{n}$ denotes a set of n agents that simultaneously learn to act in the environment. $\mathcal{A} = \{A_i\}_{i=1}^{n}$ and $\mathcal{R} = \{R_i\}_{i=1}^{n}$ where A_i and R_i denote the action space and reward function for the agent α_i. Figure 3.1 describes the architecture of MADRaS. *MADRaS Environment* consists of a *MADRaS World*

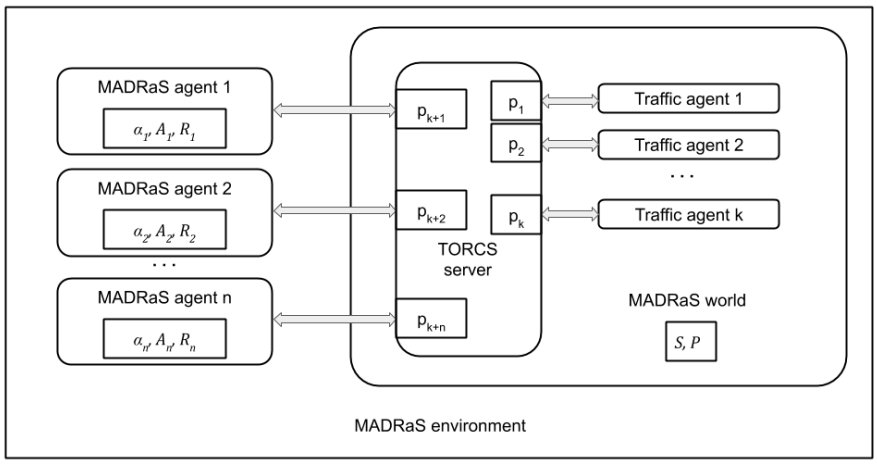


Figure 3.1: Architecture of the MADRaS simulation environment. Each double headed arrow indicates one UDP communication channel between the TORCS server and one of the clients (traffic or MADRaS agents). The server listens to the i^{th} client through a dedicated port denoted by p_i in the figure. MADRaS assigns these ports in order, first to the traffic agents and then to the learning agents. The Markov Game terms are also marked in their respective places of definition in the figure.

Table 3.1: Comparison of Gym TORCS [87] with MADRaS

Feature	**Gym TORCS**	**MADRaS**
scr-server architecture	✓	✓
observation noise	×	✓
stochastic outcomes of actions	×	✓
parallel rollout support	×	✓
multi-agent training	×	✓
inter-vehicular communication	×	✓
custom traffic cars	×	✓
domain randomization	×	✓
centralized configuration	×	✓
modular reward and done functions	×	✓
hierarchical action space	×	✓

and a given number of *MADRaS Agents* ($\{\alpha_i\}_i$). *MADRaS World* consists of a TORCS server and a given number of traffic agents each of which executes an independently configured behavior. The state space (S) and the transition dynamics (P) of the MG are defined by the *MADRaS World*. Each *MADRaS Agent* (α_i) is an SCR Client with a modified Snake Oil interface that has its own action space (A_i) and reward function (R_i) which are independent of the action spaces and reward functions of the other agents. Unlike GymTORCS, *MADRaS Agents* can not reset the TORCS server. This allows for multiple agents to complete their episodes independently. *MADRaS environment* resets its *MADRaS World* and in turn its TORCS server when all the agents have terminated their episodes. MADRaS also provides a number of ways to configure the initial state of the environment for the task at hand. The initial distance from the start line and position with respect to the track edges can be specified individually for both the learning cars as well as the traffic agents. Thus MADRaS harnesses the full potential of the SCR server-client architecture and enables multi-agent training. In the remaining part of this section, we present detailed descriptions of the salient features of MADRaS.

3.4.1 Traffic Agents in MADRaS

MADRaS introduces support for adding non-learning traffic agents in the environment that execute a pre-defined behaviour. These are different from the robot cars that come bundled with TORCS for benchmarking racing agents. MADRaS provides a base class that can be used as template to create traffic cars with interesting behavioral patterns and some sample traffic classes as free assets (see Table 3.2). The base class also comes equipped with methods for avoiding collision and going out-of-track. Each traffic agent runs as a parallel process independent of the learning agent and has an SCR client that talks to the TORCS server through a dedicated port. MADRaS takes care of the configuration and assignment of a requisite number of server ports for connecting all the learning and traffic agents properly at the beginning of each episode. The number and behavior of traffic agents can be varied between episodes.

Table 3.2: Sample traffic agents in MADRaS

Name	Behaviour
`ConstVelTrafficAgent`	Drives at a given speed at a given lane position.
`SinusoidalSpeedAgent`	Varies the speed sinusoidally while driving at a given lane position.
`RandomLaneSwitchAgent`	Agent switches lanes randomly while driving.
`DriveAndParkAgent`	Agent drives to a given distance and track-position and parks itself.
`ParkedAgent`	Agent remains parked at a given distance and track-position throughout.
`RandomStoppingAgent`	Agent halts randomly while driving.

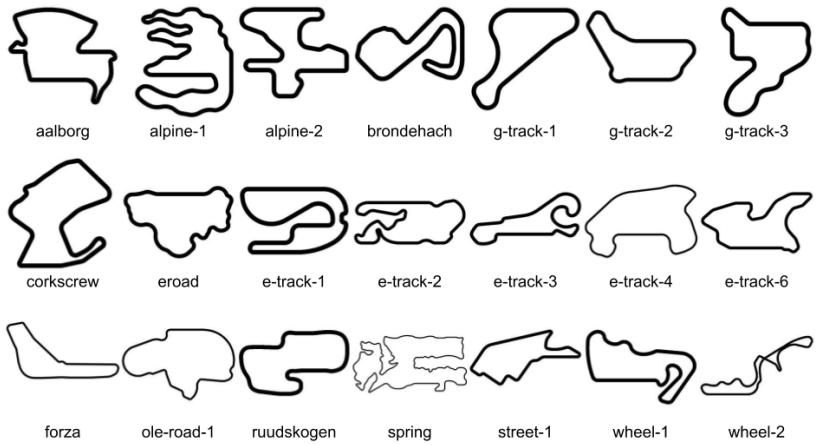


Figure 3.2: Schematic diagrams of road tracks in TORCS

3.4.2 Driving Tracks in MADRaS

One of the major advantages of TORCS as the platform of choice for building MADRaS is the availability of a large number of tracks with different geometric (see Figure 3.2) and surface properties. At the time of writing this thesis, TORCS offers 9 oval, 21 road, and 8 dirt tracks. MADRaS inherits these free assets from the TORCS project. A limitation of GymTORCS is that a track has to be chosen at the beginning of a training experiment and it remains fixed throughout. This often causes the agent to memorize the track resulting in poor generalization.

MADRaS ameliorates this by introducing an option to select a track at the beginning of each episode. Thus the agent can be exposed to multiple tracks during training.

3.4.3 Car Models Available in MADRaS

TORCS provides 42 car models with a wide range of dynamic properties. However, GymTORCS only supports a single default car type named `car1-trb1`. MADRaS is capable of changing cars at the beginning of each training episode. Thus it makes it possible to train an agent to drive cars with drastically different dynamic properties. Also, the learning and traffic agents can be assigned different car types for visual distinction.

3.4.4 Modular Configuration of MADRaS

As Reinforcement Learning (RL) is one of the most powerful and actively researched approaches for robot motion planning, MADRaS has some features tailormade for that purpose. The exercise of tuning an RL algorithm for a given task usually involves tweaking the reward function and episode termination ("done") criteria. It is important to keep accurate track of these parameters across experiments to be able to arrive at the optimal training configuration. GymTORCS has particularly poor configurability as it requires the user to make changes in the Python source code which are difficult to keep track of. On the other hand, the structure of MADRaS focuses on ease of use and encourages custom modifications. All the configuration variables are specified in the `envs/data/madras_config.yml` file. The 'yaml' (or 'yml') format provides a powerful yet convenient way of specifying most data types and basic data structures like lists and dictionaries. A copy of this configuration file can be saved in the training directory for effortless tracking across experiments.

The `madras_config.yml` file has three sections:

1. **Server configuration:** In this section contains the global configurations of the MADRaS environment. Since MADRaS can randomly vary the driving tracks, model of car for the learning agents, and the number of traffic cars between episodes, these terms are specified as lists and ranges. The maximum number of cars in the environment (including learning and traffic agents) can be specified as `max_cars` and the minimum number of traffic cars by `min_traffic_cars`. The number of learning agents (N_l) is specified in the "agent configuration" section.

   $$N_l + \texttt{min\_traffic\_cars} \leq \texttt{max\_cars}$$

   The list of car models to choose for the learning agent can be specified in `learning_car`. The list of tracks to choose for each episode can be specified in `track_names`. If `randomize_env = True` the car model, track and the number of traffic agents is chosen randomly for each episode.

2. **Agent configuration:** The `agents` section, contains the configurations of the learning agents. The `target_speed`, `pid_settings` for the low level controller if `pid_assist` is `True`, configuration of the observation space (according to the modes in `utils/observation_handler.py`), reward function (to be parsed by `utils/reward_handler.py`) and done function (to be parsed by `utils/done_handler.py`) can be specified individually for each agent in this section.

3. **Traffic configuration:** The `traffic` section can be used to specify the details of the traffic agents in the environment. If N_t traffic agents need to be chosen in a given episode, their configurations will be set to the first N_t elements from the list of agents in this section. These configurations are parsed by `traffic/traffic.py`. The `target_speed`, `target_lane_pos`, collision avoidance properties and `pid_settings` of the traffic cars can be specified here. If the traffic agents need to be parked in certain locations (specified in terms of their distance from the start line and track position) of the track before the start of each episode, that can also be specified in this section.

Please refer to Tables 3.3, 3.4 and 3.5 for some of the commonly used configuration variables.

The reward and done functions are usually composed of multiple parts that try to capture events like arrival at the goal state, crashes and damages. Modularity of these definitions in code is essential for fast iteration. MADRaS provides `MadrasReward` and `MadrasDone` base classes as templates for defining the components of the reward and done functions. Specifying a reward or done function in MADRaS is as simple as listing the names of their components in the configuration file. Each MADRaS Agent comes with a `reward_handler` and a `done_handler` that organize the listed components and set up the corresponding functions. This modular architecture makes it easy to define new reward and done functions and plug them in and out of experiments easily.

3.4.5 Observation Space of MADRaS Agents

The Snake Oil client interface provides a parser for the state information returned by the TORCS server. These state variables include odometry, range data, obstacle detection, engine statistics and metadata regarding the position of the ego vehicle relative to the other cars on the road. Such a high-level representation of the world is common in practical self-driving pipelines [93] as it helps in decoupling the perception and planning modules allowing them to be improved independently and also reduces the sample complexity of learning-based planning algorithms [63]. Raw visual input in the form of a stream of images is also available. For a full list of state variables please refer to the Simulated Car Race Championship paper [85]. The observation vector of a MADRaS agent is composed of a selection of these normalized state variables. For modularity and ease of configuration, MADRaS provides an `observation_handler` class that can toggle between different sets of observed variables. The observations can optionally be made noisy to simulate a partially observed driving scenario.

Table 3.3: Some TORCS Server configuration parameters of MADRaS

Parameters	Description	Possible Values
torcs_server_port	For setting the port of communication with the TORCS Server.	$\mathbb{Z}^+$
max_cars	Max number of vehicles to be spawned.	$\mathbb{Z}^+$
min_traffic_cars	Min number of traffic cars to be spawned.	$\mathbb{Z}^+$
track_names	List of tracks on which the simulation will run.	List of track names
track_limits	Restrict the agent to remain within a given range of track_pos values.	$(\mathbb{R}, \mathbb{R})$
distance_to_start	Starting distance of the cars from the start line.	$\mathbb{Z}^+$
torcs_server_config_dir	The location of the TORCS server racing config directory.	Path string
scr_server_config_dir	The location of available cars config directory	Path string
traffic_car	The type of car to be used for traffic	car name
learning_car	List of car models for using as the learning agent.	List of car names
randomize_env	Flag for setting randomization on. (Cycles through the selected cars and tracks in a random fashion)	boolean
add_noise_to_actions	Flag for adding a small Gaussian Noise to the actions before sending to the TORCS server.	boolean
action_noise_std	Specifies the standard deviation of the Gaussian for the noise addition.	$[0, 1]$
noisy_observations	Toggles the TORCS flag for enabling noisy observations.	boolean
visualise	Flag for setting the display on and off.	boolean
no_of_visualisations	To visualize multiple training instances	$\mathbb{Z}^+$
max_steps	Maximum steps that the environment will take before resetting.	$\mathbb{Z}^+$

Table 3.4: Some agent configuration parameters of MADRaS

Parameters	Description	Possible Values
vision	Flag for activating visual input instead of the usual sensor based one.	boolean
throttle	Flag for activating throttle control on and off.	boolean
gear_change	Flag for activating gear control on and off.	boolean
client_max_steps	Maximum steps that the client is available to take.	$\mathbb{Z}^+ \cup \{-1\}$
target_speed	Target speed setting of the agent car.	$\mathbb{Z}^+$
state_dim	Dimension of the Observation Space.	$\mathbb{Z}^+$
normalize_actions	Toggle to turn on action normalization.	boolean
pid_assist	Toggle to turn on T-S control mode.	boolean
pid_settings[accel_pid]	K_p, K_i, K_d for throttle PID.	List of floats
pid_settings[accel_pid]	K_p, K_i, K_d for steering PID.	List of floats
accel_scale	Acceleration Scaling.	$\mathbb{R}^+$
steer_scale	Steering Scaling.	$\mathbb{R}^+$
pid_latency	Number time-steps the control command sticks to the server.	$\mathbb{Z}^+$
observations[mode]	Name of the Observation Class.	string
observations[multi_flag] (multi mode only)	Toggle for turning on communication for the agent-i,	boolean
observations[buff_size]	Specifies the buffer size of action.	$\mathbb{Z}^+$
observation[normalize]	Toggle to tun on observation normalization.	boolean
obs_min	Minimum values for certain observation attributes.	dict
obs_max	Maximum values for certain observation attributes.	dict
rewards[name, scale]	List of the Reward classes and a scaling factor of the rewards.	list of names and dict
dones	Done conditions currently in use.	list of dones

Table 3.5: Common traffic configuration parameters of MADRaS

Parameters	Description	Possible Values
Name	Traffic Agent Type,	string
target_speed	Traffic Agent Speed.	$\mathbb{R}^+$
initial_distance	Initial distance range from start line.	2-Tuple (Float)
initial_trackpos	Initial track-position range.	2-Tuple (Float)
track_len	Length of the Current Track.	$\mathbb{R}^+$
pid_settings[accel_pid]	K_p, K_i, K_d values for acceleration.	List of Floats
pid_settings[steer_pid]	K_p, K_i, K_d values for steering.	List of Floats
accel_scale	Acceleration scaling.	$\mathbb{R}^+$
steer_scale	Steering scaling.	$\mathbb{R}^+$
collision_time_window	Describes the collision region for the traffic agent	$\mathbb{R}^+$

3.4.6 Action Spaces of MADRaS Agents

The Snake Oil library allows GymTORCS agents to control cars via steering, acceleration and brake commands. MADRaS inherits this primitive control mode and also adds a hierarchical track-position – speed control mode. In track-position – speed control mode, a MADRaS agent produces its desired position with respect to the left and right edges of the track and its desired speed. A PID controller takes these non-primitive actions (*desires*) as inputs and calculates a sequence of steering, acceleration and brake commands in feedback mode over a number of time steps denoted by `PID_latency`. The `PID_latency` controls the relative time scales of the higher and lower level action spaces. The track-position – speed action space is inspired by Shalev et al. [15], where the authors note that training an RL agent to generate high-level desires while relegating the low-level implementation of the desires to an analytical controller like PID significantly reduces real world risk and increases the explainability of the agent's behavior. High level actions have also been reported to show better generalizability across vehicular platforms [9]. The following is the expression of a PID controller for control variable u.

$$u(t) = K_p e(t) + K_i \int_0^t e(t')dt' + K_d \frac{de(t)}{dt} \tag{3.1}$$

K_p, K_i and K_d are the constants for the proportional, integral and derivative terms

respectively. In our implementation of the PID controller, the error function (e_{TP}) for track-position PID controller is defined as a function of the track-position (TP) and the angle (θ) that the car's heading makes with the center line. The output of this controller is the steer-angle of the vehicle for the current time-step (t) that would bring the car closer to the desired track-position ($TP_{desired}$).

$$e_{TP}(t) = \theta(t-1) - (TP(t-1) - TP_{desired}) * scale \tag{3.2}$$

The error function for the Speed PID controller (e_V) is a function of the forward velocity (V). The output of the controller is the value of acceleration and braking that would bring the speed closer to the target speed of the vehicle (V_{target}).

$$e_V(t) = (V(t-1) - V_{target}) * scale \tag{3.3}$$

Figure 3.3 and 3.4 show the responses of the PID controller over the the range of speeds used in our experiments with the high level track-position – speed action space. The track used was the "`f-speedway`" oval track and `PID_latency` was set to 5. For testing the controller response we change the input signal (position/velocity) every 100 steps and monitor the output. We use the "`f-speedway`" oval track for this study. For velocity control (Figure 3.3) we change the signal in incremental steps of 10 km/hour from 0 km/hour to 100 km/hour and back to 0 km/hour keeping the track-position input fixed at 0.0, the center of the track. For position control we increment the signal in steps of 0.4 starting from 0.4 (default initial track-position) towards the extreme left (up to 0.8) and then towards the extreme right (up to -0.8). We observe that the controller responds faithfully within the range of speeds and track positions used in our experiments.

All actions are normalized between -1 and 1 for ease of optimization of neural network policies. The outcomes of the agent's actions can optionally be made stochastic. MADRaS implements this stochasticity by adding zero-mean Gaussian noise to actions before sending them to the TORCS server.

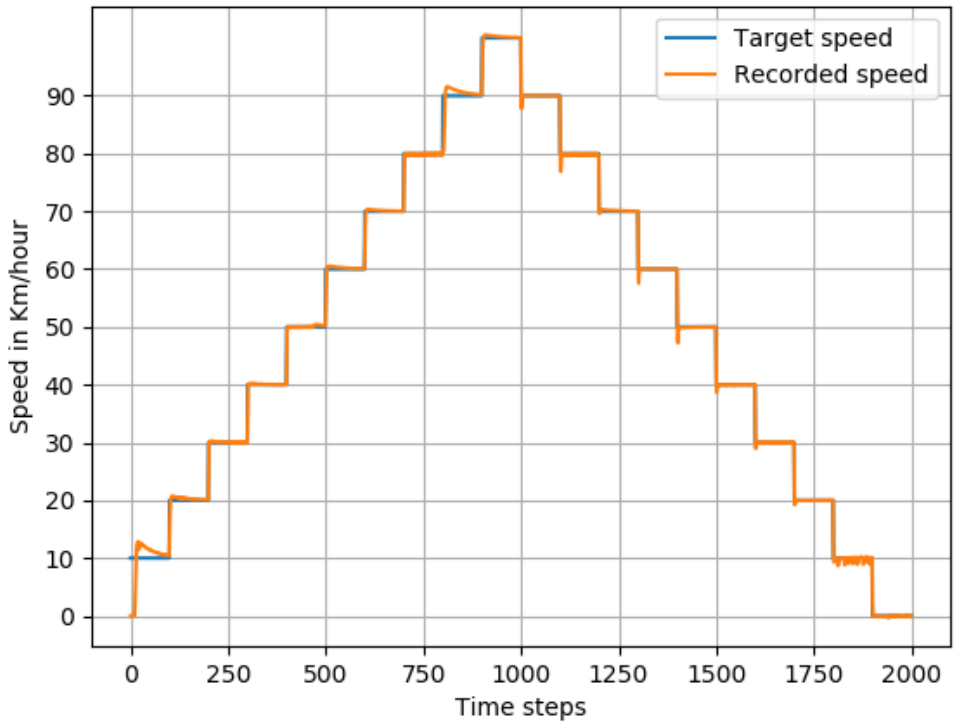


Figure 3.3: This plot demonstrates the velocity control accuracy and stability of the PID controller used in our experiments with the track-position – speed control mode and `PID_latency` was set to 5. The track used was the "`f-speedway`" oval track and the lane-position command was fixed at 0.0 which refers to the center of the track.

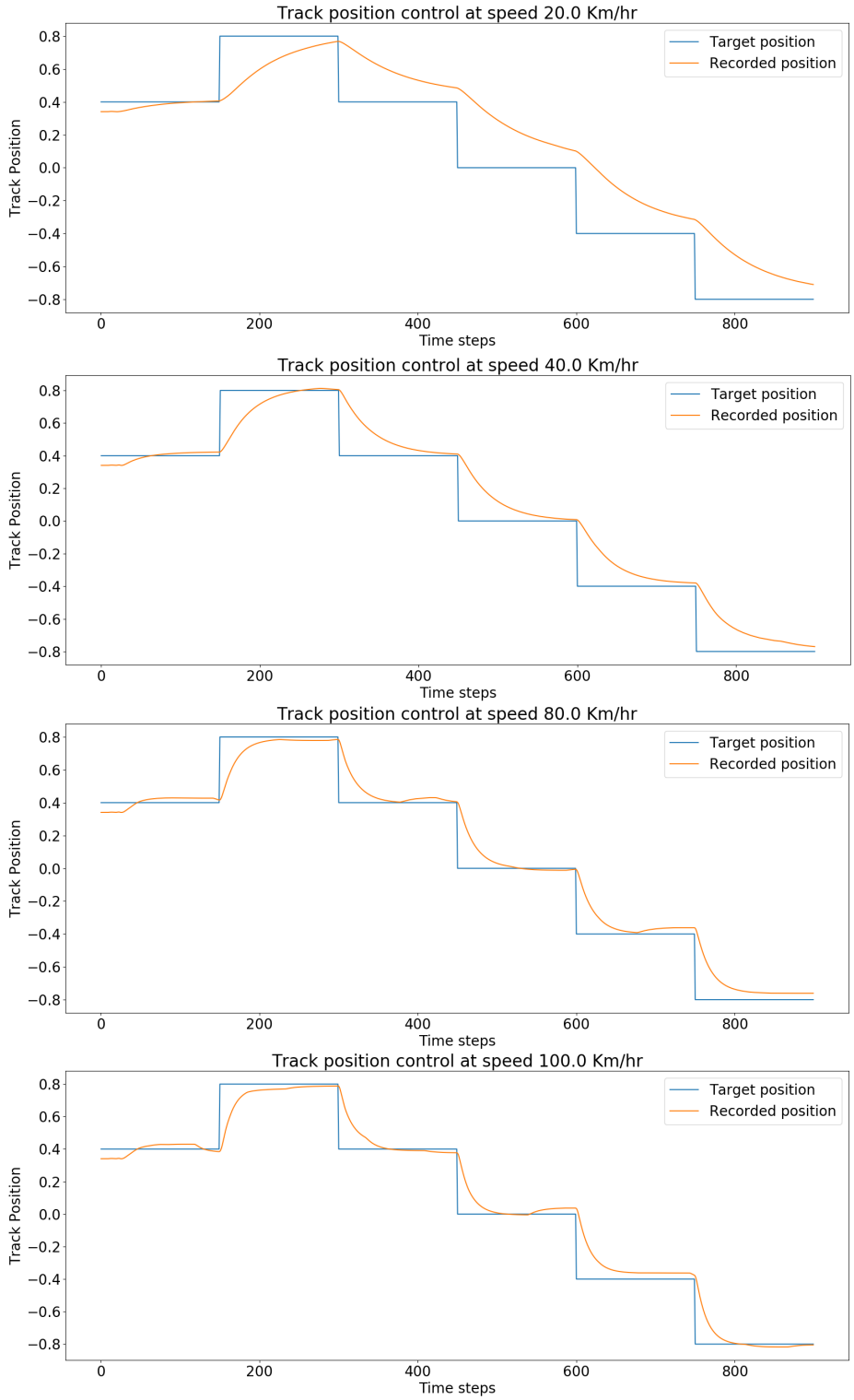


Figure 3.4: Position control accuracy at different velocities of the PID controller used in our experiments with track-position – speed control mode.

