Risk Averse Path Planning Using Lipschitz Approximated Wasserstein Distributionally Robust Deep Q-Learning

Cem Alptürk

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6166
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Abstract

We investigate the problem of risk averse robot path planning using the deep reinforcement learning and distributionally robust optimization perspectives. Our problem formulation involves modelling the robot as a stochastic linear dynamical system, assuming that a collection of process noise samples is available. We cast the risk averse motion planning problem as a Markov decision process and propose a continuous reward function design that explicitly takes into account the risk of collision with obstacles while encouraging the robot's motion towards the goal. We learn the risk-averse robot control actions through Lipschitz approximated Wasserstein distributionally robust deep Q-Learning to hedge against the noise uncertainty. The learned control actions result in a safe and risk averse trajectory from the source to the goal, avoiding all the obstacles. Various supporting numerical simulations are presented to demonstrate our proposed approach.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Venkatraman Renganathan, for supporting me in the research and execution of this thesis with his knowledge and experience.
Besides my supervisor I would also like to thank my examiner Prof. Anders Rantzer for his valuable feedback and questions during my thesis defence.
I would also like to thank the members of the faculty for making this masters program as interesting and informative as possible and giving me the necessary background in order for me to complete this thesis.
I am deeply grateful to my parents for their encouragement and support in my studies.
Special thanks to KJK.

Contents

1

Introduction

Given the continuous increase in the computing power, many computationally expensive control theory problems are increasingly now being addressed using the deep reinforcement learning approaches [Xie et al., 2019; Yan et al., 2020]. The need for handling the uncertainty in the robot motion planning has gained traction given that robots are increasingly now being deployed to solve many real-world problems. So far, the motion planning problem with uncertainty has been investigated from two different and yet closed linked perspectives namely the control theory [Luders et al., 2010] and the reinforcement learning [Raajan et al., 2020]. When stochastic uncertainties are considered in the problems such as path planning, both the above said approaches resort to the powerful stochastic optimization techniques as in [Kandel and Moura, 2020] and [Renganathan et al., 2022] to ensure satisfaction of specifications with high probability. However, when assumptions of certain functional forms for the system uncertainties are made in the name of tractability, they may lead to potentially severe miscalculation of risk when the uncertain robot is made to operate in a dynamic environment [Majumdar and Pavone, 2020]. Such shortcomings can be addressed through carefully designed risk bounded motion planning approaches using distributionally robust optimization techniques. The interested readers are referred to these non-exhaustive list of papers on risk averse motion planning [Aoude et al., 2013; Subramani and Lermusiaux, 2019; Xiao et al., 2019; Lathrop et al., 2021].

Risk averse path planning problems emphasize the need for exact propagation of uncertainties. For instance, either the distributions of all the uncertainties or the moments defining the distributions are required to be known in advance or calculated exactly for all time steps. However, things get easily complicated when robot dynamics are nonlinear as in [Safaoui et al., 2021] or in the linear setting when the uncertainty distributions being non-Gaussian as in [Renganathan et al., 2020]. Similarly, it is an usual practice to associate a particular distribution to the uncertainty (often Gaussian) just for the sake of tractability [Luders et al., 2010]. But often in reality, all we have is just a collection of samples of the uncertainty and trying to fit a distribution to it may cause undue risk. In this paper, we stick to the sample based uncertainty modeling of process noise and investigate the risk averse motion planning problem. We use the Wasserstein distributionally robust deep Q-learning to hedge against the system uncertainty and approximately solve the Bellman equation associated with the deep Q-learning approach [Mnih et al., 2013]. Specifically, we use the Lipschitz constant based approximations advocated in Theorem 5 of [Kuhn et al., 2019] to learn risk-averse robot control actions. A similar problem was investigated by [Raajan et al., 2020], albeit with usual Gaussian assumptions and no formal risk consideration.

1.1 Contributions

This article leverages powerful results in deep reinforcement learning theory and distributionally robust optimization as described in [Kandel and Moura, 2020] to learn control policies for robots to operate in a risk-averse manner in an environment. Our main contributions are:

1. we learn safe robot control actions at all the state space positions to infer a trajectory to move from source to goal by avoiding all obstacles. We account for the uncertainty due the robot initial states and the process noise through reward function design and learn the risk averse control actions using approximated Wasserstein distributionally robust Q-learning.

2. we demonstrate our proposed approach using a series of numerical simulations and show the effectiveness of our proposed approach.

1.2 Organization of Thesis

The thesis is organized as follows. In Chapter 2 the necessary background theory on Probability Theory 2.2, Reinforcement Learning 2.3 and Distributionally Robust Optimization 2.4 is presented. The risk-averse path planning problem associated with the uncertain robot system is presented in Chapter 3. The Wasserstein distributionally robust Q-Learning approach is discussed in Chapter 4. The proposed idea is then demonstrated using numerical simulations in Chapter 5. Finally the results are discussed in Chapter 6 along with directions for future research.

