MODEL-BASED IRL WITH CONTINUOUS ACTION SPACES

by

ANUJA PRADEEP NAGARE

(Under the Direction of Prashant Doshi)

Abstract

Inverse reinforcement learning (IRL) seeks to learn the preferences of an expert agent performing a task from the expert's demonstrations. More specifically, it seeks to find the reward function of the expert modelled as a Markov decision process from observations of its state-action trajectories. IRL's ability to use an expert agent's demonstrations of real-world activities, such as driving, locomotion tasks, and other robotic tasks to build intelligent agents makes IRL significant. This research provides a novel method for preference learning by developing a model-based IRL algorithm for continuous action spaces. It generalizes a previous Bayesian approach to IRL to include continuous action spaces and uses the trust region policy optimization in the method. Action space densities are generated for each state using a random walk, and an online transition model is used. Our method learns the reward function of an expert agent with a continuous action space and uses this learned function to complete the underlying MDP and predict an optimal policy. Experimental results over a benchmark problem domain called Object-World and toward modelling driver behavior on congested freeways offer evidence about the benefits of this approach.

Index words: Self-Driving Car, Machine Learning, Inverse Reinforcement Learning, Continuous Action Space, Bayesian IRL, Gaussian Process IRL

MODEL-BASED IRL WITH CONTINUOUS ACTION SPACES

by

ANUJA PRADEEP NAGARE

B.E., University of Mumbai, India, 2009

M.E., University of Mumbai, India, 2012

A Thesis Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

MASTER OF SCIENCE

Athens, Georgia

2018

MODEL-BASED IRL WITH CONTINUOUS ACTION SPACES

by

ANUJA PRADEEP NAGARE

Major Professors:	Prashant Doshi
Committee:	Khaled Rasheed
	Yi Hong

Electronic Version Approved:

Suzanne Barbour
Dean of the Graduate School
The University of Georgia
August 2018

MODEL-BASED IRL WITH CONTINUOUS ACTION SPACES

ANUJA PRADEEP NAGARE

August 6, 2018

Dedication

I dedicate this thesis to my parents Rajashree and Pradeep Nagare and my brother Pranit Nagare who have always been my pillars of strength. Also, I would like to dedicate this thesis to my mentor Dr. Prashant Doshi for always believing in me and encouraging me to achieve my goals.

Acknowledgments

I wish to express my sincere thanks and deep sense of gratitude to respected mentor and guide Dr. Prashant Doshi, Director, Thinc Lab, University of Georgia, for his in-depth and enlightening guidance. Apart from technical advice, I have got lots of encouragement and constructive suggestions, which motivated me to strive harder for excellence.

I would like to thank Tomoki Nishi for his insightful feedbacks and Toyota Research Institute – North America, Ann Arbor, Michigan for funding and collaborating on this project. Also, I am grateful to Dr. Yecheng Hunag and Suchitra Pakala for their guidance on using Sapelo2 cluster at the University of Georgia.

I would like to pay gratitude to the committee: Dr. Khaled Rasheed and Dr. Yi Hong for their time and effort they put to evaluate my work.

Also, I would like to thank my lab-mates Dr. Muthukumaran Chandrasekaran, Dr. Kenneth Bogert, Roi Ceren, Shibo Li, Keyang He, Saurabh Arora, Vinamra Jain, Sanath Bhat, Indrajit Das, Maulesh Trivedi, Shervin Shahryari, Sina Solaimanpour, Adithya Raam Sankar, Nihal Soans and Maulik Shah for providing valuable insights, comments, debates and discussions throughout my tenure at the Thinc Lab.

I would like to thank Muthukumaran Chandrasekaran, Garrison Bickerstaff, and Arun Ku Chockalingam santha kumar for helping me proofread my thesis document.

Finally, I would like to thank my friends Farah Pathan, Surbhi Wagle, Rakesh Bhavsar and Ayush Jaiswal for always being there for me and encouraging me.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Behavior and preferences of an agent who is interacting with real world are typically unknown apriori. Therefore, learning them is the key to predicting the future behavioral patterns and preferences of this agent. Machine Learning(ML) provides variety of methods to model uncertainty in the environment. A pragmatic technique could be Reinforcement Learning (RL) or Inverse Reinforcement Learning (IRL) which enables learning agent to specify an expert agent's behavior in it's environment with the help of goal accomplishment instead of hand-coding specific steps to achieve goal [24].

IRL assumes that the world is modelled as a Markov Decision Process (MDP). MDP is a tuple <States, Actions, Transitions, Discount Factor, Rewards >. MDP models decision making in a stochastic environment, and the core problem is to find an optimal policy (i.e. mapping of state's to their corresponding action's) for the decision maker.

IRL's goal is to learn reward function **r** (preference function) of an expert agent under which the policy π matches the expert's demonstration. Figure 1.1 shows how the autonomous learner receives an optimal policy or trajectories of an expert agent as an input along with the prior domain knowledge (i.e. completely observable states, actions, and fully known transition probabilities) and generates a likely reward function.

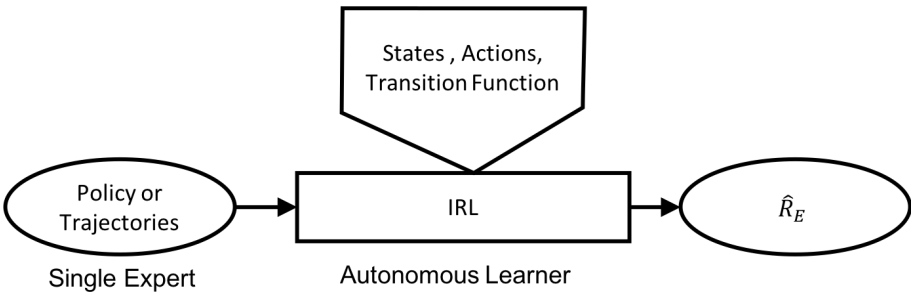


Figure 1.1: Pipeline for a classical Inverse Reinforcement Learning Process. The figure is redrawn from [4]

There are various algorithms based on the technique used for solving IRL such as Maximum Margin Optimization, Entropy Optimization, Bayesian Update, etc [4]. The Bayesian IRL algorithm is an interesting approach as it shows improvements over earlier IRL techniques, as it gives an ability to leverage machine learning literature on approximate inference, also BIRL provides better scalability than Max-margin with respect to number of states.

In the case of IRL research, continuous action spaces are less explored. This research provides a novel method for preference learning by extending Bayesian Inverse Reinforcement Learning (BIRL) for continuous action spaces. The scope of this work is to learn reward structure from an observed trajectory of the expert agent for underlying MDP to construct a policy for simplified Object-world and Highway Lane Merging environments.

1.1 Motivation

In early RL and IRL work, continuous state and action variables of the environment like position, distance, velocity, time, acceleration, etc. were discretized for an agent's learning process. However, in applications involving interaction with real world, such as autonomous driving, manoeuvring of drones or other locomotion tasks, an agent requires continuous spaces defined for better learning. Thus, continuous spaces have become an important

research track in the field of RL and IRL [2].

A clear distinction exists between the state space and the action space of a task. It is observed that continuous state spaces are easier to deal with than continuous action spaces. In RL literature considerable research effort has been devoted to the development of algorithms for continuous state as well as continuous actions spaces. Whereas, In the case of IRL research, continuous action spaces are less explored in comparison to continuous state spaces.

1.2 Problem description

Bayesian Inverse Reinforcement Learning with continuous action space is evaluated in two domains: Simplified Object-World and Highway Lane Merging environment.

Simplified Object-world is a simple toy problem that is based on grid world. It has goal states, unbiased states and hazardous states. The agent has to learn to reach goal states with an optimal number of steps. The grid has a discrete state space, and the agent can move in any direction by one unit. Direction is decided with the angle at which the agent can move (i.e. any angle between 0^o and 360^o). Thus, discrete 4 or 8 neighborhood actions are converted to continuous action space for testing purposes.

Many areas for research exist in autonomous decision making: parking in appropriate space, avoiding obstacles, breaking mechanism of a vehicle, following road symbol signs/ traffic signs, and highway lane merging among others.This research focuses on a task that is daunting even for experienced human drivers – safely merging into a congested freeway from an acceleration ramp without lengthy waits. In order to facilitate this process, an agent must account for the uncertainty both in the environment and in the driving patterns of other manned or unmanned cars.

The objective is to automatically learn the driving patterns of other cars sharing the

environment with an autonomous car in congested traffic situations with a focus on freeway lane merging from acceleration ramp using NGSim dataset [4].

For this highway lane merging problem an "A-B-C Role Model" is articulated as shown in the figure 1.2. The self-driving car (role B) which is trying to merge onto the highway from an acceleration ramp perceives the behavior of following car (role A) driving on the outermost highway lane near the merging area while comprising the preceding car's (role C) attributes in order to merge safely onto the highway.

The driving pattern of a car driving on the outermost highway lane near the merging area is called as "expert's trajectory" (i.e. Car in role A). The expert's trajectory can be thought of as a series of state-action pairs in an MDP.

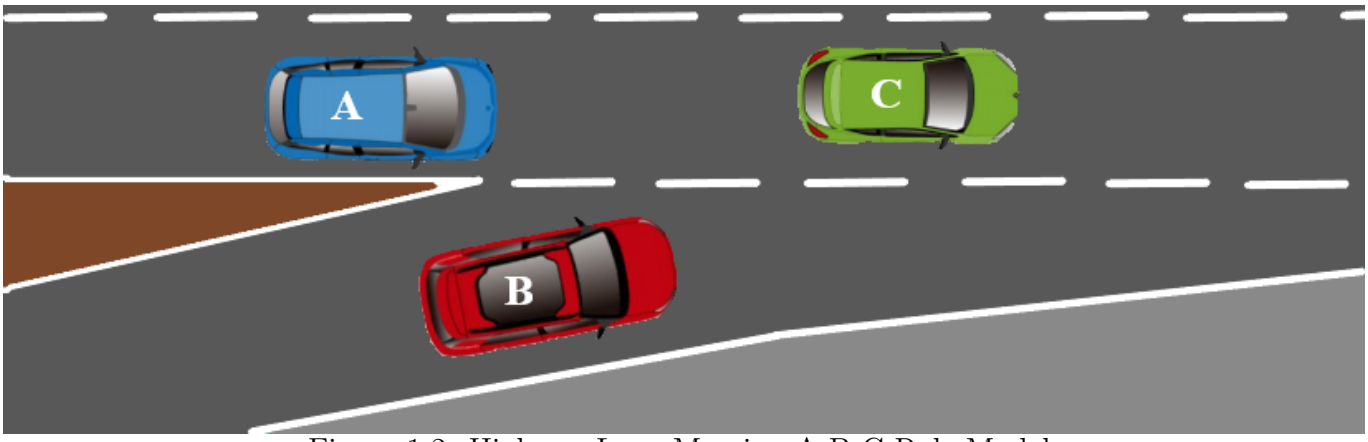


Figure 1.2: Highway Lane Merging A-B-C Role Model.

1.3 Contribution

This research provides a novel method for preference learning by extending Bayesian IRL (BIRL) for continuous action spaces. The scope of this work is to learn reward structure from an observed trajectory of the expert agent for underlying MDP to construct a policy for simplified object-world and highway lane merging environments.

For the highway lane merging environment driving patterns are extracted from a real-

world traffic data of a congested freeway at California (NGSim dataset) for each car. Every instance of a driving pattern has various attributes, such as x and y coordinates, velocity, acceleration, etc. "A-B-C Role Model" is isolated for each car that merges onto the freeway.

Further this data is filtered to isolate instances in which the car in role A slows down, maintains its speed, or speeds up to prevent B from merging in front of it. These discovered instances from the data set provide data for learning and testing corresponding aggressive driver models. Data and domain set-up of the simplified object-world and highway lane merging environment is explained in detail in chapter 5.

The scope of this work is to enable the self-driving car (i.e. car in role B) to learn and predict an acceleration control action that corresponds to an observed trajectory of the car in the role A by explicitly learning its reward structure for the underlying MDP to construct a policy π. IRL model is trained to learn risk-prone driver behavior(preference) from observed risk-prone driving demonstrations.

1.4 Structure of the thesis

The organizations of this document is as follows: Chapter 2 introduces the Markov decision process, Value iteration, Policy optimization techniques, Reinforcement learning, Trust region policy optimization and Inverse Reinforcement learning. Chapter 3 presents critical review of Reinforcement and Inverse Reinforcement learning algorithms based on continuous spaces, Also, it describes Bayesian IRL. Chapter 4 explains the devised continuous action space Inverse Reinforcement learning algorithms and implementation details behind different modules. Chapter 5 describes the data and domain setup of the simplified Object-world environment and Highway lane merging environment. Chapter 6 presents the results and evaluation. Chapter 7 discusses the conclusion and future work.

1.5 Summary

In this chapter, the focus was to give a brief idea of the motivation behind selecting model-based IRL with continuous action spaces. Next, a brief problem description is given and then the contributions are stated, which is followed by the basic outline of the rest of the thesis.

Chapter 2

BACKGROUND

Reinforcement Learning (RL) and Inverse Reinforcement Learning (IRL), assume that the world is modelled as a Markov Decision Process (MDP). First, this chapter explains the Markov Decision Process and its components. Next, expected utility of a policy is explained, which is followed by explanation of Reinforcement Learning (RL), Trust Region Policy Optimization(TRPO) and Inverse Reinforcement Learning (IRL).

2.1 Markov Decision Process (MDP)

Markov Decision Process (MDP) is a planning problem that models decision making in a stochastic environment for sequential decision making in which an action taken in the current state can affect the future states. Sequential decision making needs to maximize the utility of an agent's former actions. The MDP framework provides a well-established mathematical formulation for discrete time stochastic processes. The core problem in MDP is to find an optimal policy for the decision-making agent [24, 26, 21].

Policy gives mapping for each State 's' in the environment to it's corresponding Action 'a' and is represented with π. An Optimal policy π^* is the policy that maximizes agents'

expected utility over all other policies.

MDP is represented with a tuple: $< S, A, P, \gamma, R >$. Where,

- S is State space,
- A is Action space,
- P(s' | s,a) is Transition probability, which gives the probability distribution over the next state s' conditioned on an agent executing an action 'a' when its current state is 's'.
- γ is a Discount factor which conveys the difference in the importance of current rewards and future rewards and
- R is a Reward function that shows reward (preference function of an agent) for being in a state R(S), or a reward received for taking an action 'a' in a state 's' R(S, A).

2.1.1 Policy

Policy provides a strategy, which is a decision rule that encodes action contingencies over the state. Policy has an expected value for each state, based on the given reward criterion. There are various types of policies as shown in figure 2.1.

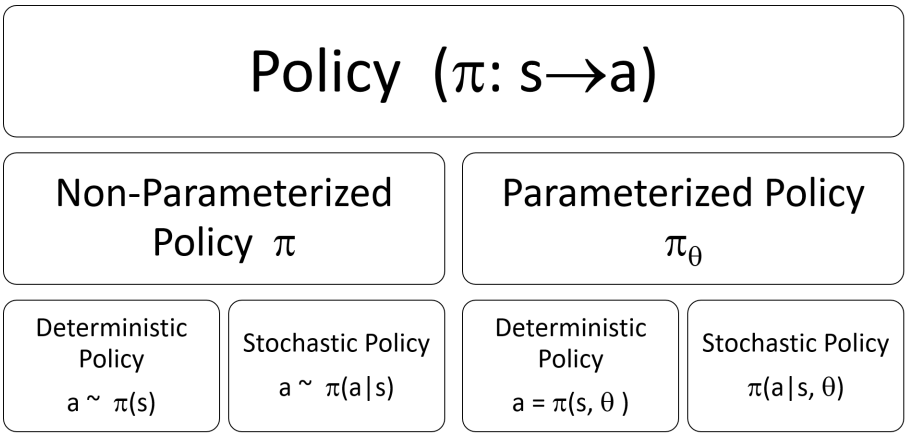


Figure 2.1: Types of Policies.

There are two main types of policy: 1. Non-parametric policy, and 2. Parametric policy. Each of these two types of policy are further divided into deterministic policy and stochastic policy.

Deterministic policy ($a = \pi(s)$) is mapping from state to corresponding action. When, there are a discrete number of states and actions in the environment, deterministic policy can be computed for an agent.

In real world control problems, actions are continuous and can be bounded by physical constraint which introduces bias. As, real world environments are stochastic (eg. autonomous driving) and involve uncertainties, here a non-deterministic policy is useful for an agent.

A Stochastic Policy ($a \sim \pi(a|s)$) selects an action according to a learned probability distribution for a given state. For example, the game of rock-paper-scissors, in which the probability of picking rock, paper, or scissors is equal in every round.

When the values of the parameters for a given state are used to compute optimal policy, it is called parameterized policy. These policies are useful for parameterized tasks where the parameters are fixed for an individual episode with a goal of learning task-dependant policy (eg. table tennis, dart game, etc. where starting positions and target positions can be different at every episode)[15]. A Gaussian Policy can be an example of a Parameterized Policy (π_θ), which can be represented as ($a \sim \mathcal{N}(\mu(s), \sigma)$), where μ is mean (linear combination of state features) and σ is a standard deviation (fixed or parameterized).

2.2 Expected Utility of a Policy

In order to evaluate a policy its expected utility is computed. Utility is a function which maps states to a real number and can be defined in terms of $V^\pi(s)$, which gives the expected value of following π in state s or $Q^\pi(s, a)$, which gives the expected value of taking action

a in state s and then following policy π [22, 24]. Equations 2.1 and 2.2 show the Bellman equations which satisfy for $V^{\pi}(s)$ and $Q^{\pi}(s,a)$, for all s $\in$ S, a $\in$ A.

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') V^{\pi}(s') \tag{2.1}$$

$$Q^{\pi}(s,a) = R(s) + \gamma \sum_{s'} T(s, a, s') V^{\pi}(s') \tag{2.2}$$

Policy π is optimal when there is no policy π' that has a greater or equal expected value at every state. (i.e. policy is optimal for the MDP iff $\forall s \in S$), see equation 2.3.

$$\pi(s) \in \underset{a \in A}{\operatorname{argmax}}\, Q^{\pi}(s,a) \tag{2.3}$$

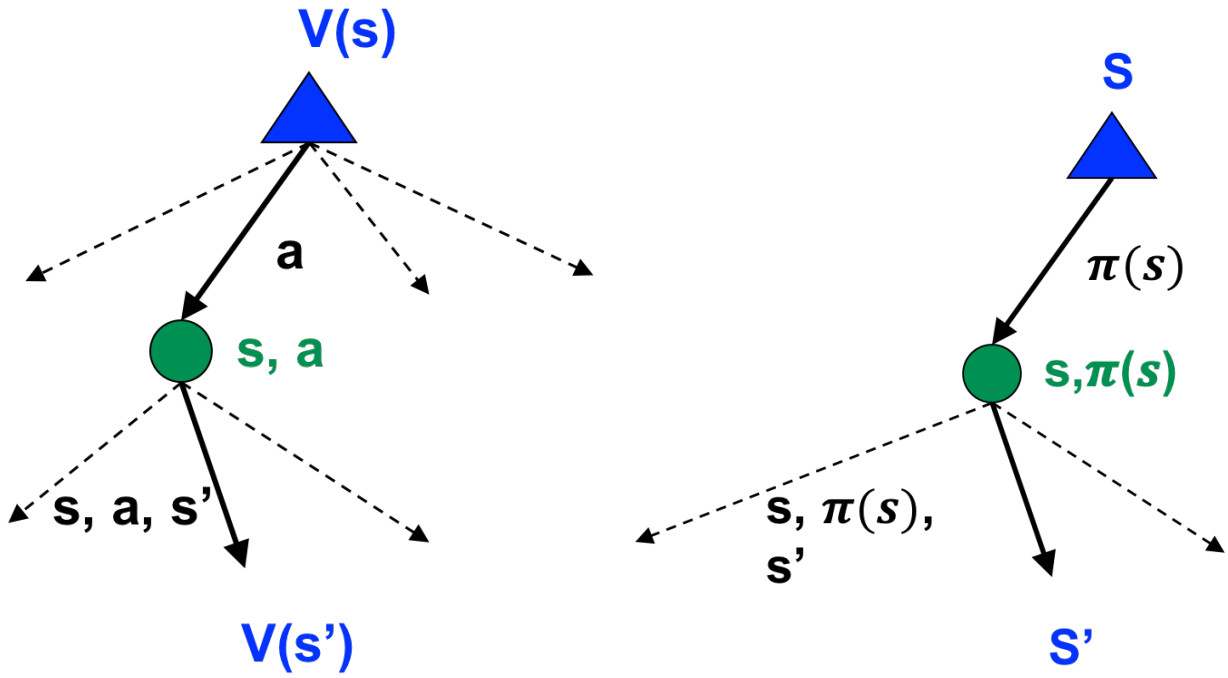
2.3 Policy Optimization

For solving MDP and obtaining optimal policy, there are two types of dynamic programming algorithms (see figure 2.2.) namely: 1. Value Iteration and 2. Policy Iteration.

2.3.1 Value Iteration

Value Iteration (VI) computes an optimal MDP policy and its value. It follows a bottom-up approach by initially generating an arbitrary value function V_0, and then it performs a Bellman update to compute the value function for all states until n+1 iterations, or until value function convergence is achieved.

Value iteration is a simple method but, is computationally heavy as it takes many iterations before convergence. Also, it is observed that policy often converges long before the values converge [24].



(a) Value Iteration.

(b) Policy Iteration.

Figure 2.2: Dynamic Programs for solving MDP. The figure is redrawn from Klein, et al. 2018 [14].

Algorithm 1: VALUE-ITERATION algorithm for calculating utilities of states

function VALUE-ITERATION(mdp, ϵ)**return** *utility function*
Input: *mdp*, an MDP with states S, transition model T, reward function R, discount γ, ϵ, the maximum error allowed in the utility of any state,
local variables: U, U', vectors of utilities for states in S, initially zero
σ, the maximum change in the utility of any state in an iteration
Output: Utility Function U

repeat
  U $\leftarrow U'; \delta \leftarrow 0$
  for each state s in S do
    $U'[s] \leftarrow R[s] + \gamma \max_a \sum_{s'} T(s,a,s')U[s']$
    if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
until $\sigma < \epsilon(1-\gamma)/\gamma$
return U

2.3.2 Policy Iteration

Policy Iteration (PI) starts with an arbitrary policy, evaluates that policy (π), and then iteratively improves on that policy until policy converges. Thus PI involves two major steps namely: 1. Policy Evaluation and 2. Policy Improvement.

Policy iteration can be computationally cheap as it can converge in a lower number of iterations compared to value iteration as PI terminates when the policy improvement step yields no change in the utilities. Also note that, PI requires more time on each iteration [24].

Policy-Iteration algorithm for calculating an optimal policy is as follows:

Algorithm 2: POLICY-ITERATION algorithm for calculating an optimal policy

function POLICY-ITERATION(mdp) **return** *a policy* **Input:** *mdp*, an MDP with states S, transition model T
local variables: U, a vector of utilities for states in S, initially zero
π, a policy vector indexed by state, initially random
Output: policy π

```
repeat
    U ← POLICY-EVALUATION (∧, U, mdp)
    unchanged? ← true
    for each STATE s IN S do
        if max, Σ_s' T(s,a,s')U[s'] > Σ_s' T(s,π[s],s')U[s'] then
        π[s] ← arg max_a Σ_s' T(s,a,s')U[s']
        unchanged? ← false
until unchanged?
return π
```

Apart from VI and PI there are two more categories of algorithms for policy optimization in RL: Policy Gradient and Derivative-free optimization (like Cross Entropy Method (CEM) and Co-variance matrix adaptation (CMA)) [8].

2.4 Reinforcement Learning

RL is primarily concerned with how to obtain an optimal policy when a perfect model is not available. The lack of a model generates a need to sample the MDP to gather statistical knowledge about this unknown model. Many model-free RL techniques exist, and they probe the environment by executing actions and there by estimate the same kind of state value and state-action value functions as model-based techniques. Thus, RL gives MDP's a focus on approximation and incomplete information and the need for sampling and exploration.

RL is defined by characterizing a learning problem where the learning agent is interacting with its environment in episodes and attempts to pick actions over time to maximize the expected rewards which are obtained long term and finally learns a high-quality policy in the environment [26]. Figure 2.3 shows RL framework, where S_t represents the current state, S_{t+1} represents the next state, R_t represents the current reward, R_{t+1} represents the next reward, and A_t represents an action executed by the learning agent.

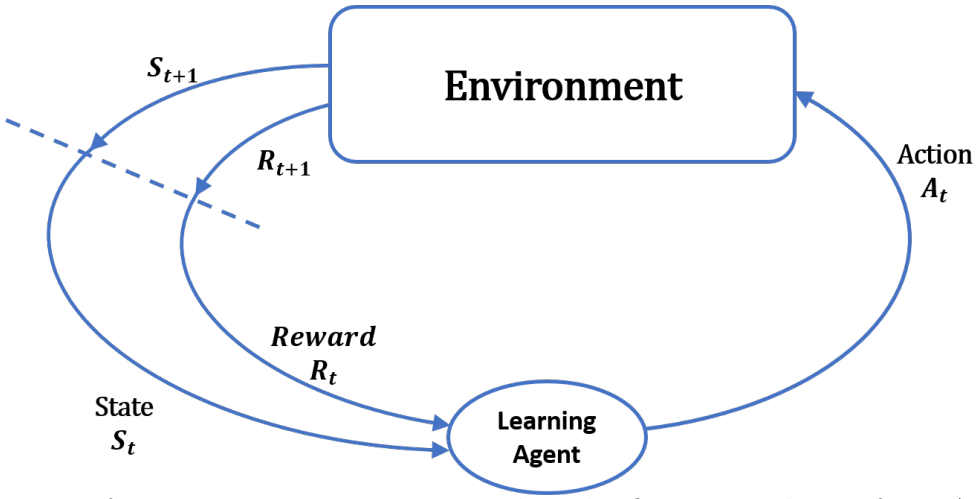


Figure 2.3: Reinforcement Learning Framework. The figure is redrawn from Arora, et al. 2018 [4]

RL algorithms are of two types : 1. Action value function based (like Q-learning and SARSA). 2. Policy Gradient based (like Actor-Critic, Trust Region Policy Optimization,

Proximal Policy Optimization).

Policy gradient method has a close connection to policy iteration method. Policy gradient method uses an estimator of gradient for the expected total reward which is obtained from generated sample demonstrations by running optimized parameterized policy. Policy Gradient methods are advantageous as they can deal with continuous states and actions with an increased learning performance and guaranteed convergence to at least local optimum. Policy Gradient methods need fewer parameters for the learning process compared to the value function based method. As Policy Gradient methods have generic formalization, they can be used in both model-free and model-based settings. Finite Difference and likelihood ratio (REINFORCE) are the most well known estimation methods of policy gradient. These methods can be disadvantageous in case of model based systems where the likelihood ratio gradient needs to maintain system model. It is difficult to maintain system model for continuous states and actions. These methods are useful when small change $\Delta\theta$ to the policy π_θ is required while improving policy, but small change is a vague term [20].