3.4.7 Initial State Distribution

The initial state of an episode in MADRaS can be configured in terms of the set of parameters listed below. The `madras_config.yml` file has the `randomize_env` flag that can be enabled to randomly assign values for these parameters at the start of each episode.

- Vehicle Model: The model of the car assigned to the learning agent(s) can be specified using the `learning_car` field. This can also be randomly selected from a categorical distribution over a list of car models when `randomize_env = True`.

- Number of Traffic Cars: The number of traffic cars can be specified using the `min_traffic_cars` field. When `randomize_env = True` the number of traffic cars is assigned randomly between `min_traffic_cars` and (`max_cars` - (number of learning agents)).

- Track Position of Traffic Cars: Some traffic cars can be assigned a certain track position to stick to. For `ParkedAgent`, it can be specified as the `parking_lane_pos` while for `ConstVelTrafficAgent`, `SinusoidalSpeedAgent` and `RandomStoppingAgent` it can be specified using the `target_lane_pos` field. If `randomize_env = True` the track position is sampled randomly from a continuous uniform distribution between specified `high` and `low` limits for these parameters.

- Parking Distance of Traffic Agents from the Start line: The distance from start of `ParkedAgent` traffic agents can be set using the `parking_dist_from_start` parameter. When `randomize_env = True` it is sampled uniformly from a fixed range specified by `high` and `low` values for the same parameter.

Table 3.6: Observed variables in TORCS with their ranges and descriptions [85].

Name	Range (unit)	Description
angle	$[-\pi, +\pi]$ (rad)	Angle between the directions of the car and the track axis
curLapTime	$[0, +\infty)$ (s)	Duration of the current lap
damage	$[0, +\infty)$ (point)	Amount of damage that has happened to the car, the higher the more
distFromStart	$[0, +\infty)$ (m)	Distance of the car from the start line along the track line
distRaced	$[0, +\infty)$ (m)	Total distance covered by the car since the start of the race
focus	$[0, 200]$ (m)	Vector of 5 sensor values that record the distance of the track edges from the car within a range of 200m and an angular span of $[-90^0, 90^0]$.
fuel	$[0, +\infty)$ (l)	Fuel level
gear	$\{-1, 0, 1, \ldots, 6\}$	Current gear: -1 is reverse, 0 is neutral, remaining 6 gears for forward motion
lastLapTime	$[0, +\infty)$ (s)	Time taken to complete the last lap
opponents	$[0, 200]$ (m)	Vector of 36 opponent-sensor values. Each sensor covers a span of 10 degrees within a range of 200 m and returns the distance of the closest opponent in the covered area. The sensors are placed -180^0 to 180^0 at 10^0 intervals and together monitor the complete 360^0 surrounding of the car.
racePos	$\{1, 2, \ldots, N\}$	Position in the race with respect to the other cars.
rpm	$[0, +\infty)$ (rpm)	Number of rotation per minute of the car engine.

speedX	$(-\infty, +\infty)$ (km/h)	Speed of the car along the longitudinal axis of the car.
speedY	$(-\infty, +\infty)$ (km/h)	Speed of the car along the transverse axis of the car.
speedZ	$(-\infty, +\infty)$ (km/h)	Speed of the car along the Z axis of the car.
track	$[0, 200]$ (m)	Vector of 19 range finder sensors that return the distance between the car and the track edge within a horizon of 200m. The sensors are placed at the following angles: -45, -19, -12, -7, -4, -2.5, -1.7, -1, -0.5, 0, 0.5, 1, 1.7, 2.5, 4, 7, 12, 19, 45
trackPos	$(-\infty, +\infty)$	The distance between the car and the track axis, normalized with respect to the track width such that it is 0 when the car is on the axis, -1 when on the right edge and $+1$ when on the left edge.
wheelSpinVel	$[0, +\infty)$	Vector of 4 sensors denoting the rotation speed of the wheels.
z	$[-\infty, +\infty]$	Distance of the car's center of mass from the surface of the track along z-axis

3.4.8 Inter-vehicular Communication in MADRaS

The most salient feature of MADRaS is its support for multi-agent training. The success of multi-agent learning is contingent on the ability of the agents to communicate among themselves and plan actions taking into account the states and actions of the other agents [94]. MADRaS provides a highly flexible framework for inter-vehicular communication through a communication buffer and an agent mapping function. The agent mapping function allows the user to specify a list of variables that the i^{th} agent wants to observe from the j^{th} agent. The communication buffer records these shared variables from the step $t-1$ and makes them a

part of the agents' observation vectors at step t.

3.4.9 Curriculum Design for Driving Agents in MADRaS

MADRaS has been designed with a goal to provide a playground for reinforcement learning agents to learn to drive any car on any track in any kind of traffic within the TORCS environment. In order to construct a driving problem of high variance, MADRaS can present an agent with a different car to drive in a different track with a different number of traffic cars of different behaviors chosen randomly or in a given order in every training episode. MADRaS can also present additional stochasticity by making the outcome of an action probabilistic. Training deep neural network policies in high variance environments poses a highly non-convex problem that is difficult to optimize. Curriculum learning [95] has been shown to be effective in reducing the sample complexity in such problems. Curriculum learning involves training the agent on a sequence of tasks of increasing complexity. MADRaS is designed with curriculum learning in mind. The complexity of the driving task in MADRaS can be systematically increased in well defined steps along the following eight dimensions:

1. Number of learning agents.
2. Number of cars to be presented to the agent to drive.
3. Number of tracks to be presented to the agent to drive.
4. Number of traffic agents.
5. Level of obstructive behavior from the traffic agents.
6. Target speed of the learning agent(s).
7. Degree of stochasticity to action-outcomes.
8. Presence of noise in observations.

In the following section we present a set of experiments to highlight the key features of MADRaS.

3.5 Case Studies

In this section, we present six experimental studies to demonstrate the capabilities of MADRaS to simulate complex driving task-environments for single and multi-agent reinforcement learning. Visual descriptions of the outcomes of some of these studies are available in this video [2].

3.5.1 General Settings

We demonstrate how MADRaS can be used to create a wide variety of driving tasks that can be addressed by RL. We use the Proximal Policy Optimization (PPO) algorithm [96] for RL in all our experiments. PPO is a trust-region based local policy optimization algorithm that has been shown to be very effective in learning policies for continuous control tasks [97]. All the performance statistics presented in this section are estimated over at least 100 episodes. All experiments with the track-position – speed action space have a `PID_latency` of 5 time steps. The reward functions of the RL agents are defined as weighted sums of reward (r) and penalty (p) components with weights w_r and w_p, respectively:

$$agent\_reward = \sum_{r \in rewards} w_r r - \sum_{p \in penalties} w_p p \tag{3.4}$$

Some general purpose reward and penalty components that are used in all the experiments are as follows:

Progress Reward: Progress Reward rewards the agent for making a finite progress at every time step. We calculate progress relative to a target speed. We reward the agent proportional to its speed until it reaches the target speed. If the speed goes beyond the target speed, we do not give the agent any extra reward. This way we prevent the agent from maximizing its cumulative rewards by running fast and crashing rather than finishing the race. Let $d(t)$ be the distance (in meters) covered by the agent in the t^{th} time step and s_{target} denote the target

[2] Accompanying video: https://youtu.be/io5mP0HUytY

speed in meters per step. Progress reward is given by:

$$progress\_reward(t) = \min\left(1, \frac{d(t)}{s_{target}}\right) \tag{3.5}$$

Average Speed Reward: Average Speed Reward rewards the agent for maintaining a high average speed only if it manages to complete a full lap of the track. Suppose the average speed of the agent for a lap is s_{avg}. Average Speed Reward is calculated as:

$$average\_reward = \frac{s_{avg}}{s_{target}} \tag{3.6}$$

The Average Speed Reward is also scaled (but not capped) relative to the target speed s_{target} of the agent.

Angular Acceleration Penalty: This penalty is meant to discourage the agent from making frequent unnecessary side-wise movements while running down a track. We calculate a numerical approximation of angular acceleration from the the past 3 recorded values of the *angle* between the car's direction and the direction of the track axis. We scale the penalty with respect to a reference $\alpha_{reference}$. Let a_{t-2}, a_{t-1}, a_t be three consecutive angles of the agent. We calculate Angular Acceleration Penalty as:

$$angular\_accleration\_penalty(t) = \frac{|a_t + a_{t-2} - 2a_{t-1}|}{\alpha_{reference}} \tag{3.7}$$

We set $\alpha_{reference}$ to 2.0 in all our experiments.

Turn Backward Penalty: A fixed penalty of -1 if the car turns backwards.

Collision Penalty: A fixed penalty of -1 if the car collides with obstacles or other cars and incurs a damage.

Apart from these we also use task specific rewards that we define separately in each experiment.

Table 3.7: Parameters of the PID controller used in our experiments

	K_p	K_i	K_d
acceleration PID	10.5	0.05	2.8
steering PID	5.1	0.001	0.000001

We terminate an episode if one of the following events happen:

- car turns backwards,
- car goes out of track,
- car collides with an obstacle,
- agent fails to complete its task within the maximum allowable duration of an episode,
- agent successfully completes the task at hand.

Unless otherwise stated, we set the learning rate to 5×10^{-5}. The policy and value functions are modelled using fully connected neural networks with 2 hidden layers and 256 *tanh*–units in each layer. We use the PPO implementation of RLLib [66] for all our experiments for its stability and support for multi-agent training. The PID parameters used for track-position – speed control are given in Table 3.7. Although ideally these parameters must be tuned for each car and for each speed range, we use the same set of parameters (originally tuned for medium-low speeds of `car1-trb1`) everywhere to check if it is possible to teach RL agents to be robust to imperfections in the low level controller.

3.5.2 Case Study 1: Generalization Across Tracks with Higher Level Actions

In our first case study, we compare two RL agents, one having the high-level track-position – speed (T-S) control mode and the other having the low-level steer – acceleration – brake (S-A-B) control mode, on their ability to generalize across

Table 3.8: RL training criteria for Case Studies 1-3. Please refer to [85] for details on the observed variables.

Reward function	**Reward Function Component**	**Weightage**
	Progress Reward	1.0
	Average Speed Reward	1.0
	Collision Penalty	10.0
	Turn Backward Penalty	10.0
	Angular Acceleration Penalty	5.0
Observed variables	angle, track, trackPos, speedX, speedY, speedZ	
Done criteria	One Lap Completed, Time Out, Collision, Turn Backward, Out of Track	

multiple driving tracks in MADRaS. We train the agents to drive `car1-stock1` in the `Alpine-1` track and evaluate them on the other road tracks. Table 3.8 lists the observed variables and the components of the reward and done functions. We set the maximum duration of an episode at 15000 time steps and the target speed at 100 km/hour. We evaluate the agents in terms of the average fraction of lap covered in an episode, average speed and successful lap completion rate.

Table 3.9 presents the results of this experiment. We see that the agent with high-level track-position – speed (T-S) control generalizes significantly better than the one with low-level steer – acceleration – brake (S-A-B) control as given by higher average scores. The low-level S-A-B control mode gives the agent tighter control of the car that can be exploited to perform maneuvers very specific to the training track in order to navigate the twists and turns while maintaining a high average speed (see the accompanying video). This results in the agent overfitting to the training track and it fails to make any significant progress in some of the test tracks. Implementing a desired track-position and speed may require different sequences of low-level actions in different tracks. Relegating the low-level control to a PID controller gives the T-S agent better generalization to track-geometries than the S-A-B agent.

Table 3.9: Generalization of an agent trained on `Alpine-1` to other road tracks (Experiment 1). S-A-B (Steering - Acceleration - Brake) and T-S (Track position - Speed) denote the control mode used. (Case Study 1).

		Average fraction of lap covered		**Average Speed**		**Lap completion rate**	
		S-A-B	**T-S**	**S-A-B**	**T-S**	**S-A-B**	**T-S**
	alpine-1	0.75	0.73	91.89	83.32	0.68	0.58
Test Tracks	aalborg	0.001	0.11	0.10	59.39	0.0	0.0
	alpine-2	0.38	0.31	89.95	72.64	0.04	0.0
	brondehach	0.001	0.72	0.1	81.01	0.0	0.3
	g-track-1	0.001	0.98	0.06	79.42	0.0	0.91
	g-track-2	0.002	0.97	0.11	75.99	0.0	0.95
	g-track-3	0.001	0.84	0.09	79.90	0.0	0.44
	corkscrew	0.0008	0.64	0.06	81.39	0.0	0.0
	e-road	0.001	0.94	0.11	85.63	0.0	0.88
	e-track-2	0.07	0.39	8.38	75.21	0.0	0.0
	e-track-3	0.31	0.68	25.88	77.96	0.03	0.57
	e-track-4	0.0005	0.95	0.08	78.41	0.0	0.85
	e-track-6	0.0009	0.83	0.09	80.65	0.0	0.58
	forza	0.001	0.79	0.08	71.63	0.0	0.70
	ole-road-1	0.29	0.40	101.22	78.06	0.0	0.11
	ruudskogen	0.97	0.97	100.87	81.15	0.95	0.93
	street-1	0.03	0.87	1.76	74.67	0.0	0.67
	wheel-1	0.0009	0.95	0.09	78.08	0.0	0.76
	wheel-2	0.36	0.81	81.69	81.51	0.0	0.64
	spring	0.14	0.29	104.76	82.55	0.0	0.0
Average Scores (Test)		**0.14**	**0.71**	**27.12**	**77.64**	**0.04**	**0.49**

3.5.3 Case Study 2: Generalization Across Vehicular Dynamics Through Random Car Selection

In our second case study, we leverage the ability of MADRaS to change the agent's car at the beginning of each episode to train a driving policy that generalizes to multiple cars with significantly different vehicular dynamics. Table 3.10 gives some physical parameters of the cars used in this experiment that characterize their handling and dynamics. Heavier cars with a low centre of gravity e.g. `car1-stock1`, `car3-trb1` and `car1-stock2` are more stable and handle better with less body-roll around tight corners. The variation of torque with the RPM (Rotations Per Minute) of a car's engine plays a crucial role in deciding its dynamics. The torque produced by an engine decides how fast the car can accelerate. Torque is usually a strong function of engine RPM. While running at a given RPM, a car can accelerate faster if its engine can produce higher torque at that RPM. Figure 3.5 gives the torque-RPM curves for the cars used in this experiment. These torque-RPM plots are generated from the engine-characteristics that were provided by the authors of TORCS in the form of a set of (RPM, torque) tuples in the "`Engine`" section of the `data/cars/models/<car_name>/<car_name>.xml` files in the TORCS code-base [31]. The cars fall in two broad categories in terms of the overall shape of this curve. Cars with a "$\cup$"-shaped curve e.g. `buggy`, `baja-bug` and `155-DTM` have high torque at low (< 1000) and high (> 10000) RPM and significantly lower values in the middle. The other category of cars e.g. `car1-stock1`, `car3-trb1` and `car1-stock2` have a "hat" ($\cap$)-shaped curve with low torque at low and high RPM and high values in the middle. When the agent needs high torque to accelerate from a standstill, speed up or climb uphill, it needs to take the engine RPM to the high-torque zone with a suitable sequence of accelerator inputs. The high-torque zones of the aforementioned categories of cars are roughly opposite to one another. This makes it challenging for a driving agent to generalize to both kinds of cars.

We choose the `Alpine-1` track for this experiment. The `Alpine-1` track is one of the hardest road tracks of MADRaS with sharp left and right turns and a few stretches of slippery road. We set the maximum duration of an episode to 20000

Table 3.10: Some physical properties of the cars used in Case Study 2 that play an important role in determining their vehicular dynamics. "RWD" and "4WD" stand for "Rear Wheel Drive" and "Four Wheel Drive", respectively.

Car Name	Drive Type	Mass (Kg)	Height of CG (m)
car1-stock1	RWD	1550.0	0.3
car1-stock2	RWD	1550.0	0.3
155-DTM	4WD	1100.0	0.2
car3-trb1	RWD	1150.0	0.2
kc-2000gt	RWD	1200.0	0.25
buggy	RWD	650.0	0.45
baja-bug	RWD	600.0	0.35

Table 3.11: Generalization of PPO policies across vehicles with different dynamics (Case Study 2). "random" refers to the setting in which the agent is presented with both `car1-stock1` and `buggy`, each with a probability of 0.5 during training.

		Avg. Fraction of Track Covered			Avg. Speed (km/h)		
	Training Car	car1-stock1	buggy	random	car1-stock1	buggy	random
Test Cars	**155-DTM**	0.37	0.01	0.37	104.22	2.14	99.78
	car3-trb1	0.002	0.003	0.62	0.12	0.25	58.95
	kc-2000gt	0.77	0.003	0.30	80.44	0.24	22.02
	car1-stock2	0.001	0.003	0.54	0.09	0.23	50.23
	baja-bug	0.35	0.04	0.55	59.45	38.40	54.91
Average Scores		0.30	0.01	**0.48**	48.86	8.25	**57.18**

time steps and the target speed to 100 km/hour. We evaluate the agent in terms of average fraction of the lap covered per episode and average speed.

First, we train two PPO agents to drive `car1-stock1` and `buggy` using the S-A-B control mode. We evaluate them on five test cars of different dynamic properties. Table 3.11 presents the results. We see that an agent trained on a car of one torque-RPM category has difficulty generalizing to the cars of the other category. In our next step, with a view to aiding in generalization through domain randomization, we leverage the ability of MADRaS to randomly switch cars between episodes and present `car1-stock1` and `buggy` to the same agent with equal probability. We observe that this training strategy brings significant improvement both in terms of average fraction of lap covered in an episode and average speed.

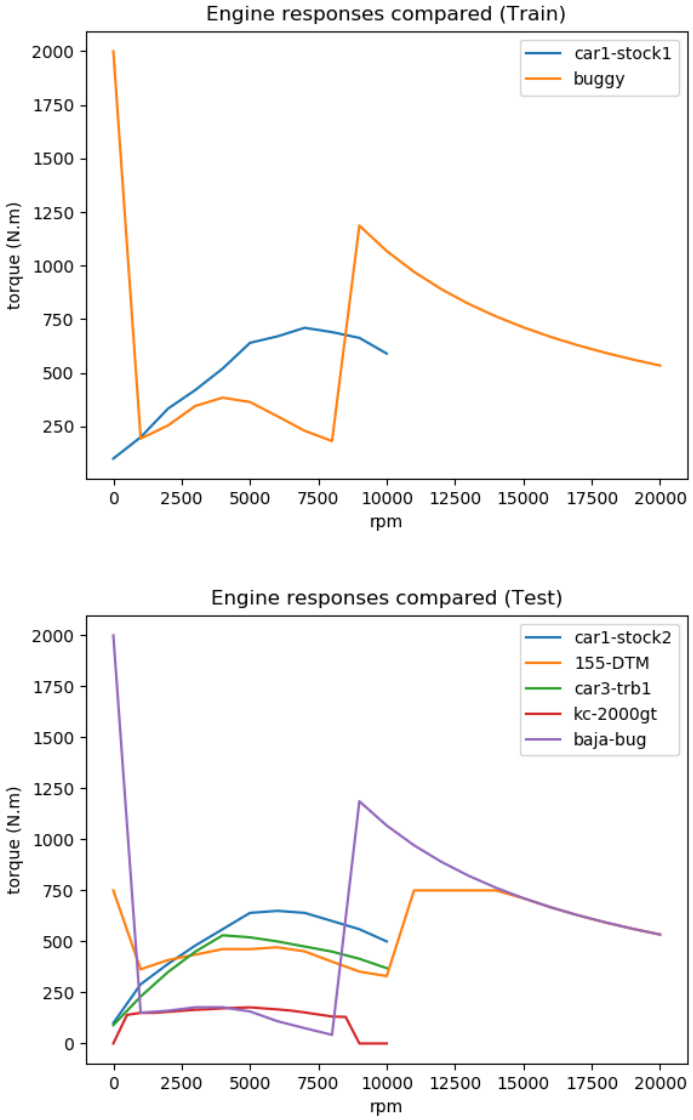


Figure 3.5: Variation of torque with engine RPM of cars in Case Study 2. (a) Torque-vs-RPM of the cars that we present our agent to drive during training with equal probability. (b) Torque-vs-RPM of the cars that we test our agent on.

3.5.4 Case Study 3: Curriculum Learning for Driving in `Spring` Track

In our third case study, we present an experiment to demonstrate how the ability of MADRaS to control the complexity of a driving task in well defined steps can be used to design curricula for an RL agent to accomplish complex tasks in a sample efficient way. We attempt to train a PPO agent to drive `car1-stock1` on `Spring` track using the primitive S-A-B action space. With a length of 22.1 km, `Spring` is the longest track in TORCS. It has the largest number of turns with different grades of sharpness, both in the left and right directions. It also has ramps and declines. The surface texture varies from place to place. These make it the toughest road track to drive in TORCS. We set the target speed to 100 Km/hr and maximum episode length to 40000 steps. Figure 3.6 and Table 3.12 show the results of this study. We see that training from scratch on `Spring` fails to complete one lap of the track even after 2500 iterations. When we use a curriculum of first training on `Alpine-1` or `Corkscrew` tracks followed by fine-tuning on `Spring` the agent learns to complete the entire lap with high success rates and average speed. In our curriculum learning experiments, we pick the policy that gives the highest mean trajectory reward in the first phase of training (obtained after 701 iterations in `Alpine-1` and 561 iterations in `Corkscrew`) and use it to initialize the policy in the second phase. The total number of training iterations and the total number of training samples for the curriculum learning strategies (considering both the pre-training and fine-tuning stages) are kept equal to that of training from scratch for fairness of comparison. For fine-tuning, we choose a learning rate of 1×10^{-6} for the `Alpine-1` policy and 5×10^{-7} for the `Corkscrew` policy. We evaluate the agents in terms of the average fraction of lap covered in an episode, average speed and successful lap completion rate.

3.5.5 Case Study 4: Learning Under Partial Observability and Stochastic Outcomes of Actions

In this case study, we compare the performances of PPO agents trained to drive `car1-stock1` around the `Corkscrew` track with and without observation noise

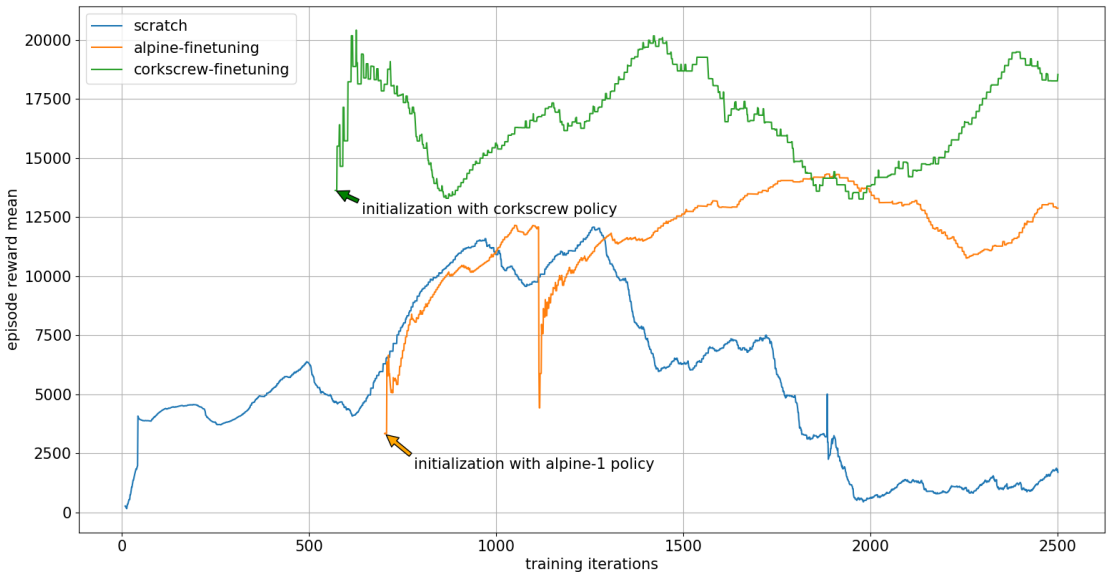


Figure 3.6: Variation of episode reward over iterations of PPO for learning from scratch on `Spring` compared with first learning on simpler tracks – `Alpine-1` and `Corkscrew` – and then fine-tuning on `Spring` (Case Study 3).

Table 3.12: Curriculum learning results for driving in `Spring` track (Case Study 3).

	Fraction of lap covered	**Average Speed (km/hr)**	**Lap completion rate (%)**
Training from scratch	0.18	101.9	0.0
Pre-training in Alpine-1	0.57	103.5	27.0
Pre-training in Corkscrew	0.54	100.6	45.8

under different levels of stochasticity of the outcome of actions. The training was performed with the primitive S-A-B action space. Observed variables, episode termination criteria and evaluation metrics are the same as in Case Study 1. The reward function is the same as in the Experiments 1-3 (see Table 3.8) with the weightage for angular acceleration penalty increased to 8. As described in Section 3.4.6, stochastic outcomes of actions is implemented by adding zero mean Gaussian noise to the actions. Figure 3.7 shows the learning curves. All these agents are tested in the same track `Corkscrew` in the presence of both observation noise and 0.5 standard deviation action noise. Table 3.13 compares the performance

statistics. We observe that the agents trained in the presence of both observation and action noise perform better than the others. This demonstrates the ability of MADRaS to serve as a platform for evaluating the resilience of learning agents to observation noise and environmental stochasticity.

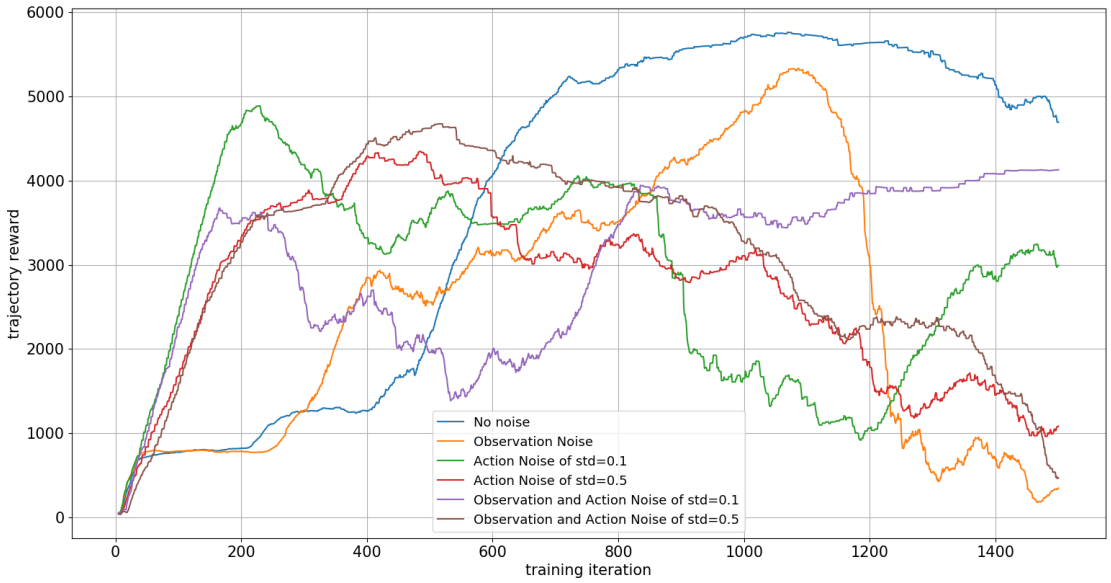


Figure 3.7: Learning to drive with under partial observability and stochastic outcomes of actions in `Corkscrew` track (Case Study 4).

3.5.6 Case Study 5: Learning to Drive in Traffic

In this case study, we use the ability of MADRaS to generate custom traffic to train an agent to navigate through a narrow road without colliding with any traffic car – moving or parked. Figure 3.8 shows a schematic diagram of the training environment. We choose the `Aalborg` track for this study since it is one of the narrowest tracks of TORCS and further reduce its width to half resulting in an effective track width of 5m.

The traffic agents used in this study are `DriveAndParkAgents` (see Table 3.2). MADRaS positions the traffic cars ahead of the learning car at the start of the race. When an episode begins, the `DriveAndParkAgents` start driving at their given target speeds (50) km/hr towards their given parking locations (specified

Table 3.13: Learning to drive in the `corkscrew` track with and without observation noise and different levels of stochasticity in the outcome of actions and evaluation with observation noise and 0.5 std action noise (Case Study 4).

	Avg. Fraction of Lap Covered	**Avg. Speed (km/hr)**
No noise	0.38	52.54
Observation noise	0.19	30.78
Stochastic actions (noise std 0.1)	0.12	29.99
Stochastic actions (noise std 0.5)	0.64	48.67
Observation noise and Stochastic actions (noise std 0.1)	0.63	48.85
Observation noise and Stochastic actions (noise std 0.5)	0.68	46.91

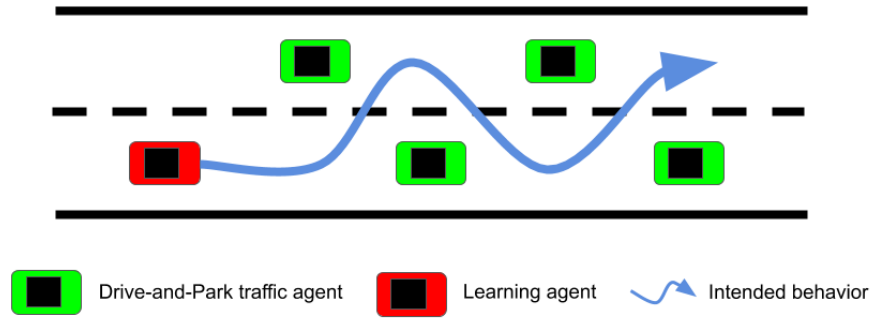


Figure 3.8: Schematic diagram of the environment design for Case Study 5. The task of the learning agent is to overtake all the traffic cars without colliding with any of them or going off track.

Table 3.14: RL training criteria for Case Study 5. Please refer to [85] for details on the observed variables.

	Reward Function Component	**Weightage**
Reward function	Progress Reward	1.0
	Average Speed Reward	1.0
	Collision Penalty	10.0
	Turn Backward Penalty	10.0
	Angular Acceleration Penalty	1.0
	Overtake Reward	5.0
	Rank 1 Reward	100.0
Observed variables	angle, track, trackPos, speedX, speedY, speedZ, opponents	
Done criteria	Rank 1, Time Out, Collision, Turn Backward, Out of Track	

in terms of distance from the start of the race and track position) using PID controllers. This way, the learning agent sees moving cars in the beginning and parked cars towards the end of each episode. This forces it to learn to avoid collision with both static and moving obstacles. We set the parking locations of the traffic cars on alternate sides of the road so that the the agent must learn to turn both left and right to overtake all the traffic cars. We maintain a gap of at least 10m between consecutive parking locations along the length of the road to make sure that the learning car has enough space to maneuver between the traffic cars. To create variance in the environment, we randomly vary each parking location within an area of 5m along the track length and 0.25m along the track width. We also switch the number of traffic cars between 4 and 5 with equal probability. Changing the number of traffic cars also makes sure that the learning agent gets initialized in the left and right halves of the track with equal probability. We use the T-S action space and set the target speed of the learning agent to 50 km/h. Table 3.14 gives the training criteria for this experiment.

The agent gets an *Overtake Reward* every time it overtakes a traffic agent and *Rank 1 Reward* at the end of the episode if it manages to overtake all the traffic agents. The agent is evaluated in terms of the fraction of times it overtakes all

Table 3.15: Results of a single PPO agent learning to drive in traffic by RL. The agent was trained to drive in the presence of 4 or 5 traffic cars with equal probability (Case Study 5).

	Number of traffic agents						
	3	**4**	**5**	**6**	**7**	**8**	**9**
Successful task completion rate	99.5%	98.1%	95.5%	96%	95.5%	95.7%	92.8%

Table 3.16: Dimensions of cars used in Case Study 6.

	Car Model	**Length (m)**	**Width (m)**
Traffic Car	`car1-trb1`	4.52	1.94
PPO Agent-1	`car3-trb1`	4.55	1.95
PPO Agent-2	`car5-trb1`	4.67	1.94

the traffic cars successfully.

Table 3.15 presents the results of this case study. We observe that the agent learns to generalize to both fewer and more traffic agents than it encountered during training and navigate its way through them collision-free with a high success rate.

3.5.7 Case Study 6: Avoiding Traffic Obstruction Through Multi-agent Cooperation

One of the biggest aspirations of autonomous driving is the avoidance of traffic congestion through cooperation. In this case study, we utilize the multi-agent training ability of MADRaS and its framework for inter-vehicular communication to solve a simplified version of this task by multi-agent reinforcement learning.

Table 3.17: Curriculum for multi-agent RL in Case Study 6.

Iterations of training	**Parking Distance (m)**	**Gap Width (m)**
1–240	30–40	2.76–4.06
240–300	30–35	2.76–3.46

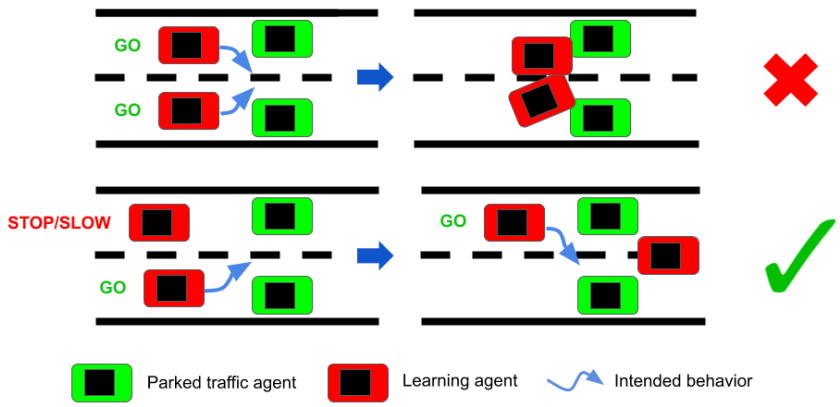


Figure 3.9: Schematic diagram of the multi-agent task of Case Study 6. The task for the two learning agents is to coordinate with each other and pass through the gap between the parked traffic cars without making any collision. The top row shows an example of undesirable behavior in which both the agents attempt to pass through the bottleneck at the same time and result in a collision. The bottom row gives a feasible solution to the problem in which one of the agents stops or slows down to wait for the other agent to pass through the gap. Only after the gap is clear does it attempt to pass through – thus avoiding collision with any of the other cars.

Table 3.18: RL training criteria for Case Study 6. *peerActions* refers to the actions of the other learning agent from the previous time step. Please refer to [85] for details on the other observed variables.

	Reward Function Component	**Weightage**
Reward function	Progress Reward	1.0
	Average Speed Reward	1.0
	Collision Penalty	10.0
	Turn Backward Penalty	10.0
	Angular Acceleration Penalty	5.0
Observed variables	angle, track, trackPos, opponents, speedX, speedY, speedZ, peerActions	
Done criteria	Race Over, Time Out, Collision, Turn Backward, Out of Track	

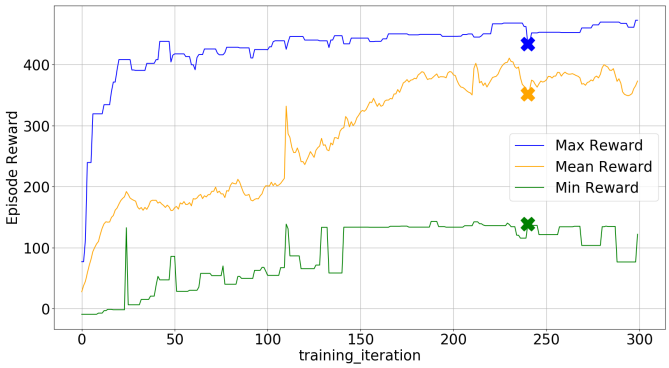
The training environment consists of two PPO agents and two traffic agents on the `Corkscrew` track. The PPO agents communicate their actions to each other at every step. We park the traffic agents next to each other with a small gap in between that is sufficient only for one car to pass through. The task of the PPO agents is to pass through the gap one by one without colliding with each other or with any traffic agent (see Figure 3.9). Thus the agents must learn a collaborative strategy in which the agent trying to pass through the gap first should be given enough time to pass through completely by the other agent before it makes its own attempt.

Table 3.16 gives the cars assigned to the learning and traffic agents and their dimensions. Both the PPO agents have T-S action space. Table 3.17 describes the curriculum used for the training. We randomly vary the parking distance of each traffic car and the gap between them at the start of each episode for improved generalization. Table 3.18 gives the details of the observed variables, reward functions and done criteria. The agents must learn the following distinct skills to be able to accomplish this task.

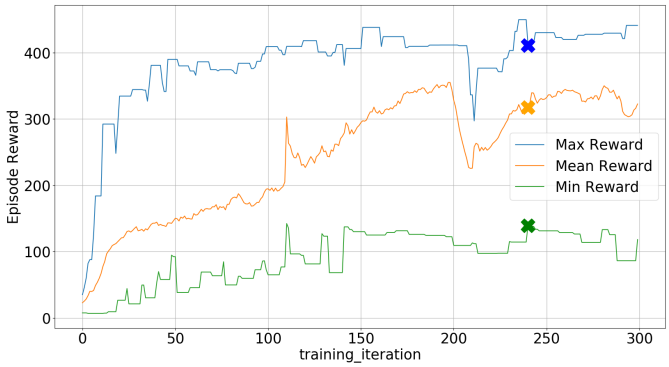
- Running forward without going off track.
- Not colliding with each other.

- Not colliding with any of the parked cars.
- Learning to collaborate and pass through the bottleneck one by one.

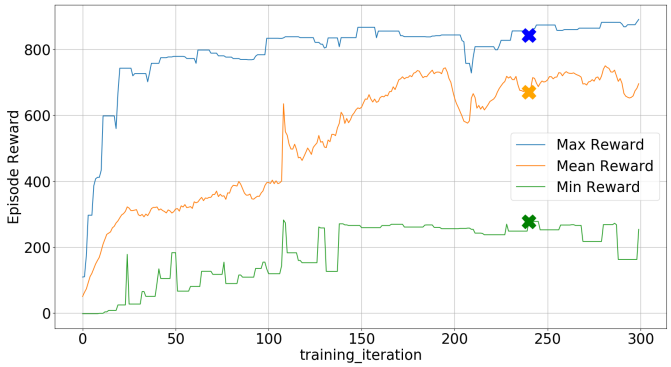
We jointly evaluate the agents in terms of the rate of successful passage of both the agents through the traffic bottleneck. Figure 3.10 shows the individual and joint learning curves respectively during training. The final evaluation is done over 100 episodes of stochastically parked agents and the PPO agents demonstrate a joint task completion rate of 83.3%.



(a) Agent-1 Training Curves



(b) Agent-2 Training Curves



(c) Joint learning Curves

Figure 3.10: Learning curves for multi-agent training in Case Study 6. The cross symbol denotes transition point in the agent's curriculum where the first task ends and the second task begins.

3.6 Summary

In this chapter we present MADRaS, an open-source Multi-Agent Driving Simulator for autonomous driving. MADRaS builds on TORCS, a popular car racing platform, and adds a suite of features like hierarchical control modes, domain randomization, custom traffic, partial observability, stochastic outcomes of actions and support for multi-agent training. We present six case studies in which we train reinforcement learning agents to accomplish challenging tasks like generalizing across a wide range of track geometries and vehicular dynamics, driving under stochasticity and partial observability, navigating through static and moving obstacles and negotiating with other agents to pass through a traffic bottleneck. These studies demonstrate the viability of MADRaS to simulate rich highway and track driving scenarios of high variance and complexity that are valuable for autonomous driving research.

CHAPTER 4

RAIL: Risk Averse Imitation Learning

4.1 Introduction

In Reinforcement learning (RL) [13], the goal is communicated to the agent through a scalar reward function. The negative of the reward function is often referred to as *cost* function. The agent learns a policy that minimizes the expected cost. In classical RL, the cost function is handcrafted based on heuristic assumptions about the goal and the environment. This is challenging in most real-world applications and also prone to subjective bias [25]. Imitation learning or Learning from Demonstration (LfD) [26, 27, 47, 48, 28, 49] addresses this challenge by providing methods of learning policies through imitation of an expert's behavior without the need of a handcrafted cost function. In this chapter, we study the reliability of imitation learning algorithms that learn from a fixed set of expert-demonstrated trajectories.

We focus our investigation on Generative Adversarial Imitation Learning (GAIL) [29]. GAIL is a state-of-the-art algorithm for learning policies from a fixed set of expert-trajectories. We evaluate in terms of the expert's cost function and

observe that the distribution of trajectory-costs is often more heavy-tailed for GAIL agents than the expert at a number of benchmark continuous-control tasks. This translates to the fact that, high-cost trajectories, corresponding to tail-end events of catastrophic failure, are more likely to be encountered by GAIL agents than the expert. This makes the reliability of GAIL agents questionable in risk-sensitive applications like autonomous driving. In this chapter, we aim to make GAIL agents aware of possibly catastrophic tail-end events by minimizing tail-risk within the GAIL framework. We quantify tail-risk by the Conditional-Value-at-Risk ($CVaR$) [32] and develop the Risk-Averse Imitation Learning (RAIL) algorithm. We demonstrate that the policies learned with RAIL show lower tail-risk than GAIL at five benchmark continuous control tasks of OpenAI Gym [33].

4.2 Background

In this section we describe the mathematical background of Generative Adversarial Imitation Learning (GAIL) [29] and Conditional Value at Risk ($CVaR$) [32] that form the basis of the theory developed in this chapter. Please refer to Section 2.5 for an introduction to Imitation Learning.

4.2.1 Generative Adversarial Imitation Learning (GAIL)

Generative Adversarial Imitation Learning (GAIL) [29] is an Apprenticeship Learning algorithm. Apprenticeship learning algorithms [28] first learn a model of the expert's reward function using Inverse Reinforcement Learning (IRL) [58, 49, 48]. This model of the expert's reward is then maximized using Reinforcement Learning (RL) to obtain the output policy. Mathematically, this can be described as:

$$RL \circ IRL(\pi_E) = \underset{\pi \in \Pi}{argmin} \; \underset{c \in \mathcal{C}}{max} \; \mathbb{E}_{\pi}[c(s,a)] - \mathbb{E}_{\pi_E}[c(s,a)] - H(\pi) \tag{4.1}$$

where, π_E denotes the expert-policy and $c(\cdot,\cdot)$ denotes the cost function. Π and $\mathcal{C}$ denote the hypothesis classes for policy and cost functions. $H(\pi)$ denotes entropy of policy π. The term $-H(\pi)$ provides causal-entropy regularization [59, 60] which

helps in making the policy optimization algorithm unbiased to factors other than the expected reward. Ho and Ermon [29] showed that the entropy-regularized Apprenticeship Learning problem formulation in equation 4.1 can be represented as in the following form:

$$RL \circ IRL_{\psi}(\pi_E) = \underset{\pi \in \Pi}{argmin} - H(\pi) + \psi^*(\rho_\pi - \rho_{\pi_E}) \tag{4.2}$$

where $\psi : \mathcal{S} \times \mathcal{A} \to \mathbb{R} \cup \{\infty\}$ denotes a closed, proper convex regularizer on the cost function and $\rho_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the occupancy measure for policy π defined in equation 4.3.

$$\rho_\pi(s, a) = \pi(a|s) \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\pi) \tag{4.3}$$

The authors also proposed a regularizer, $\psi_{GA}(c)$, that scales well in large environments as well as matches the expert's occupancy measure closely:

$$\psi_{GA}(c) \triangleq \begin{cases} \mathbb{E}_{\pi_E}\left[g(c(s,a))\right], & \text{if } c < 0 \\ +\infty, & \text{otherwise} \end{cases} \tag{4.4}$$

where $g(\cdot)$ is defined as follows:

$$g(x) = \begin{cases} -x - log(1 - e^x), & \text{if } x < 0 \\ +\infty, & \text{otherwise} \end{cases} \tag{4.5}$$

For this choice of $\psi = \psi_{GA}$, $\psi^*(\rho_\pi - \rho_{\pi_E})$ is given by:

$$\psi^*(\rho_\pi - \rho_{\pi_E}) = \max_{\mathcal{D} \in (0,1)^{\mathcal{S} \times \mathcal{A}}} \mathbb{E}_\pi[log(\mathcal{D}(s,a))] + \mathbb{E}_{\pi_E}[log(1 - \mathcal{D}(s,a))] \tag{4.6}$$

The maximum in equation 4.6 ranges over discriminative binary classifiers of the form $\mathcal{D} : \mathcal{S} \times \mathcal{A} \to (0, 1)$. Substituting equation 4.6 in 4.2, we get:

$$\underset{\pi \in \Pi}{argmin} \max_{\mathcal{D} \in (0,1)^{\mathcal{S} \times \mathcal{A}}} \mathbb{E}_\pi[log(\mathcal{D}(s,a))] + \mathbb{E}_{\pi_E}[log(1 - \mathcal{D}(s,a))] - H(\pi) \tag{4.7}$$

Equation 4.7 is the objective of the Generative Adversarial Imitation Learning (GAIL) algorithm. Here, the agent's policy, π, acts as a *generator* of state-action pairs. $\mathcal{D}$ is a discriminative binary classifier of the form $\mathcal{D} : \mathcal{S} \times \mathcal{A} \rightarrow (0, 1)$, known as *discriminator*, which given a state-action pair (s, a), predicts the likelihood of it being generated by the generator.