In the Natural Policy Gradient method, the natural gradient is defined by Amari (1998), as an update $\Delta\widetilde{\theta}$ that is most similar to the true gradient ∇_θ while the change in path distribution is limited to ϵ [12, 2].

2.5 Trust Region Policy Optimization

Trust Region (TR) represents a subset of the region of the objective function, which is approximated using a model function[8]. TR method finds direction and step size with the help of a quadratic model of the objective function.

At each iteration TR first defines a region within which the model is trusted to be an adequate representation of the objective function, and step is selected which tries to minimize the model in this defined trust region. Thus, the direction and length of the step is selected

simultaneously. In case a step is not acceptable, the size of region is reduced, and a new minimizer is found. Whenever the size of the trust region is modified, step direction changes. TR method is also known as Restricted-Step Method.

Figure 2.4 shows an illustration of TR method in finding direction and step size with the help of a quadratic model of the objective function.

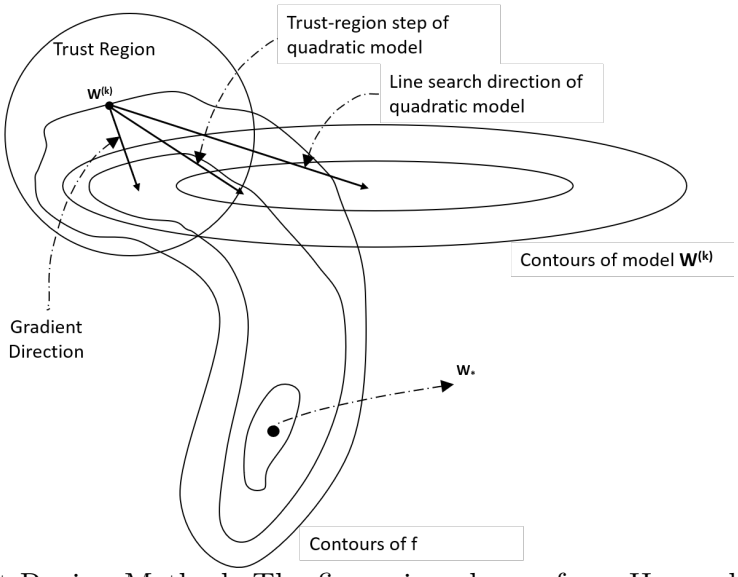


Figure 2.4: Trust Region Method. The figure is redrawn from Heeyoul et al. 2004 [7]

Trust Region Policy Optimization (TRPO) algorithm is similar to natural policy gradient methods and provides a principled approach to control the rate of policy change by taking steps in the direction that improves the policy, while simultaneously not staying too far away from the original policy [25]. This is achieved by adding a constraint over average KL divergence between the new and the old policy for each policy update step. Thus, TRPO helps in increasing the probability of good actions (i.e. positive advantage value) and decreases the probability of bad actions (i.e. negative advantage value).

When a change that is too extreme from the previous policy is made in a high-dimensional, nonlinear environment (like autonomous driving), then that change can lead to a dramatic decrease in performance (eg. little increase in an acceleration helps to increase the speed of car, but too much increase in an acceleration can lead to a crash or an accident). Use of δ in

vanilla policy gradient is disadvantageous as it can cause a huge deviation in policy. TRPO can be advantageous over vanilla policy gradient in achieving good results in challenging continuous control tasks with non-linear policies, such as Neural Networks.

TRPO uses advantage learning for finding an optimal policy for a given MDP. It has two variants, namely: 1. Single Path: which can be used in model free setting and 2. Vine method: which requires the system to be restored to particular states.

Advantage learning is a form of reinforcement learning similar to Q-learning and uses advantage values while learning. For state s and action a, the advantage for that state-action pair A(s,a) is related to the Q value as show in equation 2.5

If the policy updating gives a non-negative expected advantage value at every state, then policy updating is guaranteed to increase the policy performance η or leave η constant in the case that the expected advantage value is zero everywhere. Thus, it can be observed that the classic result that the update performed by exact policy-iteration which uses the deterministic policy, improves the policy in the case where there is at least one state-action pair with a positive advantage value and nonzero state visitation probability. Otherwise, the algorithm is considered to be converged to the optimal policy.

The expected discounted reward for a given policy π is given as follows:

$$\eta(\pi) = \mathop{\mathbb{E}}_{s_0, a_0, \ldots} \Big[\sum_{t=0}^{\infty} \gamma^t r(s_t) \Big] \tag{2.4}$$

where,

$$s_0 \sim \rho(s_0), a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t)$$

Equation 2.5, 2.6 and 2.7 show Advantage function A_π, value function Q_π and value function V_π respectively.

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s) \tag{2.5}$$

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}\Big[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})\Big] \tag{2.6}$$

$$V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, \ldots}\Big[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})\Big] \tag{2.7}$$

where,

$a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t)$ for $t \geqslant 0$.

For a guaranteed increase in the performance, expected return of another policy can be expressed in terms of an advantage over the original policy, and equation 2.4 can be rewritten with sum over states instead of time steps as follows:

$$\eta(\widetilde{\pi}) = \eta(\pi) + \sum_s \rho_{\widetilde{\pi}}(s) \sum_a \widetilde{\pi}(a|s) A_\pi(s, a) \tag{2.8}$$

Estimating both the state visitation based on new policy $\rho_{\widetilde{\pi}}(s)$ and the new policy $\widetilde{\pi}$ at the same time increases the complexity of the problem as $\rho_{\widetilde{\pi}}(s)$ is dependent on $\widetilde{\pi}$. This problem is resolved by using the state visitation based on an old policy and only focusing upon the improvement of policy.

$$L_\pi(\widetilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \widetilde{\pi}(a|s) A_\pi(s, a) \tag{2.9}$$

Trust regions are required as above approximation can be acceptable for small step sizes and using trust regions will help an agent to stay close to the earlier policy, so any policy update in this region can be considered as trust worthy, for not leading to a dramatically bad policy degradation. TRPO is conservative policy iteration technique and gives an explicit lower bound on improvement of η (Expected discounted reward). Conservative policy iteration defines new policy as shown in equation 2.10:

$$\pi_{new}(a|s) = (1 - \alpha)\pi_{old}(a|s) + \alpha\pi'(a|s) \tag{2.10}$$

where, π_{old} is a current policy, π_{new} is a new policy and $\pi' = \arg\max_{\pi'} L_{\pi_{old}}(\pi')$

Policy improvement bound can be given as shown in equation 2.11 (derived by Kakade and Langford) for the mixture policy in equation 2.10:

$$\eta(\pi_{new}) \geq L_{\pi_{old}}(\pi_{new}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha^2 \tag{2.11}$$

where, $\epsilon = \max_s |\mathbb{E}_{a\sim\pi'(a|s)}[A_\pi(s,a)]|$

To extend this equation to general stochastic policies, α can be replaced with distance measure between π and $\widetilde{\pi}$ which is total variational divergence (i.e. the distance between two distributions) as given by equation 2.12:

$$D_{TV}^{max}(\pi, \widetilde{\pi}) = \max_s D_{TV}(\pi(\cdot|s)||\widetilde{\pi}(\cdot|s)) \tag{2.12}$$

Divergence is expressed in terms of Kullback-Leibler (KL) divergence, so that the optimization brings the old and new distribution closer in KL sense. KL divergence measures the difference between two distributions, for example, p(x) and q(x) are probability distributions of a discrete random variable x then $D_{KL}(p(x), q(x))$ gives the measure of information loss when q(x) is used to approximate p(x) as given in equation 2.13.

$$D_{KL}(p(x)||q(x)) = \sum_x p(x) log \frac{p(x)}{q(x)} \tag{2.13}$$

The relationship between the total variation divergence and the KL divergence is given as: $D_{TV}(p||q)^2 \leq D_{KL}(p||q)$

$$\therefore D_{KL}^{max}(\pi, \widetilde{\pi}) = \max_s D_{KL}(\pi(\cdot|s)||\widetilde{\pi}(\cdot|s)) \tag{2.14}$$

Algorithm 3: POLICY-ITERATION : guaranteeing non-decreasing expected return η

Initialize π_0
for $i = 1, 2, ...$ until convergence **do**
    Compute all advantage values $A_{\pi_i}(s, a)$.
    Solve the constraint optimization problem
        $\pi_{i+1} = \underset{\pi}{\text{argmax}}[L_{\pi_i}(\pi) - CD_{KL}^{max}(\pi_i, \pi)]$
        where, $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$
        and, $L_{\pi_i}(\pi) = \eta(\pi_i) - \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s) A_{\pi_i}(s, a)$
end for

Thus the policy improvement bound can be given as in equation 2.15:

$$\eta(\widetilde{\pi}) \geq L_{\pi}(\widetilde{\pi}) - CD_{KL}^{max}(\pi, \widetilde{\pi}) \tag{2.15}$$

$$where, C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$$

Algorithm 3 describes an approximate policy iteration algorithm that is guaranteed to generate monotonically improving sequence of policies. True objective η is guaranteed to improve as the surrogate function is maximized at each iteration. This type of algorithm is called Minorization Maximization Algorithm.

TRPO is an approximation to Algorithm 3 and it has finite sample counts and arbitrary parameterization for policy. Trust region constraint is given in equation 2.16. To have reasonable updates, it maximizes local approximation of expected discounted reward with respect to maximum KL divergence between an old and a new policy.

$$\underset{\theta}{\text{maximize}} L_{\theta_{old}}(\theta) \tag{2.16}$$

$$\text{subject to } \overline{D}_{KL}^{max}(\theta_{old}, \theta) \leq \delta$$

As optimizing maximum KL divergence is impractical due the large number of constraints,

average KL divergence is used to generate policy update as shown in equation 2.17.

$$\underset{\theta}{\text{maximize}} L_{\theta_{old}}(\theta) \tag{2.17}$$

$$\text{subject to } \overline{D}_{KL}^{\rho_{\theta_{old}}}(\theta_{old}, \theta) \leq \delta$$

This constraint optimization problem needs to be transformed into sample-based estimation in order to solve it in a heuristic manner. Expanding $L_{\theta_{old}}$ from equation 2.17, equation 2.18 is obtained.

$$\underset{\theta}{\text{maximize}} \sum_{s} \rho_{\theta_{old}}(s) \sum_{a} \pi_{\theta}(a|s) A_{\theta_{old}}(s, a) \tag{2.18}$$

$$\text{subject to } \overline{D}_{KL}^{\rho_{\theta_{old}}}(\theta_{old}, \theta) \leq \delta$$

Replacing $\sum_s \rho_{\theta_{old}}(s)$ in equation 2.18 by expectation $\mathbb{E}_{s\sim\rho_{\theta_{old}}, a\sim q}$ and the sum over the actions by an importance sampling estimator the following equation 2.19 is obtained:

$$\underset{\theta}{\text{maximize}} \mathbb{E}_{s\sim\rho_{\theta_{old}}, a\sim q}\left[\frac{\pi_{\theta}(a|s)}{q(a|s)} A_{\theta_{old}}(s|a)\right] \tag{2.19}$$

where,

$$\sum_{a} \pi_{\theta}(a|s_n) A_{\theta_{old}}(s_n, a) = \mathbb{E}_{a\sim q}\left[\frac{\pi_{\theta}(a|s_n)}{q(a|s_n)} A_{\theta_{old}}(s_n|a)\right]$$

q denotes the sampling distribution.

Now replacing $A_{\theta_{old}}$ by $Q_{\theta_{old}}$ in equation 2.19 we get equation 2.20,

$$\underset{\theta}{\text{maximize}} \mathbb{E}_{s\sim\rho_{\theta_{old}}, a\sim q}\left[\frac{\pi_{\theta}(a|s)}{q(a|s)} Q_{\theta_{old}}(s|a)\right] \tag{2.20}$$

$$\text{subject to } \mathbb{E}_{s\sim\rho_{\theta_{old}}}[D_{KL}(\pi_{\theta_{old}}(\cdot|s)||\pi_{\theta}(\cdot|s))] \leq \delta$$

Further expectations are replaced by sample averages and the Q value is replaced by empirical estimate, which is information that is based on the experience of an agent. An agent can pick any random start state from state space while generating a trajectory. There

are two ways for performing estimation: 1. Single Path, and 2. Vine.

1. The Single Path procedure generates a set of trajectories via simulation of policy and incorporates all state action pairs (s_n, a_n) into an objective. Figure 2.5, shows trajectory generation here, $q(a|s) = \pi_{\theta_{old}}(a|s)$.

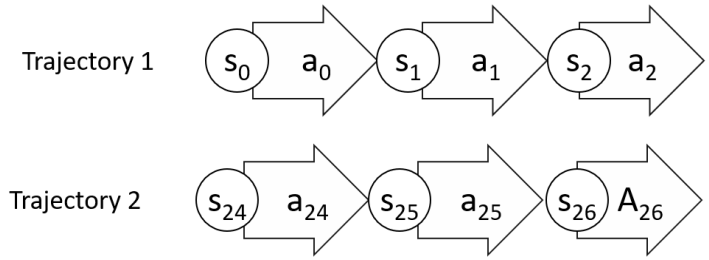


Figure 2.5: Single Path Estimation. The figure is redrawn from Schulman, et al. 2018 [25].

2. Vine procedure generates a set of trunk trajectories and then generates branch rollouts from subset of the reached states. For each state s, multiple actions are performed and then rollout is performed for each action as shown in figure 2.6.

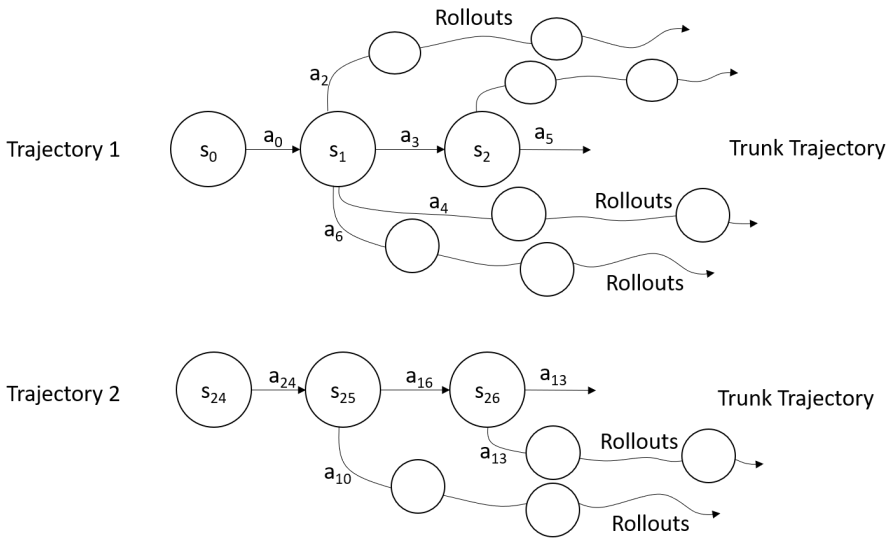


Figure 2.6: Vine Estimation. The figure is redrawn from Schulman, et al. 2018 [25].

Algorithm 4 illustrates Trust Region Policy Optimization. Where, step 2 shows how to construct an estimated objective with KL divergence constraint and step 3 uses conjugate gradient algorithm followed by line search to solve the optimization problem.

Algorithm 4: TRPO : Trust Region Policy Optimization

1 Collect a set of state-action pairs(using Single Path/ Vine procedure) along with Monte Carlo estimates of their Q-values

2 Plug in the calculated Q values and old action probability for KL divergence then average over samples and construct estimated objective and constraint as shown:
$\underset{\theta}{\text{maximize}} \mathbb{E}_{s\sim\rho_{\theta_{old}},a\sim q}\left[\frac{\pi_\theta(a|s)}{q(a|s)}Q_{\theta_{old}}(s|a)\right]$
subject to $\mathbb{E}_{s\sim\rho_{\theta_{old}}}[D_{KL}(\pi_{\theta_{old}}(\cdot|s)||\pi_\theta(\cdot|s))] \leq \delta$

3 Use the conjugate gradient algorithm followed by line search to solve this constrained optimization problem to update policy parameter vector θ. Policy update directions are conjugate with respect to the Fisher Information Matrix.
$\frac{1}{N}\sum_{n-1}^{N}\frac{\partial^2}{\partial\theta_i\partial\theta_j}D_{KL}(\pi_{\theta_{old}}(\cdot|s_n)||\pi_\theta(\cdot|s_n))$

2.5.1 Gaussian MLP Policy

Policy used in TRPO is a conditional probability distribution and is parameterized with a Gaussian Multi-Layer Perceptron (GMLP).

This feed forward Neural Network (NN) with fully-connected (dense) layers maps state vectors to a vector , which is a mean of Gaussian distribution. A separate set of parameters specifies the log standard deviation of each element.

These parameters include a set of weights and biases for the neural network that is computing the mean and a vector (log standard deviation) with the same dimension as a. The policy is defined by the normal distribution, $\mathcal{N}(mean = NeuralNet(s; \{W_i, b_i\}_i^L = 1), stdev = exp(r))$(here μ = [mean, stdev]) and is trained with weighted policy gradient. Using log standard deviation is canonical as all values between -infinity to infinity are accepted and the gradients flow more smoothly.

Figure 2.7 shows the Neural Network used for the Object-Word Environment.

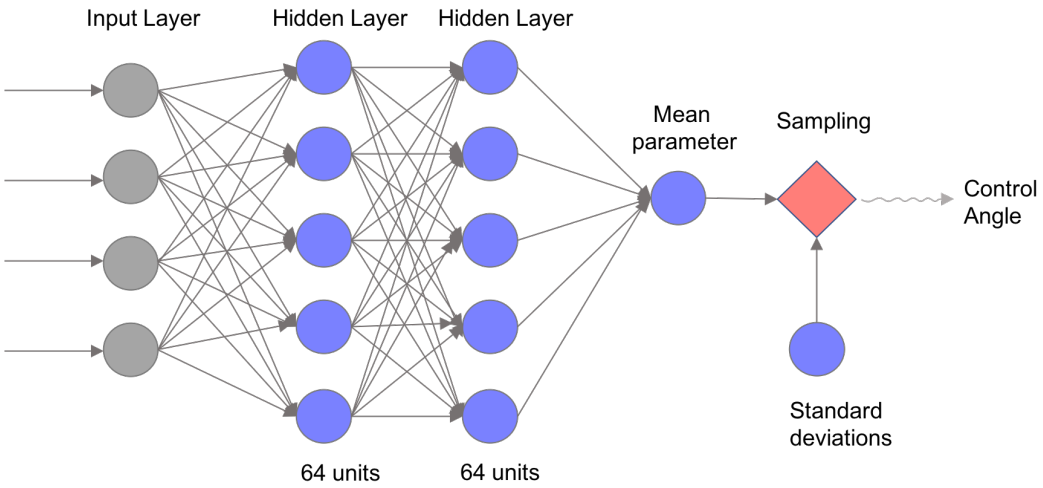


Figure 2.7: Neural Network used for the Object-Word Environment

2.6 Inverse Reinforcement Learning

Inverse Reinforcement Learning (IRL) is the problem of learning an unknown reward function (preference function) in a MDP from an expert agent based on its policy which maps states to actions or behavior (demonstration) [23].

IRL is formally defined by Russell as follows: "Given: 1) measurements of an agent's behavior over time, in a variety of circumstances; 2) if needed, measurements of the sensory inputs to that agent; 3) if available, a model of the environment. Determine: the reward function being optimized"

Presuppositions: The reward function is a succinct, compact, and transferable component of the task definition. Learning a reward function of demonstrator may be simpler than directly learning the decision mapping (control function) from observations, Often, it is not easily specified especially when multiple attributes are involved [23].

Figure 2.8 shows an IRL framework where learning agent perceives trajectory (state and action pairs) $\{(s_1, a_1), (s_2, a_2), ..., (s_t, a_t)\}$ of an expert agent E (an alternative could be

experts policy π_E) and learns a reward function, $\hat{R_E}$, of E. Note that the learned reward function may not exactly correspond to the true reward function of expert agent as it depends upon the trajectories that are used in training. Blue color lines indicate the RL framework where an expert agent is interacting with an environment. Red color lines indicate the IRL part where, the learning agent takes trajectory of an expert agent as input in order to predict expert agent's reward function.

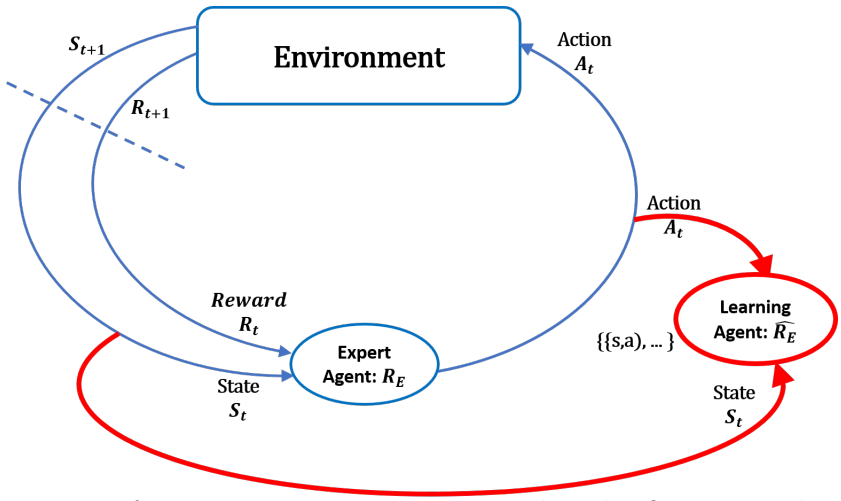


Figure 2.8: Inverse Reinforcement Learning Framework. The figure is redrawn from Arora, et al. 2018 [4].

In case of various real life decision-making activities (such as driving, various locomotion tasks, etc.), it is difficult to have a pre-defined reward function, but it is feasible to have data of the desired sample behaviors in the form of expert demonstrations. IRL is useful in such cases as it attempts to model preferences (rewards) of an expert agent based on its demonstrations (behaviour) thus avoiding manual specification of a preference function [3]. IRL assumes that the remaining parameters of the model are known.

IRL is advantageous over earlier methods as expert demonstrations substitute manual specification of a reward function, and reward functions are believed to have an inherent ability to be more conveyable than the expert agent's policy, so IRL provides better generalization.

IRL has some challenges, such as accurate inference, generalizability, correctness of prior knowledge, and growth in solution complexity with problem size because of optimization that is associated with finding a reward function that best explains observations is essentially ill-posed [23, 4].

There are various algorithms based on the technique used for solving IRL as follows [4]:

1. Maximum Margin Optimization (MMO): The learning agent tries to address the ambiguity of a solution by deciding on a preference function that maximizes a margin from others. In other words, the MMO seeks for the reward which makes the demonstrated policy look better than alternative policies by a margin. Linear Programming IRL (LPIRL) [19] (Appendix I explains LPIRL), Apprenticeship learning and Maximum Margin Planning (MMP) are some of the algorithms which use MMO.

2. Entropy Optimization: MMO methods generally introduce bias into the learned reward function. Multiple methods take recourse to the maximum entropy principle in order to obtain a distribution over potential reward functions while avoiding any bias. The distribution that maximizes entropy is considered more accurate. Maximum Entropy IRL (Max-Ent IRL) recovers a distribution over all trajectories, which has the maximum entropy among all distributions under the constraint that feature expectations of a learned policy match that of demonstrated behavior [28]. Relative entropy IRL is another method that is based on Entropy Optimization.

   The MaxEnt approch is straightforward when the information comes as a set of hard constraints on a hypothesis space and gives the lowest worst-case log-loss when its constraints are satisfied (i.e. The worst taken over all possible priors) [6].

3. Bayesian Update: These methods treat the state-action pair in the given demonstration as an observation (evidence) which facilitates a Bayesian update of prior reward distribution (hypothesis). Bayesian IRL, Bayesian IRL via Bisimulation, Non-parametric

Bayesian IRL algorithms are all based on Bayesian Update.

Bayesian IRL approach has spawned various improvements over earlier IRL techniques, as it gives an ability to leverage machine learning literature on approximate inference. Also, BIRL provides better scalability than Max-margin with respect to the number of states.

The Bayesian approach is straightforward to apply if prior knowledge comes in the form of a measurable real-valued function over a given hypothesis space. The Bayesian model of averaging gives the lowest average log-loss (averaged over many model draws) when the prior matches the true distribution of hypotheses [6]. Section 3.3 explains Bayesian IRL in detail.

2.7 Summary

This chapter covers the background concepts of this thesis. First, this chapter explains the Markov decision process and its components. Next, expected utility of a policy is explained, which is followed by explanation of Reinforcement Learning (RL), Trust Region Policy Optimization(TRPO) and Inverse Reinforcement Learning (IRL).

Chapter 3

REVIEW OF LITERATURE

This chapter presents a review on various reinforcement learning and inverse reinforcement learning algorithms for continuous spaces and is organized as follows: Subsection 3.1. presents various reinforcement learning algorithms with continuous spaces and subsection 3.2 describes existing inverse reinforcement learning algorithms with continuous spaces. Last subsections describe Bayesian IRL and Gaussian process IRL.

3.1 Reinforcement Learning with Continuous Spaces

This section is organized into 4 subsections and discusses the different reinforcement learning techniques. Subsection 3.1.1. presents Fast Reinforcement Learning in Continuous Action Spaces, subsection 3.1.2. discusses Reinforcement Learning in Continuous Action Spaces, subsection 3.1.3 covers Reinforcement Learning in Continuous Action Spaces through Sequential Monte Carlo Methods and subsection 3.1.4 describes about Continuous-action reinforcement learning with fast policy search and adaptive basis function selection.

3.1.1 Fast Reinforcement Learning in Continuous Action Spaces [9]

This paper presents an algorithm similar to Doya's approach, (which does not consider Temporal and Spatial Differencing) for Reinforcement Learning with continuous state and action spaces. It couples the underlying Hamilton Jacobi Bellman (HJB) partial differential equation with the structure of the function approximator, using following three techniques: 1. Techniques like Finite Element Method (FEM), 2. Locally weighted learning, and 3. Constrain the control to a constant radius hypersphere in control space and apply Pontryagin's optimality principle on this restricted control space only. Because of these three techniques, the HJB equation can be reduced to a linear system of algebraic equations and its iterative solution gives the optimal value function of the problem.

According to Doya's derivation value of a state under policy is a scalar expressing expected discounted reward if that policy is followed as shown in equation 3.1.

$$V^{\pi}(x) = \int_0^{\infty} \frac{1}{\tau} e^{-\frac{t}{\tau} r(x^t)} dt, x^0 = x \tag{3.1}$$

Discounted version of the HJB equation 3.2.

$$V^*(x) = r(x) + \tau \max_{u(x)} [\bigtriangledown_x V^*|_x \cdot f(x, u(x))] \tag{3.2}$$

Following two features of FEM makes it preferable to Finite Differencing (FD) for RL problems: 1. FEM decomposes space by means of irregular volume elements and if the environment involves of low dimensional manifold of state space then whole space need not be covered, only manifold is covered, thus circumventing the curse of dimensionality; 2. Universal approximators like MLP, RBF, etc. are used in RL to represent value function which can be differentiated instead of expressing them as a finite difference in state space.