The optimization of this objective function starts a two-player adversarial game between the generator and the discriminator. Figure 4.1 describes the set-up. The generator tries to generate state-action pairs that closely match the expert. The discriminator tries to correctly classify incoming state-action pairs as whether they are from the expert or the agent. The generator tries to improve its predictions and fool the discriminator into thinking that its actions came from the expert. The discriminator, on the contrary, keeps improving its classification accuracy at distinguishing real expert actions from the fake ones being produced by the generator. As both the players get better at their performances, the generator's actions gradually become indistinguishable from those of the expert in any given state. Eventually, at convergence, the discriminator's classification accuracy drops to that of a coin-toss (0.5) and the solution is said to have arrived at its Nash equilibrium.

The generator and the discriminator are assigned parameterized models π_θ and $\mathcal{D}_w$ respectively. The training algorithm alternates between a *gradient ascent* step with respect to the discriminator parameters, w, and a *policy-gradient descent* step with respect to the generator parameters, θ. The authors demonstrate that GAIL agents with neural network function approximators are capable of imitating complex behaviors in large, high-dimensional environments.

4.2.2 Conditional-Value-at-Risk ($CVaR$)

Conditional Value at Risk ($CVaR$) [32] is a measure of tail-risk popularly used in the portfolio-risk optimization literature. *Tail risk* is a form of portfolio risk that arises when the possibility that an investment moving more than three standard deviations away from the mean is greater than what is shown by a normal

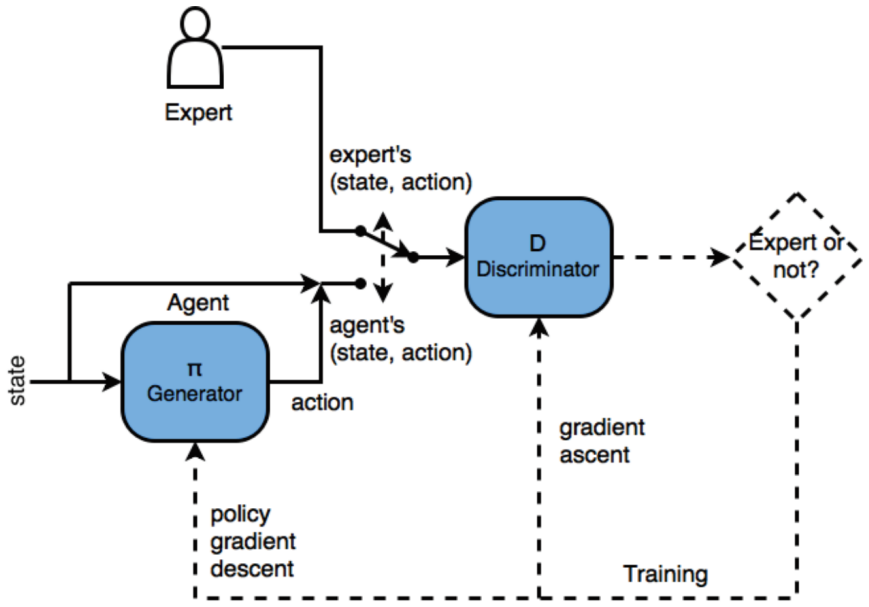


Figure 4.1: Schematic diagram of the two-player adversarial game set-up of Generative Adversarial Imitation Learning (GAIL) [29]

distribution [98]. Tail risk corresponds to events that have a small probability of occurring. When the distribution of market returns is heavy-tailed, tail risk is high because there is a probability, which may be small, that an investment will move beyond three standard deviations. Conditional-Value-at-Risk ($CVaR$) is the most conservative measure of tail risk [99] and unlike other measures like Variance and Value at Risk (VaR), it can be applied when the distribution of returns is not normal. Let Z be a random variable and let $\alpha \in [0, 1]$ denote a probability value. The *Value-at-Risk* of Z with respect to confidence level α, denoted by $VaR_\alpha(Z)$, is defined as the minimum value $z \in \mathbb{R}$ such that with probability α, Z will not exceed z.

$$VaR_\alpha(Z) = \min(z \mid P(Z \leq z) \geq \alpha) \tag{4.8}$$

The Conditional Value at Risk, $CVaR_\alpha(Z)$, is defined as the conditional expectation of losses above $VaR_\alpha(Z)$:

$$CVaR_\alpha(Z) = \mathbb{E}\left[Z \mid Z \geq VaR_\alpha(Z)\right] = \min_{\nu \in \mathbb{R}} H_\alpha(Z, \nu) \tag{4.9}$$

where $H_\alpha(Z, \nu)$ is given by:

$$H_\alpha(Z, \nu) \triangleq \{\nu + \frac{1}{1-\alpha}\mathbb{E}\left[(Z-\nu)^+\right]\}; \qquad (x)^+ = max(x, 0) \tag{4.10}$$

We use $CVaR$ to quantify tail-risk of GAIL-learned policies in this work.

4.2.3 Risk-Awareness in Imitation Learning

Risk sensitivity is integral to human learning ([100, 101]). Risk-sensitive decision-making problems, in the context of MDPs, have been investigated in various fields, e.g., in finance ([102]), operations research ([103, 104]), machine learning ([105, 106]) and robotics ([15, 107, 108, 109]). Garcia et al. [110] give a comprehensive overview of different risk-sensitive RL algorithms. They fall in two broad categories. The first category includes methods that constrain the agent to safe states during exploration while the second modifies the optimality criterion of the agent to embed a term for minimizing risk. Studies on risk-minimization are rather scarce in the imitation learning literature. Majumdar et al. [111] take inspiration from studies like [112, 113, 114] on modeling risk in human decision-making and conservatively approximate the expert's risk preferences by finding an outer approximation of the risk envelope. Much of the literature on imitation learning has been developed with average-case performance at the center, overlooking tail-end events. In this work, we aim to take an inclusive and direct approach to minimizing tail-risk of GAIL-learned policies at test time irrespective of the expert's risk preferences.

4.3 Tail-Risk of GAIL Agents

In order to evaluate the reliability of GAIL-learned policies, we study the histograms of trajectory-costs for the GAIL agents and experts at some benchmark continuous control tasks of OpenAI Gym (see Table 4.1). The expert's cost function is used to calculate these trajectory-costs. Figure 4.2 presents the histograms.

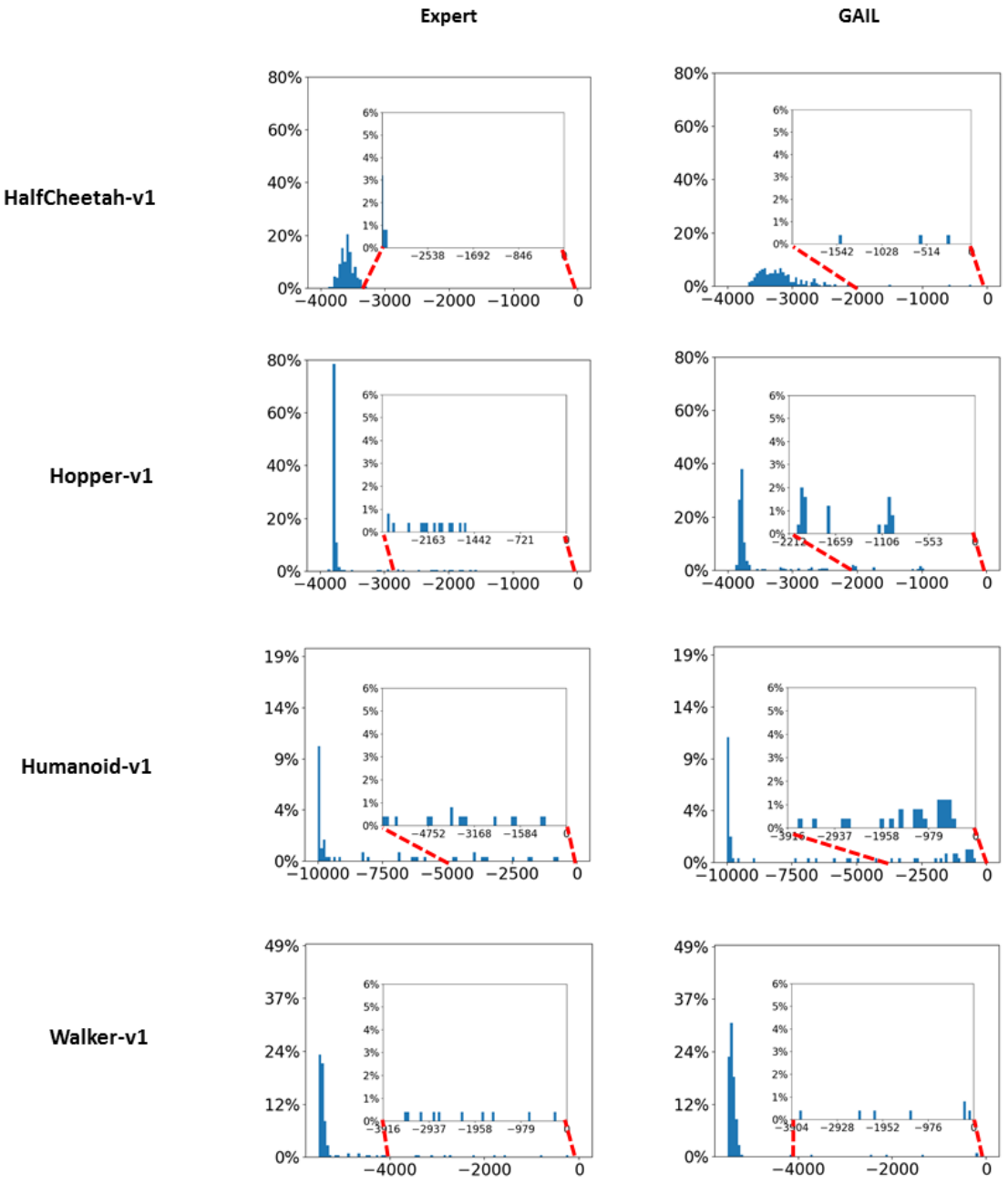


Figure 4.2: Histograms of the costs of 250 trajectories generated by different agents for four continuous-control tasks from OpenAI Gym - HalfCheetah-v1, Hopper-v1, Humanoid-v1 and Walker-v1. The inset diagrams show zoomed-in views of the tails of these distributions (the region beyond 2σ of the mean). We observe that the GAIL agents produce heavy tails, heavier than the expert in most cases, and hence, they have high tail-risk.

Algorithm 6 Risk-Averse Imitation learning (RAIL)

Input: Expert trajectories $\xi_E \sim \pi_E$, hyper-parameters α and β, λ_{CVaR}
Output: Optimized learner's policy π

1: Initialization: $\theta \leftarrow \theta_0$, $w \leftarrow w_0$, $\nu \leftarrow \nu_0$, $\lambda \leftarrow \lambda_{CVaR}$, $i \leftarrow 0$
2: **repeat**
3: Sample trajectories $\xi_i \sim \pi_{\theta_i}$
4: Estimate $\hat{H}_\alpha(\mathcal{R}^\pi(\xi|c(\mathcal{D})), \nu) = \nu + \frac{1}{1-\alpha}\mathbb{E}_{\xi_i}[(\mathcal{R}^\pi(\xi|c(\mathcal{D})) - \nu)^+]$
5: Gradient ascent on discriminator parameters using:

$$\nabla_{w_i}\mathcal{J} = \hat{\mathbb{E}}_{\xi_i}[\nabla_{w_i}\log(\mathcal{D}(s,a))] + \hat{\mathbb{E}}_{\xi_E}[\nabla_{w_i}\log(1-\mathcal{D}(s,a))] + \lambda_{CVaR}\nabla_{w_i}\hat{H}_\alpha(\mathcal{R}^\pi(\xi|c(\mathcal{D})), \nu)$$

6: KL-constrained natural gradient descent step (TRPO) on policy parameters using:

$$\nabla_{\theta_i}\mathcal{J} = \hat{\mathbb{E}}_{(s,a)\sim\xi_i}\left[\nabla_{\theta_i} log(\pi_\theta(a|s)Q(s,a)\right] - \nabla_{\theta_i}H(\pi_\theta) + \lambda_{CVaR}\nabla_{\theta_i}\hat{H}_\alpha(\mathcal{R}^\pi(\xi|c(\mathcal{D})), \nu)$$

where $Q(\bar{s},\bar{a}) = \mathbb{E}_{(s,a)\sim\xi_i}[log(\mathcal{D}_{w_{i+1}}(s,a))|s_0 = \bar{s}, a_0 = \bar{a}]$

7: Gradient descent on CVaR parameters:

$$\nabla_{\nu_i}\mathcal{J} = \nabla_{\nu_i}\hat{H}_\alpha(\mathcal{R}^\pi(\xi|c(\mathcal{D})), \nu)$$

8: $i \leftarrow i + 1$
9: **until** $i ==$ max_iter

The tails of these histograms correspond to the occurrences of high-cost trajectories. We observe that the distribution of trajectory-costs for GAIL-learned policies are more heavy-tailed than the expert's. Since high trajectory-costs may correspond to events of catastrophic failure, GAIL agents are not reliable in risk-sensitive applications like autonomous driving where a single instance of bad performance can cost lives. We aim to explicitly minimize $CVaR$ within the framework of GAIL such that the learned policies are more reliable and still preserve the average performance of GAIL.

4.4 Risk-Averse Imitation Learning

In this section we develop the mathematical formulation of the proposed Risk-Averse Imitation Learning (RAIL) algorithm. We consider stochastic policies $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$. $\pi(a|s)$ gives a probability distribution over actions, $a \in \mathcal{A}$

in a given state, $s \in \mathcal{S}$. Let $\xi = (s_0, a_0, s_1, \ldots, s_{L_\xi})$ denote a trajectory of length L_ξ, obtained by following a policy π. We define expectation of a function $f(\cdot, \cdot)$ defined on $\mathcal{S} \times \mathcal{A}$ with respect to a policy π as follows:

$$\mathbb{E}_\pi[f(s,a)] \triangleq \mathbb{E}_{\xi \sim \pi}\left[\sum_{t=0}^{L_\xi - 1} \gamma^t f(s_t, a_t)\right] \tag{4.11}$$

We define the trajectory-cost variable $\mathcal{R}^\pi(\xi|c(\mathcal{D}))$, in the context of GAIL as:

$$\mathcal{R}^\pi(\xi|c(\mathcal{D})) = \sum_{t=0}^{L_\xi - 1} \gamma^t c(\mathcal{D}(s_t, a_t)) \tag{4.12}$$

where $c(\cdot)$ is an order-preserving function. As the output of the discriminator, $\mathcal{D}$, denotes the likelihood of the input action not coming from the expert, $c(\mathcal{D})$ quantifies the badness of the action. Since the expert's cost function is unknown to the learning agent, we use $c(\mathcal{D})$ to estimate trajectory-cost.

We define an optimization problem that finds a policy that minimizes the maximum tail-risk under the entire class of cost functions $\mathcal{C}$ of which $c(\mathcal{D})$ is a member.

$$\min_\pi \max_c \; CVaR_\alpha(\mathcal{R}^\pi(\xi|c(\mathcal{D}))) = \min_{\pi,\nu} \max_c \; H_\alpha(\mathcal{R}^\pi(\xi|c(\mathcal{D})), \nu) \tag{4.13}$$

Integrating this with the GAIL objective of equation 4.7, we have:

$$\min_{\pi,\nu} \max_{\mathcal{D} \in (0,1)^{\mathcal{S}\times\mathcal{A}}} \mathcal{J} = \min_{\pi,\nu} \max_{\mathcal{D} \in (0,1)^{\mathcal{S}\times\mathcal{A}}} -H(\pi) + \mathbb{E}_\pi[log(\mathcal{D}(s,a))] + \mathbb{E}_{\pi_E}[log(1-\mathcal{D}(s,a))] + \lambda_{CVaR}\; H_\alpha(\mathcal{R}^\pi(\xi|c(\mathcal{D})), \nu) \tag{4.14}$$

Note that as $c(\cdot)$ is order-preserving, the maximization with respect to c in equation 4.13 is equivalent to maximization with respect to $\mathcal{D}$ in equation 4.14. λ_{CVaR} is a constant that controls the amount of weightage given to $CVaR$ optimization relative to the original GAIL objective. Equation 4.14 comprises the objective function of the proposed Risk-Averse Imitation Learning (RAIL) algorithm. Algorithm 6 gives the pseudo-code. We use Adam algorithm [115] for gradient ascent

in the discriminator and Trust Region Policy Optimization (TRPO) [22] for policy gradient descent in the generator. The $CVaR$ term α is trained by batch gradient descent [116].

4.5 Calculating Gradients of CVaR

In this section we derive expressions of gradients of the CVaR term in equation 4.13 with respect to π, $\mathcal{D}$, and ν. Let us denote $H_\alpha(\mathcal{R}^\pi(\xi|c(\mathcal{D})), \nu)$ by $\mathcal{L}_{CVaR}$. Our derivations are inspired by those shown by Chow et al. [117].

Gradient of $\mathcal{L}_{CVaR}$ w.r.t. $\mathcal{D}$:

$$\begin{aligned} \nabla_{\mathcal{D}}\, \mathcal{L}_{CVaR} &= \nabla_{\mathcal{D}} \left[\nu + \frac{1}{1-\alpha} \mathbb{E}_{\xi\sim\pi} \left[(\mathcal{R}^\pi(\xi|c(\mathcal{D})) - \nu)^+ \right] \right] \\ &= \frac{1}{1-\alpha} \mathbb{E}_{\xi\sim\pi} \left[\nabla_{\mathcal{D}}\, \mathcal{R}^\pi(\xi|c(\mathcal{D})) \mathbf{1}(\mathcal{R}^\pi(\xi|c(\mathcal{D})) \geq \nu) \right] \end{aligned} \tag{4.15}$$

where $\mathbf{1}(\cdot)$ denotes the *indicator function*.

Now,

$$\nabla_{\mathcal{D}}\, \mathcal{R}^\pi(\xi|c(\mathcal{D})) = \nabla_c\, \mathcal{R}^\pi(\xi|c(\mathcal{D}))\, \nabla_{\mathcal{D}}\, c(\mathcal{D}) \tag{4.16}$$

We have:

$$\begin{aligned} \nabla_c\, \mathcal{R}^\pi(\xi|c(\mathcal{D})) &= \nabla_c \sum_{t=0}^{L_\xi - 1} \gamma^t c(s_t, a_t) \\ &= \sum_{t=0}^{L_\xi - 1} \gamma^t \end{aligned}$$

$$= \frac{1-\gamma^{L_\xi}}{1-\gamma} \tag{4.17}$$

Substituting equation 4.17 in 4.16 and then 4.16 in 4.15, we have the following:

$$\nabla_{\mathcal{D}} \quad \mathcal{L}_{CVaR} \quad = \quad \frac{1}{1-\alpha}\mathbb{E}_{\xi\sim\pi}\left[\frac{1-\gamma^{L_\xi}}{1-\gamma}\ \mathbf{1}(\mathcal{R}^\pi(\xi|c(\mathcal{D})) \geq \nu)\nabla_{\mathcal{D}}\ c(\mathcal{D})\ \right] \tag{4.18}$$

Gradient of $\mathcal{L}_{CVaR}$ w.r.t. π:

$$\begin{aligned}
\nabla_\pi\ \mathcal{L}_{CVaR} &= \nabla_\pi\ H_\alpha(\mathcal{R}^\pi(\xi|c(\mathcal{D})), \nu) \\
&= \nabla_\pi\ \left[\nu + \frac{1}{1-\alpha}\mathbb{E}_{\xi\sim\pi}\left[(\mathcal{R}^\pi(\xi|c(\mathcal{D})) - \nu)^+\right]\right] \\
&= \frac{1}{1-\alpha}\nabla_\pi\ \mathbb{E}_{\xi\sim\pi}\left[(\mathcal{R}^\pi(\xi|c(\mathcal{D})) - \nu)^+\right] \\
&= \frac{1}{1-\alpha}\mathbb{E}_{\xi\sim\pi}\left[(\nabla_\pi\ logP(\xi|\pi))(\mathcal{R}^\pi(\xi|c(\mathcal{D})) - \nu)^+\right]
\end{aligned} \tag{4.19}$$

Gradient of $\mathcal{L}_{CVaR}$ w.r.t. ν:

$$\begin{aligned}
\nabla_\nu\ \mathcal{L}_{CVaR} &= \nabla_\nu\left[\nu + \frac{1}{1-\alpha}\mathbb{E}_{\xi\sim\pi}\left[(\mathcal{R}^\pi(\xi|c(\mathcal{D})) - \nu)^+\right]\right] \\
&= 1 + \frac{1}{1-\alpha}\mathbb{E}_{\xi\sim\pi}\left[\nabla_\nu\ (\mathcal{R}^\pi(\xi|c(\mathcal{D})) - \nu)^+\right] \\
&= 1 - \frac{1}{1-\alpha}\mathbb{E}_{\xi\sim\pi}\left[\mathbf{1}(\mathcal{R}^\pi(\xi|c(\mathcal{D})) \geq \nu)\right]
\end{aligned} \tag{4.20}$$

4.6 Experiments

We compare the tail-risk of policies learned by GAIL and RAIL for 5 continuous control tasks listed in Table 4.1. All these environments, were simulated using MuJoCo Physics Simulator [118]. Each of these environments come packed with a

"true" reward function in OpenAI Gym [119]. Ho and Ermon [29] trained neural network policies using Trust Region Policy Optimization (TRPO) [22] on these reward functions to achieve state-of-the-art performance and made the pre-trained models publicly available for all these environments as a part of their repository [120]. They used these policies to generate the expert trajectories in their work on GAIL [29]. We also use the same policies to generate expert trajectories in our experiments. Table 4.1 gives the number of expert trajectories sampled for each environment. These numbers correspond to the best results reported in Ho and Ermon [29].

Again, following Ho and Ermon [29], we model the generator (policy), discriminator and value function (used for advantage estimation [121] for the generator) with multi-layer perceptrons of the following architecture: `observationDim - fc_100 - tanh - fc_100 - tanh - outDim`. `fc_100` means fully connected layer with 100 nodes. `tanh` is the hyperbolic tangent activation function of the hidden layers. `observationDim` stands for the dimensionality of the observed feature space. The output dimension, `outDim`, is equal to 1 for the discriminator and value function networks. As we have a stochatic policy, we predict the mean and standard deviation of the Gaussian distribution from which the action must be sampled. Hence, `outDim` is equal to the twice of the dimensionality of the action space for the policy network. For example, in the case of `Humanoid-v1` environment, `observationDim` = 376 and `outDim` = 34 in the policy network. The value of the $CVaR$ coefficient λ_{CVaR} is set as given by Table 4.1. All the hyperparameters corresponding to the GAIL component of the algorithm are set identical to those used in Ho and Ermon [29] and [120] for all the experiments. The value of α in the $CVaR_\alpha$ term is set to 0.9 and its lone parameter, ν, is trained by gradient descent with learning rate 0.01.