To solve the HJB equation, the dynamics of the system are linearized around the current state x and by plugging in the linearized dynamics into the HJB equation 3.3 is obtained.

$$V^*(x) = r(x) + \tau \bigtriangledown_x V^*|_x \cdot f(x,0) + \tau \max_{u(x)} [\bigtriangledown_x V^*|_x \cdot \frac{\partial f}{\partial u}|_{x,0} u] \tag{3.3}$$

After determining optimal control, the differential constraint defined by means of HJB equation can be enforced in the direction of application of optimal control as shown in equation 3.4.

$$V^*(x) = r(x) + \tau \bigtriangledown_x V^*|_x \cdot [f(x,0) + \frac{\partial f}{\partial u}|_{x,0} u_{opt}] \tag{3.4}$$

The algorithm was tested on a simple 2D system with 2 controls. The control gains were assumed to be known and constant throughout state space. The goal of the experiment was to verify if a good approximation of the value function can be learned without a regular cell grid and if the proper continuous controls can be determined from that value function. Results show learned value function over 25 irregularly spaced nodes, after 9 iterations.

3.1.2 Reinforcement Learning in Continuous Action Spaces [27]

This paper presents a new class of algorithms named Continuous Actor Critic Learning Automaton (CACLA) that can handle continuous state and action spaces. It is showed that continuous state spaces and continuous action spaces make use of parameterized function approximator (FA's). For exploration of the continuous spaces 2 methods are used: 1. ϵgreedy exploration [where, random action is selected with probability ϵ and greedy current approximation of optimal action is selected with probability (1- ϵ] and 2. Gaussian exploration [which explores around the current approximation of the optimal action and the performed action is sampled from the distribution with μ at the action output]

In this paper CALCA is used for tasks like Continuous Tracking and Cart Pole. Comparison between proposed algorithm and other algorithms (like Wire Fitting, Gradient Ascent

Table 3.1: Tracking and Cartpole Results. The table is redrawn from Hasselt, et al. 2007 [27].

	Tracking Results			Cartpole Results	
	ϵ-Greedy			ϵ-Greedy	
	γ	Mean	Std Dev	γ	Success
WF	0.80	0.861	0.056	0.80	10 %
GAV	0.00	0.783	0.163	0.00	15 %
CACLA	0.95	0.829	0.012	0.95	40 %
CACLA+Var	0.90	0.843	0.014	0.99	80 %
RND	0.00	0.467	0.020	0.00	0 %
	Gaussian			Gaussian	
	γ	Mean	Std Dev	γ	Success
WF	0.00	0.816	0.261	0.80	0 %
GAV	0.95	0.088	0.265	0.80	20 %
CACLA	0.95	0.922	0.011	0.80	100 %
CACLA+Var	0.80	0.938	0.011	0.95	100 %
RND	0.00	0.467	0.020	0.00	0 %

on the value) of the tracking and Cartpole results are shown in table 3.1.

Experimental results show that CACLA performs much better than the other algorithms, especially when it is combined with a Gaussian exploration method, also CACLA is resistant to noisy interactions with an environment, figure 3.1 shows the results for Tracking and cartpole example where, the average intermediate rewards are plotted against the number of training iterations.

Furthermore, this paper defines Continuous Actor Critic (CAC) systems and show that its performance is inferior to the CACLA algorithms. CACLA and CAC only update when the TD-error (t) is positive. Negative updates result in worse behavior.

The main difference between CAC and CACLA lies in their intended optimization. CAC tries to optimize the policy in terms of t, while CACLA optimizes the probability of positive t. This approach increases learning rates, especially in areas where t is relatively small.

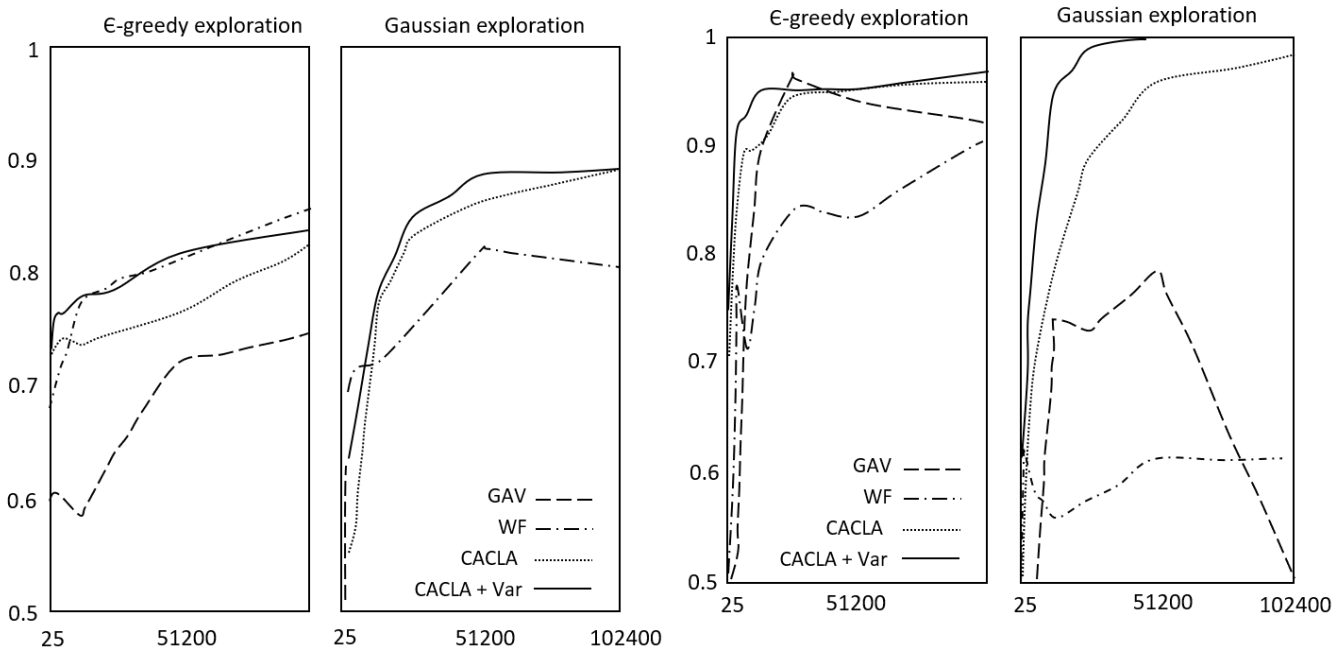


Figure 3.1: Tracking and Cartpole Results. The figure is redrawn from Hasselt, et al. 2007 [27].

Additionally, an extension to the CACLA and CAC algorithms is presented which uses the variance t to determine if an update has been significantly large table 3.2 shows cartpole results for CACLA and CAC.

Table 3.2: Cartpole results for CACLA and CAC. The table is redrawn from Hasselt, et al. 2007 [27].

	Update only when $\delta_t > 0$		Update always	
	γ	Success	γ	Success
CACLA	0.90	100 %	0.90	0 %
CACLA+Var	0.95	100 %	0.90	80 %
CAC	0.90	50 %	0.90	45 %
CAC+Var	0.99	60 %	0.90	60 %

Paper shows that this new version performs slightly better than the versions of the algorithms that treat all updates equally. Instead of using true variance of states, the variance

is updated as a running average over all visited states, figure 3.2 shows the results for cartpole with CACLA and CAC.

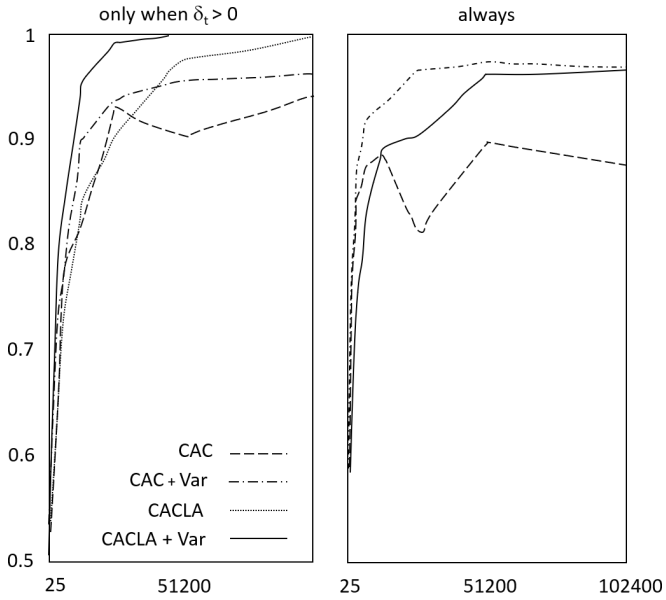


Figure 3.2: Cartpole Results for CACLA and CAC. The figure is redrawn from Hasselt, et al. 2007 [27].

It is mentioned in the paper that better results might be obtained when an approximation of the current variance is stored per state, using FA's to map states to approximations of the variance of δ_t.

3.1.3 Reinforcement Learning in Continuous Action Spaces through Sequential Monte Carlo Methods [16]

This paper, proposes a novel Actor-Critic (AC) approach where, the policy of the actor is estimated through Sequential Monte Carlo (SMC) methods and re-sampling step modifies the actor's policy. Actor represents current policy using finite set of available actions associated to weights. While the importance sampling step is performed based on the values learned by the critic. Critic updates actions associated to weights with its computed utility values.

Experimental results show that SMC-learning can identify highest valued actions with

the help of importance sampling and re-sampling. It is observed that the convergence time and performance of proposed approach is better compared to Continuous Q-learning and Tile Coding and SARSA. Results are reported for 'steering a boat across a river' control problem as shown in figure 3.3.

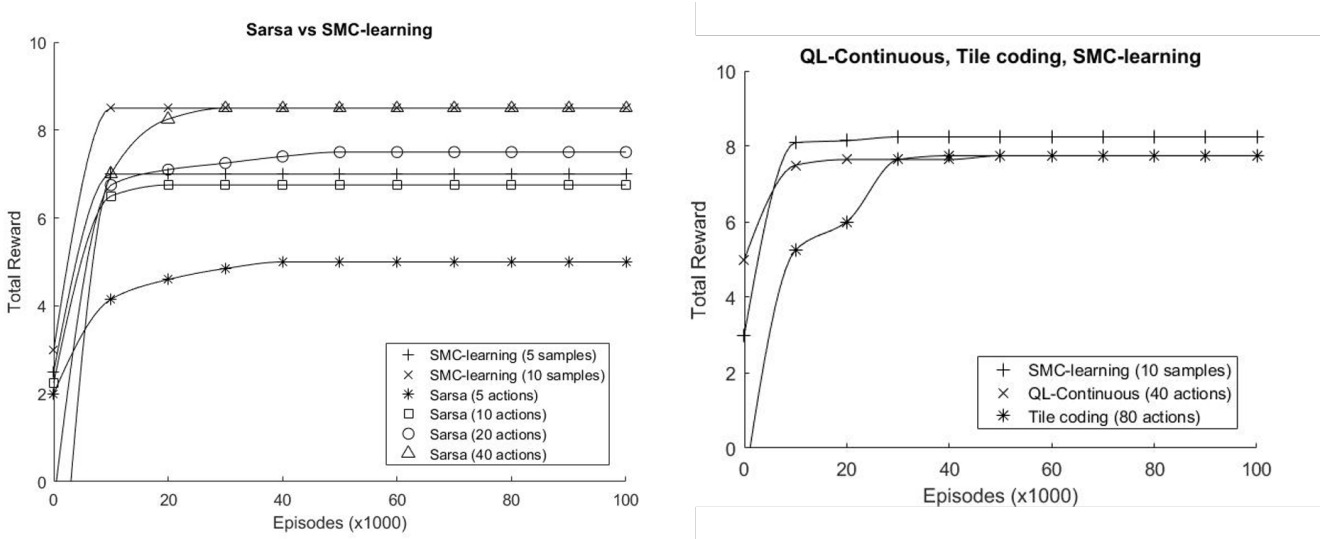


Figure 3.3: Performance comparing between SMC-learning and other algorithms. The figure is redrawn from Lazaric, et al. 2008 [16].

Two main directions are thought of for the future work: extending SMC-learning for the problems with poor knowledge of discretization of the state space, and investigating multi-agent problems in continuous action.

3.2 Inverse Reinforcement Learning with Continuous Spaces

This section is organized into 4 subsections and discusses the different inverse reinforcement learning techniques. Subsection 3.2.1. presents Continuous Inverse Optimal Control with Locally Optimal Examples, subsection 3.2.2. discusses Relative Entropy Inverse Reinforcement Learning, subsection 3.2.3 gives insights on Learning objective functions for manipulation

and subsection 3.2.4. discusses about Maximum entropy inverse reinforcement learning in continuous state spaces with path integrals.

3.2.1 Continuous Inverse Optimal Control with Locally Optimal Examples [17]

This paper introduces a probabilistic Inverse Optimal Control (IOC) algorithm that is suitable for high dimensional and continuous domains. By using a local approximation to the reward function likelihood, it has been shown that the demonstrations requiring only local optimality.

Presented algorithm relies on the derivatives of the reward features to learn the reward function. Reward features need to be differentiable with respect to the states and actions. These derivatives are pre-computed only once, thus it is quite practical to use finite differences when analytic derivatives are unavailable. This algorithm addresses deterministic, fixed horizon control tasks, with continuous states and actions, and discrete time. Optimal actions are given by equation 3.5.

$$u = \arg\max_{u} \sum_{t} r(x_t, u_t) \tag{3.5}$$

MaxEnt model is employed for the expert's behavior, as shown in equation 3.6. Where, Z is a partial function, x represents states and u represents actions.

$$
\begin{aligned}
P(u|x_0) &= \frac{1}{Z}\exp(\sum_t r(x_t, u_t)) \\
P(u|x_0) &= e^{r(u)}[\int e^{r(\widetilde{u})} d\widetilde{u}]^{-1} \\
P(u|x_0) &\approx e^{r(u)}[\int e^{r(u)+(\widetilde{u}-u)^T g+\frac{1}{2}(\widetilde{u}-u)^T H(\widetilde{u}-u)} d\widetilde{u}]^{-1} \\
&= e^{\frac{1}{2}g^T H^{-1} g}|-H|^{\frac{1}{2}}(2\pi)^{\frac{-du}{2}}
\end{aligned}
\tag{3.6}
$$

Equation 3.7 obtains the Approximate log likelihood and the most likely parameters for a given parameterization can be learnt by maximizing it.

$$
\mathcal{L} = \frac{1}{2}g^T H^{-1} g + \frac{1}{2}\log|-H| - \frac{du}{2}\log(2\pi) \tag{3.7}
$$

This paper shows 2 efficient ways for likelihood optimization: 1. Direction Likelihood evaluation, 2. LQR-Based likelihood evaluation. Rewards can be learnt as a linear combination of features as well as by using Gaussian process to learn nonlinear reward functions. Another proposed future work can be to apply presented approach to other high dimensional, continuous problems. Nonlinear variant of the algorithm relies on features to generalize the reward to unseen regions of the state space.

Evaluation is done on simulated robot arm control, planar navigation, and simulated driving. Evaluation presents that the prior methods do not converge to the underlying reward function when the examples are only locally optimal, regardless of how many examples are provided.

In the robot arm task, the expert sets continuous torques on each joint of an n-link planar robot arm. The reward depends on the position of the end-effector.

Figure 3.4 (a) and (b) shows that the prior methods do not converge to the expert's policy when the examples are not globally optimal. Figure 3.4 (c) and (d) shows that

only the non-linear variant of the proposed algorithm learns the reward, only using position features.

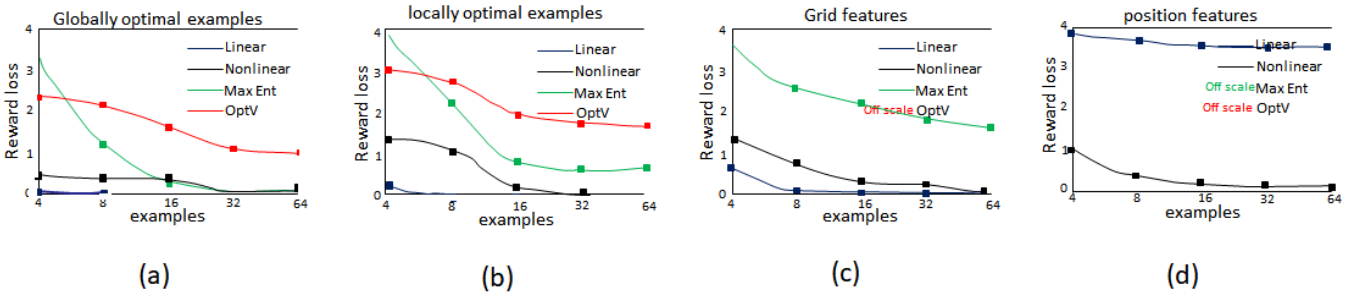


Figure 3.4: Reward loss for each algorithm with (a) globally optimal planar navigation examples (b) locally optimal planar navigation examples (c) Gaussian grid features on the 2 link robot arm task (d) End-effector position features on the 2 link robot arm task. The figure is redrawn from Levine, et al. 2012 [17].

Table 3.3: Statistics for sample paths for the learned driving rewards and the corresponding human demonstrations starting in the same initial states. The table is redrawn from Levine, et al. 2012 [17].

Driving Style	Aggressive		Evasive		Tailgater	
Path	Learned	Human	Learned	Human	Learned	Human
Average Speed (kph)	158.1	158.2	150.1	149.5	97.5	115.3
Time behind (sec)	3.5	3.5	7.2	4.5	111.0	99.5
Time in front (sec)	12.5	16.7	3.7	2.8	0.0	7.0

Table 3.3 shows the statistics of the learned paths which closely resemble the holdout demonstrations. And the figures in the table show the corresponding learned driving style rewards. It can be observed from the figure that the aggressive reward is high in front of other cars, the evasive one avoids while the tailgater reward is high behind the cars.

Given method is not well suited for features that exhibit discontinuities and the formulation only considers: deterministic, fixed-horizon control problems. It is mentioned that extension to stochastic or infinite- horizon problems can be a good direction for the future work.

3.2.2 Relative Entropy Inverse Reinforcement Learning [5]

This paper proposes a model-free IRL algorithm, where the cost function is minimized by stochastic gradient descent (SGD). This new approach is compared to a well-known IRL algorithms (like model based Maximum Entropy (MaxEnt), naïve model-free adaptation of Maximum Map Planning (MMP) using learned MDP models.

The proposed algorithm is based on minimizing the relative entropy (KL divergence) between the empirical distribution of the state-action trajectories under a baseline policy and the distribution of the trajectories under a policy that matches the reward feature counts of the demonstration. It uses an inaccurate model, which is learned from a small number of sample and shows that it may lead to learning reward functions that are completely different from the true ones.

Empirical results on simulated car racing, grid world and ball-in-a-cup problems show that this new approach can learn good policies from a small number of demonstrations. Ball-in-a-cup problem has continuous state and action space, as the actions correspond to the Cartesian accelerations of the cup and the states consist of Cartesian positions and velocities of ball and cup.

Figure 3.5 shows schematic drawing of the Ball-in-a-Cup motion.

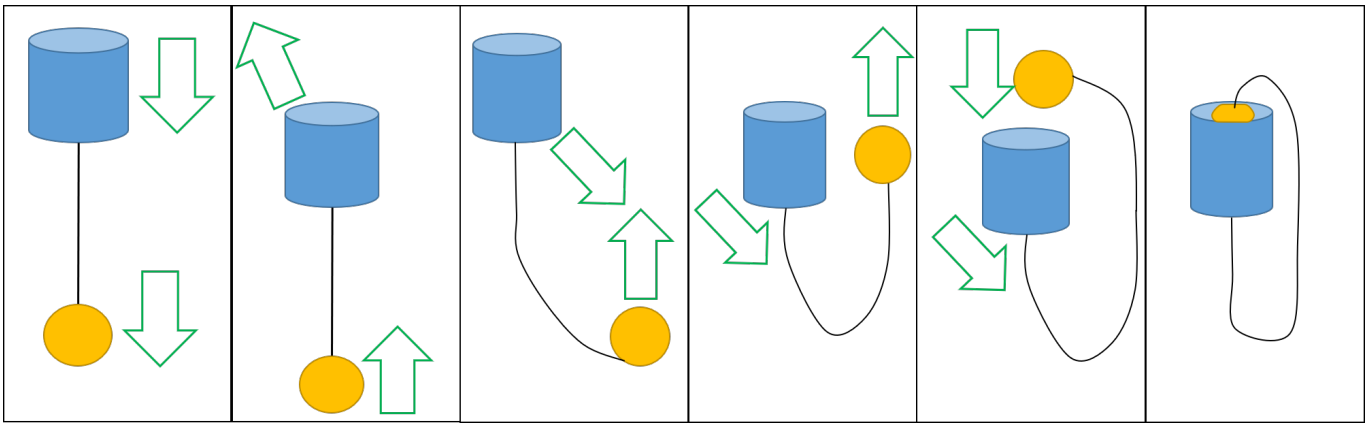


Figure 3.5: Schematic drawing of the Ball-in-a-Cup motion. The figure is redrawn from Boularias, et al. 2011 [5].

Figure 3.6 shows the resulting success rate with Model-free Relative Entropy IRL (RE IRL) and Model-free Maximum Map Planning (MMP).

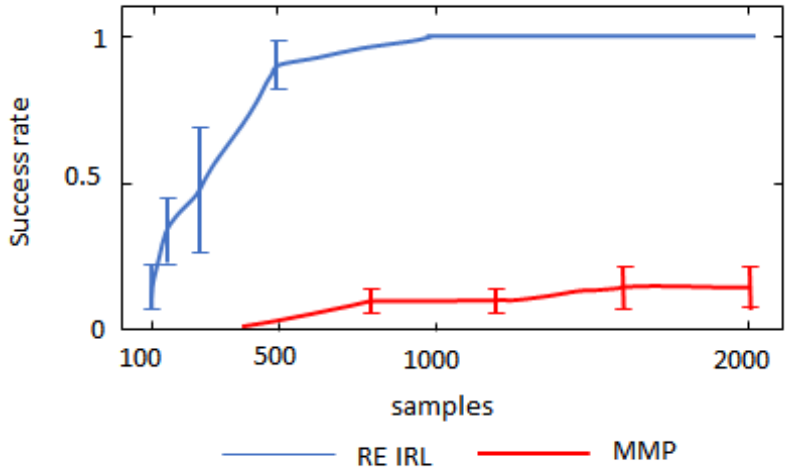


Figure 3.6: Ball-in-a-cup success rate. The figure is redrawn from Boularias, et al. 2011 [5].

Proposed future work is to experiment on more difficult problems and to explore other techniques of stochastic optimization.

3.2.3 Learning objective functions for manipulation [13]

This paper presents an approach to learning objective functions for robotic manipulation based on IRL, called path integral IRL algorithm, which can deal with high-dimensional continuous state-action spaces, and only requires local optimality of demonstrated trajectories. In-order to achieve feature selection, L1 regularization is used. An efficient algorithm is proposed to minimize the resulting convex objective function.

Path integral IRL assumes that the state-dependent cost function q is linearly parameterized. Overall trajectory likelihood is given in equation 3.8, where the cumulative feature counts starting at time-step t.

$$L(\tau^*|w) = \Pi_{t=1}^{N} \frac{e^{-w^T \Phi_t^*}}{\int e^{-w^T \Phi_t} d\tau_t} \tag{3.8}$$

Objective is to minimize the weighted sum of negative log likelihood and the l1 norm of w, as shown in equation 3.9. Here, λ indicates the strength of the regularizer, D is the number of demonstration.

$$\min_{w} - \sum_{i=1}^{D} \log \frac{e^{-w^T \Phi_i^*}}{\sum_{k=1}^{K} e^{-w^T \Phi_{i,k}}} + \lambda ||w||_1 \tag{3.9}$$

Demonstrated End-effector trajectory from home to grasp configuration in green, 20 samples around the demonstration are shown in blue and the optimized trajectory based on the learnt objective function is shown in red.

Presented approach is demonstrated by applying on two core problems in robotic manipulation: 1. Cost function is learnt for redundancy resolution in inverse kinematics, 2. Cost function is learnt over trajectories and is used in optimization-based motion planning for grasping and manipulation tasks.

Experimental results show that proposed method outperforms previous algorithms in high-dimensional settings and demonstrates faster learning with lower variance than previous algorithms.

3.2.4 Maximum entropy inverse reinforcement learning in continuous state spaces with path integrals [1]

In this paper, IRL is applied for a particular class of continuous-time stochastic systems with continuous state and action spaces, to produce a cost function for which a given policy is optimal. It is assumed that both the cost function and the optimal control policy are parametric with known basis functions.

MaxEnt distribution is produced over the space of all sample paths with a constraint that input cost is inversely proportional to the noise variance. Then, maximum likelihood estimation is applied to approximate the parameters of this distribution given a finite set of

sample paths. At each iteration a new batch of sample path is added to improve approximation of the estimates of the cost function. IRL problem proposed is to optimize the most likely set of weights which define the cost function, given observation of the optimal policy, parameterized by , and a set of sampled trajectories , here as shown in equation 3.10.

$$\begin{aligned} \widetilde{\beta} &= \arg\max_{\beta} P(\theta^*|\beta,\Omega) \\ \max_{\beta} P(T=\tau^*|\beta,\Omega) &= \max_{\beta} \frac{e^{-\frac{1}{\lambda}(\beta^T\Phi(\tau^*))}}{\sum_{k=1}^{K} e^{-\frac{1}{\lambda}(\beta^T\Phi(\tau_k))}} \end{aligned} \tag{3.10}$$

Formal comparison between presented approach and existing IRL approaches are topics of future work of this paper.

3.3 Bayesian IRL [22]

As described in Chapter 2, IRL tries to learn an expert agent's reward function by observing it's behaviour. Bayesian Inverse Reinforcement Learning (BIRL) models IRL from Bayesian perspective. BIRL incorporates domain knowledge of an expert to learn rewarding decisions.

Bayes Rule is based on the concept of updating learning agents understanding of the world, as more and more evidences become available, in order to make learning agent's understanding of the world perfect. While learning, an agent builds it's perception about the world based on, it's own belief which, it forms based on the evidences it gathers about the world from an expert agent.

BIRL approach can find solutions for both reward learning and apprenticeship learning tasks by using prior knowledge and evidence from the expert's actions to derive a probability distribution over the space of reward functions. Inference is performed for these tasks using, a modified Markov Chain Monte Carlo (MCMC) algorithm. MCMC converges to the correct

answer in polynomial time as the Markov Chain for distribution with a uniform prior mixes rapidly.

BIRL is advantageous over previous work, since completely specified optimal policy is not needed as an input to the IRL agent and it need not be assumed that the expert is dependable (i.e. expert can be sub-optimal). Also evidence from multiple experts can be used for learning.

Figure 3.7 shows Bayesian IRL model. Here, expert agent χ operates in MDP $M =< S, A, T, \gamma >$. As seen in the figure posterior distribution for the rewards are derived from a prior distribution and a probabilistic model of the expert's actions given the reward function. It is assumed that reward function R for χ is selected from known prior distribution P_R.

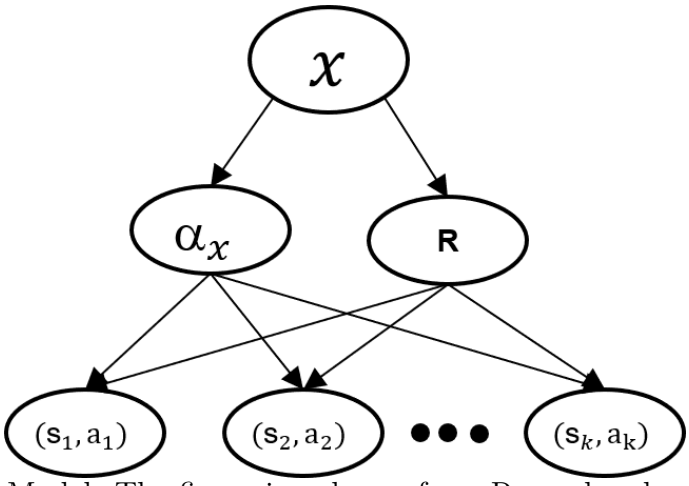


Figure 3.7: BIRL Model. The figure is redrawn from Ramachandran, et al. 2007 [22].

Series of observations of the expert behaviour $O_\chi = \{(s_1, a_1), (s_2, a_2), ..., (s_k, a_k)\}$ are given as input to an IRL agent, χ was in state s_i and took action a_i at time step i. BIRL makes following assumptions about the expert: χ tries to maximize accumulated reward according to R and χ executes stationary policy (eg. χ executes a policy learned from reinforcement learning algorithm).

Equation 3.11 presents an independence assumption that, given reward function all actions are in-dependant (i.e. probability of (agent selects an action 1 in state 1) is independent

of (agent selects an action 2 in state 2)).

$$Pr_x(O_X|R) = Pr_x((s_1, a_1)|R)Pr_x((s_2, a_2)|R)...Pr_x((s_k, a_k)|R) \tag{3.11}$$

Rewards can be maximized by finding an action for which Q* value at each state is maximum. Thus, for larger Q*(s,a) χ is likely to take action a at state s, which increases confidence in χ's ability to select good actions as shown in equation 3.12.

$$Pr_\chi((s_i, a_i)|R) = \frac{1}{Z_t} e^{\alpha_\chi Q^*(s_t, a_t, R)} \tag{3.12}$$

Here, α_χ gives the measure of expertness of expert agents to choose actions with high value. Expert agent is allowed to be sub-optimal and α_χ helps to measure number of time agent can be wrong.

Equation 3.13 has $E(O_\chi) = \sum_i Q * (s_i, a_i, R)$ and Z is an appropriate normalization constant.

$$Pr_\chi(O_\chi|R) = \frac{1}{Z} e^{\alpha_\chi E(O_\chi, R)} \tag{3.13}$$

Equation 3.14 show how to computer posterior probability of reward function R by applying Bayes Theorem:

$$\begin{aligned} Pr_\chi(R|O_\chi) &= \frac{Pr_\chi(O_\chi|R)P_R(R)}{Pr(O_\chi)} \\ &= \frac{1}{Z'} e^{\alpha_\chi E(O_\chi, R)} P_R(R) \end{aligned} \tag{3.14}$$

Selection of prior depends on the characteristics of the problem. In case of no available information, rewards are assumed to be independently identically distributed. If completely agnostic about prior then, use uniform distribution over the space $-R_{max} \leq R(s) \leq R_{max}$

for each s ϵ S. When MDP has parsimonious reward structure with states having negligible rewards, Gaussian Laplacian prior is used and if the underlying MDP represents a planning type problem, where most states have low rewards and few states have high rewards then it can be modelled as Beta distribution.

General procedure for reward learning and apprenticeship learning in BIRL, is to derive minimal solution for appropriate loss functions over the posterior as shown in equation 3.14.

Reward learning is an estimation task. Linear and squared error loss functions are the most common loss functions for estimation problems as shown in equation 3.15

$$\begin{aligned} L_{linear}(R, \widehat{R}) &= ||R - \widehat{R}||_1 \\ L_{SE}(R, \widehat{R}) &= ||R - \widehat{R}||_2 \end{aligned} \tag{3.15}$$

Here, R is true reward and $\widehat{R}$ is estimated reward. Bayesian estimation problems can use the maximum a posteriori (MAP) value as the estimator.

Apprenticeship learning task attempts to learn policy π, by trying to find π that minimizes expected L_{policy} loss over posterior distribution of R. Policy loss functions is given in equation 3.16.

$$L^p_{policy}(R, \pi) = ||V^*(R) - V^\pi(R)||_p \tag{3.16}$$

Here, $V^*(R)$ is vector of optimal values for each state achieved by the optimal policy for R and p is a norm.

Instead of minimizing expected policy loss, optimal policy for mean reward function can be found. To compute mean of posterior distribution, generate samples from these distributions and then return the sample mean as an estimate of the true mean of the distribution. BIRL uses Markov chain Monte Carlo (MCMC) sampling technique. In the case of MCMC

sampling: posterior = prior x likelihood. Trace plot of the prior is like a random walk and each prior value update is dependent on the previous prior value in the sequence. Policy Walk Sampling algorithm is given as in Algorithm 5 where, Metropolis Hasting algorithm is used to decide which proposed value of prior to accept or reject.

Algorithm 5: PolicyWalk sampling algorithm [22]

Input: Distribution P, MDP M, Step Size δ
Output: Rewards $\mathbf{R}$

1 Pick a random reward vector $\mathbf{R}\ \epsilon\ \mathbb{R}^{|s|}\ /\ \delta$
2 π := PolicyIteration(M, $\mathbf{R}$)
3 Repeat
4 (a) Pick a reward vector $\widetilde{\mathbf{R}}$ uniformly at random from the neighbours of
5 $\mathbf{R}$ in $\mathbb{R}^{|s|}\ /\ \delta$
6 (b) Compute $Q^{\pi}(s, a, \widetilde{\mathbf{R}})$ for all $(s, a)\epsilon S, A$.
7 (c) **If** $\exists(s, a)\epsilon(S, A)$,Q(s, $\pi(s), \widetilde{\mathbf{R}}$) $<Q(s, a, \widetilde{\mathbf{R}})$
8 (i) $\widetilde{\pi}$:= PolicyIteration(M, $\widetilde{\mathbf{R}}$. π)
9 (ii) Set $\mathbf{R} := \widetilde{\mathbf{R}}$ and $\pi := \widetilde{\pi}$ with probability min $\left\{1, \frac{P(\widetilde{\mathbf{R}}, \widetilde{\pi})}{P(\mathbf{R}, \widetilde{\pi})}\right\}$
10 **else**
11 (i) Set $\mathbf{R} := \widetilde{\mathbf{R}}$ with probability min $\left\{1, \frac{P(\widetilde{\mathbf{R}}, \pi)}{P(\mathbf{R}, \widetilde{\pi})}\right\}$
12 **return** $\boldsymbol{R}$

PolicyWalk Sampling algorithm keeps track of optimal policy π^* for the current reward vector R, while moving along the Markov Chain. Change in π^* can be detected because for some (s,a) ϵ (S,A), $Q^{\pi}(s, \pi(s), \tilde{R}) < Q^{\pi}(s, a, \tilde{R})$ see line 7 in the algorithm 5. The new optimal policy π^* is usually only slightly different from the old one and can be computed by just a few steps of policy iteration.

3.4 Gaussian Process IRL

The Gaussian Process Inverse Reinforcement Learning(GPIRL) algorithm uses Gaussian process(GP) to learn the reward as a nonlinear function, while also determining the relevance of each feature to the expert's policy. GPIRL enables learning agent to capture complex

behaviors from sub-optimal stochastic trajectories.

IRL finds a reward with a meaningful structure, many of the prior IRL methods impose structure by describing the reward as a linear combination of hand selected features. In GPIRL Gaussian process model is extended to learn highly nonlinear reward functions, it only observes the expert's actions, not the rewards, so the GP model is extended to account for the stochastic relationship between actions and the underlying rewards. This allows GPIRL to balance the simplicity of the learned reward function against its consistency with the expert's actions, without assuming the expert to be optimal.

The learned GP kernel hyper-parameters capture the structure of the reward, including the relevance of each feature. Once learned, the GP can recover the reward for the current state space and can predict the reward for any unseen state space within the domain of the features. Figure 3.8 shows the GPIRL framework.

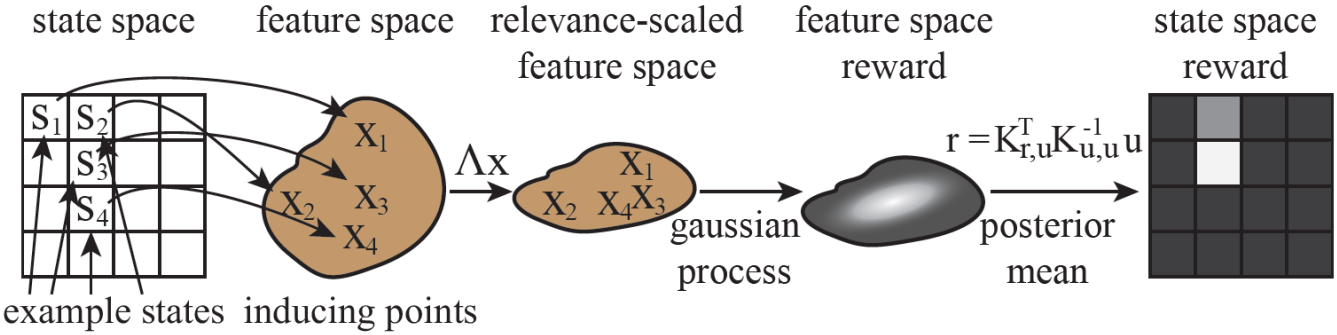


Figure 3.8: GPIRL Framework. The figure is redrawn from Levine, et al. 2011 [18].

GPIRL is the first method to combine probabilistic reasoning about stochastic expert behavior with the ability to learn the reward as a nonlinear function of features, allowing it to outperform prior methods on tasks with inherently nonlinear rewards and sub-optimal examples.

The GP prior prevents over fitting , selects relevant features with an Automatic Relevance Detection (ARD) kernel and can represent complex and nonlinear rewards. Unique challenges associated with learning GP outputs are handled with an novel hyper parameter prior. A

probabilistic IRL model allows the method to handle sub optimal real-world demonstrations.

The full reward can be recovered as a posterior mean of the GP,given by $r = K_{r,u}^T K_{u,u}^{-1} u$, where $K_{r,u}$ is the co-variance between all states and the inducing points. The likelihood of u and θ is given by equation 3.17.

$$\log P(\mathcal{D}, u, \theta | X_u) = \underbrace{\log P(\mathcal{D} | r = K_{r,u}^T K_{u,u}^{-1} u)}_{\text{IRL log likelihood}} + \underbrace{\log P(u, \theta | X_u)}_{\text{GP log likelihood}} \tag{3.17}$$

Noise can be observed in X_u, which reflects uncertainty about the location of no-redundant inducing points. GPIRL assumes Gaussian noise with variance σ^2, expected distance in k^{th} feature is $(x_{ik} - x_{jk})^2 + 2\sigma^2$ and the kernel is as given in equation 3.18.

$$k(x_i, x_j) = \beta \exp(-\frac{1}{2}(x_i - x_j)^T \Lambda (x_i, x_j) - 1_{i \neq j} \sigma^2 tr(\Lambda) \tag{3.18}$$

Degeneracies can occur as $\Lambda \to 0$ or $\beta \to 0$. This can be avoided with a hyperparameter prior which captures belief that no inducing points are deterministically related, which implies infinite partial correlation, so partial correlation $[K_{u,u}^{-1}]_{ij}$ is penalized as illustrated in equation 3.19

$$\log P(\theta) = -\frac{1}{2} \sum_{ij} [K_{u,u}^{-1}]_{ij}^2 = -\frac{1}{2} tr(K_{u,u}^{-2}) \tag{3.19}$$

Policy returns a dictionary, whose values are symbolic expressions for quantities related to the distribution of the actions and for a Gaussian policy, it is the mean and (log) standard deviation [18].

3.5 Summary

This chapter presented a review on various reinforcement learning and inverse reinforcement learning algorithms for continuous spaces. First, various reinforcement learning algorithms with continuous spaces are explained in detail, then the existing inverse reinforcement learning algorithms with continuous spaces are explained, which is followed by description of Bayesian IRL and Gaussian process IRL.

Chapter 4

MODEL-BASED IRL WITH CONTINUOUS ACTION SPACES

As mentioned in chapter 1, in the case of applications involving interaction with the real world, such as autonomous driving, maneuvering of drones or other locomotion tasks, an agent requires continuous spaces that are defined for better learning processes. There exists a clear distinction in the state space and the action space of a task. It is observed that continuous state spaces are easier to deal with than a continuous action spaces because of this reason in the case of IRL research continuous action spaces, such as angle, velocity, acceleration, etc. are less explored compared to continuous state spaces.

This chapter presents the proposed model-based IRL with continuous action space for preference(reward) learning. A model-based IRL algorithm for continuous action spaces extends the Bayesian IRL Policy-Walk algorithm. The proposed algorithm uses Trust Region Policy Optimization in the policy optimization step. Action space distributions are generated for each state with a random walk algorithm, and an online transition function is used.

Model free approach can be biased by algorithm and based on the parameters that are used for learning policy. Therefore a model-based approach can provide better results.

In the case of Bayesian learning an agent learns the most probable hypothesis (reward) given evidence (expert's behavior). Equation 3.14 illustrates how to compute the posterior probability of reward function R by applying Bayes theorem.

4.1 MDP model

MDP tuple is explained in chapter 2, MDP tuple of an IRL problem is without reward function and is represented as $< S, A, P, \gamma >$. Where,

- S is a discrete state space,
- A is a continuous action space,
- P(s' | s,a) is transition probability, which gives the probability distribution over next state s' conditioned on an agent executing an action 'a' when its current state is 's'.
- γ is a discount factor which conveys the difference in the importance of current rewards and future rewards.

Figure 4.1 and 4.2 illustrate an MDP model of ObjectWorld and Highway lane merging environment respectively, where actions are stochastic. Sampled action has 0.9 probability of going to expected next state and 0.1 probability of deviating from the expected next state.

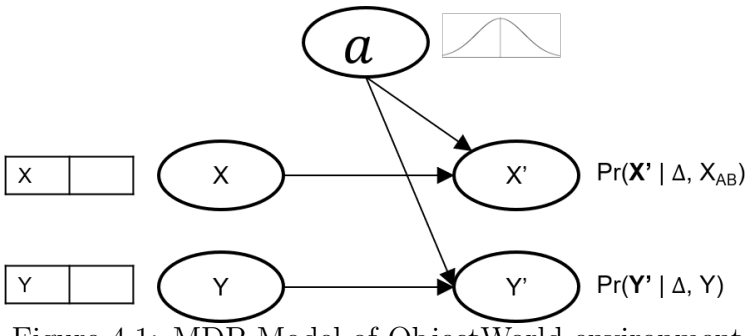


Figure 4.1: MDP Model of ObjectWorld environment.

In the case of ObjectWorld environment setting the X and Y coordinates are the state variables for a current state of an agent, action taken by an agent is sampled from normal

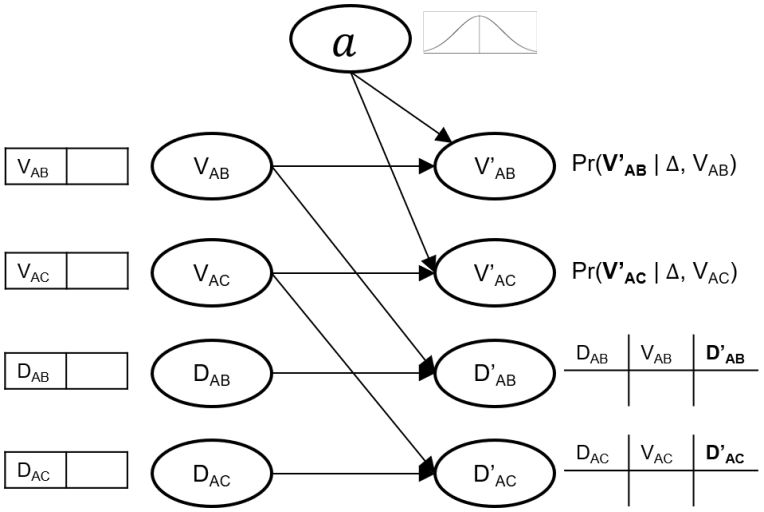


Figure 4.2: MDP Model of highway lane merging environment.

distribution over an angle. The angle determines the direction in which an agent moves by one unit. The X' and Y' coordinates are the state variables for next state of an agent. Whereas, in the case of Highway lane merging environment setting $V_{AB}, V_{AC}, D_{AB}, D_{AC}$ represent the state variables for current state of an agent. Δ indicates distribution over actions. Action is an acceleration of a car which is sampled from the distribution Δ for each state. $V'_{AB}, V'_{AC}, D'_{AB}, D'_{AC}$ represent state variables for the next state of an agent after taking an action that is sampled from the distribution. Refer to chapter 5 for a detailed description of ObjectWorld and highway lane merging environment settings.

Bellman equations for all $s \epsilon S$ and continuous action space satisfy equation 4.1 where, $\pi : S \rightarrow A$.

$$\begin{aligned} Q_\pi(s, \Delta, R) &= \int_a R(s)\Delta(a) + \gamma \sum_{s' \epsilon S} \int_a T(s, a, s')\Delta(a)V^\pi_{(s')} \\ V_\pi(s) &= \int_a R(s)\pi(s, a) + \gamma \sum_{s' \epsilon S} \int_a T(s, a, s')\pi(s, a)V^\pi_{(s')} \end{aligned} \tag{4.1}$$

4.2 Prior over rewards

Prior expresses one's belief about the uncertain quantity before some evidence is taken into account. Bayesian IRL (BIRL) has an ability to impose an informative prior over rewards which shows how BIRL can integrate domain knowledge to reduce uncertainty and ill-posing of IRL's reward estimation problem by limiting the space of possible reward functions. BIRL attempts to determine minimum distance from the real reward function and tries to maximize expected total reward.

While using continuous action space in IRL problem rewards are a function of the state alone ($R : S \rightarrow R$) because such IRL problems typically have limited information about the value of an action which is sampled from a distribution for each state. Also, having rewards as a function of the state alone helps to avoid over fitting.

Prior rewards are dependent on the characteristics of the selected problem and can be elicited from a purely subjective assessment of an experienced expert or is chosen according to some principle.

When we are completely agnostic about the prior, uniform distribution over the space $-R_{max} \leq R(s) \leq R_{max}$ for each state $s \epsilon S$ is used. Implementation uses an improper prior $P(R) = 1 \forall R \epsilon \mathbb{R}^n$. Many real world MDP's have parsimonious reward structures with most states having negligible reward values. In such situations, a Gaussian prior is used as shown in equation 4.2 [22].

$$P_{Gaussian}(R(s) = r) = \frac{1}{\sqrt{1\pi}\sigma} \exp^{-\frac{r^2}{2\sigma^2}}, \forall_{s \epsilon S} \tag{4.2}$$

.

4.2.1 Random Walk

To find if an action exists which satisfies following condition $Q_\pi(s, \pi(s), \widetilde{R}) < Q_\pi(s, a, \widetilde{R})$ in continuous action space, we incorporate a random walk algorithm in finite action space.

Finite continuous action spaces have upper and lower bounds that are defined for an action in an MDP. ObjectWorld environment has $\angle 0^o$ as a lower bound and $\angle 360^o$ as an upper bound on actions. While in the case of highway lane merging environment $-11.2 ft/sec^2$ is the lower bound on an acceleration and $11.2 ft/sec^2$ is the upper bound on an acceleration as observed in the NGSim dataset.

Random walk algorithm generates space of distributions ∂ for each state 's'. To decide the stopping criteria of the random walk algorithm we need to find the space covered by the random walk in finite space, which is a challenging problem. For a discrete action space we can compute x_n as shown in equation 4.3 [10].

$$x_n = \sum_{i=0}^{n} y_i \tag{4.3}$$

Where, n is number of steps in random walk, x_n gives cardinality of the walk, x_n is the number of distinct sites visited in 'n' step walk. y_n is the number of virgin sites visited on the n^{th} step. Here, $y_0 = 1$ because the starting event is counted as a visit to the starting site. For $n > 1$, y_n can take only values 0 or 1. x_n is non- decreasing and constrained to satisfy inequality $1 \leq x_n \leq n + 1$.

Expectation over x_n is represented by $< x_n >$ and is shown in equation 4.4.

$$\begin{aligned} < x_n > &= \sum_{i=0}^{n} < y_i > \\ &= \sum_{i=0}^{n} Pr\{y_i = 1\} \end{aligned} \tag{4.4}$$

Equation 4.5 illustrates the constraint for a RandomWalk at step 12, as it gives the space covered by RandomWalk in finite space. Where, π is a mathematical constant 22/7.

$$< y_i > \sim \begin{cases} \left(\frac{2}{\pi n}\right)^{\frac{1}{2}} & \text{1D linear chain} \\ \frac{\pi}{logn} & \text{2D square lattice} \end{cases} \tag{4.5}$$

Algorithm 6 illustrates the random walk algorithm, which generates an action distribution space ∂ for each state s. Δ represents the normal distribution over actions with μ as mean and $\delta = 0.1$ as the standard deviation. Line 12 indicates the stopping criteria for the random walk, if maximum possible space of a 2D lattice is covered by the random walk then random walk stops exploring action space further for a given state.

Algorithm 6: RandomWalk algorithm

Input: Number of steps n, Lower bound of action lb, Upper bound on action ub, State s

Output: Action distribution space ∂

1 Pick a mean μ uniformly at random from range$[lb, ub]$ for an action distribution Δ_0
2 $\partial[0] = \Delta_0; y_0 = 1; x_0 = y_0$
3 **for** $i = 1; i++; i < n$
4 Pick a new mean $\widetilde{\mu}$ uniformly at random from the neighbors of μ for an action distribution Δ_i
5 $\mu = \widetilde{\mu}$
6 $\partial[i] = \Delta_i$
7 **if** Δ_i is not in ∂
8 $y_i = 1$
9 **else**
10 $y_i = 0$
11 $x_i = x_{i-1} + y_i$
12 **if** $x_i > \frac{\pi}{logn}$
13 break;
14 **else**
15 continue;
16 $\partial_s = \partial$
17 **return** ∂_s

4.2.2 Transition Function

RandomWalk gives an action distribution space ∂ for each state 's' as $\partial_s = \{\Delta_0, \Delta_1, \Delta_2, ...\}$. Algorithm 7 helps to generate a transition function using continuous action spaces for object-world environment. Line 4 to 8 in the algorithm are computing the visitation frequencies. Based on the selected orientation in step 5, one unit distance from the current state in x-direction and y-direction is computed. Line 9 to 11 are computing the transition probabilities.

Algorithm 7: TRANSITION FUNCTION algorithm for object-world environment

Output: Transition function tp

1 **for** trajectory t in trajectorySet
2 $s \leftarrow trajectorySet[t].state$
3 **for** each Δ_i in ∂_s
4 **Repeat** for epochs e
5 $a_A \leftarrow$ sample action from$\Delta_i$distribution, update $x_{dir}$and y_{dir}using a_A
6 $s'[X] \leftarrow s[X] + x_{dir}$ // sampled x value
7 $s'[Y] \leftarrow s[Y] + y_{dir}$ // sampled y value
8 $vf[s, \Delta_i, s'] \leftarrow vf[s, \Delta_i, s'] + 1$
9 **for** each Δ_i in ∂_s
10 **for** each s' in all possible states
11 $tp[s, \Delta_i, s'] \leftarrow vf[s, \Delta_i, s']/sum(vf[s, \Delta_i, :])$

Algorithm 8 helps to generate a transition function using continuous action spaces for highway lane merging environment. Line 4 to 8 in the algorithm are computing the visitation frequencies. State variable V_{AC}'s value changes based on the selected acceleration for car in role A in step 5. As relative velocity between car in role A and car in role C changes in step 6, the distance between car in role A and car in role C changes. Line 9 to 11 are computing the transition probabilities.

Algorithm 8: TRANSITION FUNCTION algorithm for highway lane merging

Output: Transition function tp

for trajectory t in trajectorySet
  $s \leftarrow trajectorySet[t].state$
  for each Δ_i in ∂_s
    Repeat for epochs e
      $a_A \leftarrow$ sample action from$\Delta_i$distribution
      $s'[V_{AC}] \leftarrow s[V_{AC}] + a_A * t$ // sampled relative velocity between A & C
      $s'[X_{AC}] \leftarrow s[X_{AC}] - s[V_{AC}] * t - \frac{1}{2} a_A * t^2$ // sampled distance between A & C
      $vf[s, \Delta_i, s'] \leftarrow vf[s, \Delta_i, s'] + 1$
  for each Δ_i in ∂_s
    for each s' in all possible states
      $tp[s, \Delta_i, s'] \leftarrow vf[s, \Delta_i, s']/sum(vf[s, \Delta_i, :])$

4.3 BIRL with Continuous Actions

In order to learn rewards, the mean of posterior distribution must be computed, but it is analytically hard and complex to derive posterior. Posterior distribution is conditioned distribution of the uncertain quantity given the data. The BIRL technique generates the sample from posterior distribution and then returns the sample mean as an estimate of the true mean of the distribution instead of computing the mean of the posterior distribution.

BIRL uses Markov Chain Monte Carlo (MCMC) sampling technique to generate Markov Chain with step size δ in the region $\mathbb{R}^{|s|}/\delta$. the Policy Walk sampling algorithm that was explained in chapter 3 is considered an efficient method as it keeps track of optimal policy for the current reward function using MCMC sampling. Algorithm 5 used a policy iteration algorithm for the policy optimization step. The policy iteration algorithm makes use of line search technique in order to optimize policy, by first finding the direction in which objective function can be optimized and then by determining the step size to move in that direction using methods, such as gradient ascent/descent.

Policy Walk with policy iteration can be expensive for a continuous or large action space. So, Policy Walk is modified to use Trust Region Policy Optimization (TRPO) for evaluating

optimal policy. TRPO first selects the step size, and then selects the step direction in order to restrict each step for preventing the guess from stepping too far away from the desired optimal policy.