4.6.1 Evaluation Metrics

In this section we define the metrics used to evaluate the efficacy of RAIL at reducing the tail risk of GAIL learned policies. Given an agent A's policy π_A we roll out N trajectories $T = \{\xi_i\}_{i=1}^N$ from it and estimate VaR_α and $CVaR_\alpha$ as defined

Table 4.1: Hyperparameters for RAIL experiments. Number of training iterations and expert trajectories are same as those used by Ho and Ermon [29]

Task	**No. of training iterations**	**No. of expert trajectories**	λ_{CVaR}
`Reacher-v1`	200	18	0.25
`HalfCheetah-v1`	500	25	0.5
`Hopper-v1`	500	25	0.5
`Walker-v1`	500	25	0.25
`Humanoid-v1`	1500	240	0.75

in section 4.2.2. VaR_α denotes the value under which the trajectory-cost remains with probability α and $CVaR_\alpha$ gives the conditional expected value of cost above VaR_α. Intuitively, $CVaR_\alpha$ gives the average value of cost of the worst cases that have a total probability no more than $(1-\alpha)$. The lower the value of both these metrics, the lower is the tail risk.

In order to compare tail risk of an agent with respect to the expert, E, we define percentage relative-VaR_α as follows:

$$VaR_\alpha(A|E) = 100 \times \frac{VaR_\alpha(E) - VaR_\alpha(A)}{|VaR_\alpha(E)|}\% \tag{4.21}$$

Similarly, we define percentage relative-$CVaR_\alpha$ as follows:

$$CVaR_\alpha(A|E) = 100 \times \frac{CVaR_\alpha(E) - CVaR_\alpha(A)}{|CVaR_\alpha(E)|}\% \tag{4.22}$$

The higher these numbers, the lesser is the tail-risk of agent A. We define *Gain in Reliability* (GR) as the difference in percentage relative tail-risk between RAIL and GAIL agents.

$$GR\text{-}VaR = VaR_\alpha(RAIL|E) - VaR_\alpha(GAIL|E) \tag{4.23}$$

$$GR\text{-}CVaR = CVaR_\alpha(RAIL|E) - CVaR_\alpha(GAIL|E) \tag{4.24}$$

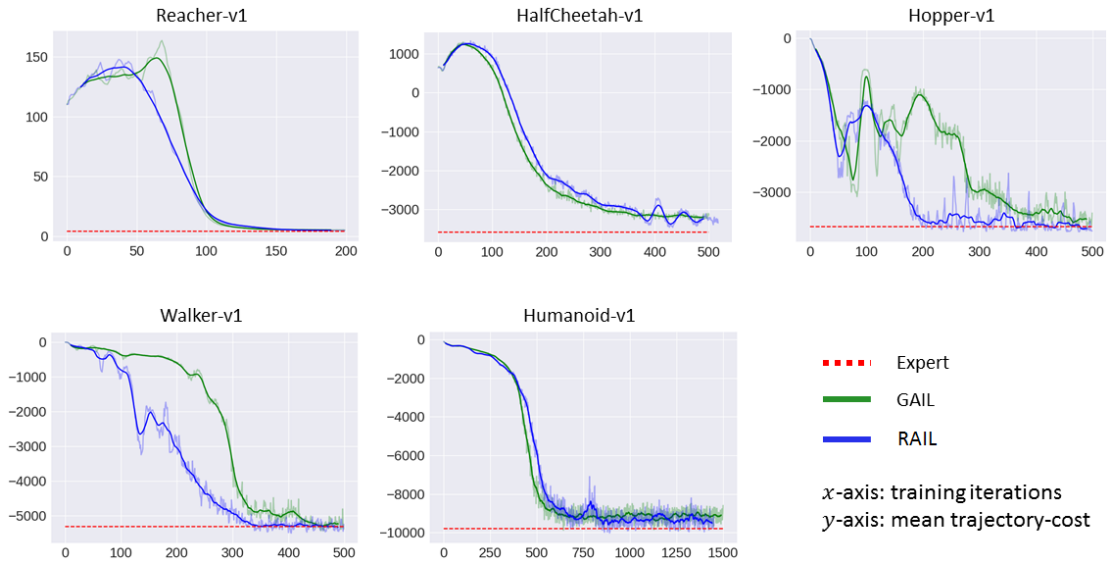


Figure 4.3: Convergence of mean trajectory-cost during training of GAIL vs. RAIL. The faded curves corresponds to the original value of mean which varies highly between successive iterations. The data is smoothened with a moving average filter of window size 21 to demonstrate the prevalent behavior and plotted with solid curves. The light curves show the actual variation of the data.

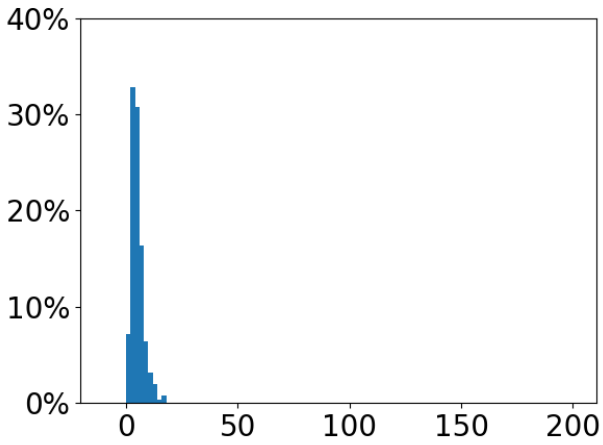


Figure 4.4: Histogram of costs of 250 trajectories generated by a GAIL-learned policy for `Reacher-v1`. The distribution shows no heavy tail.

Table 4.2: Comparison of expert, vanilla GAIL (GAIL), and RAIL in terms of the tail-end risk metrics - $VaR_{0.9}$ and $CVaR_{0.9}$. All the scores are calculated on samples of 50 trajectories. With smaller values of VaR and $CVaR$, our method outperforms GAIL in all the 5 continuous control tasks and also outperforms the expert in many cases.

Environment	**Dimensionality**		**VaR**			**CVaR**		
	Observation	**Action**	**Expert**	**GAIL**	**Ours**	**Expert**	**GAIL**	**Ours**
Reacher-v1	11	2	5.88	9.55	7.28	6.34	13.25	9.41
Hopper-v1	11	3	-3754.71	-1758.19	-3745.90	-2674.65	-1347.60	-3727.94
HalfCheetah-v1	17	6	-3431.59	-2688.34	-3150.31	-3356.67	-2220.64	-2945.76
Walker-v1	17	6	-5402.52	-5314.05	-5404.00	-2310.54	-3359.29	-3939.99
Humanoid-v1	376	17	-9839.79	-2641.14	-9252.29	-4591.43	-1298.80	-4640.42

Table 4.3: Values of percentage relative tail-risk measures and gains in reliability on using RAIL over GAIL for different continuous control tasks.

Environment	$VaR_{0.9}(A\|E)$**(%)**		**GR-VaR** (%)	$CVaR_{0.9}(A\|E)$ **(%)**		**GR-CVaR** (%)
	GAIL	**RAIL**		**GAIL**	**RAIL**	
Reacher-v1	-62.41	-23.81	**38.61**	-108.99	-48.42	**60.57**
Hopper-v1	-53.17	-0.23	**52.94**	-49.62	39.38	**89.00**
HalfCheetah-v1	-21.66	-8.20	**13.46**	-33.84	-12.24	**21.60**
Walker-v1	-1.64	0.03	**1.66**	45.39	70.52	**25.13**
Humanoid-v1	-73.16	-5.97	**67.19**	-71.71	1.07	**72.78**

4.6.2 Results and Discussion

In this section we present and discuss the results of comparison between GAIL and RAIL. The expert's performance is used as a benchmark. Tables 4.2 and 4.3 present the values of our evaluation metrics for different continuous-control tasks. We set $\alpha = 0.9$ for VaR_α and $CVaR_\alpha$ and estimate all metrics with $N = 50$ sampled trajectories (following the example of Ho and Ermon [29]). The following are some interesting observations that we make:

- RAIL obtains superior performance than GAIL at both tail-risk measures – $VaR_{0.9}$ and $CVaR_{0.9}$, across a wide range of continuous-control tasks. This shows that RAIL is a superior choice than GAIL for imitation learning in risk-sensitive applications.

- The applicability of RAIL is not limited to environments in which the distribution of trajectory-cost is heavy-tailed for GAIL. Rockafellar and Urya-

sev [32] showed that if the distribution of the risk variable Z be normal, $CVaR_\alpha(Z) = \mu_Z + a(\alpha)\sigma_Z$, where $a(\alpha)$ is a constant for a given α, μ_Z and σ_Z are the mean and standard deviation of Z. Thus, in the absence of a heavy tail, minimization of $CVaR_\alpha$ of the trajectory cost aids in learning better policies by contributing to the minimization of the mean and standard deviation of trajectory cost. The results on `Reacher-v1` corroborate our claims. Although the histogram does not show a heavy tail (Figure 4.4), the mean converges fine (Figure 4.3) and tail-risk scores are improved (Table 4.2) which in this case indicates the distribution of trajectory-costs is more condensed around the mean than GAIL. Thus we can use RAIL instead of GAIL, no matter whether the distribution of trajectory rewards is heavy-tailed for GAIL or not.

- Figure 4.3 shows the variation of mean trajectory cost over training iterations for GAIL and RAIL. We observe that RAIL converges at least as fast as GAIL, at all the continuous-control tasks in discussion.

- The success of RAIL in learning a viable policy for `Humanoid-v1` suggests that RAIL is scalable to large environments. Scalability is one of the salient features of GAIL. RAIL preserves the scalability of GAIL while showing lower tail-risk.

4.7 Summary

This chapter presents the RAIL algorithm which incorporates $CVaR$ optimization within the original GAIL algorithm to minimize tail risk and thus improve reliability of learned policies. We report significant improvement over GAIL at a number of evaluation metrics on five continuous-control tasks. Thus the proposed algorithm is a viable step in the direction of learning low-risk policies by imitation learning in complex environments, especially in risk-sensitive applications like autonomous driving.

CHAPTER 5

ExTra: Transfer-guided Exploration

5.1 Introduction

Reinforcement Learning (RL) provides a powerful set of tools for training agents to plan in highly complex task-environments. An RL agent learns by trial and error using reward signals that are indicative of progress or accomplishment of the target task [13]. The agent uses two policies to act in the environment during the learning phase. The policy that the agent learns to exploit for solving the target task is known as the target policy. The agent also has a behavioral policy that it uses for exploration in order to find better solutions for the target policy. The sample efficiency and convergence time of an RL algorithm depends heavily on the exploration method used by the behavioral policy [122].

While attempting to solve a new task human beings tend to take actions motivated by similar situations faced in the past. These actions often happen to be good starting points even if the prior experiences are not in the exact same task-environment. In this chapter we present a novel approach for transfer-guided exploration in reinforcement learning that is inspired by the human tendency

to leverage experiences from similar encounters in the past while navigating a new task. Given an optimal policy in a related task-environment, we show that its bisimulation distance from the current task-environment gives a lower bound on the optimal advantage of state-action pairs in the current task-environment. Transfer-guided Exploration (ExTra) samples actions from a Softmax distribution over these lower bounds. In this way, actions with potentially higher optimum advantage are sampled more frequently. In our experiments on gridworld environments, we demonstrate that given access to an optimal policy in a related task-environment, ExTra can outperform popular domain-specific exploration strategies like epsilon greedy, Model-Based Interval Estimation – Exploration Bonus (MBIE-EB), Pursuit and Boltzmann at the rate of convergence. We further show that ExTra is robust to choices of source task and shows a graceful degradation of performance as the dissimilarity of the source task increases. We also demonstrate that ExTra, when used alongside traditional exploration algorithms, improves their rate of convergence. Thus it is capable of complementing the efficacy of traditional exploration methods.

5.2 Related Work

A large body of research in reinforcement learning has been dedicated to the formulation of sample-efficient algorithms. Some of the notable developments include count based exploration [123], curiosity driven exploration [124], optimism in the face of uncertainty [125], Thompson sampling or posterior sampling and bootstrapped methods [126], parameter space exploration [127] and intrinsic motivation [128]. However these methods are based on heuristics specific to the current environment and do not use any prior experiences of the agent in other environments.

Leveraging prior experiences for improving the efficiency of exploration has been studied in the past. This includes learning *action priors* from one or more source tasks to identify useful actions or affordances for the target task [129], policy and value function initialization [130], policy reuse [131], policy transfer through reward shaping [132] and transfer of samples [133]. Recent works in

the Meta Reinforcement Learning community have approached this problem from the perspectives of representation learning [134] and introduction of structured stochasticity via a learned latent space [135]. However most of these methods are limited to using source tasks that share the same state and action spaces as the target task. We are interested in the more general setting of cross-domain transfer where the source environment can have different state and action spaces. Cross-domain transfer in RL has been studied by [136] and [137] who propose methods involving transfer of rules and adaptation of the source policy respectively. We take a different approach in our paper that is based on the theory of bisimulation based policy transfer [34, 138].

5.3 Background

In this section we present a brief introduction to the essential theoretical concepts used in this chapter.

5.3.1 Exploration in Reinforcement Learning

A reinforcement learning agent learns through trial and error in the environment. At any step of decision making, the agent either "exploits" the best policy it has learned or "explores" other actions in search of a better strategy. Balancing exploration and exploitation is a key challenge in RL and a large body of literature has been dedicated to the formulation of strategies that address this dilemma. Exploration strategies can be widely classified into two categories: *directed* and *undirected* [139]. While directed exploration methods utilize exploration specific knowledge collected online, undirected exploration methods are driven almost purely by randomness with the occasional usage of estimates of utility of a state-action pair [139, 122]. Popular undirected exploration algorithms include random walk [140], ϵ-greedy [141, 13] and softmax or Boltzmann exploration [142, 143]. On the other hand, some notable directed exploration methods are count-based [144, 145], error-based [146, 147], and recency based [144]. In the rest of this section, we briefly describe the exploration algorithms that are used as baselines in the current work. For a detailed review of exploration algorithms

in RL, please refer to [122].

ϵ-greedy:

In *epsilon*-greedy exploration, the agent explores by choosing random actions with probability ϵ and follows the learned policy greedily the rest of the time.

Model Based Interval Estimation - Exploration Bonus (MBIE-EB):

MBIE-EB [148] is a count-based exploration algorithm that supplies the agent with count-based reward bonuses for favouring exploration of less visited states and actions. The reward bonus is calculated as:

$$r_{bonus}(s,a) = \frac{\beta}{\sqrt{n(s,a)}} \tag{5.1}$$

where $n(s,a)$ is the number of times the agent chose the state action pair (s,a).

Pursuit:

Pursuit [13] is an undirected exploration algorithm for Multi-Arm Bandits, adapted for MDP by Tijsma et al. [122]. In Pursuit, the agent follows a stochastic policy $\pi(s,a)$. After the update step, t, if $a^*_{t+1} = arg\max_a Q_{t+1}(s_t,a)$, Pursuit updates $\pi_{t+1}(s_t,a)$ as follows:

$$\pi_{t+1}(s_t,a) = \begin{cases} \pi_t(s_t,a) + \beta[1 - \pi_t(s_t,a)], & \text{if } a = a^*_{t+1} \\ \pi_t(s_t,a) + \beta[0 - \pi_t(s_t,a)], & \text{if } a \neq a^*_{t+1} \end{cases} \tag{5.2}$$

where β is a hyperparameter.

Boltzmann exploration:

Boltzmann or Softmax exploration [13] is another undirected exploration algorithm in which the probabilities of the different actions are assigned by a Boltzmann distribution over the state-action value function $Q_t(s_t,a)$.

$$\pi(s_t,a) = \frac{e^{Q_t(s_t,a)/T}}{\sum_{i=1}^{m} e^{Q_t(s_t,a^i)/T}} \tag{5.3}$$

5.3.2 Transfer in Reinforcement Learning

Transfer learning aims to achieve generalization of skills across tasks by using knowledge gained in one task to accelerate learning of a different task. In reinforcement learning, the transferred knowledge can be high level, such as, rules or advice, sub-task definitions, shaping reward, or low level, such as, task features, experience instances, task models, policies, value functions and distribution priors. Most transfer algorithms for RL make certain assumptions about the relationship between the source and target MDPs. Taylor et al. [138] gives a comprehensive overview. In this work, we study the general case of transfer of knowledge between MDPs with discrete state and action spaces and make no assumptions about their structures or relationships. We use the bisimulation transfer framework of Castro et al. [34] as it provides a principled method of transfer for the general setting and a way to estimate the relative goodness of actions under the transferred model [149, 150].

5.3.3 Bisimulation Metric

Bisimulation, first introduced for MDP by Givan et al. [151], is a relation that draws equivalence between states of an MDP that have the same long-term behavior. The following is the definition of the bisimulation relation:

Definition 1. ***Bisimulation relation*** *[151]:*
For an MDP, $\mathcal{M} = \langle S, A, P, R, \gamma \rangle$, a relation $E \subset S \times S$ is a bisimulation relation if, for $s_1, s_2 \in S$, whenever $s_1 E s_2$, the following conditions hold:

$$\begin{aligned} \forall a \in A, \quad R(s_1, a) &= R(s_2, a) \\ \forall a \in A, \forall C \in S/E, \quad \sum_{s' \in C} P(s_1, a)(s') &= \sum_{s' \in C} P(s_2, a)(s') \end{aligned}$$

S/E is the set of equivalence classes induced by E in S.

The theory of bisimulation is equivalent to the theory of MDP homomorphism [152, 153] that studies equivalence relations based on reward structure and tran-

sition dynamics. Ferns et al. [149] proposed bisimulation metric as a quantitative analogue of the bisimulation relation that can be used as a notion of distance between states of an MDP.

Definition 2. ***Bisimulation metric*** *[149]:*
For an MDP, $\mathcal{M} = \langle S, A, P, R, \gamma \rangle$, let M be the set of all semi-metrics on $S \times S$. $\forall s, s' \in S$, the bisimulation metric is given by:

$$d(s, s') = \max_{a \in A} \; c_R \; |R(s, a) - R(s', a)| + c_T \; d_P(P(s, a), P(s', a))$$

Where, d_P is some probability metric and c_R and c_T are two positive 1-bounded constants. Under this metric, two states are bisimilar if their bisimulation distance is zero. The higher the distance, the more dissimilar the states are. In order to measure the bisimulation distance between state-action pairs of different MDPs, Taylor et al. [150] introduced the lax bisimulation metric. Considering two MDPs, $\mathcal{M}_1 = \langle S_1, A_1, P_1, R_1, \gamma_1 \rangle$ and $\mathcal{M}_2 = \langle S_2, A_2, P_2, R_2, \gamma_2 \rangle$ the lax bisimulation metric is defined as follows.

Definition 3. ***Lax Bisimulation metric*** *[150, 34]:*
Let M be the set of all semi-metrics on $S_1 \times A_1 \times S_2 \times A_2$. $\forall s_1 \in S_1, a_1 \in A_1, s_2 \in S_2, a_2 \in A_2, d \in M$, $F : M \to M$ is defined as:

$$F(d)(s_1, a_1, s_2, a_2) = c_R |R_1(s_1, a_1) - R_2(s_2, a_2)| + c_T T_K(d')(P_1(s_1, a_1), P_2(s_2, a_2))$$

Where

$$d'(s_1, s_2) = \max(\max_{a_1 \in A_1} \min_{a_2 \in A_2} d((s_1, a_1), (s_2, a_2)), \min_{a_1 \in A_1} \max_{a_2 \in A_2} d((s_1, a_1), (s_2, a_2)))$$

and $T_K(d')(P_1(s_1, a_1), P_2(s_2, a_2))$ is the Kantorovich distance [154] between the transition probability distributions. $c_R, c_T \in \mathbb{R}$ are tunable hyperparameters representing relative weightages for the reward and Kantorovich components. F has a least-fixed point d_L and d_L is called the Lax Bisimulation metric between $\mathcal{M}_1$ and $\mathcal{M}_2$. $d'_L(s_1, s_2)$ is the state lax bisimulation distance.

Lax bisimulation metric forms the basis of our transfer-guided exploration algorithm presented in this chapter.

5.3.4 Bisimulation Based Policy Transfer

The lax bisimulation metric gives a measure of similarity between state-action pairs of two different task-environments, based on their long-term behavior. Castro et al. [34] propose a method of policy transfer between MDPs that is based on the bisimulation metric. Let $\mathcal{M}_1 = \langle S_1, A_1, P_1, R_1, \gamma_1 \rangle$ be a source MDP with known optimal policy π_1^*. Let $\mathcal{M}_2 = \langle S_2, A_2, P_2, R_2, \gamma_2 \rangle$ be the target MDP. The first step of the algorithm finds the matching state in the source environment, $s_{match}(s_2)$, for every target state $s_2 \in S_2$ as follows:

$$s_{match}(s_2) = arg \min_{s \in S_1} d'_L(s, s_2) \tag{5.4}$$

In the next step it determines the transferred action a_2^T for each target state s_2 as the target action that minimizes the lax bisimulation distance with the matching source state $s_{match}(s_2)$ and its optimal action $\pi_1^*(s_{match}(s_2))$.

$$a_2^T = arg \min_{a \in A_2} d_L((s_{match}(s_2), \pi_1^*(s_{match}(s_2))), (s_2, a)) \tag{5.5}$$

Algorithm 7 gives the pseudo-code. As only optimal actions are transferred from the source environment, Castro et al. [34] suggest to calculate lax bisimulation distance only for the optimal source actions in the interest of faster computation. They define a simplified notation for the lax bisimulation distance as follows:

$$d_{\approx}(s_1, (s_2, a_2)) = d_L((s_1, \pi_1^*(s_1)), (s_2, a_2)) \tag{5.6}$$

The state lax bisimulation distance $d'_{\approx}(s_1, s_2)$ becomes:

$$d'_{\approx}(s_1, s_2) = \max_{a \in A_2} d_{\approx}(s_1, (s_2, a)) \tag{5.7}$$

The authors derive a lower bound on the optimum value of the lax bisimulation transferred actions in terms of the optimum value of source states and bisimulation distance (Corollary 1).

Algorithm 7 Lax Bisimulation Transfer (Algorithm 1 of [34]).

1: **for** All $s_1 \in S_1$, $a_1 \in A_1$, $s_2 \in S_2$, $a_2 \in A_2$ **do**
2: Compute $d_L((s_1, a_1)(s_2, a_2))$
3: **end for**
4: **for** All $s_2 \in S_2$ **do**
5: $s_{match} \leftarrow arg\min_{s \in S_1} d'_L(s, s_2)$
6: $a_2^T \leftarrow arg\min_{a \in A_2} d_L((s_{match}, \pi_1^*(s_{match})), (s_2, a))$
7: $\pi_L(s_2) \leftarrow a_2^T$
8: **end for**
9: **return** π_L

Corollary 1. *(Corollary 11 from [34]):* $\forall s_1 \in S_1, \forall s_2 \in S_2$,

$$Q_2^*(s_2, a_2^T) \geq V_1^*(s_1) - d'_{\approx}(s_1, s_2)$$

This lower bound is interesting because incorporates the behavioral similarity of the source and target states as well as the value of the source state. The authors improve the lax bisimulation transfer algorithm (Algorithm 7) by using this lower bound for state-matching. For each target state $s_2 \in S_2$, the matching source state is determined by maximizing this lower bound:

$$s_{match}(s_2) = arg\max_{s \in S_1} V_1^*(s) - d'_{\approx}(s, s_2) \tag{5.8}$$

We present the modified algorithm in Algorithm 8. LB in line 6 stands for "lower bound". As the state bisimulation metric is calculated by maximizing the distance between the source and target states (Equation 5.7) the authors call this bisimulation transfer "*pessimistic*". The authors note that this pessimism in the definition of bisimulation metric may lead to a poor transfer. They explain it using the following example situation. Suppose $\exists s \in S_1$ and $a \in A_2$ for which $d_{\approx}(s, (s_2, a)) \to 0$. However, there exists a target action $b \in A_2$ for which $d_{\approx}(s, (s_2, b))$ is large. This would cause $d'_{\approx}(s_1, s_2)$ to be large. As a result, s might not be chosen as s_{match} although it could have very well qualified as the best choice.