Algorithm 9 shows revised Policy Walk algorithm with continuous actions. Step 2 is using TRPO instead of policy iteration algorithm in order to find an optimal policy. In the case of continuous action space the policy is in the form of parameterized stochastic policy. Here, parameterization is based on the Gaussian Multi-Layered Perceptron. Step 6 is an additional step and is required for continuous action space to generate action distribution space with random walk for a given state. Step 7 and 8 are also revised to incorporate continuous action spaces, as Q value computation requires a transition function with continuous action space.

Algorithm 9: PolicyWalk algorithm with Continuous Actions

Input: Distribution P, MDP M/R, Step Size δ, Epochs i
Output: Rewards $\mathbf{R}$

1 Pick random reward vector $\mathbf{R}\ \epsilon\ \mathbb{R}^{|s|}\ /\ \delta$
2 $\pi :=$ TRPO(M/R, $\mathbf{R}$)
3 Repeat
4 (a) Pick a reward vector $\widetilde{\mathbf{R}}$ uniformly at random from the neighbors of
5 $\mathbf{R}$ in $\mathbb{R}^{|s|}\ /\ \delta$
6 (b) Using RandomWalk(π) Generate action distributions space ∂, $\forall(s)\epsilon(S)$
7 (c) Compute $Q^{\pi}(s, \Delta, \widetilde{\mathbf{R}})$ for all $(s, \Delta)\epsilon S, \partial$.
8 (d) **If** $\exists(s, \Delta)\epsilon(S, \partial)$, $Q(s, \pi(s), \widetilde{\mathbf{R}}) < Q(s, \Delta, \widetilde{\mathbf{R}})$
9 (i) $\widetilde{\pi} :=$ TRPO(M/R, $\widetilde{\mathbf{R}}$)
10 (ii) Set $\mathbf{R} := \widetilde{\mathbf{R}}$ and $\pi := \widetilde{\pi}$ with probability min $\left\{1, \frac{P(\widetilde{\mathbf{R}}, \widetilde{\pi})}{P(\mathbf{R}, \pi)}\right\}$
11 **else**
12 (i) Set $\mathbf{R} := \widetilde{\mathbf{R}}$ with probability min $\left\{1, \frac{P(\widetilde{\mathbf{R}}, \pi)}{P(\mathbf{R}, \pi)}\right\}$
13 **return** $\boldsymbol{R}$

After predicting the posterior rewards of an expert agent with continuous action space these rewards are used to complete underlying MDP and predict an optimal policy.

4.4 Summary

This chapter explains the proposed model-based IRL with continuous action space for preference(reward) learning in detail. First, MDP model and assignment of prior over rewards is explained. Then, description of the random walk algorithm which is used to generate action spaces is covered. After that, transition function algorithm for object-world and highway lane merging is explained, which is followed by the explanation of the PolicyWalk algorithm with continuous action spaces

Chapter 5

DATA AND DOMAIN SETUP

This chapter describes two environment settings used for conducting experiments: Simplified Object-World Environment and Highway Lane Merging Environment. While Simplified Object-World is not meant to be the major challenging task, it allows us to compare our approach to other methods of generalizing the demonstrations.

5.1 Simplified Object-World Environment

Object-world is a NxN grid of states with five actions per state, corresponding to steps in each direction and staying in place [18].

Randomly placed objects populate the Object-World, and each is assigned a color. Agent gains positive reward (represented with Yellow color) in states that are both within 2 cells range of object color 1 (Blue) and 1 cell range of object color 2 (red). Negative reward (represented with Purple color) is obtained by an agent when it is within 2 cells of object color 1 (Blue), and zero otherwise. Figure 5.1 illustrates how the true reward is displayed with the brightness of color of grid cell where, purple is the lowest reward and yellow is the highest reward. Figure 5.1 shows various settings of Object-World.

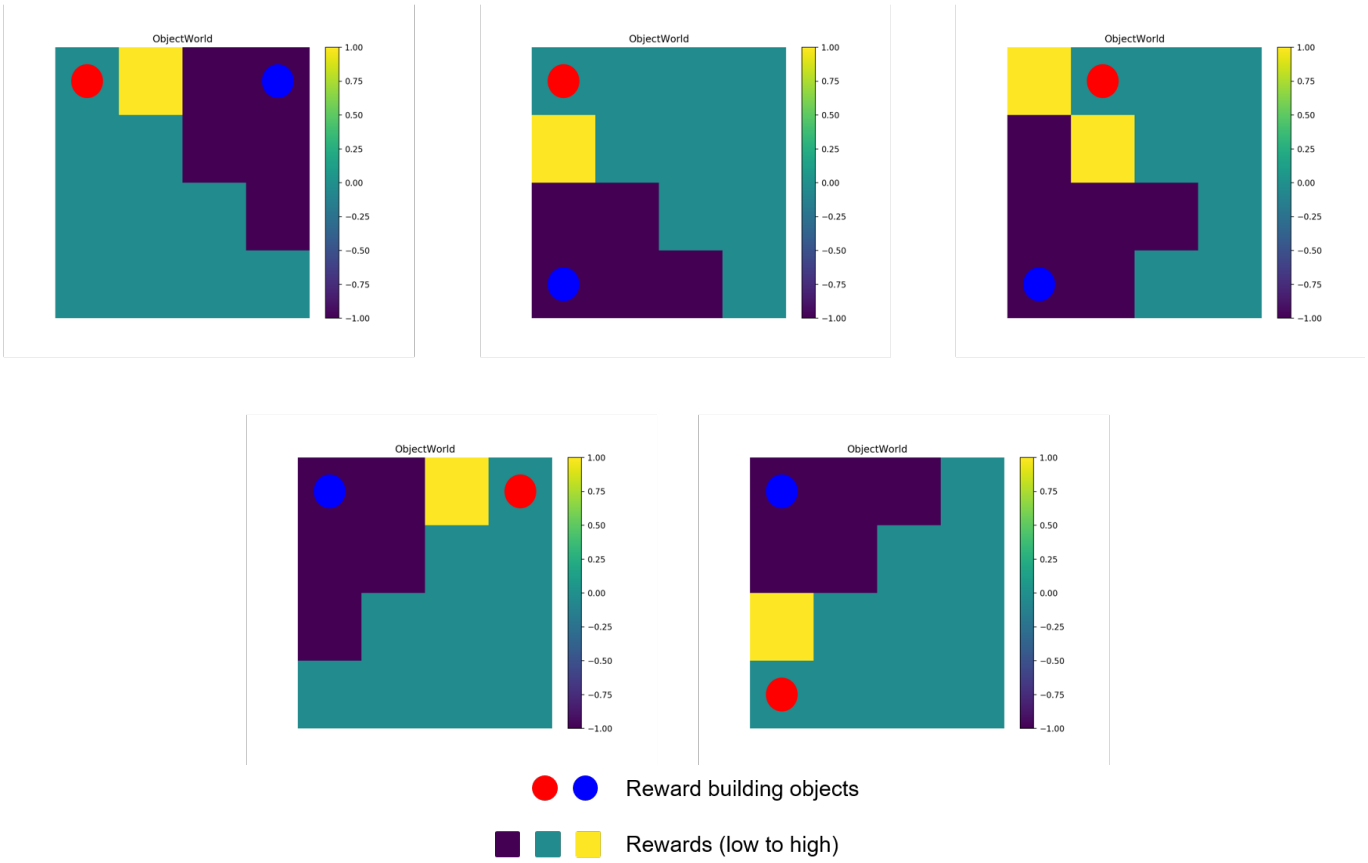


Figure 5.1: Example 4x4 Object-World.Cells colored green have reward 0, cells colored yellow have reward 1, and cells colored purple have reward -1. Objects are represented by circles with red or blue color.

5.1.1 Object-World Model Description

Object-World state space depends upon the size of the grid. For 4x4 Object-World, state space is 16 grid cells. These 16 states have two state variables each: x-coordinate and y-coordinate.

Object-World environment is extended for continuous action space, with 1-dimensional actions, where action is an angle which gives the direction in which agent takes a step. Action space has 0^o as a lower bound and 360^o as an upper bound. Each action helps an agent to move by 1 unit in specified direction(angle) inside the grid.

The algorithms were provided sample paths of length 4. The goal is to reach states with

positive reward and avoid states with negative reward.

5.2 Highway Lane Merging Environment

This research focuses on a task that is daunting even for experienced human drivers – safely merging into a congested freeway from an acceleration ramp without lengthy waits. In order to facilitate this, an agent must account for the uncertainty both in the environment and in the driving patterns of other manned or unmanned cars.

The objective is to automatically learn the driving pattern of other cars sharing the environment with an autonomous car in congested traffic situations, with a focus on freeway lane merging from acceleration ramp using NGSim dataset.

5.2.1 Next Generation SIMulation [11]

Federal Highway Administration's (FHWA's) Traffic Analysis Tools Program launched the Next Generation SIMulation (NGSIM) program in order to encourage the use of microsimulation systems and to confirm that the tools provide precise results. NGSIM data-sets are freely available for the academic research community.

NGSIM program provides high-quality real-world traffic datasets with their corresponding data descriptions. The dataset has detailed vehicle trajectories, which were collected using digital video cameras and the specific location of each vehicle on approximately 500-meter (1640 feet) section of roadway is recorded every 1/10th of a second. Figure 5.2 shows digital video camera used to record vehicle trajectory data.

This work uses Interstate 80 (I-80) Freeway Dataset which was the first of several datasets collected under the NGSIM program. Figure 5.3 shows the aerial photograph of the I-80 study area covered by, each of the seven video cameras and the schematic drawing of the number of lanes and location of the Powell Street on ramp within the I-80 study area.

Figure 5.2: A digital video camera mounted on top of a building that overlooks I-80 is recording vehicle trajectory data [11].

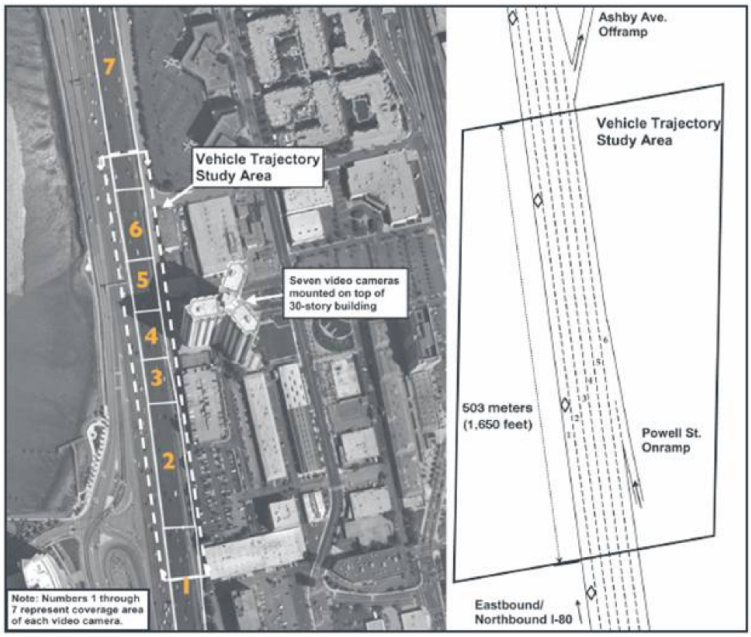


Figure 5.3: The aerial photograph on the left shows the extent of the I-80 study area in relation to the building from which the video cameras were mounted and the coverage area for each of the seven video cameras. The schematic drawing on the right shows the number of lanes and location of the Powell Street onramp within the I-80 study area [11].

5.2.2 NGSIM Metadata Description

Vehicle trajectory data is collected from eastbound I-80 in the San Francisco Bay area in Emeryville, CA, on April 13, 2005 and consists of total 45 minutes of data, recorded in three 15-minutes time intervals: 4:00pm to 4:15pm; 5:00pm to 5:15pm; and 5:15pm to 5:30pm.

Each of these three data sets have more than a million rows containing 2,052, 1,836 and 1,790 tracked cars, respectively.

NG-VIDEO transcribed the vehicle trajectory data from the video and provided the precise location of each vehicle within the study area every 1/10th of a second. Figure 5.4 shows snapshot of processed video which shows all the vehicles detected in the frame and each vehicle marked with an unique ID.

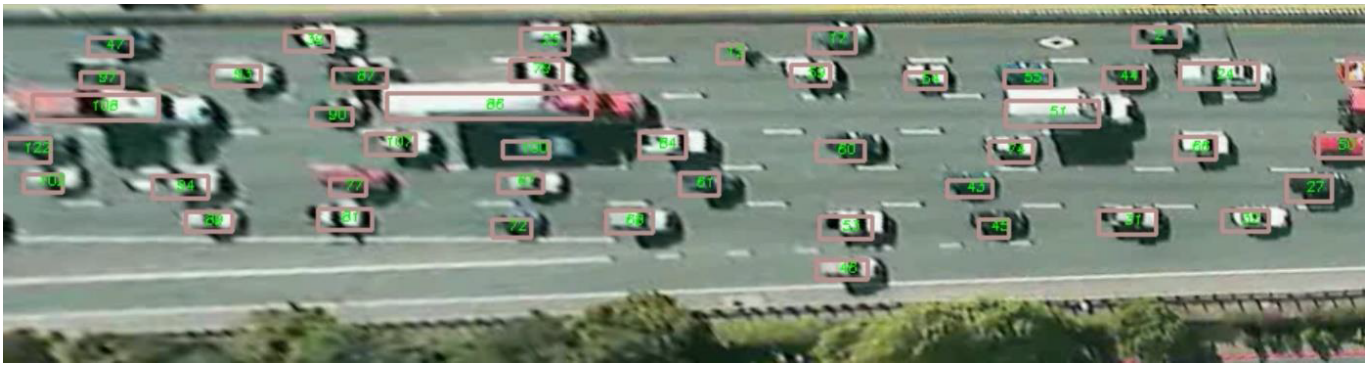


Figure 5.4: Snapshot of the processed video from NGSIM I-80, where vehicles in a frame are marked with unique IDs [11].

The recorded video data was then transcribed into vehicle trajectory data using NG-VIDEO, a customized software application developed for NGSIM program. Figure 5.5 illustrates Snapshot of transcribed I-80 freeway merging dataset from 5:00pm to 5:15pm time slot.

284	63	526	1113436773200	41.375	66.469	6042864.125	2133138.844	15.4	5.9	2	44.55	0.00	4	0	0	0.00	0.00
284	64	526	1113436773300	41.375	71.069	6042863.571	2133143.309	15.4	5.9	2	44.55	0.00	4	0	0	0.00	0.00
284	65	526	1113436773400	41.374	75.570	6042863.016	2133147.775	15.4	5.9	2	44.55	0.00	4	0	0	0.00	0.00
284	66	526	1113436773500	41.373	80.070	6042862.462	2133152.241	15.4	5.9	2	44.55	0.00	4	0	0	0.00	0.00
284	67	526	1113436773600	41.372	84.607	6042861.904	2133156.736	15.4	5.9	2	44.55	-6.14	4	0	0	0.00	0.00
284	68	526	1113436773700	41.369	89.058	6042861.350	2133161.189	15.4	5.9	2	43.77	-10.10	4	0	0	0.00	0.00
284	69	526	1113436773800	41.369	93.368	6042860.818	2133165.481	15.4	5.9	2	43.00	-7.12	4	0	0	0.00	0.00
284	70	526	1113436773900	41.369	97.565	6042860.307	2133169.608	15.4	5.9	2	42.89	3.56	4	0	0	0.00	0.00
284	71	526	1113436774000	41.376	101.844	6042859.800	2133173.755	15.4	5.9	2	43.84	11.20	4	0	0	0.00	0.00
284	72	526	1113436774100	41.396	106.346	6042859.274	2133178.163	15.4	5.9	2	45.83	11.20	4	0	0	0.00	0.00
284	73	526	1113436774200	41.388	111.064	6042858.686	2133182.834	15.4	5.9	2	48.40	11.20	4	0	0	0.00	0.00

Figure 5.5: Snapshot of transcribed I-80 freeway merging dataset of 5:00pm from 5:15pm time slot [11].

This transcribed data consists of 18 attributes for each tuple of the dataset. Table 5.1 shows the Vehicle Trajectory File Data Dictionary, for these 18 attributes.

Table 5.1: Vehicle Trajectory File Data Dictionary of I-80 Freeway Dataset [11].

Column No.	Name	Description	Units
1	Vehicle ID	Vehicle identification number (ascending by time of entry into section)	Number
2	Frame ID	Frame Identification number (ascending by start time)	1/10 of a second
3	Total Frames	Total number of frames in which the vehicle appears in this data set.	1/10 of a second
4	Global Time (Epoch Time)	Elapsed time since Jan 1, 1970.	Milliseconds
5	Local X	Lateral (X) coordinate of the front center of the vehicle with respect to the left-most edge of the section in the direction of travel.	Feet
6	Local Y	Longitudinal (Y) coordinate of the front center of the vehicle with respect to the entry edge of the section in the direction of travel.	Feet
7	Global X	X Coordinate of the front center of the vehicle based on CA State Plane III in NAD83.	Feet
8	Global Y	Y Coordinate of the front center of the vehicle based on CA State Plane III in NAD83.	Feet
9	Vehicle Length	Length of vehicle	Feet
10	Vehicle Width	Width of vehicle	Feet
11	Vehicle Class	Vehicle type: 1 - motorcycle, 2 - auto, 3 - truck	Text
12	Vehicle Velocity	Instantaneous velocity of vehicle	Feet/Second
13	Vehicle Acceleration	Instantaneous acceleration of vehicle	Feet/Second Square
14	Lane Identification	Current lane position of vehicle. Lane 1 is farthest left lane; lane 6 is farthest right lane. Lane 7 is the on-ramp at Powell Street, and Lane 9 is the shoulder on the right-side.	Number
15	Preceding Vehicle	Vehicle Id of the lead vehicle in the same lane. A value of '0' represents no preceding vehicle - occurs at the end of the study section and off-ramp due to the fact that only complete trajectories were recorded by this data collection effort (vehicles already in the section at the start of the study period were not recorded).	Number
16	Following Vehicle	Vehicle Id of the vehicle following the subject vehicle in the same lane. A value of '0' represents no following vehicle - occurs at the beginning of the study section and on-ramp due to the fact that only complete trajectories were recorded by this data collection effort (vehicle that did not traverse the downstream boundaries of the section by the end of the study period were not recorded).	Number
17	Spacing (Space Headway)	Spacing provides the distance between the front-center of a vehicle to the front-center of the preceding vehicle.	Feet
18	Headway (Time Headway)	Headway provides the time to travel from the front-center of a vehicle (at the speed of the vehicle) to the front-center of the preceding vehicle. A headway value of 9999.99 means that the vehicle is traveling at zero speed (congested conditions).	Seconds

5.2.3 A-B-C Role System

After developing a deep understanding of NGSIM data set, it was scrubbed to use in this research and A-B-C role systems for each car that merges into the freeway was isolated. A-B-C Role System is articulated as shown in the figure 5.6. Self-driving car (role B: Red Car) which is trying to merge onto the highway from an acceleration ramp perceives behavior of following car (role A: Blue car) driving on the outermost highway lane near the merging area while comprising preceding car's (role C: Green Car) attributes, in order to merge safely onto the highway.

Driving pattern of a car driving on the outermost highway lane near the merging area is called as "expert's trajectory" (i.e. Car in role A). Expert's trajectory can be thought of as a series of state-action pairs in a MDP.

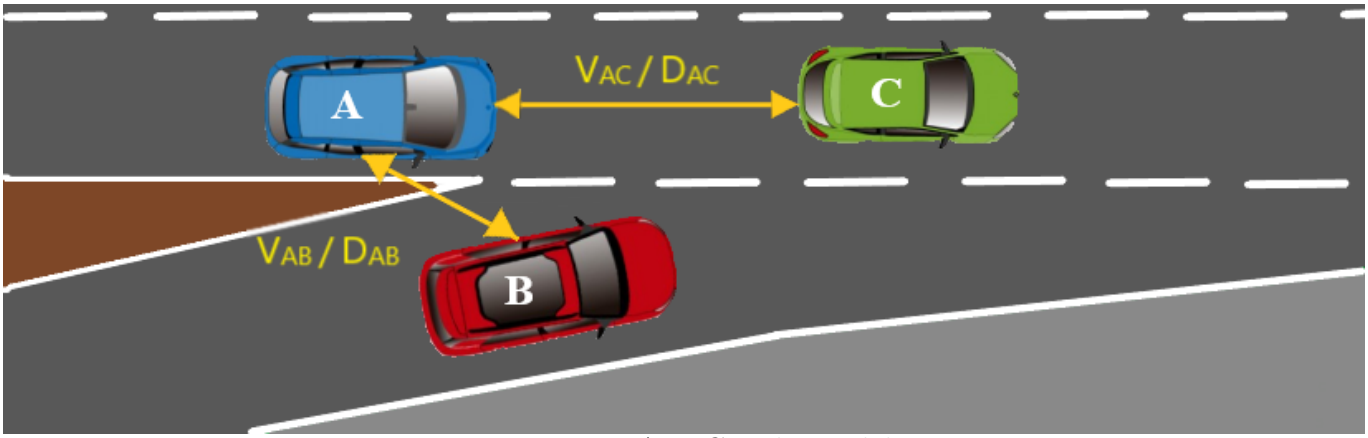


Figure 5.6: A-B-C Role Model

Further, the data is filtered to isolate instances where the car in role A speeds up to prevent B from merging in front of it. 60 such instances were discovered in the entire data set. These provides sufficient data for learning a corresponding aggregate driver model. For experiments of this research, 12 trajectories with risk prone Role A behaviour were extracted from NGSIM's 5:00 to 5:15 dataset.

5.2.4 Highway Lane Merging Model Description

Goal is to learn preferences of car in role A using IRL. Thus the state space of Highway Lane Merging environment is based on car in role A. State space is discrete and is characterized by values of four state variables(D_{AC}, D_{AB}, V_{AC} and V_{AB}), these variables help in deciding the distance and relative velocity between the respective cars. Recall that the y coordinate is from front center of the vehicle with respect to the entry edge of the section in the direction of travel for any car.

Figure 5.7 shows State space variables: Spacing (D_{AC} and D_{AB}) and Relative velocity (V_{AC} and V_{AB}) along with their corresponding bins. Each state variable $D_{AC}, D_{AB}, V_{AC}, V_{AB}$ is discretized by dividing into 5 bins.

- D_{AC} is the horizontal Distance between vehicle in role A and C. and is computed for each frame with: $D_{AC} = Y_A - Y_C$.
- D_{AB} is the horizontal Distance between vehicle in role A and B. It is computed for each frame with: $D_{AB} = Y_A - Y_B$.
- V_{AC} is the instantaneous relative Velocity of vehicle in role A and C. It is computed for each frame with: $V_{AC} = V_A - V_C$.
- V_{AB} is the instantaneous relative Velocity of vehicle in role A and B. It is computed for each frame with: $V_{AB} = V_A - V_B$.

		Bin 1	Bin 2	Bin 3	Bin 4	Bin 5
State Variable 1	DAC	(-∞, -85.000)	[-85.000 ,-65.000)	[-65.000, -50.000)	[-50.000, -35.000)	[-35.000, ∞)
State Variable 2	DAB	(-∞, -45.000)	[-45.000,-35.000)	[-35.000,-25.000)	[-25.000,-15.000)	[-15.000, -∞)
State Variable 3	VAC	(-∞ ,-5.000)	[-5.000,-2.000)	[-2.000,0.00)	[0.00,3.00)	[3.000, ∞]
State Variable 4	VAB	(∞ , -5.000)	[-5.000,-2.000)	[-2.000,0.00)	[0.00,3.00)	[3.000, ∞]

Figure 5.7: State space variables: Spacing (D_{AC} and D_{AB}) and Relative velocity (V_{AC} and V_{AB}) along with their corresponding bins

These 4 state variables each divided into 5 bins gives a state space of $5^4 = 625$ states. A state number is obtained using the bin numbers of the state variables by using equation: $(V_{AB_{binNum}} - 1) \times (5^3) + (D_{AB_{binNum}} - 1) \times (5^2) + (V_{AC_{binNum}} - 1) \times (5^1) + (D_{AC_{binNum}}) \times (5^0)$.

Action space is continuous and is based on the instantaneous acceleration values. Lower bound of the accelerations based on the NGSIM dataset is -11.2 $Feet/second^2$ and upper bound is 11.2 $Feet/second^2$.

5.2.5 Trajectory Extraction of Risk-prone Vehicle Role A

In-order to model highway lane merging problem with A-B-C role model, we need to extract data of only vehicles into one of the three A, B or C vehicle roles. The following steps illustrate trajectory extraction of risk-prone vehicle role A with respect to vehicle role B from the NGSIM dataset.

1. Select all the tuples in which Lane Identification column has values 6 or 7. Vehicle lane 6 is the rightmost lane on the highway and lane 7 is the onramp.

2. Identify and extract all the vehicles in role 'B' (i.e. the vehicle about to merge onto the highway from ramp).

3. Identify and extract vehicles in role A and vehicles in role C with the same frame ID as that of the vehicle B. Vehicle's in role A will have y-coordinate value just less than (following car for B) that for vehicle B in same frame whereas, vehicle in role C has y-coordinate value just more than (preceding car for B) that for vehicle B.

4. Back propagate to get the complete trajectories for all the three vehicles from the time they entered study area till the end of the study area.

5. Concatenate required attributes (showed in 5.8) for tuples of vehicle role A, B and, C with the common frame ID's.

6. Extract all trajectories of vehicle in role A, with details of the corresponding vehicle B and vehicle C in each frame.

1. B's Vehicle ID	2. Frame ID	3. B's x-coordinate	4. B's y-coordinate
5. B's Vehicle Length	6. B's Vehicle Width	7. B's Vehicle Velocity	8. B's Vehicle Acceleration
9. B's Lane Identification	10. B's Preceding Vehicle	11. B's Following Vehicle	12. B's Spacing
13. A's Vehicle ID	14. A's x-coordinate	15. A's y-coordinate	16. A's Vehicle Length
17. A's Vehicle Width	18. A's Vehicle Velocity	19. A's Vehicle Acceleration	20. A's Lane Identification
21. A's Preceding Vehicle	22. A's Following Vehicle	23. A's Spacing	24. C's Vehicle ID
25. C's x-coordinate	26. C's y-coordinate	27. C's Vehicle Length	28. C's Vehicle Width
29. C's Vehicle Velocity	30. C's Vehicle Acceleration	31. C's Lane Identification	32. C's Preceding Vehicle
33. C's Following Vehicle	34. C's Spacing	35. Distance between A & B	36. Distance between A & C
37. Velocity difference between A & B	38. Velocity difference between A & C	39. State of A (Expert Agent)	

Figure 5.8: Attribute names of extracted 39 columns

7. Identify and extract vehicles in Role A which show risk-prone behavior with respect to vehicle role B by increasing the speed and by not letting vehicles in role B merge onto the highway.

8. Verify the extracted trajectories by simulating them in MATSim Senozon's Via Simulator.

5.3 Summary

This chapter describes two environment settings used for conducting experiments: simplified object-world environment and highway lane merging environment.

Chapter 6

RESULTS AND DISCUSSION

Section 6.1 illustrates Inverse Learning Error results on simplified object-world environment using proposed algorithm: Bayesian IRL with continuous action spaces. Section 6.2 gives learnt reward structure of simplified object world environment and highway lane merging environment.

6.1 Inverse Learning Error

Inverse Learning Error(ILE) is a method used to compare results of IRL, ILE compares value functions of the policies obtained from optimally solving the MDP generated by using the learnt reward function, to that of the value function obtained from solving the expert's MDP.

If R_E^L is the reward function learnt by a learning agent and π_E^* is the optimal policy followed by an expert (Agent 'E') and π_E^L is the policy of the learner (Agent 'L') then, Inverse Learning Error (ILE) can be defined as showed in equation 6.1

$$ILE = ||V^{\pi_E^*} - V^{\pi_E^L}|| \tag{6.1}$$

Where, $V^{\pi_E^*}$ is optimal value function of the expert E's MDP and $V^{\pi_E^L}$ is the value function obtained by following the learnt policy π_E^L on E's MDP.

An interesting property of ILE is that it monotonically increases as the policies of the expert and the learner diverge. On the other hand, when $\pi_E^* = \pi_E^L$ and the learner follows the exact same policy as that of the expert, ILE comes out to be 0.

Following Table illustrates results for ILE metric, averaged over 100 trials, for 5 different settings of Object World.

Table 6.1: Inverse Learning Error for ObjectWorld Domain.

	BIRL with continuous actions	GPIRL
Object World Setting 1	3.093	200.420
Object World Setting 2	2.288	65.857
Object World Setting 3	2.6746	103.13
Object World Setting 4	2.954	51.464
Object World Setting 5	3.229	25.533

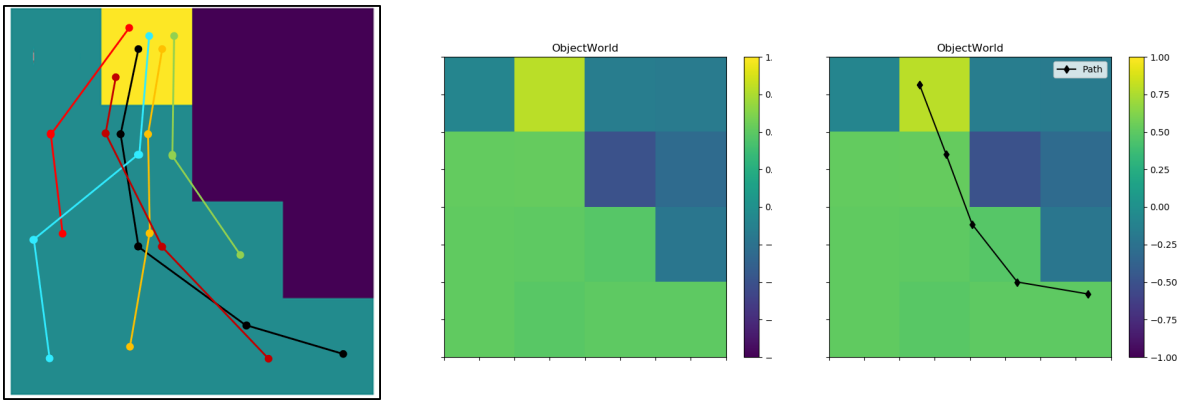
6.2 Learnt Reward Structure

This section shows the learnt reward structure of an agent along with one sample trajectory which is simulated using the learnt reward structure. For each environment setting the Policy Walk algorithm is executed for 25 times to acquire 25 samples of the learnt reward structure. As the prior over rewards is obtained with an uniform distribution, there was a need to run PolicyWalk multiple times. An average of learnt reward structures is computed for each environment setting.

6.2.1 Simplified Object World Environment

6.2.1.1 Results: BIRL with continuous action spaces

Figure 6.1, 6.2, 6.3, 6.4, and 6.5 illustrate simplified object-world setting 1, 2, 3 ,4, and 5 respectively. Where, sub-figure (a) shows sample trajectories used for training purpose, the different colored lines indicate the trajectory followed by an expert agent at an instance, also, these trajectories can have different start state, sub-figure (b) shows learned reward structure, and (c) shows sample trajectory generated using learned reward structure.

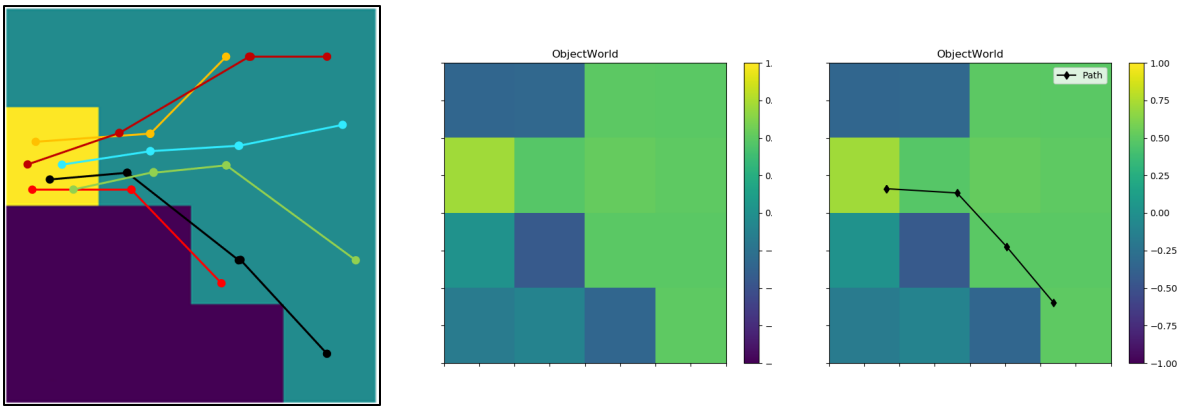


(a) (b) (c)

Figure 6.1: ObjectWorld Setting 1: (a) Simulated sample trajectories used for training , (b) Learned reward structure, (c) Sample trajectory generated using learned reward structure

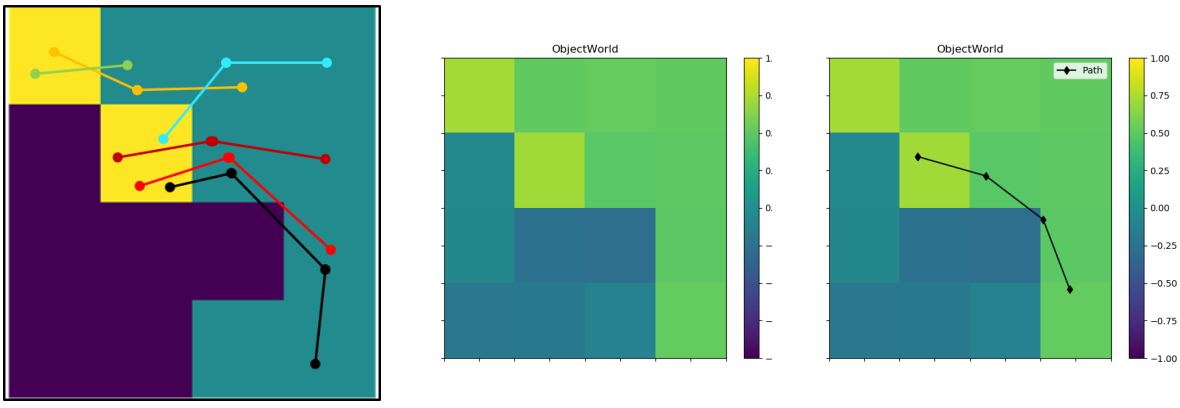
Reward values range from -1 to 1. As can be seen in these figures, yellow color represents an optimal positive reward, purple color represents an optimal negative reward, and the green indicates neutral rewards.

It can be observed from the figures that the learnt reward structure has a similar arrangement as seen in the true reward structure of the environment.



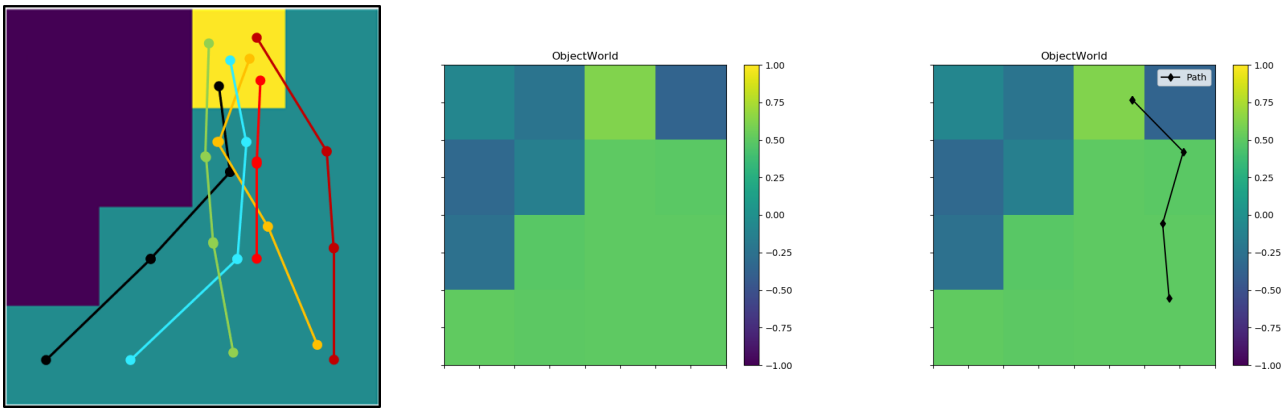
(a) (b) (c)

Figure 6.2: ObjectWorld Setting 2: (a) Simulated sample trajectories used for training , (b) Learned reward structure, (c) Sample trajectory generated using learned reward structure



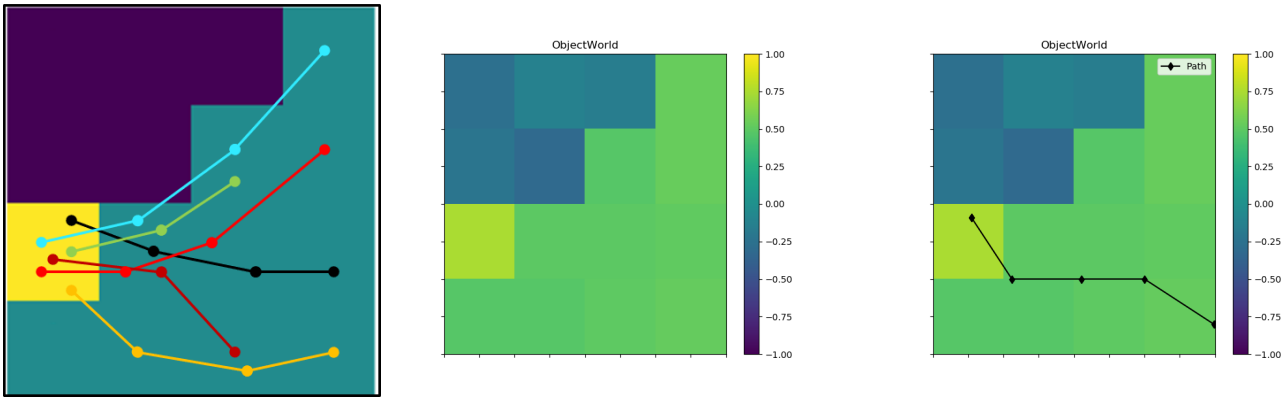
(a) (b) (c)

Figure 6.3: ObjectWorld Setting 3: (a) Simulated sample trajectories used for training , (b) Learned reward structure, (c) Sample trajectory generated using learned reward structure



(a) (b) (c)

Figure 6.4: ObjectWorld Setting 4: (a) Simulated sample trajectories used for training , (b) Learned reward structure, (c) Sample trajectory generated using learned reward structure



(a) (b) (c)

Figure 6.5: ObjectWorld Setting 5: (a) Simulated sample trajectories used for training , (b) Learned reward structure, (c) Sample trajectory generated using learned reward structure

6.2.1.2 Results: GPIRL

Figure 6.6, 6.7, 6.8, 6.9, and 6.10 illustrate simplified object-world setting 1, 2, 3 ,4, and 5 respectively. Where, sub-figure (a) shows learned reward structure, and (b) shows sample policy generated using learned reward structure. Here, rewards have discrete values and actions are discrete.

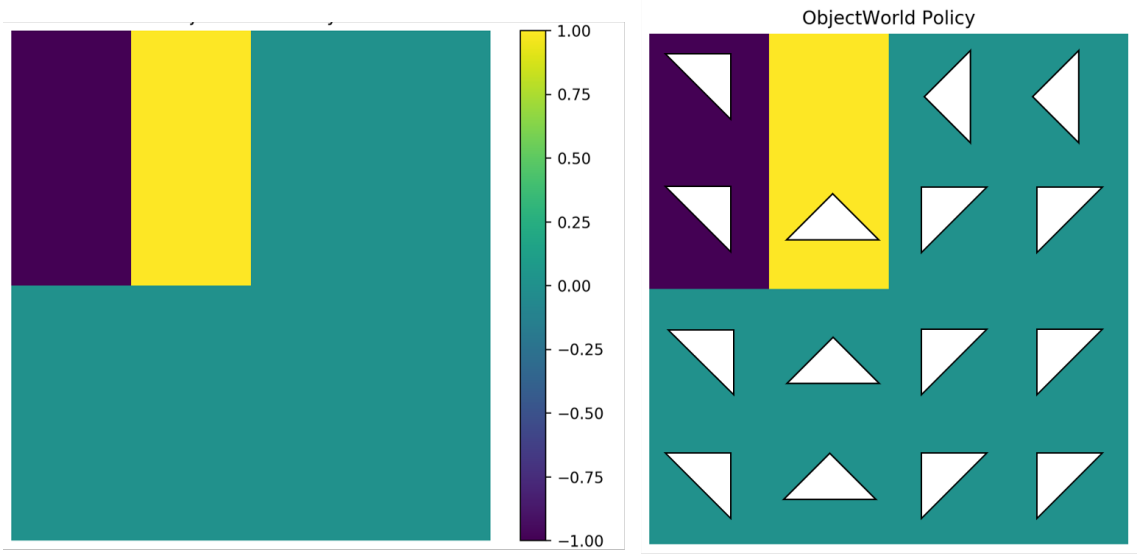
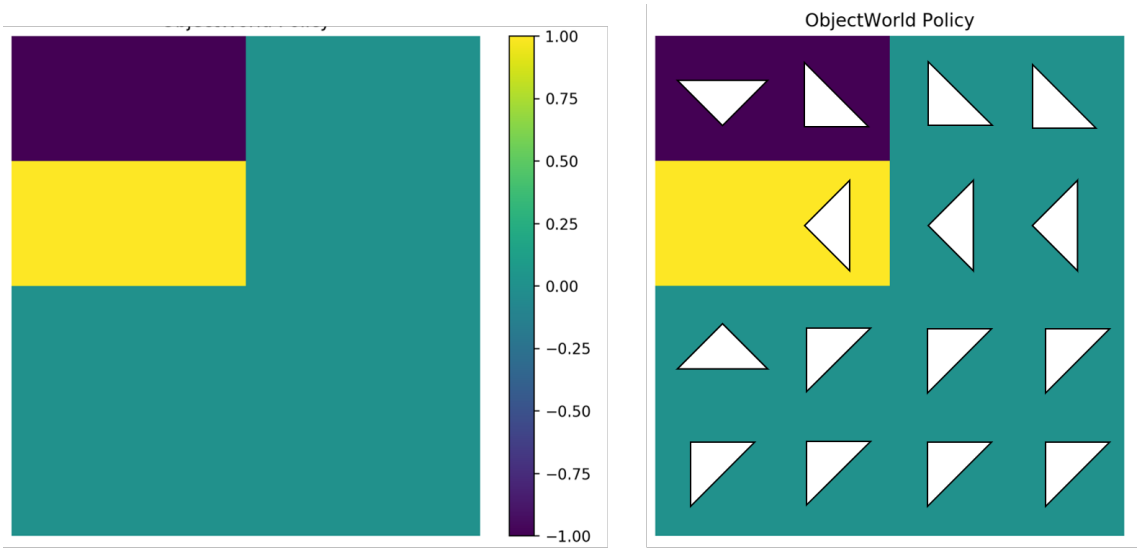
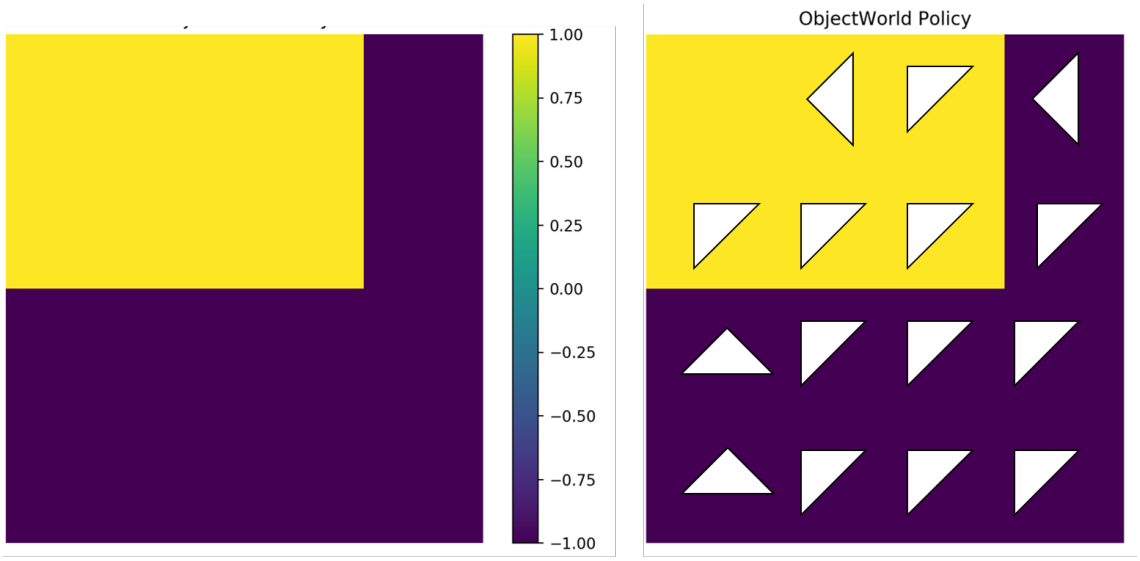


Figure 6.6: y-coordinate difference between expert agent's trajectory and learning agent's trajectory for highway lane merging scenario 1



(a) (b)

Figure 6.7: y-coordinate difference between expert agent's trajectory and learning agent's trajectory for highway lane merging scenario 2



(a) (b)

Figure 6.8: y-coordinate difference between expert agent's trajectory and learning agent's trajectory for highway lane merging scenario 3

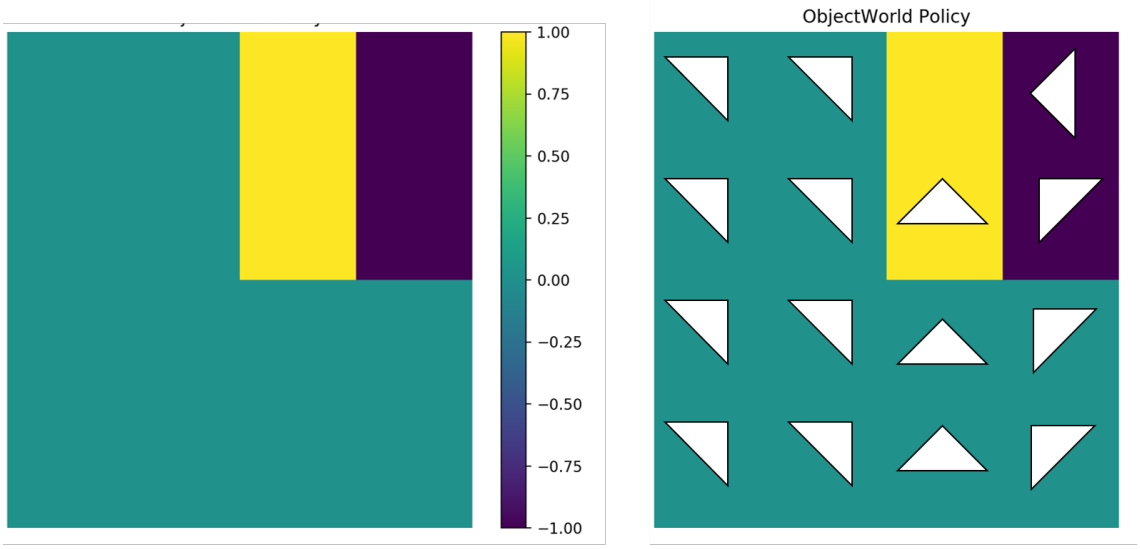


Figure 6.9: y-coordinate difference between expert agent's trajectory and learning agent's trajectory for highway lane merging scenario 4

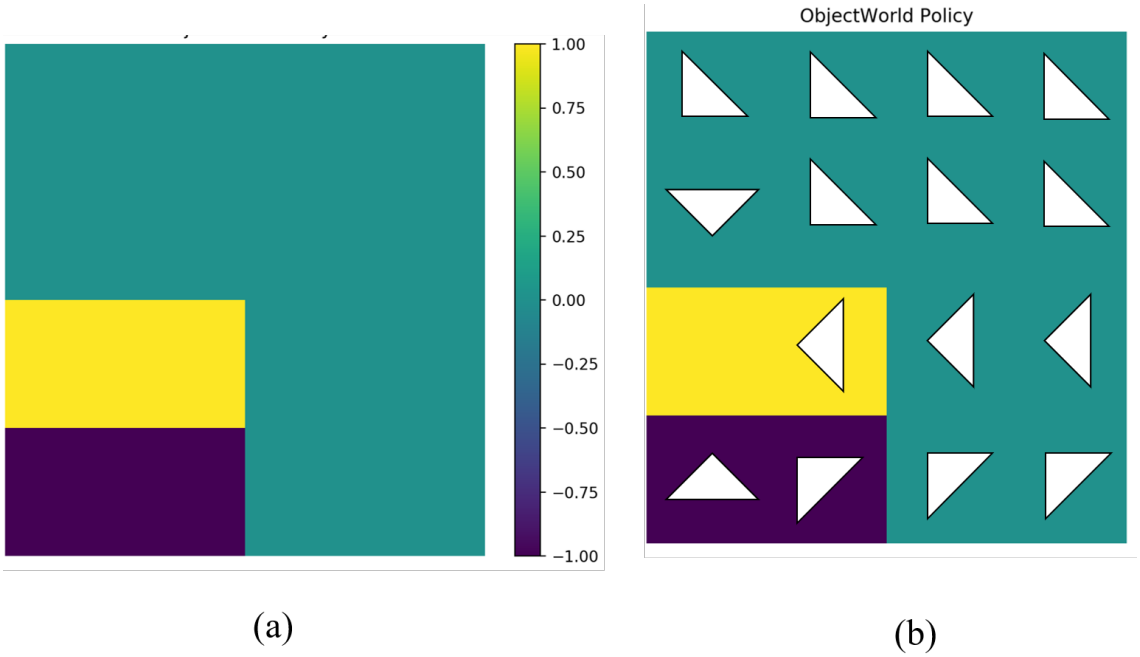


Figure 6.10: y-coordinate difference between expert agent's trajectory and learning agent's trajectory for highway lane merging scenario 5

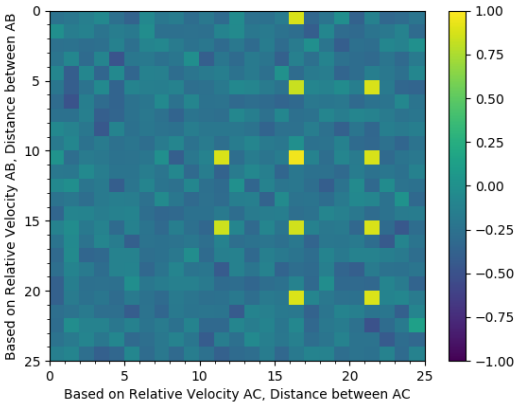
6.2.2 Highway Lane Merging Environment

This section presents learnt reward structure of highway lane merging environments. As mentioned in chapter 4 rewards are functions of state alone because IRL problems typically have limited information about the value of an action.

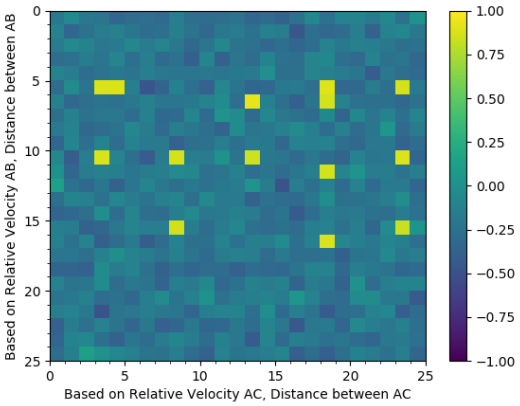
The state space layout of 625 states is shown in the form of a 25×25 grid where, state numbers are calculated sequentially and the state space layout is used to illustrate learned reward structure in figure 6.11a to 6.12f.

Snapshots in figure 6.13 to 6.16 show the highway lane merging scenario 1 to 12. Where, horizontal axis indicates the y-coordinates values and vertical axis indicates the x-coordinate values from NGSim dataset. Trajectory of car in role B is indicated with a Red color, trajectory of a car in role C is shown with a Green color. An original trajectory of car in role A is shown with Cyan color and a learnt trajectory of a car in role A is in Blue color. It can be observed that the trajectories of car in role A (indicated with Cyan color and Blue color) match closely to each other, thus it can be deduced that the learnt reward structure is informative and useful in learning process of the learning agent.

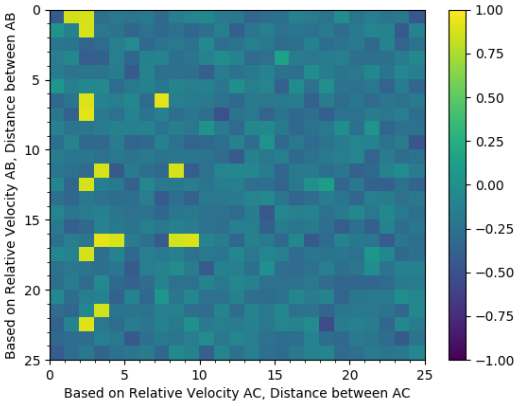
Figure 6.17 to 6.28 show y-coordinate difference between expert agent's trajectory and learning agent's trajectory for respective highway lane merging scenario. Blue line indicates the original car role A trajectory and red line indicates the learned car role A trajectory. The x-coordinate shows the time steps of the trajectory and y-coordinate gives the y-coordinate of the car in role A.