In order to overcome this inherent pessimism of bisimulation, the authors propose an *optimistic* definition of the state lax bisimulation metric as follows:

$$d'_{\approx}(s_1, s_2) = \min_{a_2 \in A_2} d_{\approx}(s_1, (s_2, a_2)) \tag{5.9}$$

Using this definition in Algorithm 8 we have the optimistic bisimulation policy transfer algorithm. Although this lacks the theoretical guarantees of pessimistic transfer, the authors note that it yields superior transfer quality in practice.

Algorithm 8 Bisimulation based policy transfer [34].

for All $s_1 \in S_1$, $s_2 \in S_2$, $a_2 \in A_2$ **do**
    Compute $d_{\approx}(s_1, (s_2, a_2))$
end for
for All $s_2 \in S_2$ **do**
    for All $s_1 \in S_1$ **do**
        $LB(s_1, s_2) \leftarrow V_1^*(s_1) - d'_{\approx}(s_1, s_2)$
    end for
    $s_{match} \leftarrow arg\max_{s_1 \in S_1} LB(s_1, s_2)$
    $a_2^T \leftarrow arg\min_{a_2 \in A_2} d_{\approx}(s_{match}, (s_2, a_2))$
end for

5.4 Proposed Work

In this section we present Transfer-guided Exploration (ExTra) as a novel directed exploration method for reinforcement learning that is based on the bisimulation transfer framework of Castro et al. [34]. The motivation is to use transferred knowledge from the optimal policy in a source domain to accelerate RL in a target domain – especially during the initial stage, when the domain-specific statistics used by traditional directed exploration methods are yet to be consistently estimated. We first present some results which relate bisimulation distance to the optimal advantage of an action in a target state.

Lemma 1. $\forall s_1 \in S_1$, $\forall s_2 \in S_2$, $\forall a_2 \in A_2$, $|V_1^*(s_1) - Q_2^*(s_2, a_2)| \leq d_{\approx}(s_1, (s_2, a_2))$.

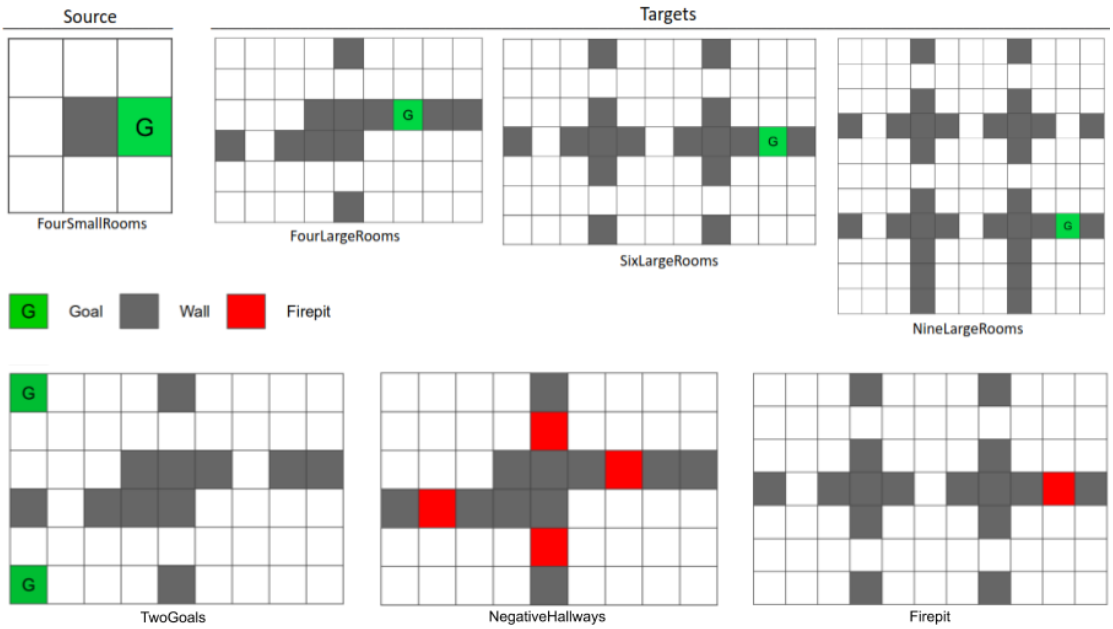


Figure 5.1: Some gridworld environments used in our experiments on ExTra.

Proof:

$$\begin{aligned}
|V_1^*(s_1) - Q_2^*(s_2, a_2)| &= |Q_1^*(s_1, \pi_1^*(s_1)) - Q_2^*(s_2, a_2)| \\
&\leq |R_1(s_1, \pi_1^*(s_1)) - R_2(s_2, a_2)| \\
&\quad + \gamma T_K(d_{\approx})(P_1(s_1, \pi_1^*(s_1)), P_2(s_2, a_2)) \\
&\text{by similar argument as for Lemma 4 in [34]} \\
&= d_{\approx}(s_1, (s_2, a_2)). \square
\end{aligned}$$

Corollary 2. $\forall s_1 \in S_1, \forall s_2 \in S_2, |V_1^*(s_1) - V_2^*(s_2)| \leq d_{\approx}(s_1, (s_2, \pi_2^*(s_2)))$.

The goal of reinforcement learning is to get the agent to take optimal actions and the goal of efficient exploration is to achieve that fast. Hence, efficient exploration algorithms should be able to weigh the actions available in a given state on the basis of their potential optimality. In an MDP, $\mathcal{M} = \langle S, A, P, R, \gamma \rangle$, the advantage of an action $a \in A$ in a state $s \in S$ with respect to a policy π is defined as $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$. It quantifies the relative superiority of action a

compared to the expectation under the policy over next actions. The optimal policy maximizes the optimal Q function: $\pi^*(s) = \arg\max_a Q^*(s,a)$. Substituting the expression of Q in terms of advantage, we have: $\pi^*(s) = \arg\max_a A^*(s,a) + V^*(s) = \arg\max_a A^*(s,a)$. Hence the optimal policy also maximizes the optimal advantage function. Given a source MDP, we derive a lower bound on the optimal advantage of available actions in a target state in terms of the lax-bisimulation distance function and use it as a measure of potential optimality.

Theorem 1. *Given MDPs,* $\mathcal{M}_1 = \langle S_1, A_1, P_1, R_1, \gamma_1 \rangle$ *and* $\mathcal{M}_2 = \langle S_2, A_2, P_2, R_2, \gamma_2 \rangle$ *and bisimulation metric* $d_\approx : S_1 \times S_2 \times A_2 \to \mathbb{R}$ *we have* $\forall s_2 \in S_2, \forall a_2 \in A_2$

$$A_2^*(s_2, a_2) \geq -d_\approx(s_{match}, (s_2, a_2)) - \beta(s_2)$$

Where $A_2^*(s_2, a_2)$ *is the optimum advantage function in* $\mathcal{M}_2$, $s_{match} \in arg\max_{s_1 \in S_1} V_1^*(s_1) - d'_\approx(s_1, s_2)$ *and* $\beta(s_2) = d_\approx(s_{match}, (s_2, \pi_2^*(s_2)))$.

Proof:
Since, $V_2^*(s_2) = arg\max_{a_2 \in A_2} Q_2^*(s_2, a_2) \implies V_2^*(s_2) - Q_2^*(s_2, a_2) \geq 0$.

We have:

$$\begin{aligned}
V_2^*(s_2) - Q_2^*(s_2, a_2) &= |V_2^*(s_2) - Q_2^*(s_2, a_2)| \\
&= |(V_2^*(s_2) - V_1^*(s_{match})) + \\
&\quad (V_1^*(s_{match}) - Q_2^*(s_2, a_2))| \\
&\leq |V_2^*(s_2) - V_1^*(s_{match})| + \\
&\quad |V_1^*(s_{match}) - Q_2^*(s_2, a_2)| \\
&\leq d_\approx(s_{match}, (s_2, \pi_2^*(s_2))) + d_\approx(s_{match}, (s_2, a_2)) \\
&\text{(from Lemma 1 and Corollary 2)} \\
&= \beta(s_2) + d_\approx(s_{match}, (s_2, a_2))
\end{aligned}$$

$\therefore \quad A_2^*(s_2, a_2) = Q_2^*(s_2, a_2) - V_2^*(s_2) \geq -d_\approx(s_{match}, (s_2, a_2)) - \beta(s_2)$ $\square$

Theorem 1 gives a lower bound on the optimal advantage of an action in a target state in terms of the bisimulation distance to a source MDP. As explained

in Section 5.3.4, Castro et al. [34] assign matching source state s_{match} to target state-action pair (s_2, a_2) following a minimum performance criterion in the form of a lower bound on $Q_2^*(s_2, a_2)$ (Equation 5.8). We choose to use s_{match} in Theorem 1 to make sure that this minimum performance criterion is satisfied. While other bounds are possible for choices of $s_1 \in S_1, s_1 \neq s_{match}$, they might not be usable, since they do not guarantee the performance bounds identified by Castro et al. [34].

Corollary 3. *The bisimulation transfer algorithm of Castro et al. [34] maximizes a lower bound on the optimum advantage function $A_2^*(s_2, a_2)$ of the target environment.*

Proof:

In bisimulation transfer, the transferred action a_2^T for target state s_2 is given by:

$$\begin{aligned} a_2^T &= arg \min_{a_2 \in A_2} d_{\approx}(s_{match}, (s_2, a_2)) \\ &= arg \max_{a_2 \in A_2} -d_{\approx}(s_{match}, (s_2, a_2)) - \beta(s_2) \end{aligned}$$

Since, $\beta(s_2)$ is independent of a_2.□

Note that Theorem 1 and Corollaries 2 and 3 hold for both optimistic (Equation 5.9) and pessimistic (Equation 5.7) definitions of the lax bisimulation metric [34].

Definition 4. ***Bisimulation Advantage:*** *Given MDPs, $\mathcal{M}_1 = \langle S_1, A_1, P_1, R_1, \gamma_1 \rangle$ and $\mathcal{M}_2 = \langle S_2, A_2, P_2, R_2, \gamma_2 \rangle$ and bisimulation metric $d_{\approx} : S_1 \times S_2 \times A_2 \to \mathbb{R}$ we define the bisimulation advantage of an action $a_2 \in A_2$ in a state $s_2 \in S_2$ as:*

$$A_{\approx}(s_2, a_2) = \max_{s_{match} \in S_{match}} -d_{\approx}(s_{match}, (s_2, a_2)) - \beta(s_2)$$

Where, $S_{match} = arg \max_{s_1 \in S_1} V_1^*(s_1) - d'_{\approx}(s_1, s_2)$ is the set of matching states in M_1 for state-action pair (s_2, a_2) in $\mathcal{M}_2$. We look to define a probability distribution

over target actions that assigns higher probability to actions having higher bisimulation advantages. As, in general, the solution to $arg\min_{a_2} d_{\approx}(s_{match}, (s_2, a_2))$ is non-unique, under bisimulation, multiple actions in the target environment could potentially map to the optimal action in the source environment. Among all policies that pick at least one of the actions that map to the optimal action under the bisimulation, by the principle of maximum entropy [155] , we choose one that assigns equal probability to all those actions. Softmax distribution satisfies this property. Hence we define the behavioral policy in Transfer-guided Exploration (ExTra) as a Softmax distribution over bisimulation advantages.

$$\pi_{\text{ExTra}}(a_2|s_2, \mathcal{M}_1, \pi_1^*) = \frac{e^{A_{\approx}(s_2,a_2)}}{\sum_{b\in A_2} e^{A_{\approx}(s_2,b)}} \tag{5.10}$$

In transfer-guided exploration (ExTra), the agent samples actions from $\pi_{\text{ExTra}}(\cdot|s_2, \mathcal{M}_1, \pi_1^*)$. Since the optimal policy in the target MDP, π_2^*, is not known during learning, $\beta(s_2)$ can not be known exactly. Since $\beta(s_2)(\geq 0)$ is the same for all actions in a given state s_2, replacing $\beta(s_2)$ with a real positive number preserves the order of probability values assigned to different actions by π_{ExTra}. If transfer is successful, π_{ExTra} would assign higher probabilities to the optimal actions and thus help the agent to quickly arrive at the optimal policy. However, in the event of an unsuccessful transfer, π_{ExTra} may be biased away from the optimal actions. This may cause the agent to remain stuck with the wrong actions for long periods of time. In order to help the agent recover from the effect of unsuccessful transfer, we set $\beta = \alpha n$, where $\alpha \in \mathbb{R}^+$ is a tunable hyperparameter and n is the current step number. As n grows, $\pi_{\text{ExTra}}(\cdot|s_2, \mathcal{M}_1, \pi_1^*)$ tends to a uniform distribution over actions, thus annealing the influence of transferred knowledge on exploration with time. This does not hurt the agent's learning in states where the transfer was successful because the agent happens to have explored the optimal actions early on in training in those states. Note that changing β does not affect the rate of exploration; instead, it merely changes the shape of the probability distribution from which the agent samples actions during exploration. Algorithm 9 gives an example use case of ExTra as an alternative to random exploration in ϵ-greedy Q-learning. Algorithms 10, 11 and 12 give more ways of using ExTra in conjunction with other traditional exploration methods.

Algorithm 9 ϵ-greedy Q-learning with Transfer-guided Exploration (ExTra)

Given: $d_{\approx}(s_1, (s_2, a_2)), \quad \forall s_1 \in S_1, s_2 \in S_2, a_2 \in A_2$
for All $s_2 \in S_2$ **do**
    for All $s_1 \in S_1$ **do**
        $LB(s_1, s_2) \leftarrow V_1^*(s_1) - d'_{\approx}(s_1, s_2)$
    end for
    $s_{match} \leftarrow arg\max_{s_1 \in S_1} LB(s_1, s_2)$
end for
step = 0
while step < MAXSTEPS **do**
    with probability ϵ
        $a_2 \sim \pi_{ExTra}(\cdot|s_2, \mathcal{M}_1, \pi_1^*)$
    with probability $1 - \epsilon$
        $a_2 \leftarrow arg\max_b Q_2(s_2, b)$
    $r = take\_step(a_2)$
    $update\_Q(Q_2(s_2, a_2), r)$
    step = step + 1
end while

Algorithm 10 MBIE-EB Q-learning interleaved with Transfer-guided Exploration.

step = 0
while step < MAXSTEPS **do**
    with probability ϵ
        $a_2 \sim \pi_{ExTra}(\cdot|s_2, \mathcal{M}_1, \pi_1^*)$
    with probability $1 - \epsilon$
        $a_2 \leftarrow arg\max_{b'} Q_2(s_2, a'_2)$
    $r = take\_step(a_2) + \frac{\beta}{\sqrt{n(s_2, a_2)}}$
    $update\_Q(Q_2(s_2, a_2), r)$
    step = step + 1
end while

Algorithm 11 Pursuit Q-learning interleaved with Transfer-guided Exploration

step = 0
$\pi_{pursuit} = Uniform(A_2)$
while step < MAXSTEPS **do**
    with probability ϵ
        $a_2 \sim \pi_{ExTra}(\cdot|s_2, \mathcal{M}_1, \pi_1^*)$
    with probability $1 - \epsilon$
        $a \leftarrow arg\max_{a_2'} Q_2(s_2, a_2')$
        $update\_\pi_{pursuit}(a)$
        $a_2 \leftarrow Sample(\pi_{pursuit})$
    $r = take\_step(a_2)$
    $update\_Q(Q_2(s_2, a_2), r)$
    step = step + 1
end while

Algorithm 12 Softmax Q-learning interleaved with Transfer-guided Exploration

step = 0
while step < MAXSTEPS **do**
    with probability ϵ
        $a_2 \sim \pi_{ExTra}(\cdot|s_2, \mathcal{M}_1, \pi_1^*)$
    with probability $1 - \epsilon$
        $a_2 \sim softmax(Q_2(s_2, \cdot))$
    $r = take\_step(a_2)$
    $update\_Q(Q_2(s_2, a_2), r)$
    step = step + 1
end while

5.5 Experimental Results

In this section we present an empirical analysis of the performance of ExTra. We aim to address the following questions:

1. How does ExTra compare against traditional exploration methods?

2. How sensitive is ExTra to the choice of source task?

3. Can ExTra enhance the performance of other exploration algorithms that only use local information?

4. How does ExTra compare against Bisimulation Transfer?

We use the optimistic definition of the bisimulation metric in all our experiments as Castro et al. [34] note that it gives superior transfer results than the pessimistic definition.

5.5.1 Evaluation Measures

Our goal is to improve the rate of convergence of RL with the help of external knowledge transferred from a different task. The rate of convergence of an RL algorithm can be judged from a plot of Mean Average Reward (MAR) obtained by the agent over steps of training. Mean Average Reward is defined as follows:

Definition 5. ***Mean Average Reward:*** *The average reward of a trajectory τ obtained by following a policy π is the average of all the rewards received by the agent in the trajectory. Mean Average Reward (MAR) of π is the mean of average reward for multiple trajectories rolled out from the policy. A sample-average based estimate of the mean average reward of a policy π can be calculated as follows:*

$$MAR(\pi) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{T_i}\sum_{t=1}^{T_i} R(s_t^i, a_t^i)$$

Where N is the number of trajectories rolled out using policy π and T_i is the length of the i^{th} trajectory. The value of N should be large enough for a consistent estimate of MAR.

The Area under the MAR Curve (AuC-MAR) is an objective measure of rate of convergence when the highest MAR values achieved asymptotically by all the candidate algorithms are the same [138]. A higher AuC-MAR implies higher rate of convergence. We use AuC-MAR for comparing the rates of convergence of RL algorithms using ExTra with the baseline methods. For the convenience of

comparison, we report AuC-MAR as a percentage of AuC for the optimal policy (whose MAR is represented as a straight horizontal line through the steps of training but not shown in the plots to avoid clutter).

For measuring the relative improvement of the rate of convergence achieved by ExTra, we use Transfer Ratio, denoted by TR, defined as follows:

$$TR = \frac{\text{AuC-MAR with ExTra} - \text{AuC-MAR without ExTra}}{\text{AuC-MAR without ExTra}} \tag{5.11}$$

5.5.2 Experimental Design, Results and Discussion

We use stochastic grid-world environments of different levels of complexity (Figures 5.1 and 5.3) for analysing the viability of ExTra. All of these environments have the common task of avoiding obstacles and reaching a goal. After being initialised with uniform probability from any of the grid-cells, the agent gets a reward of $+1$ on reaching the goal, -1 for stepping into a "fire-pit" and 0 elsewhere. The agent has four primitive actions: up, down, left and right. When one of the actions is chosen, the agent moves in the desired direction with 0.9 probability, and with 0.1 probability it moves uniformly in one of the other three directions or stays in the same place. If the agent bumps into a wall, it does not move and remains in the same state.

We choose Q-learning with four traditional exploration algorithms viz. ϵ-greedy uniform random exploration, MBIE-EB [148], Pursuit [13] and Softmax [13] as baselines for comparison. For ExTra, we train the optimal policy for the source domain using value iteration and calculate the optimistic bisimulation distance $d_{\approx}(s_1, (s_2, a_2))$, $\forall s_1 \in S_1, s_2 \in S_2, a_2 \in A_2$, tuning c_R and c_T for maximum transfer accuracy. We use the PyEMD library by [156] to calculate earth mover distance for the estimation of $T_K(d)$ [157, 158, 159]. Our code is available open-source on GitHub[1]. The results reported for each baseline are obtained after rigorous tuning of their respective hyperparameters for the target domain. We

[1]https://github.com/madan96/ExTra

report all MAR and AuC-MAR numbers as their mean ± standard deviation calculated over 10 different experiments with different random seeds. We tabulate the hyperparameter values in Tables 5.1 and 5.2.

Table 5.1: Hyperparameters for Optimistic Bisimulation Transfer

Transfer Parameters	**FourLargeRooms**	**SixLargeRooms**	**NineLargeRooms**
c_R	0.1	0.2	0.1
c_T	0.9	0.9	0.9
Fixed-Point Iter. Threshold	0.01	0.01	0.01

Table 5.2: Hyperparameters used for comparing ExTra with traditional exploration algorithms (Section 5.5.2.1).

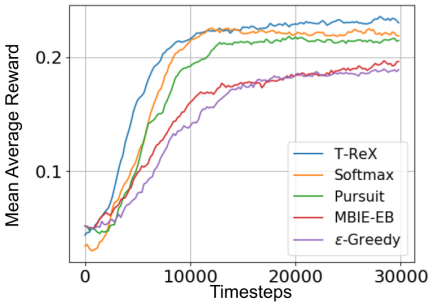
ϵ-greedy	Q Learning Rate	0.2
	ϵ	0.5
Softmax	Q Learning Rate	0.2
	τ	8.1
MBIE-EB	Q Learning Rate	0.2
	cb-β	0.005
	ϵ	0.2
Pursuit	Q Learning Rate	0.2
	β	0.007
ExTra	Q Learning Rate	0.5
	ϵ	0.2
	α	1e-6

5.5.2.1 Comparison of ExTra with Traditional Exploration Methods

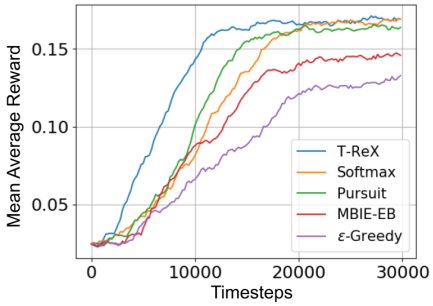
In our first set of experiments, we show that given an optimal policy for a related task-environment, ExTra can obtain faster convergence than traditional exploration methods that only use local information. We choose the `FourLargeRooms`, `SixLargeRooms`, and `NineLargeRooms` environments shown in Figure 5.1 as benchmarks. We train Q-learning agents with ϵ-greedy, MBIE-EB, Pursuit and Softmax explorations in these environments as baselines. For ExTra, we choose `FourSmallRooms` in Figure 5.1 as the source environment. We train an ϵ-greedy Q-learning agent ($\epsilon = 0.2$) that uses ExTra instead of uniform random for exploration

Table 5.3: Percentage AuC-MAR of ϵ-greedy Q-learning with ExTra versus traditional exploration methods (Section 5.5.2.1).