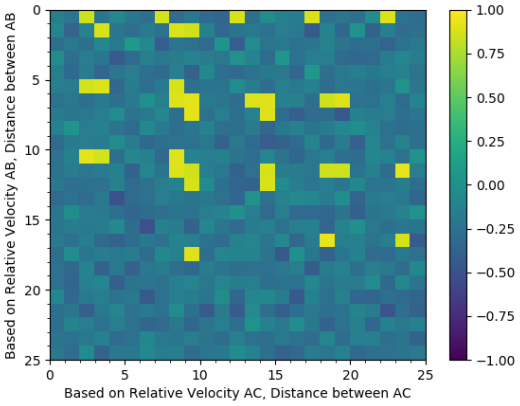
(a) Scenario 1



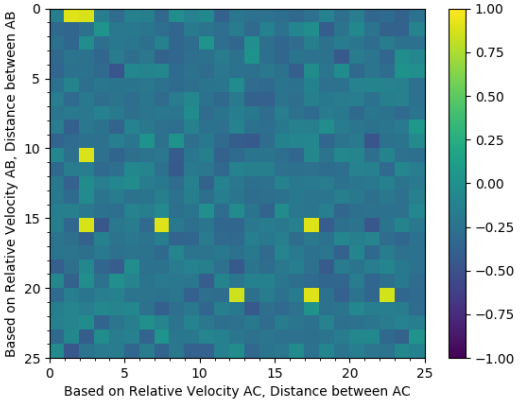
(b) Scenario 2



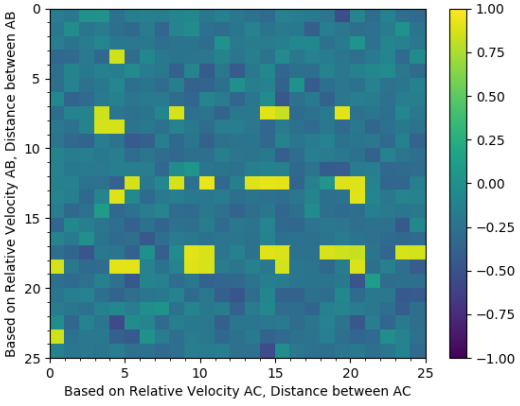
(c) Scenario 3



(d) Scenario 4

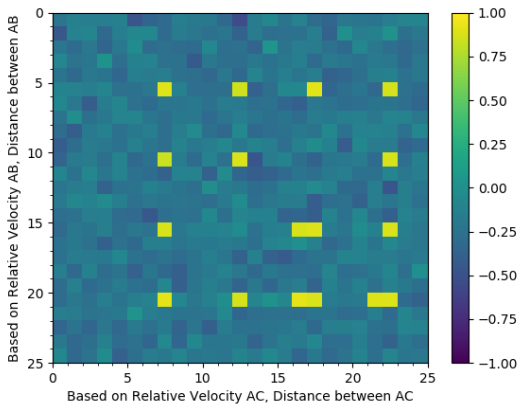


(e) Scenario 5

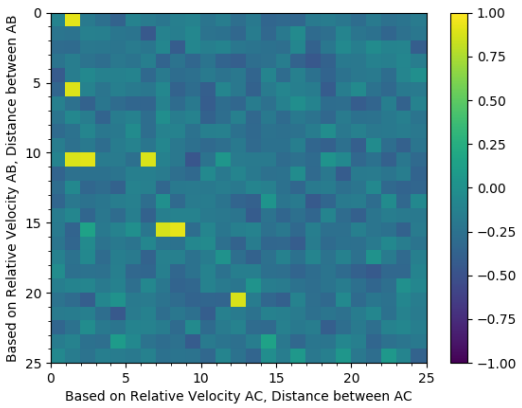


(f) Scenario 6

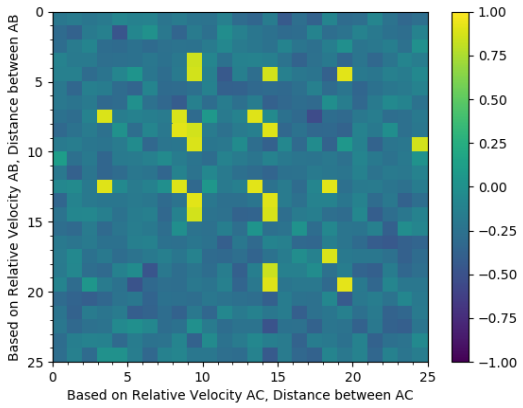
Figure 6.11: Learned reward structure of highway lane merging scenario's



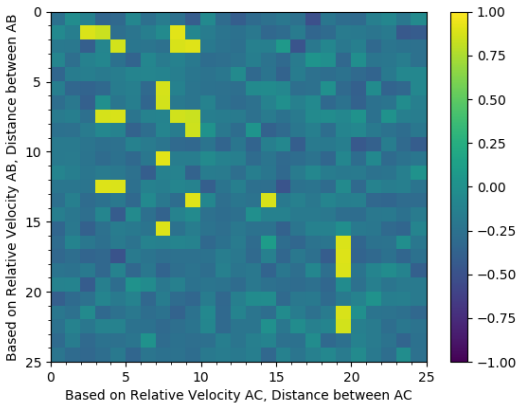
(a) Scenario 7



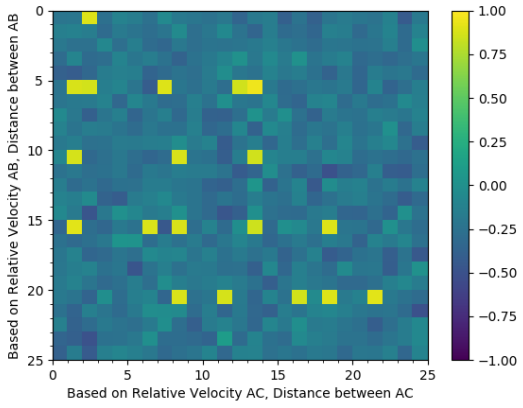
(b) Scenario 8



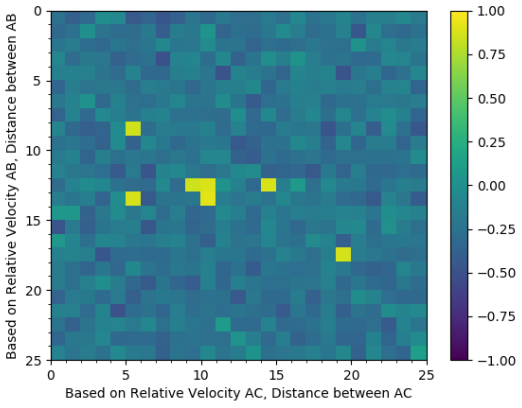
(c) Scenario 9



(d) Scenario 10

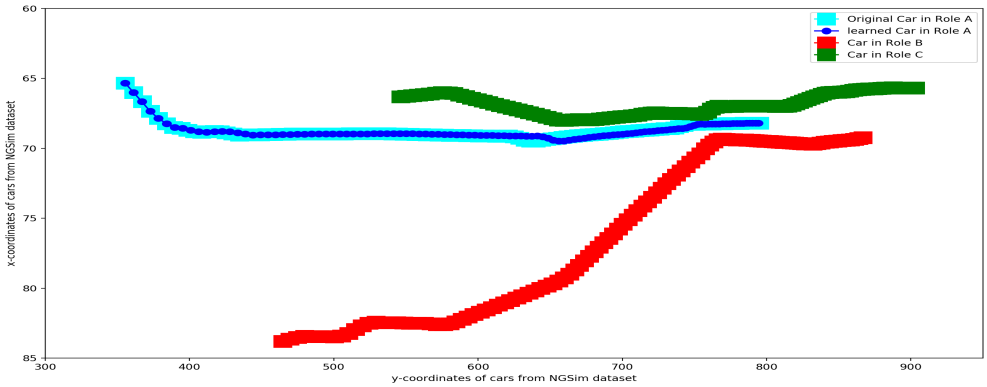


(e) Scenario 11

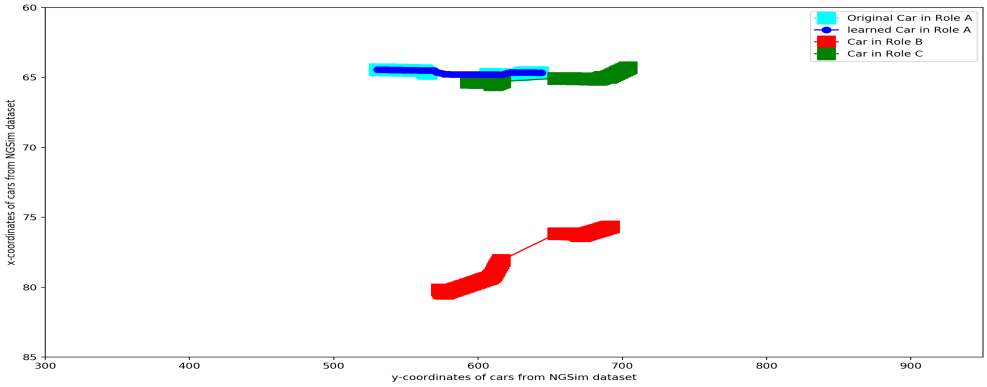


(f) Scenario 12

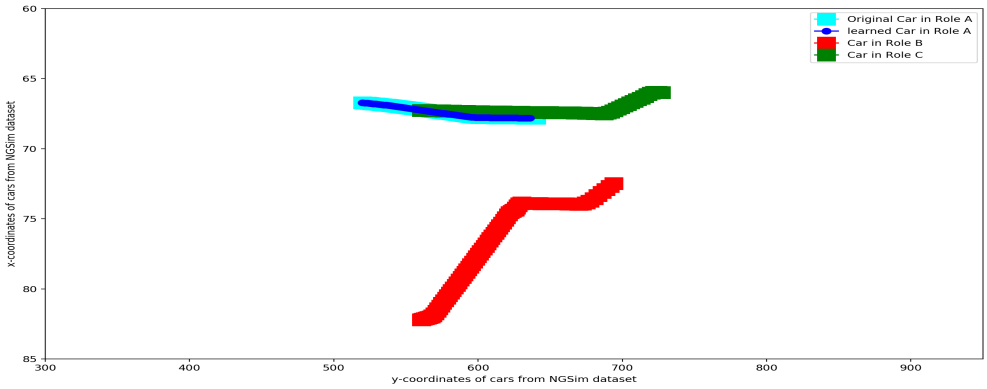
Figure 6.12: Learned reward structure of highway lane merging scenario's



(a) Scenario 1

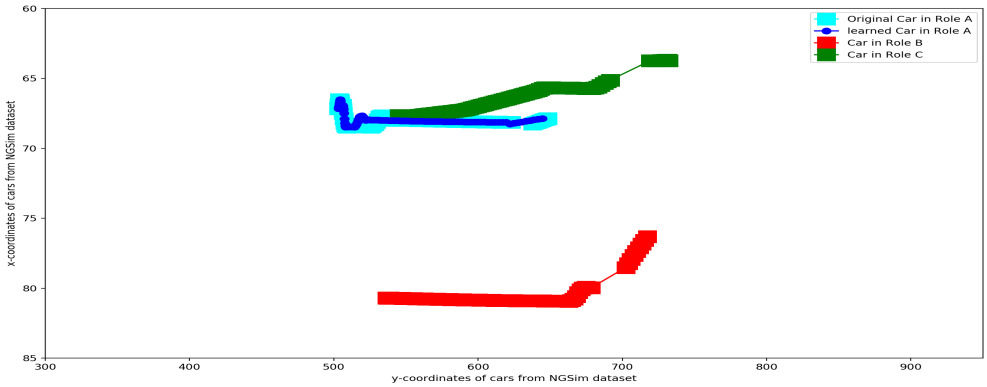


(b) Scenario 2

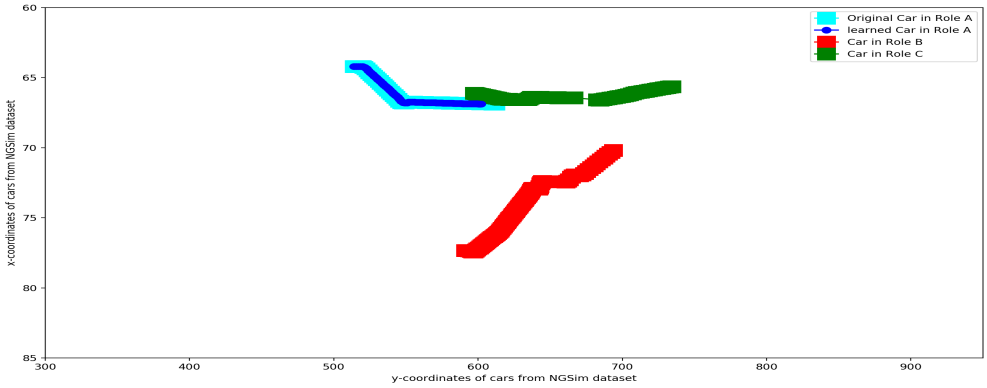


(c) Scenario 3

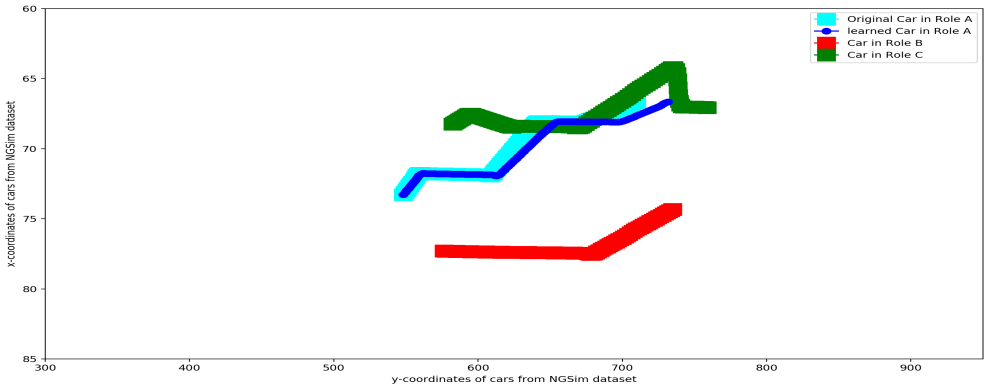
Figure 6.13: Original trajectories of cars in role A, B, and C along with the trajectory of a car in role A that is generated using the learnt rewards for the highway lane merging scenario's



(a) Scenario 4

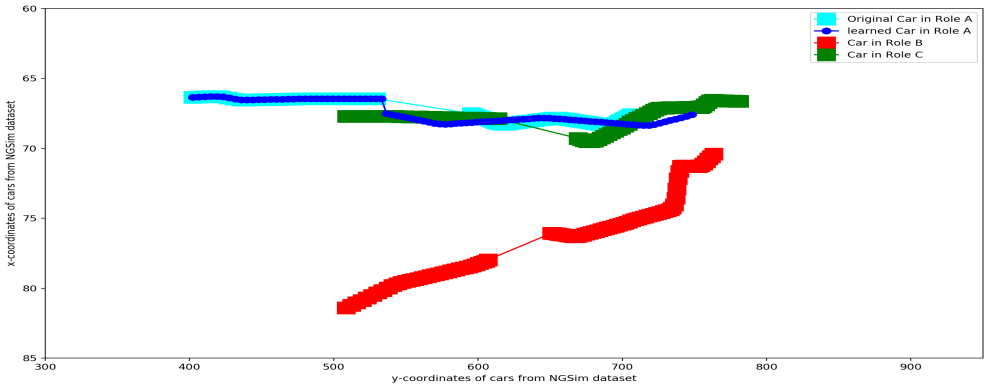


(b) Scenario 5

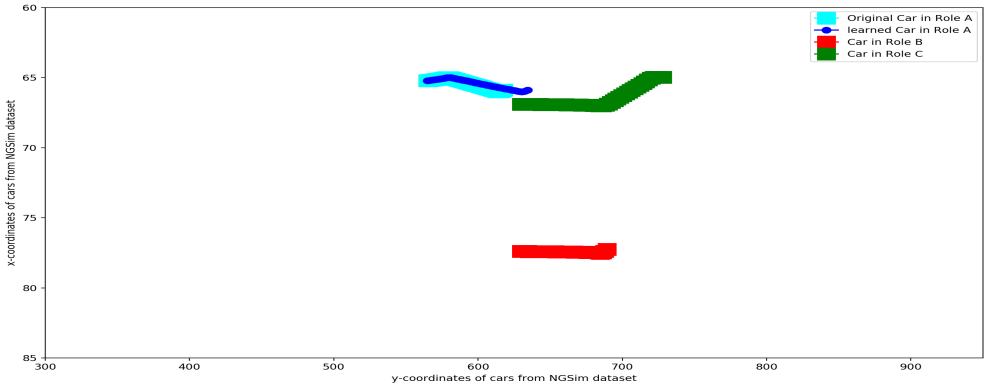


(c) Scenario 6

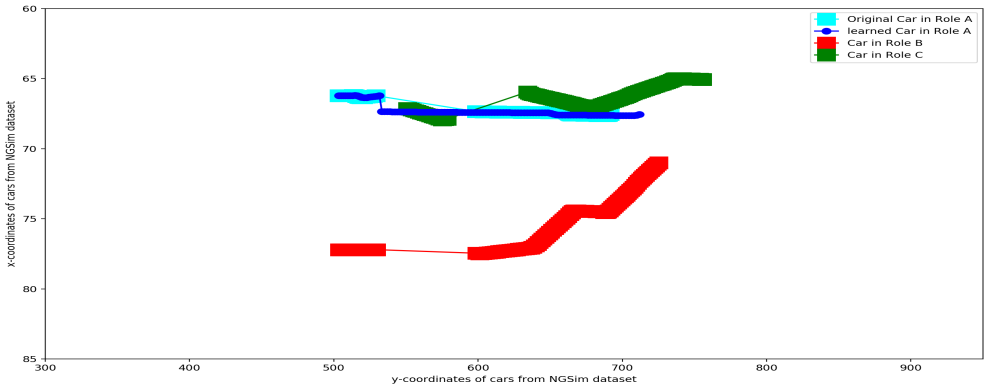
Figure 6.14: Original trajectories of cars in role A, B, and C along with the trajectory of a car in role A that is generated using the learnt rewards for the highway lane merging scenario's



(a) Scenario 7

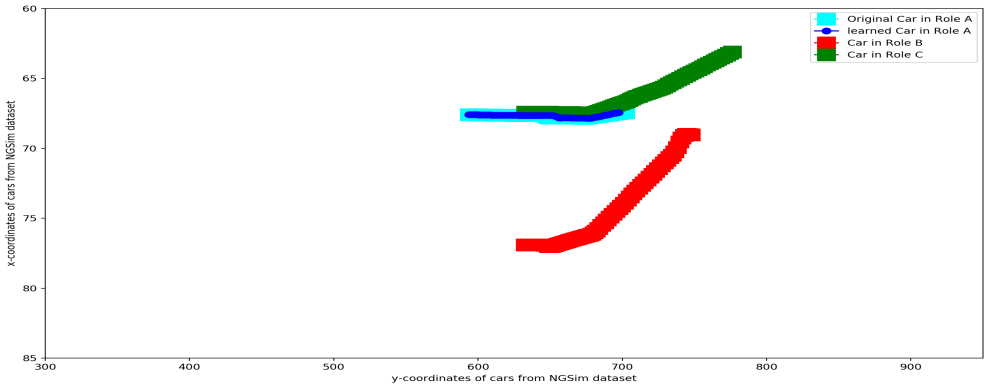


(b) Scenario 8

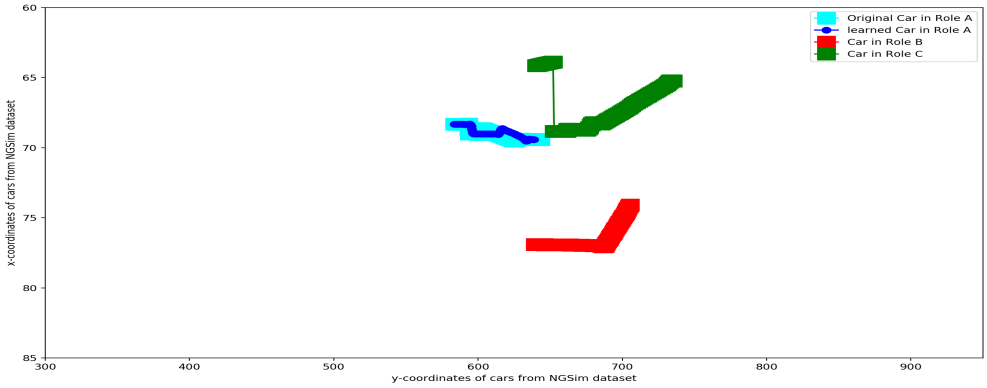


(c) Scenario 9

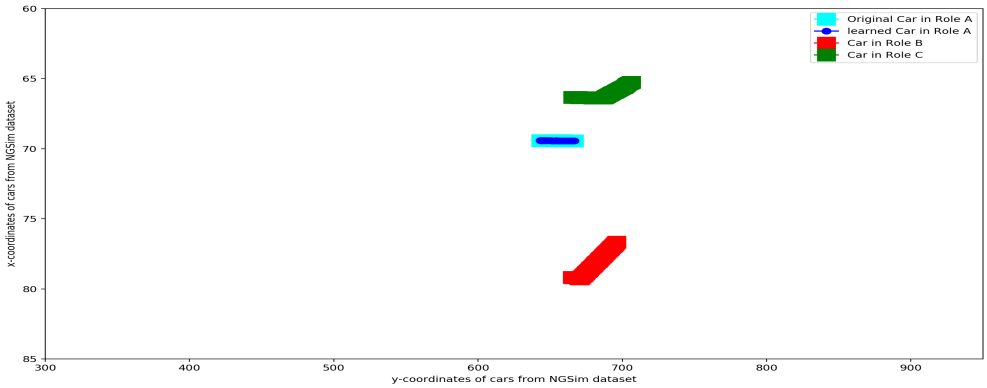
Figure 6.15: Original trajectories of cars in role A, B, and C along with the trajectory of a car in role A that is generated using the learnt rewards for the highway lane merging scenario's



(a) Scenario 10



(b) Scenario 11



(c) Scenario 12

Figure 6.16: Original trajectories of cars in role A, B, and C along with the trajectory of a car in role A that is generated using the learnt rewards for the highway lane merging scenario's

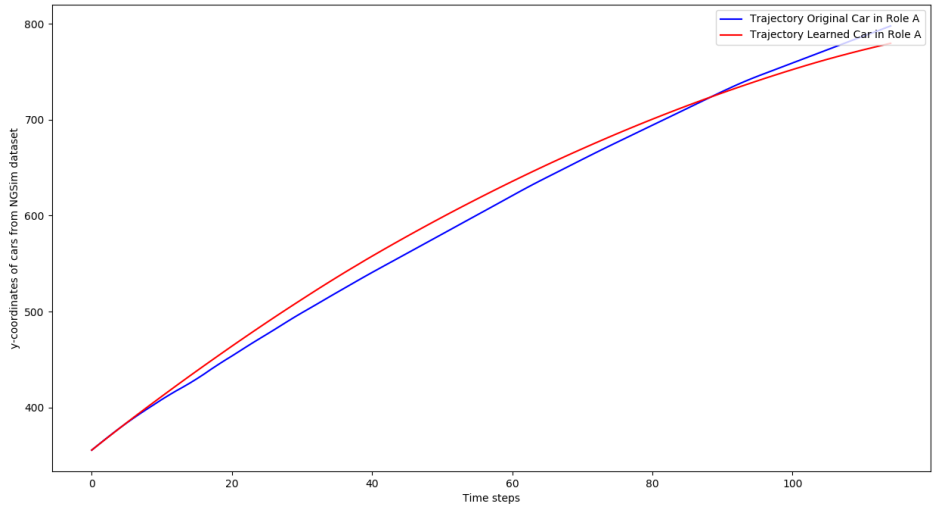


Figure 6.17: y-coordinate difference between expert agent's trajectory and learning agent's trajectory for highway lane merging scenario 1

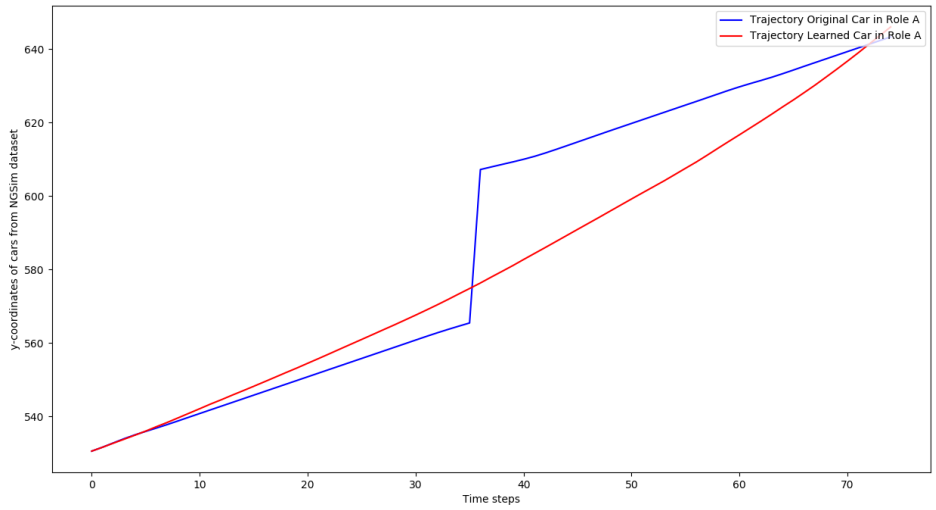


Figure 6.18: y-coordinate difference between expert agent's trajectory and learning agent's trajectory for highway lane merging scenario 2

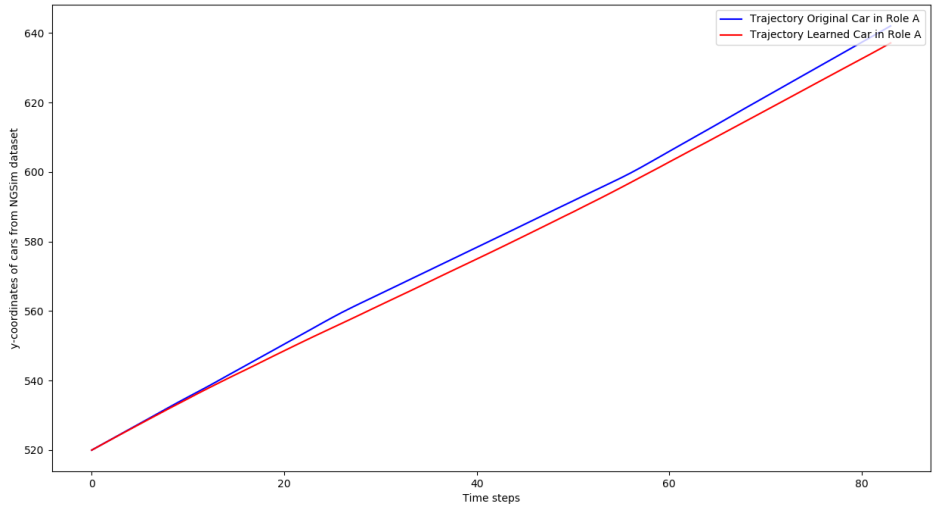


Figure 6.19: y-coordinate difference between expert agent's trajectory and learning agent's trajectory for highway lane merging scenario 3