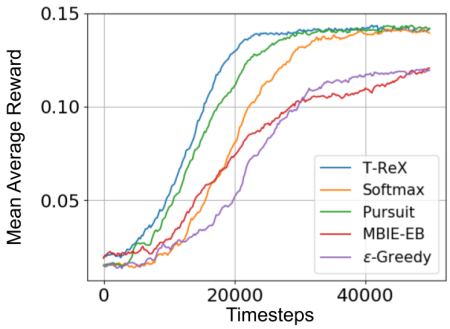
Target Environment	AuC-MAR(%)				
	ϵ-greedy	MBIE-EB	Pursuit	Softmax	ExTra
FourLargeRooms	61.08 ± 3.37	63.91 ± 1.62	73.96 ± 2.59	77.64 ± 1.70	$\mathbf{82.79} \pm 1.68$
SixLargeRooms	45.81 ± 3.07	54.36 ± 2.69	63.44 ± 1.44	61.46 ± 1.31	$\mathbf{72.52} \pm 1.28$
NineLargeRooms	43.20 ± 2.45	45.87 ± 5.00	61.52 ± 2.08	53.66 ± 2.01	$\mathbf{64.81} \pm 1.11$



(a) `FourSmallRooms` → `FourLargeRooms`



(b) `FourSmallRooms` → `SixLargeRooms`



(c) `FourSmallRooms` → `NineLargeRooms`

Figure 5.2: Comparison of ExTra with traditional exploration methods: variation of MAR with steps of training for different `source` → `target` pairs of environments (Section 5.5.2.1).

(Algorithm 6) on each of the three target environments. Figure 5.2 shows the variation of MAR over steps of training for the baseline as well as our ExTra agents. Table 5.3 shows AuC-MAR values. We observe that our ExTra agent consistently achieves faster convergence in all the three environments. This corroborates our claim that ExTra can achieve faster convergence and hence superior sample efficiency if we have access to the optimal policy in a related task-environment.

5.5.2.2 Sensitivity of ExTra to the Choice of Source Task

In this experiment we study the sensitivity of ExTra to the selection of source task. In our *first* study we consider transfer between tasks that share the same state-action space and reward structure but differ in goal positions. We construct 6 tasks in the `SixLargeRooms` environment in which the i^{th} task has its goal at the center of the i^{th} room (see Figure 5.3). We use tasks 1 through 5 as source

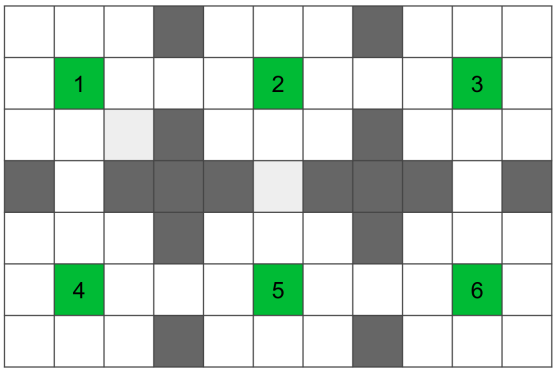


Figure 5.3: Goal locations in the six tasks used to study the sensitivity of ExTra to the choice of source task (Section 5.5.2.2).

and task 6 as target. In our *second* study, we compare transfer from source tasks that differ in state space (`FourSmallRooms`), action space (`3Actions`), reward structure (`Firepit`, `NoGoal`, `NegativeHallways`), goal distribution (`Two Goals`) and transition dynamics (`Gravity`). `3Actions`, `NoGoal` and `Gravity` are variants of the `FourLargeRooms` environment with just three allowed actions (left, right, down), no goal at all and an added probability of 0.1 to slide downwards, respectively. The rest of the environments are as depicted in Figure 5.1. The target task-environment is `FourLargeRooms`. We report AuC-MAR values obtained by ϵ-greedy Q-learning with ExTra (Algorithm 6) for each of these source tasks in Tables 5.4 and 5.5.

We make the following observations:

- In our first study (Table 5.4), each of our ExTra agents fetch higher AuC-MAR values than any of the baseline methods thus demonstrating the efficacy of ExTra. Also there is a rough trend of the AuC-MAR values decreas-

Table 5.4: Variation of percentage AuC-MAR of ϵ-greedy Q-learning with ExTra exploration for different source task goal positions in `SixLargeRooms` environment (Section 5.5.2.2).

		AuC-MAR(%)
Baselines	ϵ-greedy	52.92 ± 3.83
	MBIE-EB	34.80 ± 3.98
	Pursuit	69.05 ± 2.84
	Softmax	64.85 ± 3.29
Source task #	1	73.70 ± 2.21
	2	75.73 ± 2.08
	3	78.54 ± 2.78
	4	74.59 ± 2.81
	5	81.94 ± 1.70

ing with increasing distance of goal in the source task. This demonstrates graceful degradation of performance.

- In our second study, ExTra beats both $\epsilon-$greedy and MBIE-EB for even the worst case choice of source task – `NegativeHallways` (compare Table 5.5 with Row 1 of Table 5.3). This is possible because ExTra is able to leverage knowledge about the transition dynamics of the source MDP (that are identical to the target MDP) even when the reward structures and goal distributions are drastically different. It also reflects the robustness of ExTra to the choice of source task.

- We note that in Table 5.5 the reward structure of the source environment has a more profound effect on the performance of ExTra than transition dynamics or goal distribution. This observation serves as a guide for choosing source environments for bisimulation transfer.

- Higher AuC-MAR for `FourSmallRooms` than `3Actions` suggests that source MDPs with the same action space as the target are more favourable for ExTra even if the state spaces are different.

Table 5.5: Variation of percentage AuC-MAR of ϵ-greedy Q-learning with ExTra exploration for different choices of source task and `FourLargeRooms` as target (Section 5.5.2.2).

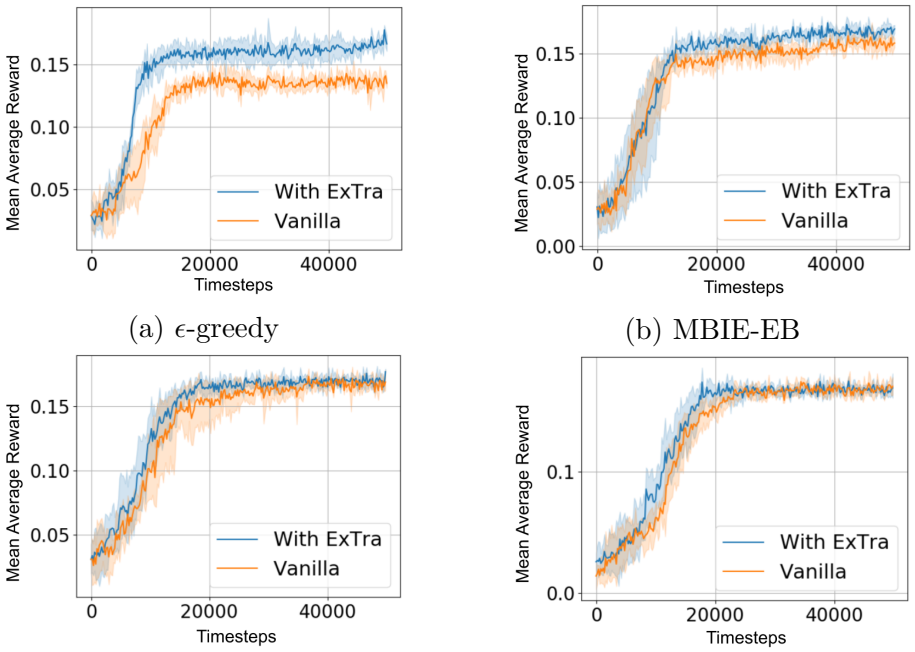
Source Task	**Difference**	**AuC-MAR**(%)
`TwoGoals`	Goal distribution	77.83 ± 1.54
`Firepit`	Reward structure	75.54 ± 1.59
`NoGoal`	Reward structure	75.59 ± 1.81
`NegativeHallways`	Reward structure and Goal distribution	73.36 ± 1.62
`3Actions`	Action space	77.60 ± 1.78
`Gravity`	Transition dynamics	77.78 ± 1.56
`FourSmallRooms`	State space	82.79 ± 1.68

Table 5.6: Comparison of percentage AuC-MAR scores of traditional exploration methods with and without ExTra (Section 5.5.2.3).

	AuC-MAR(%)		TR(%)
	vanilla	**with ExTra**	
ϵ-greedy	61.91 ± 1.30	76.10 ± 1.44	23.41
MBIE-EB	70.55 ± 1.30	74.87 ± 1.68	6.11
Pursuit	72.80 ± 2.02	76.87 ± 1.16	5.59
Softmax	69.51 ± 1.60	72.59 ± 1.14	4.43

5.5.2.3 Enhancing the Performance of Traditional Exploration Algorithms Using ExTra

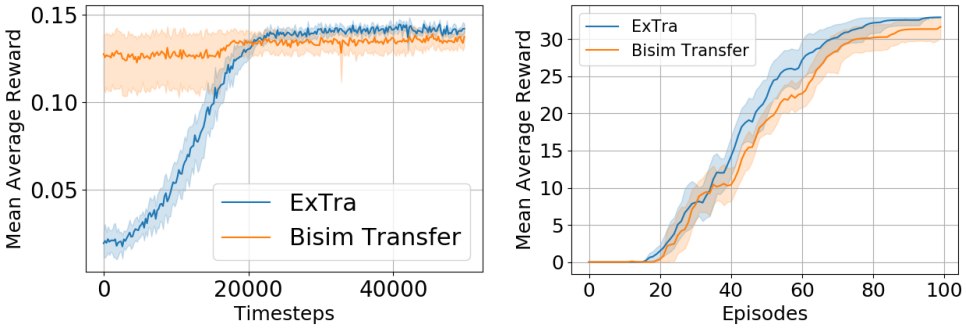
With a view to test if ExTra has a complementary effect when used with traditional exploration algorithms, we formulate ϵ-greedy versions of each of our baseline algorithms ($\epsilon = 0.5$). The agent samples actions from π_{ExTra} with probability ϵ ($\epsilon = 0.5$) and follows the main algorithm rest of the time (Algorithms 10, 11 and 12). We choose `SixLargeRooms` as our benchmark environment and `FourSmallRooms` (Figure 5.1) as source environment for ExTra. Table 5.6 and Figure 5.4 present the results of these experiments. We see that ExTra can provide improved rate of convergence of the traditional methods and hence can indeed be used as a complementary exploration method.



(a) ϵ-greedy (b) MBIE-EB

(c) Pursuit (d) Softmax

Figure 5.4: Complementary effect of using ExTra: improved rate of convergence of traditional exploration algorithms when used in conjunction with ExTra (Section 5.5.2.3).



(a) `FourLargeRooms` (b) `Taxi-v2`

Figure 5.5: Comparison of rate of convergence between $\epsilon-$greedy Q-learning with ExTra and the bisimulation policy transfer algorithm of [34] (Section 5.5.2.4).

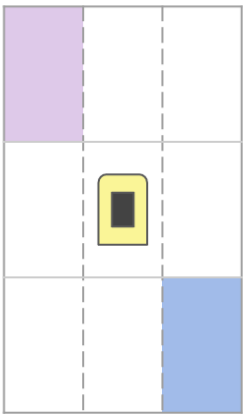


Figure 5.6: Modified `Taxi-v2` environment of OpenAI Gym used in our study on comparing ExTra with Bisimulation Policy Transfer (Section 5.5.2.4)

5.5.2.4 Comparison of ExTra with Bisimulation Transfer

In this experiment, we compare the efficacy of ExTra with Bisimulation Policy Transfer [34] for different levels of similarity of the source and target environments. We choose `FourSmallRooms` as source environment. As targets, we choose `FourLargeRooms` and a modified version of the `Taxi-v2` environment of OpenAI Gym [33] with 54 states (see Figure 5.6). The modified `Taxi-v2` environment has 3 *rows*, 3 *columns* and 2 *drop locations* at the top-left and bottom-right corners. The action space of this environment remains same as the original version, where the agent can choose actions from `South`, `North`, `West`, `East`, `Pickup` and `Drop`. We compare the rates of convergence of $\epsilon-$greedy Q-learning with ExTra and the bisimulation transfer algorithm of Castro et al. [34] that initializes the Q-matrix with the Q-value of the transferred policy.

Figure 5.5 presents the results of the experiments. We see that when the source and target tasks are similar (`FourSmallRooms` and `FourLargeRooms`) and bisimulation policy transfer is successful, Q-learning gets an initial jumpstart while ExTra catches up. When the source and target tasks are drastically different (`FourSmallRooms` and `Taxi-v2`), the bisimulation distances are large and π_{ExTra} tends to a uniform distribution over target actions. As a result, ExTra falls back

to vanilla $\epsilon-$greedy Q-learning with uniform sampling. On the other hand, Q-learning initialized with bisimulation transfer has to first recover from the effect of negative transfer using $\epsilon-$greedy uniform exploration before it can start learning. In this case, it does not get any jumpstart and also converges slower than ExTra which does not need to correct for the damage done by negative transfer and proceeds as vanilla $\epsilon-$greedy Q-learning. We observe that, while bisimulation transfer can be both effective and fatal depending on how the source and target tasks are related, ExTra does not negatively affect the learning process even when the source and target task-environments are drastically different.

5.6 Summary

In this chapter we investigated the fundamental possibility of using transfer to guide exploration in RL and formulate a novel transfer guided exploration algorithm, ExTra, based on the theory of bisimulation based policy transfer in MDPs. We demonstrated that our method achieves faster convergence compared to traditional exploration methods that only use local information. We also showed that ExTra is robust to source task selection and can complement traditional exploration methods by improving their rates of convergence. We also provided theoretical guarantees in the form of a lower bound on the optimal advantage of an action in the target domain in terms of bisimulation distance from the source environment. With access to scalable methods of calculating the bisimulation distance [160], the framework of ExTra and its guarantees can potentially be used in autonomous driving for transfer of knowledge gained in simulation to the real world.

CHAPTER 6

Conclusion

6.1 Summary of Contributions

The dissertation aims at improving the safety and sample complexity of reinforcement learning algorithms for use in autonomous driving. The *first* contribution is MADRaS, an open-source Multi-Agent Driving Simulator capable of creating driving tasks of high variance. It offers a wide variety of traffic conditions, noisy observations, stochastic actions, inter-vehicular communication and multi-agent training. We present six case studies to demonstrate the features of MADRaS:

- Generalization across tracks with higher level actions
- Generalization across vehicular dynamics through random car selection
- Curriculum learning for driving in `Spring` track of MADRaS
- Learning under partial observability and stochastic outcomes of actions
- Learning to drive in traffic
- Avoiding traffic obstruction through multi-agent cooperation.

The *second* contribution of this thesis is RAIL, a Risk-Averse Imitation Learning algorithm that aims to increase the reliability of Imitation Learning for risk-sensitive applications like autonomous driving. RAIL is based on Generative Adversarial Imitation Learning (GAIL), a state-of-the-art imitation learning algorithm. We show that RAIL is a superior choice than GAIL for imitation learning in risk-sensitive applications. We also demonstrate the following:

- The applicability of RAIL is not limited to environments in which the distribution of trajectory-cost is heavy-tailed for GAIL.
- RAIL converges at least as fast as GAIL.
- RAIL preserves the scalability of GAIL while showing lower tail-risk.

The *third* contribution of this thesis is ExTra, a framework for Transfer-guided Exploration, that uses prior knowledge from related tasks to guide exploration in a new task-environment. ExTra is based on the theory of transfer learning using bisimulation metrics [34]. Efficient exploration is crucial for success of reinforcement learning in real-world environments of high-variance such as those faced in autonomous driving. We evaluate the viability of ExTra through empirical analysis of its performance on stochastic grid-world environments and arrive at the following conclusions:

1. Given the optimal policy in a related task-environment (even if the MDP is different) ExTra can achieve faster convergence and hence superior sample efficiency than domain-specific exploration algorithms like ϵ-greedy, MBIE-EB [148], Pursuit [13] and Softmax [13].
2. ExTra is able to leverage knowledge about the transition dynamics of the source MDP even when the reward structures and goal distributions are different. ExTra is more sensitive to the reward structure of the source MDP than transition dynamics or goal distribution. Also, source MDPs with the same action space as the target are more preferable even if the state spaces are different.

3. ExTra achieves gains in performance of traditional exploration algorithms like ϵ-greedy, MBIE-EB [148], Pursuit [13] and Softmax [13] when used in conjunction. This proves its viability as a complementary exploration method for accelerating the rate of convergence of traditional RL algorithms.

4. We observe that bisimulation transfer [34] can be both effective and detrimental depending on how the source and target tasks are related. Unlike bisimulation transfer, ExTra does not affect the learning process negatively even when the source and target environments are drastically different. Since bisimulation distances are larger for dissimilar environments, π_{ExTra} tends to a uniform distribution over target actions. As a result, ExTra falls back to vanilla $\epsilon-$greedy Q-learning with uniform sampling.

The computational complexity involved in the estimation of the bisimulation distance function is a major hindrance to the application of bisimulation based transfer algorithms in large environments and continuous state-action spaces. Scalable methods of computing the bisimulation metric between large task-environments in the absence of complete knowledge about their reward functions and transition dynamics is a topic of active research [160]. When such methods become available, the framework of ExTra and its guarantees can potentially be used in autonomous driving for transfer of knowledge gained in simulation to the real world.

6.2 Future Scopes

The work presented in this thesis can serve as a starting point for the following lines of research:

- Development of multi-agent driving simulators for planning in urban environments.
- Development of scalable methods for provably safe reinforcement learning.
- Development of scalable algorithms for bisimulation-based transfer of policies between complex environments.

The primary goal of autonomous driving is to make life safer on this planet. However, achieving this goal is not just a technological challenge. Making fully autonomous driving mainstream would require a complete overhaul of our existing transport infrastructures [161]. This poses an immense socio-economic challenge that may take several decades to achieve. However, autonomous sensing and decision making technologies can be incorporated into existing Advanced Driver-Assistance Systems (ADAS) right away and can be made affordable and ubiquitous. This can save millions of lives without having to hand over full control to the robots.

References

[1] "Health and sustainable development," *World Health Organization*, 2020 (accessed February 11, 2020). [Online]. Available: https://www.who.int/sustainable-development/transport/health-risks/climate-impacts/en/

[2] D. D. Zhang, H. F. Lee, C. Wang, B. Li, Q. Pei, J. Zhang, and Y. An, "The causality analysis of climate change and large-scale human crisis," *Proceedings of the National Academy of Sciences*, vol. 108, no. 42, pp. 17 296–17 301, 2011.

[3] "Road traffic injuries," *World Health Organization*, 2020 (accessed February 11, 2020). [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries

[4] "Annual global road crash statistics," *Association for Safe International Road Travel*, 2020 (accessed February 11, 2020). [Online]. Available: https://www.asirt.org/safe-travel/road-safety-facts/

[5] R. Pal, A. Ghosh, R. Kumar, S. Galwankar, S. K. Paul, S. Pal, D. Sinha, A. Jaiswal, L. R. Moscote-Salazar, and A. Agrawal, "Public health crisis of road traffic accidents in india: Risk factor assessment and recommendations on prevention on the behalf of the academy of family physicians of india," *Journal of family medicine and primary care*, vol. 8, no. 3, p. 775, 2019.

[6] "Automated driving systems 2.0: A vision for safety," *National Highway Traffic Safety Administration, US Department of Transportation, Washington, DC (DOT HS)*, vol. 812, p. 442, 2017.

[7] L. Fulton, J. Mason, and I. D. Meroux, "Three revolutions in urban transportation," *UC Davis Institute for Transportation & Development Policy*, 2017.

[8] M. Hutson, *Watch just a few self-driving cars stop traffic jams*, 2018 (accessed February 11, 2020). [Online]. Available: https://www.sciencemag.org/news/2018/11/watch-just-few-self-driving-cars-stop-traffic-jams

[9] S. Behere and M. Törngren, "A functional reference architecture for autonomous driving," *Information and Software Technology*, vol. 73, pp. 136–150, 2016.

[10] M. Dikmen and C. M. Burns, "Autonomous driving in the real world: Experiences with tesla autopilot and summon," in *Proceedings of the 8th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*. ACM, 2016, pp. 225–228.

[11] G. Minster, S. Haghighat, K. Chu, and K. Vogt, "System and method for autonomous vehicle driving behavior modification," Jul. 31 2018, uS Patent 10,035,519.

[12] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[14] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

[15] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv preprint arXiv:1610.03295*, 2016.

[16] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," *arXiv preprint arXiv:1704.07911*, 2017.

[17] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," *arXiv preprint arXiv:1711.03938*, 2017.

[18] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*. Springer, 2018, pp. 621–635.

[19] "Deepdrive 2.0," https://deepdrive.io, accessed: 2018-11-03.

[20] "Udacity's self-driving car simulator," https://github.com/udacity/self-driving-car-sim, accessed: 2018-01-15.

[21] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[22] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

[23] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[25] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," *arXiv preprint arXiv:1606.06565*, 2016.

[26] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469 – 483, 2009.

[27] S. Schaal, "Learning from demonstration," in *Advances in Neural Information Processing Systems*, 1997, pp. 1040–1046.

[28] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," *Proceedings of the 22nd International Conference on Machine learning*, 2004.

[29] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 4565–4573.

[30] S. M. Kakade, "On the sample complexity of reinforcement learning," Ph.D. dissertation, University of London London, England, 2003.

[31] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, "Torcs, the open racing car simulator," *Software available at http://torcs.sourceforge.net*, vol. 4, no. 6, 2000.

[32] R. T. Rockafellar and S. Uryasev, "Optimization of conditional value-at-risk," *Journal of risk*, vol. 2, pp. 21–42, 2000.

[33] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[34] P. S. Castro and D. Precup, "Using bisimulation for policy transfer in mdps," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[35] L. Bu, R. Babu, B. De Schutter *et al.*, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.

[36] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *arXiv preprint arXiv:1911.10635*, 2019.

[37] M. T. Heath, *Scientific computing: an introductory survey*. SIAM, 2018, vol. 80.

[38] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 157–163.

[39] L. S. Shapley, "Stochastic games," *Proceedings of the national academy of sciences*, vol. 39, no. 10, pp. 1095–1100, 1953.

[40] T. Basar and G. J. Olsder, *Dynamic noncooperative game theory*. SIAM, 1999, vol. 23.

[41] J. Filar and K. Vrieze, *Competitive Markov decision processes*. Springer Science & Business Media, 2012.

[42] Y. Shoham, R. Powers, and T. Grenager, "Multi-agent reinforcement learning: a critical survey," *Web manuscript*, 2003. [Online]. Available: http://jmvidal.cse.sc.edu/library/shoham03a.pdf

[43] P. Hernandez-Leal, M. Kaisers, T. Baarslag, and E. M. de Cote, "A survey of learning in multiagent environments: Dealing with non-stationarity," *arXiv preprint arXiv:1707.09183*, 2017.

[44] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 33, no. 6, pp. 750–797, 2019.

[45] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth International Conference on Machine Learning*, 1993, pp. 330–337.

[46] A. D. Laud, "Theory and application of reward shaping in reinforcement learning," Tech. Rep., 2004.

[47] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *ICML*, vol. 97, 1997, pp. 12–20.

[48] P. Abbeel and A. Y. Ng, "Inverse reinforcement learning," in *Encyclopedia of machine learning*. Springer, 2011, pp. 554–558.

[49] A. Y. Ng, S. J. Russell *et al.*, "Algorithms for inverse reinforcement learning." in *Icml*, 2000, pp. 663–670.

[50] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems*, 1989, pp. 305–313.

[51] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[52] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," *arXiv preprint arXiv:1704.07911*, 2017.

[53] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 661–668.

[54] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 627–635.

[55] H. Daumé, J. Langford, and D. Marcu, "Search-based structured prediction," *Machine learning*, vol. 75, no. 3, pp. 297–325, 2009.

[56] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 1171–1179.

[57] K. Lange, *MM optimization algorithms*. SIAM, 2016, vol. 147.

[58] S. Russell, "Learning agents for uncertain environments," in *Proceedings of the eleventh annual conference on Computational Learning Theory*. ACM, 1998, pp. 101–103.

[59] B. D. Ziebart, "Modeling purposeful adaptive behavior with the principle of maximum causal entropy," Ph.D. dissertation, 2010.

[60] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning." in *AAAI*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[61] A. R. Fayjie, S. Hossain, D. Oualid, and D. Lee, "Driverless car: Autonomous driving using deep reinforcement learning in urban environment," in *2018 15th International Conference on Ubiquitous Robots (UR)*, 2018, pp. 896–901.

[62] V. Talpaert., I. Sobh., B. R. Kiran., P. Mannion., S. Yogamani., A. El-Sallab., and P. Perez., "Exploring applications of deep reinforcement learning for real-world autonomous driving systems," in *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 5: VISAPP,*, INSTICC. SciTePress, 2019, pp. 564–572.

[63] S. Shalev-Shwartz and A. Shashua, "On the sample complexity of end-to-end training vs. semantic abstraction training," *arXiv preprint arXiv:1604.06915*, 2016.

[64] A. Santara, A. Naik, B. Ravindran, D. Das, D. Mudigere, S. Avancha, and B. Kaul, "Rail: Risk-averse imitation learning," *arXiv preprint arXiv:1707.06658*, 2017.

[65] C. M. Kozierok, *The TCP/IP guide: a comprehensive, illustrated Internet protocols reference*. No Starch Press, 2005.

[66] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, "Rllib: Abstractions for distributed reinforcement learning," *arXiv preprint arXiv:1712.09381*, 2017.

[67] A. Stooke and P. Abbeel, "Rlpyt: A research code base for deep reinforcement learning in pytorch," *arXiv preprint arXiv:1909.01500*, 2019.

[68] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," 2017.

[69] I. Caspi, G. Leibovich, G. Novik, and S. Endrawis, "Reinforcement learning coach," Dec. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.1134899

[70] T. Sulkowski, P. Bugiel, and J. Izydorczyk, "In search of the ultimate autonomous driving simulator," in *2018 International Conference on Signals and Electronic Systems (ICSES)*. IEEE, 2018, pp. 252–256.

[71] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems*, 1989, pp. 305–313.

[72] "Grand theft auto v," https://www.rockstargames.com/V/, accessed: 2018-11-03.

[73] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2015, pp. 2722–2730.

[74] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *European Conference on Computer Vision*. Springer, 2016, pp. 102–118.

[75] S. R. Richter, Z. Hayder, and V. Koltun, "Playing for benchmarks," in *International Conference on Computer Vision (ICCV)*, vol. 2, 2017.

[76] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3234–3243.

[77] K. Dresner and P. Stone, "A multiagent approach to autonomous intersection management," *Journal of Artificial Intelligence Research*, vol. 31, pp. 591–656, 2008.

[78] J. Neider, T. Davis, and M. Woo, *OpenGL programming guide*. Addison-Wesley Reading, MA, 1993, vol. 14.

[79] Y. Li, J. Song, and S. Ermon, "Infogail: Interpretable imitation learning from visual demonstrations," in *Advances in Neural Information Processing Systems*, 2017, pp. 3812–3822.

[80] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[81] D. Loiacono, A. Prete, P. L. Lanzi, and L. Cardamone, "Learning to overtake in torcs using simple reinforcement learning," in *IEEE Congress on Evolutionary Computation*. IEEE, 2010, pp. 1–8.

[82] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez, "Evolving large-scale neural networks for vision-based reinforcement learning," in *Proceedings of the 15th annual Conference on Genetic and Evolutionary Computation*. ACM, 2013, pp. 1061–1068.

[83] J. Koutník, J. Schmidhuber, and F. Gomez, "Evolving deep unsupervised convolutional networks for vision-based reinforcement learning," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. ACM, 2014, pp. 541–548.

[84] D. Loiacono, P. L. Lanzi, J. Togelius, E. Onieva, D. A. Pelta, M. V. Butz, T. D. Lonneker, L. Cardamone, D. Perez, Y. Sáez *et al.*, "The 2009 simulated car racing championship," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 2, pp. 131–147, 2010.

[85] D. Loiacono, L. Cardamone, and P. L. Lanzi, "Simulated car racing championship: Competition software manual," *arXiv preprint arXiv:1304.1672*, 2013.

[86] M. Kaushik, V. Prasad, K. M. Krishna, and B. Ravindran, "Overtaking maneuvers in simulated highway driving using deep reinforcement learning," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1885–1890.

[87] N. Yoshida, "Gym-torcs," *Software available at https://github. com/ugonama-kun/gym torcs*, 2016.

[88] G.-H. Liu, A. Siravuru, S. Prabhakar, M. Veloso, and G. Kantor, "Learning end-to-end multimodal sensor policies for autonomous navigation," *arXiv preprint arXiv:1705.10422*, 2017.

[89] T. de Bruin, J. Kober, K. Tuyls, and R. Babuška, "Integrating state representation learning into deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1394–1401, 2018.

[90] R. F. J. Dossa, X. Lian, H. Nomoto, T. Matsubara, and K. Uehara, "A human-like agent based on a hybrid of reinforcement and imitation learning," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.

[91] M. Bowling and M. Veloso, "An analysis of stochastic game theory for multiagent reinforcement learning," Carnegie-Mellon Univ Pittsburgh Pa School of Computer Science, Tech. Rep., 2000.

[92] F. L. Da Silva and A. H. R. Costa, "A survey on transfer learning for multiagent reinforcement learning systems," *Journal of Artificial Intelligence Research*, vol. 64, pp. 645–703, 2019.

[93] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," *arXiv preprint arXiv:1812.03079*, 2018.

[94] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Neural Information Processing Systems (NIPS)*, 2017.

[95] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual International Conference on Machine Learning*. ACM, 2009, pp. 41–48.

[96] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[97] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[98] "Investopedia article on tail risk," http://www.investopedia.com/terms/t/tailrisk.asp, accessed: 2017-06-27.

[99] N. Dalleh, "Why is cvar superior to var?" Ph.D. dissertation.

[100] A. J. Nagengast, D. A. Braun, and D. M. Wolpert, "Risk-sensitive optimal feedback control accounts for sensorimotor behavior under uncertainty," *PLoS computational biology*, vol. 6, no. 7, p. e1000857, 2010.

[101] Y. Niv, J. A. Edlund, P. Dayan, and J. P. O'Doherty, "Neural prediction errors reveal a risk-sensitive reinforcement-learning process in the human brain," *Journal of Neuroscience*, vol. 32, no. 2, pp. 551–562, 2012.

[102] A. Ruszczyński, "Risk-averse dynamic programming for markov decision processes," *Mathematical programming*, vol. 125, no. 2, pp. 235–261, 2010.

[103] R. A. Howard and J. E. Matheson, "Risk-sensitive markov decision processes," *Management science*, vol. 18, no. 7, pp. 356–369, 1972.

[104] V. S. Borkar, "Q-learning for risk-sensitive control," *Mathematics of operations research*, vol. 27, no. 2, pp. 294–311, 2002.

[105] M. Heger, "Consideration of risk in reinforcement learning," in *Proceedings of the Eleventh International Conference on Machine Learning*, 1994, pp. 105–111.

[106] O. Mihatsch and R. Neuneier, "Risk-sensitive reinforcement learning," *Machine learning*, vol. 49, no. 2-3, pp. 267–290, 2002.

[107] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *arXiv preprint arXiv:1708.06374*, 2017.

[108] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in *Advances in neural information processing systems*, 2007, pp. 1–8.

[109] A. Rajeswaran, S. Ghotra, S. Levine, and B. Ravindran, "Epopt: Learning robust neural network policies using model ensembles," *5th International Conference on Learning Representations*, 2016.

[110] J. Garcia and F. Fernandez, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, pp. 1437–1480, 2015.

[111] A. Majumdar, S. Singh, A. Mandlekar, and M. Pavone, "Risk-sensitive inverse reinforcement learning via coherent risk models."

[112] P. W. Glimcher and E. Fehr, *Neuroeconomics: Decision making and the brain.* Academic Press, 2013.

[113] Y. Shen, M. J. Tobia, T. Sommer, and K. Obermayer, "Risk-sensitive reinforcement learning," *Neural computation*, vol. 26, no. 7, pp. 1298–1328, 2014.

[114] M. Hsu, M. Bhatt, R. Adolphs, D. Tranel, and C. F. Camerer, "Neural systems responding to degrees of uncertainty in human decision-making," *Science*, vol. 310, no. 5754, pp. 1680–1683, 2005.

[115] J. B. Diederik Kingma, "Adam: A method for stochastic optimization," *arXiv:1310.5107 [cs.CV]*, 2015.

[116] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

[117] Y. Chow and M. Ghavamzadeh, "Algorithms for cvar optimization in mdps," in *Advances in neural information processing systems*, 2014, pp. 3509–3517.

[118] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5026–5033.

[119] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[120] "Openai imitation learning github repository," https://github.com/openai/imitation.git, accessed: 2017-06-27.

[121] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, ser. A Bradford book. Bradford Book, 1998. [Online]. Available: https://books.google.co.in/books?id=CAFR6IBF4xYC

[122] A. D. Tijsma, M. M. Drugan, and M. A. Wiering, "Comparing exploration strategies for q-learning in random stochastic mazes," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2016, pp. 1–8.

[123] A. L. Strehl and M. L. Littman, "An analysis of model-based interval estimation for markov decision processes," *Journal of Computer and System Sciences*, vol. 74, no. 8, pp. 1309–1331, 2008.

[124] M. Gregor and J. Spalek, "Curiosity-driven exploration in reinforcement learning," in *2014 ELEKTRO*. IEEE, 2014, pp. 435–440.

[125] M. Kearns and S. Singh, "Near-optimal reinforcement learning in polynomial time," *Machine learning*, vol. 49, no. 2-3, pp. 209–232, 2002.

[126] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped dqn," in *Advances in Neural Information Processing Systems*, 2016, pp. 4026–4034.

[127] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin *et al.*, "Noisy networks for exploration," *arXiv preprint arXiv:1706.10295*, 2017.

[128] P.-Y. Oudeyer and F. Kaplan, "What is intrinsic motivation? a typology of computational approaches," *Frontiers in neurorobotics*, vol. 1, p. 6, 2009.

[129] D. Abel, D. E. Hershkowitz, G. Barth-Maron, S. Brawner, K. O'Farrell, J. MacGlashan, and S. Tellex, "Goal-based action priors," in *Twenty-Fifth International Conference on Automated Planning and Scheduling*, 2015.

[130] D. Abel, Y. Jinnai, S. Y. Guo, G. Konidaris, and M. Littman, "Policy and value transfer in lifelong reinforcement learning," in *International Conference on Machine Learning*, 2018, pp. 20–29.

[131] F. Fernández and M. Veloso, "Probabilistic policy reuse in a reinforcement learning agent," in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. ACM, 2006, pp. 720–727.

[132] T. Brys, A. Harutyunyan, M. E. Taylor, and A. Nowé, "Policy transfer using reward shaping," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2015, pp. 181–188.

[133] A. Tirinzoni, M. Salvini, and M. Restelli, "Transfer of samples in policy search via multiple importance sampling," in *International Conference on Machine Learning*, 2019, pp. 6264–6274.

[134] G. Vezzani, A. Gupta, L. Natale, and P. Abbeel, "Learning latent state representation for speeding up exploration," *arXiv preprint arXiv:1905.12621*, 2019.

[135] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-reinforcement learning of structured exploration strategies," in *Advances in Neural Information Processing Systems*, 2018, pp. 5302–5311.

[136] M. E. Taylor and P. Stone, "Cross-domain transfer for reinforcement learning," in *Proceedings of the 24th International Conference on Machine learning*. ACM, 2007, pp. 879–886.

[137] G. Joshi and G. Chowdhary, "Cross-domain transfer in reinforcement learning using target apprentice," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7525–7532.

[138] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, no. Jul, pp. 1633–1685, 2009.

[139] S. B. Thrun, "Efficient exploration in reinforcement learning," Tech. Rep., 1992.

[140] M. C. Mozer and J. Bachrach, "Discovering the structure of a reactive environment by exploration," in *Advances in Neural Information Processing Systems*, 1990, pp. 439–446.

[141] S. D. Whitehead and D. H. Ballard, "Learning to perceive and act by trial and error," *Machine Learning*, vol. 7, no. 1, pp. 45–83, 1991.

[142] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Machine Learning Proceedings 1990*. Elsevier, 1990, pp. 216–224.

[143] N. Cesa-Bianchi, C. Gentile, G. Lugosi, and G. Neu, "Boltzmann exploration done right," in *Advances in Neural Information Processing Systems*, 2017, pp. 6284–6293.

[144] M. Wiering and J. Schmidhuber, "Efficient model-based exploration," in *Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, vol. 6, 1998, pp. 223–228.

[145] M. Sato, K. Abe, and H. Takeda, "Learning control of finite markov chains with an explicit trade-off between estimation and control," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, no. 5, pp. 677–684, 1988.

[146] J. Schmidhuber, "Adaptive confidence and adaptive curiosity," in *Institut fur Informatik, Technische Universitat Munchen, Arcisstr. 21, 800 Munchen 2*. Citeseer, 1991.

[147] S. B. Thrun and K. Möller, "Active exploration in dynamic environments," in *Advances in Neural Information Processing Systems*, 1992, pp. 531–538.

[148] A. L. Strehl and M. L. Littman, "A theoretical analysis of model-based interval estimation," in *Proceedings of the 22nd International Conference on Machine learning*. ACM, 2005, pp. 856–863.

[149] N. Ferns, P. Panangaden, and D. Precup, "Metrics for finite markov decision processes," in *Proceedings of the 20th conference on Uncertainty in artificial intelligence*. AUAI Press, 2004, pp. 162–169.

[150] J. Taylor, D. Precup, and P. Panagaden, "Bounding performance loss in approximate mdp homomorphisms," in *Advances in Neural Information Processing Systems*, 2009, pp. 1649–1656.

[151] R. Givan, T. Dean, and M. Greig, "Equivalence notions and model minimization in markov decision processes," *Artificial Intelligence*, vol. 147, no. 1-2, pp. 163–223, 2003.

[152] B. Ravindran and A. G. Barto, "Model minimization in hierarchical reinforcement learning," in *International Symposium on Abstraction, Reformulation, and Approximation*. Springer, 2002, pp. 196–211.

[153] B. Ravindran, "Smdp homomorphisms: An algebraic approach to abstraction in semi markov decision processes," Ph.D. dissertation, 2003.

[154] A. L. Gibbs and F. E. Su, "On choosing and bounding probability metrics," *International statistical review*, vol. 70, no. 3, pp. 419–435, 2002.

[155] T. M. Cover, *Elements of information theory*. John Wiley & Sons, 1999.

[156] W. Mayner, "Fast emd for python: a wrapper for pele and werman's c++ implementation of the earth mover's distance metric," https://github.com/wmayner/pyemd, 2018.

[157] O. Pele and M. Werman, "A linear time histogram metric for improved sift matching," in *Computer Vision–ECCV 2008*. Springer, October 2008, pp. 495–508.

[158] A. Andoni, P. Indyk, and R. Krauthgamer, "Earth mover distance over high-dimensional spaces." in *SODA*, vol. 8, 2008, pp. 343–352.

[159] O. Pele and M. Werman, "Fast and robust earth mover's distances," in *2009 IEEE 12th International Conference on Computer Vision*. IEEE, September 2009, pp. 460–467.

[160] P. S. Castro, "Scalable methods for computing state similarity in deterministic markov decision processes," *arXiv preprint arXiv:1911.09291*, 2019.

[161] T. Duvall, E. Hannon, J. Katseff, B. Safran, and T. Wallace, "A new look at autonomous-vehicle infrastructure," 2019. [Online]. Available: https://www.mckinsey.com/industries/capital-projects-and-infrastructure/our-insights/a-new-look-at-autonomous-vehicle-infrastructure

Publications by the Author

Journals

- *Anirban Santara*, Kaustubh Mani, Pranoot Hatwar, Ankit Singh, Ankur Garg, Kirti Padia, Pabitra Mitra, (2017). "BASSNet: Band-Adaptive Spectral-Spatial Feature Learning Neural Network for Hyperspectral Image Classification". In: IEEE Transactions on Geoscience and Remote Sensing 55.9, pp.5293–5301.

Conferences

- *Anirban Santara*, Rishabh Madan, Pabitra Mitra, Balaraman Ravindran. "ExTra: Transfer-guided Exploration". Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems. 2020.

- *Anirban Santara*, Abhishek Naik, Balaraman Ravindran, Dipankar Das, Dheevatsa Mudigere, Sasikanth Avancha, Bharat Kaul. "RAIL: Risk-Averse Imitation Learning". Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. 2018.

- Debapriya Maji, *Anirban Santara*, Sambuddha Ghosh, Debdoot Sheet, Pabitra Mitra. "Deep neural network and random forest hybrid architecture for learning to detect retinal vessels in fundus images". In: Engineering in Medicine and Biology Society (EMBC), 2015, 37th Annual International Conference of the IEEE. pp.3029–3032.

Submitted Papers

- *Anirban Santara*, Sohan Rudra, Sree Aditya Buridi, Meha Kaushik, Abhishek Naik, Bharat Kaul, Balaraman Ravindran. "MADRaS: Multi-Agent Driving Simulator", *communicated to* Journal of Artificial Intelligence Research, 2020

- *Anirban Santara*, Jayeeta Datta, Sourav Sarkar, Ankur Garg, Kirti Padia, Pabitra Mitra,(2019). "PUNCH: Positive UNlabelled Classification based information retrieval in Hyperspectral images.", arXiv:1904.04547

Author's Biography

Anirban Santara is a Google India PhD Fellow at the Department of Computer Science and Engineering, Indian Institute of Technology (IIT) Kharagpur. He graduated from the Department of Electronics and Electrical Communications Engineering, IIT Kharagpur in 2015.

Research Interests

Reinforcement Learning, Robotics, Deep Learning & Applications, AI Safety.

Education

- **Doctor of Philosophy (PhD)**, Computer Science and Engineering
  Indian Institute of Technology Kharagpur, India
  Dissertation title: Reinforcement Learning for Safe and Efficient Planning in Autonomous Driving
  Year: 2015–2020

- **Bachelor of Technology (B-Tech)**, in Electronics and Electrical Communication Engineering
  Year: 2011-2015
  Indian Institute of Technology Kharagpur, India
  CGPA: 9.30/10

Awards & Honours

- **Heidelberg Laureate Forum**: One of 200 students selected for participation in the 6th Heidelberg Laureate Forum (2018).
- **Indian Ambassador to Russia**: Represented the AI community of India at the XIX World Festival of Youth and Students in Sochi (2017)
- **Google India Ph.D. Fellowship**: from Google, for leadership in Machine Learning research (2016)
- **Rajendra Nath Das MCM Award**: From IIT Kharagpur, for outstanding academic performance (2014)
- **Batch of '85 Scholarship**: From IIT Kharagpur, for outstanding academic performance (2013)

Research Experience

- **Google India Ph.D. Fellow**
  Duration: Indian Institute of Technology, Kharagpur, India
  Duration: July 2015 – Ongoing
  Dissertation Topic: Reinforcement Learning for Safe and Efficient Planning in Autonomous Driving
  Advisors: Prof. Pabitra Mitra, Prof. Balaraman Ravindran
  Sponsor: Google India

- **Google Brain**
  Internship 1 Duration: Jul 2019–November 2019
  Position: Research Intern
  Internship 2 Duration: Nov 2018–Mar 2019
  Position: Software Engineering Intern
  Location: Mountain View, CA, USA

  – Developed a pipeline for data efficient learning of high-dimensional long-horizon continuous control tasks that involve a hierarchy of goals at different time scales.

- The pipeline comprises unsupervised learning of human motion primitives, supervised learning of fine grained motor control and reinforcement learning of a high-level policy.
- The pipeline achieves superior sample efficiency and human-like motion than any single learning paradigm.

- **Graduate Research Intern for Autonomous Driving**
  Location: Parallel Computing Lab – Intel Labs, Bangalore, India
  Duration: Jan 2017–Dec 2017

  - Developed RAIL, a framework for risk-averse imitation learning in autonomous agents deployed in risk-sensitive applications: `https://intel.ly/2lDyQ34`
  - Achieved upto 89 percent improvement in Conditional Value-at-Risk (a measure of tail-risk) over the previous state-of-the-art algorithm with RAIL at benchmark continuous control tasks
  - Developed MADRaS, the world's first open-source fully-customizable Multi-Agent DRiving Simulator: `https://github.com/madras-simulator`

- **Research Consultant for Deep Learning**
  Location: Indian Institute of Technology, Kharagpur, India
  Duration: 2015–2016

  - Developed a novel Deep Neural Network architecture, BASS-Net (`https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7938656`), and information retrieval framework, PUNCH (`https://arxiv.org/abs/1904.04547`) for land-cover classification in hyper-spectral images (HSI) for Indian Space Research Organization (ISRO).
  - Designed a state-of-the-art Deep Neural Network ensemble for retinal vessel segmentation (`https://ieeexplore.ieee.org/abstract/document/7319030/`, `https://arxiv.org/abs/1603.04833`) in Diabetic Retinopathy diagnosis in collaboration with Apollo Gleneagles Hospitals.

- **Undergraduate Researcher in Deep Learning**
  Location: Indian Institute of Technology, Kharagpur, India
  Duration: 2013–2015
  - Optimization of Deep Learning algorithms on multi-core CPUs.
  - Deep learning for Diabetic Retinopathy screening
- **Project Trainee**
  Location: Texas Instruments, Bangalore, India
  Duration: 2014
  - Adaptive Grayscale Level Adjustment in DLP Based 3D Scanning System for Improved Reconstruction of Object Shape.

VOLUNTEERING

- **Program Chair at up.AI Summit 2018 at IIT Kharagpur**
  Year: 2018–2019
  - Ideated and Organized IIT Kharagpur's first Artificial Intelligence Summit.
  - Event witnessed record turnover of 1000+ students and got covered by national media: `http://bit.ly/2OlBVTg`, `http://bit.ly/2ukVNfN`, `http://bit.ly/2HNfuow`
- **Intel Student Ambassador for AI at Intel AI Academy**
  Year: 2018–Ongoing
  - Wrote blogs, delivered blitz talks and tutorials on Deep Learning theory and implementation on Intel hardware and software platforms online and at several Intel AI Academy events around the globe.
  - Blog on MADRaS: A Multi Agent DRiving Simulator: `https://intel.ly/2KvB4MX`.
  - Intel AI Acedemy Spotlight Video: `https://intel.ly/2lDyQ34`.

- **Guard Commander as part of 1 Bengal EME COY NCC at IIT Kharagpur**
  Year: 2011–2012
  - Managed weekly drills and assigned duties of 300 cadets from the Electrical and Mechanical Engineering (EME) branch of the Indian Army
  - Received NCC-B certificate