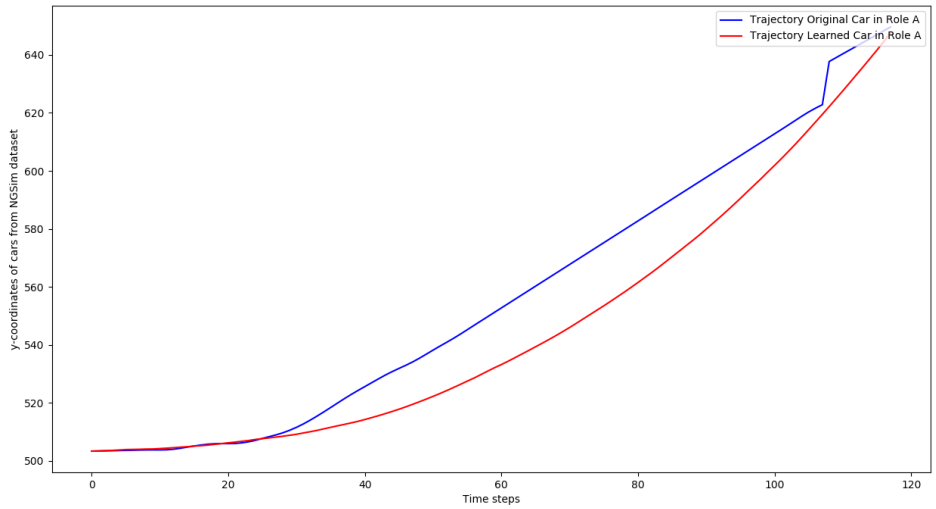


Figure 6.20: y-coordinate difference between expert agent's trajectory and learning agent's trajectory for highway lane merging scenario 4

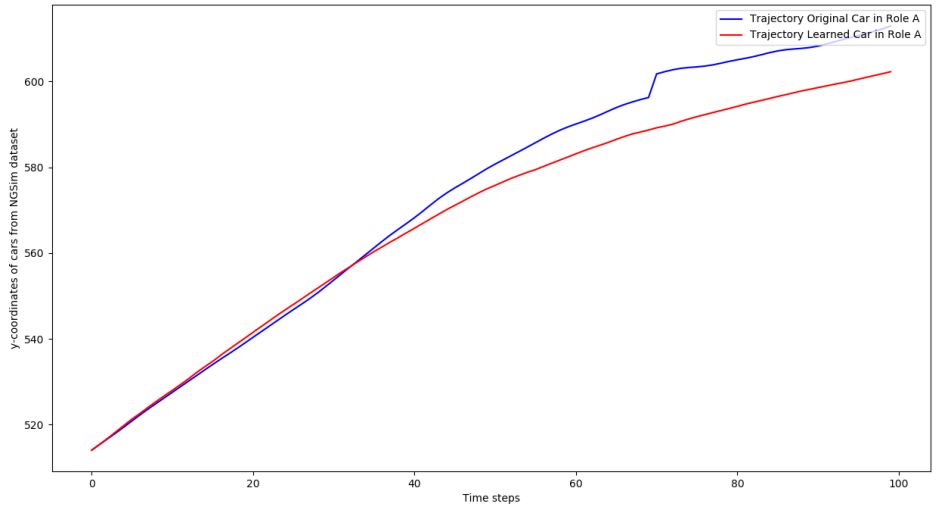


Figure 6.21: y-coordinate difference between expert agent's trajectory and learning agent's trajectory for highway lane merging scenario 5

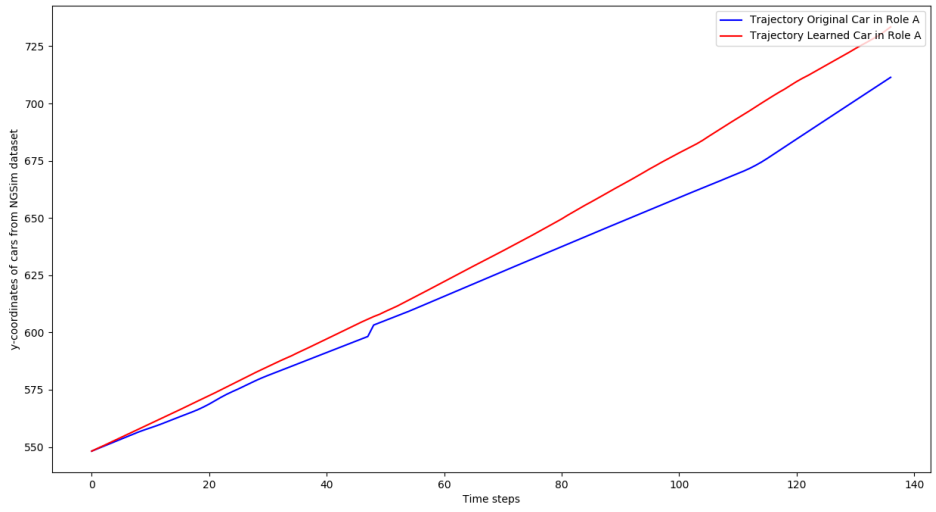


Figure 6.22: y-coordinate difference between expert agent's trajectory and learning agent's trajectory for highway lane merging scenario 6

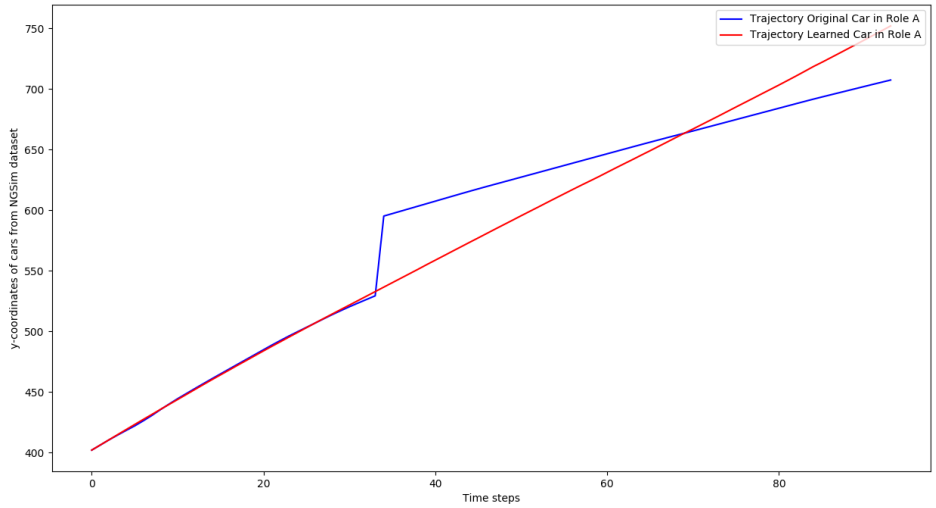


Figure 6.23: y-coordinate difference between expert agent's trajectory and learning agent's trajectory for highway lane merging scenario 7

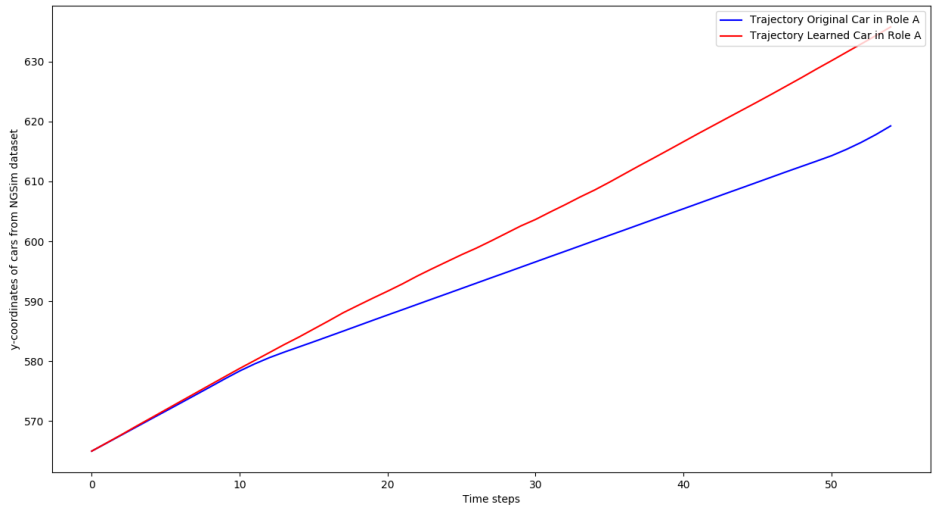


Figure 6.24: y-coordinate difference between expert agent's trajectory and learning agent's trajectory for highway lane merging scenario 8

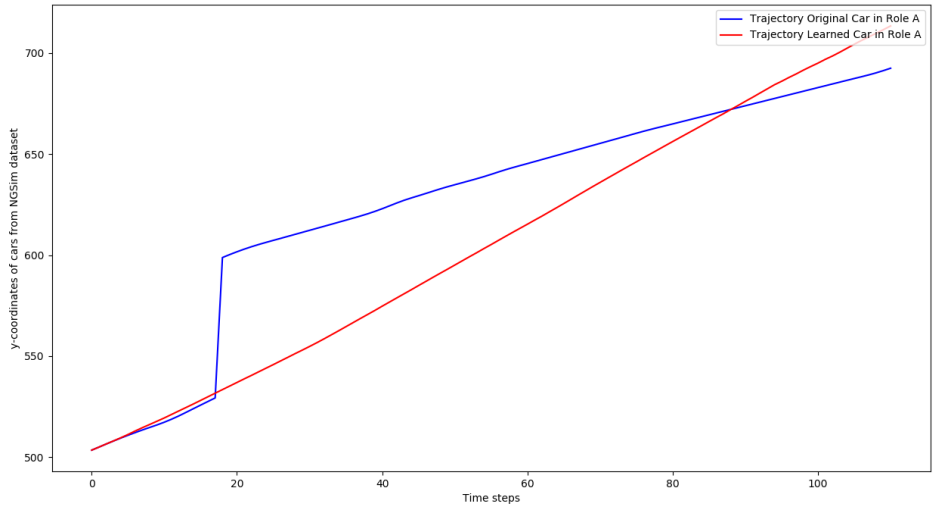


Figure 6.25: y-coordinate difference between expert agent's trajectory and learning agent's trajectory for highway lane merging scenario 9

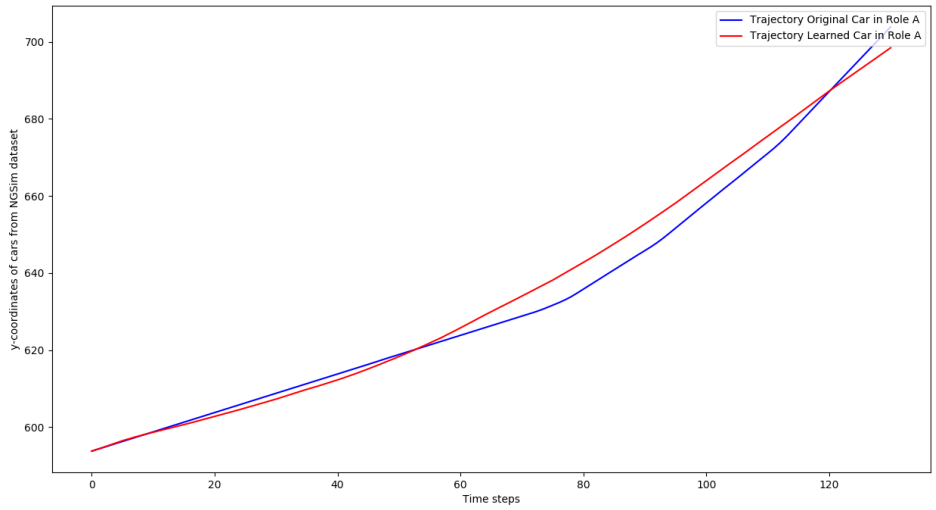


Figure 6.26: y-coordinate difference between expert agent's trajectory and learning agent's trajectory for highway lane merging scenario 10

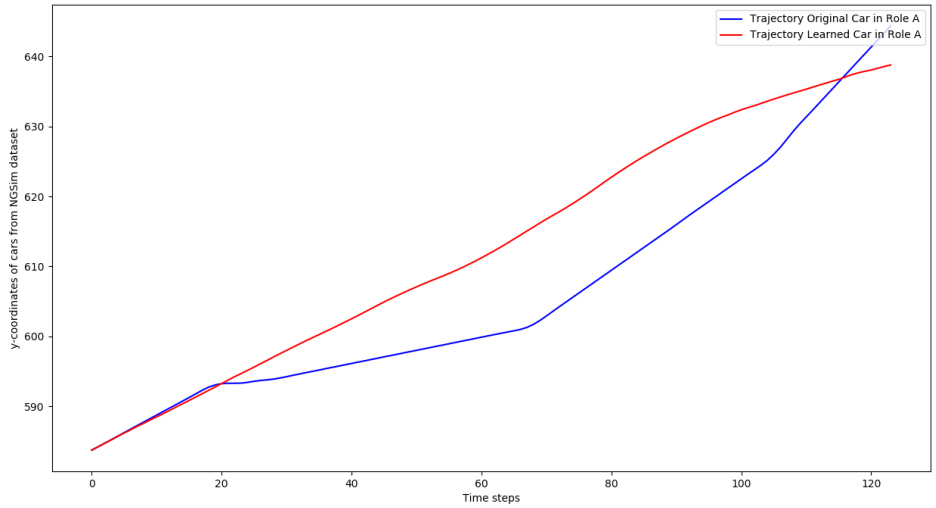


Figure 6.27: y-coordinate difference between expert agent's trajectory and learning agent's trajectory for highway lane merging scenario 11

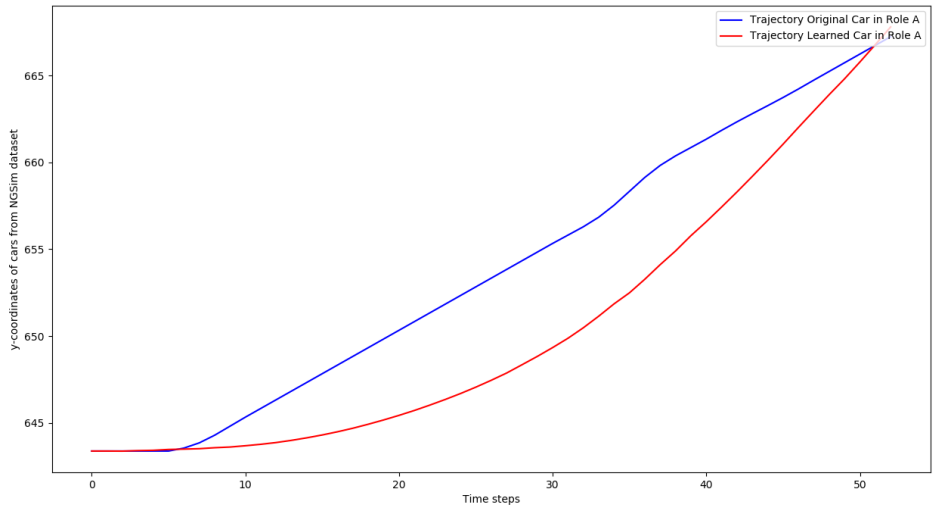


Figure 6.28: y-coordinate difference between expert agent's trajectory and learning agent's trajectory for highway lane merging scenario 12

6.3 Summary

This chapter illustrates the results obtained by implementing the model-based IRL with continuous action spaces. Along with GPIRL results for simplified object-world setting. First, inverse learning error results on simplified object-world environment are presented, which are followed by results on the learned reward structure of simplified object world environment and highway lane merging environment.

Chapter 7

CONCLUSION

In this chapter conclusion is drawn based on the results obtained using the Bayesian Inverse Reinforcement Learning algorithm with continuous action spaces on a simplified object-world environment and highway lane merging environment. Further, this chapter also gives an indication of the direction in which further work can be carried out.

7.1 Conclusion

This work provides a novel and tractable model based inverse reinforcement learning algorithm with continuous action spaces by revising Bayesian IRL. Preference functions are learnt for simplified object world and highway lane merging environment and our solutions show informative reward structure.

Results of simplified object world environment are compared using Inverse Learning Error(ILE) with proposed algorithm.

Our experiments verify that our solutions are close to the true reward functions based on the expert demonstrations and yield good policies for apprenticeship learning.

It is observed that the reward evaluation is formative as it takes place during the devel-

opment of the rewards, with the intention of improving the effectiveness of the rewards.

BIRL with continuous action spaces has better accuracy in preference learning.

7.2 Future work

Conduct more experiments with proposed algorithm on highway lane merging domain and evaluate the learnt preference function.

Explore if more informative priors can be constructed for specific environments using background knowledge of that environment. Investigate how well does IRL generalize when the transition function of the expert agent and learning agent are different also find out how robust would the reward function or policy learned from an expert agent be, with respect to the learning agents state space.

Explore other model based IRL algorithms which can incorporate continuous action spaces and compute inverse learning error for them. Also, results can be compared with a model free IRL algorithm with continuous action space to examine the performance.

Bibliography

[1] Navid Aghasadeghi and Timothy Bretl. "Maximum entropy inverse reinforcement learning in continuous state spaces with path integrals". In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE. 2011, pp. 1561–1566.

[2] Shun-Ichi Amari. "Natural gradient works efficiently in learning". In: *Neural computation* 10.2 (1998), pp. 251–276.

[3] Brenna D Argall et al. "A survey of robot learning from demonstration". In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.

[4] Saurabh Arora and Prashant Doshi. "A Survey of Inverse Reinforcement Learning: Challenges, Methods and Progress". In: *arXiv preprint arXiv:1806.06877* (2018).

[5] Abdeslam Boularias, Jens Kober, and Jan Peters. "Relative entropy inverse reinforcement learning". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011, pp. 182–189.

[6] Peter Cheeseman and John Stutz. "On the relationship between bayesian and maximum entropy inference". In: *AIP Conference Proceedings*. Vol. 735. 1. AIP. 2004, pp. 445–461.

[7] Heeyoul "Henry Choi, Sookjeong Kim, and Seungjin Choi. "Trust-region learning for ICA". In: 1 (Aug. 2004), p. 46.

[8] Andrew R Conn, Nicholas IM Gould, and Ph L Toint. *Trust region methods*. Vol. 1. Siam, 2000.

[9] Nikovski D.N. "Fast Reinforcement Learning in continuous action spaces". In: *Robotics Institute Carnegie Mellon University, Pittsburgh PA.* (1998).

[10] Barry D Hughes. *Random walks and random environments*. Vol. 1: Random Walks. Clarendon Press; Oxford University Press, 1995.

[11] James Colyar John Halkias. *NGSIM-Next Generation SIMulation*. URL: `https://www.fhwa.dot.gov/publications/research/operations/its/06135/index.cfm`. (accessed: 07.30.2018).

[12] Sham M Kakade. "A natural policy gradient". In: *Advances in neural information processing systems*. 2002, pp. 1531–1538.

[13] Mrinal Kalakrishnan et al. "Learning objective functions for manipulation". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE. 2013, pp. 1331–1336.

[14] Dan Klein and Pieter Abbeel. *CS188 Artificial Intelligence: Markov Decision Processes II*. URL: `http://ai.berkeley.edu/slides/Lecture%209%20--%20MDPs%20II/`. (accessed: 04.17.2018).

[15] Jens Kober et al. "Reinforcement Learning to Adjust Parametrized Motor Primitives to New Situations". In: 33 (Nov. 2012), pp. 361–379.

[16] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. "Reinforcement learning in continuous action spaces through sequential monte carlo methods". In: *Advances in neural information processing systems*. 2008, pp. 833–840.

[17] Sergey Levine and Vladlen Koltun. "Continuous inverse optimal control with locally optimal examples". In: *arXiv preprint arXiv:1206.4617* (2012).

[18] Sergey Levine, Zoran Popovic, and Vladlen Koltun. "Nonlinear inverse reinforcement learning with gaussian processes". In: *Advances in Neural Information Processing Systems*. 2011, pp. 19–27.

[19] Andrew Y Ng, Stuart J Russell, et al. "Algorithms for inverse reinforcement learning." In:

[20] J. Peters. "Policy gradient methods". In: *Scholarpedia* 5.11 (2010). revision #137199, p. 3698. DOI: 10.4249/scholarpedia.3698.

[21] David L. Poole and Alan K. Mackworth. *"Artificial Intelligence: Foundations of Computational Agents"*. Cambridge University Press, 2017.

[22] Deepak Ramachandran and Eyal Amir. "Bayesian inverse reinforcement learning". In: *Urbana* 51.61801 (), pp. 1–4.

[23] Stuart Russell. "Learning agents for uncertain environments". In: *Proceedings of the eleventh annual conference on Computational learning theory*. ACM. 1998, pp. 101–103.

[24] Stuart Russell and Peter Norvig. *"Artificial Intelligence: A Modern Approach"*. 3rd ed. Prentice Hall, Englewood Cliffs, NJ, 2009.

[25] John Schulman et al. "Trust region policy optimization". In: *International Conference on Machine Learning*. 2015, pp. 1889–1897.

[26] Richard S. Sutton and Andrew G. Barto. *"Reinforcement Learning: An Introduction"*. The MIT Press, 2017.

[27] Hado Van Hasselt and Marco A Wiering. "Reinforcement learning in continuous action spaces". In: *Approximate Dynamic Programming and Reinforcement Learning, ADPRL 2007*. IEEE. 2007, pp. 272–279.

[28] Brian D Ziebart et al. "Maximum Entropy Inverse Reinforcement Learning." In: 2008.

Appendices

Appendix A

MATSim Simulator

The NGSim dataset is very large and does not yield much understanding for analysis in its raw format. Therefore, simulation of this data is useful for analysis and evaluation. We developed a new MATSim based simulator that is more comprehensive simulator compared to the previously-developed MATLAB based simulator. This simulator is playing a crucial role in our understanding of the data set and in spotting specific driving styles in the data.

"MATSim + Senozon's Via" provide a complete agent-based, real-time and dynamic visualization for NGSim dataset, which helps in understanding the data. Also, it allows a professional and robust simulation for only the cars in roles A, B and C, over different frames in a shortened interval.

Figure A.1 shows the merge lane 6 and the entry ramp (lane 7) with several cars on it. It shows risk-prone behavior of car's in role "A" as car's in role "B" have to reduce their speed. Car's color range in as follows: Red (for low speed of car which is near zero), Yellow (for moving car with minor speed), Green (for moving car with higher speed), Blue (for moving car with very high speed).

We need to load two layers in the simulator: network layer and the events layer. The network layer describes the road network on which cars move, it consists of nodes and links.

Figure A.1: Snapshot of MATSim simulator: 5:00 pm - 5:15 pm dataset

The events layer describes actions of an object(car) at a specific time, for example, a vehicle enters a node, a vehicle arrives at a location, or a vehicle leaves a node. These layers are in the form of XML files.

Our custom program in MATLAB allows us to generate the network.xml and events.xml files. These files are imported in "MATSim Senozon Via" in-order to render the visualizations. A complete workflow allows any road network to be visualized in Via and A-B-C models from NGSim to be rendered on the network. Figure A.2 illustrates the flowchart for generating events.xml file from data file.

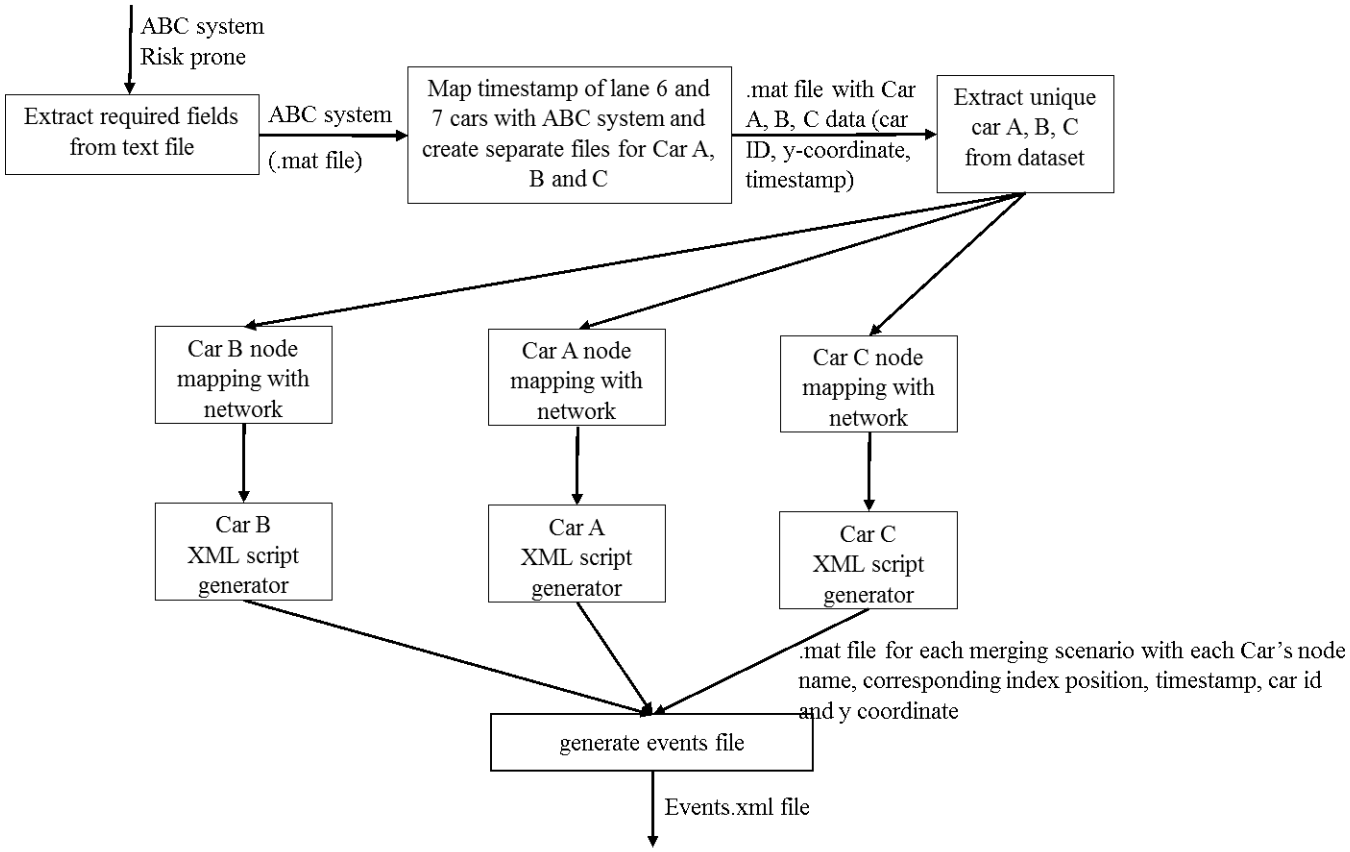


Figure A.2: Flowchart for generating events.xml file from data file