2

Background

In this chapter we describe our notations and the background theory that is necessary to understand this thesis. This chapter is divided into four sections namely; 2.1 Notations, where we define the notations used in this paper, 2.2 Probability theory, where we go over the basics of probability theory, 2.3 Reinforcement Learning, where we explain the basic idea behind reinforcement learning, and 2.4 Stochastic Optimization, where we introduce the field of stochastic optimization where the project is built upon.

2.1 Notations

The set of real numbers, integers are denoted by $\mathbb{R}, \mathbb{Z}$. The subset of real numbers greater than $a \in \mathbb{R}$ is denoted by $\mathbb{R}_{>a}$. The set of integers between two values $a, b \in \mathbb{Z}$ with $a < b$ is denoted by $[a : b]$. The set of non-negative integers is denoted by $\mathbb{Z}_+$. We denote by $\mathbf{B}(\mathbb{R}^d)$ and $\mathcal{P}(\mathbb{R}^d)$ the Borel σ-algebra on $\mathbb{R}^d$ and the space of probability measures on $(\mathbb{R}^d, \mathbf{B}(\mathbb{R}^d))$ respectively. We denote the uniform probability distribution over a set $\mathcal{X}$ as $\mathcal{U}(\mathcal{X})$. By $\|\cdot\|$ we denote the 2-norm or the euclidian distance of a vector.

2.2 Probability Theory

Formally, a probability space is defined using the triple $(\Omega, \mathcal{F}, \mathbb{P})$, where

1. Ω is called sample space and denotes the set of all possible outcomes.
2. $\mathcal{F}$ is called an event space and is a set of events where an event is a set of outcomes in the sample space.
3. $\mathbb{P}$ is called the probability function and assigns a probability between 0 and 1 to each event in the event space.

A random variable X is a measurable function $X : \Omega \to E$ that maps a set of possible outcomes Ω to a measurable space E. The probability that X takes on a value in a measurable set $S \subseteq E$ is written as

$$\mathbb{P}(X \in S) = \mathbb{P}\left(\{\omega \in \Omega \mid X(\omega) \in S\}\right). \tag{2.1}$$

Any random variable can be described by its cumulative distribution function, which describes the probability that the random variable will be less than or equal to a certain value. The probability distribution of a random variable can be characterized by a small number of parameters with a practical interpretation instead of a known probability distribution. For a distribution $\mathbb{P} \in \mathcal{P}(\mathcal{X})$ that is defined on a space $\mathcal{X} \subseteq \mathbb{R}^d$ with random variable $x \in \mathcal{X}$, the probability of x occurring is denoted as $\mathbb{P}(x) : \mathcal{X} \to \mathbb{R}$.

For a discrete $\mathcal{X}$, $\mathbb{P}(x)$ represents the probability of x occurring. The distribution $\mathbb{P}$ has to satisfy the following summation property. That is, for a discrete probability distribution,

$$\sum_{x \in \mathcal{X}} \mathbb{P}(x) = 1, \tag{2.2}$$

and for a continuous probability distribution,

$$\int_{\mathcal{X}} \mathbb{P}(x) dx = 1. \tag{2.3}$$

The expected value of a random variable $x \in \mathbb{R}^d$ with distribution $\mathbb{P}(x)$ is denoted as $\mathbb{E}_x[x] \in \mathbb{R}^d$ and is defined as,

$$\mathbb{E}_x[x] \triangleq \begin{cases} \sum_{x \in \mathcal{X}} x\mathbb{P}(x), & \text{for discrete } \mathcal{X}, \\ \int_{\mathcal{X}} x\mathbb{P}(x)dx, & \text{for continuous } \mathcal{X}. \end{cases} \tag{2.4}$$

2.3 Reinforcement Learning

Reinforcement learning is a machine learning method that is based on rewarding desired behaviors and penalizing undesired ones. This method differs from other machine learning methods such as `supervised learning` and `unsupervised learning`, in not needing labelled input/output pairs. In reinforcement learning problems, an `agent` interacts with an environment by taking actions, and receives a reward each time an action is taken. With each action the environment changes. The goal is to find the best policy or decision rule such that the cumulative rewards received from the environment is maximized. The agent tries to learn the relationship between the environment states, actions and the rewards. Figure 2.1 illustrates the main principle in reinforcement learning methods.

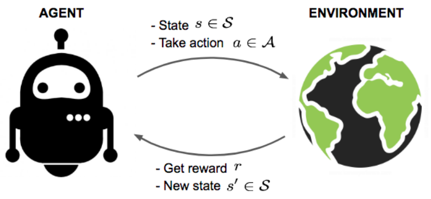

AGENT
ENVIRONMENT
- State $s \in \mathcal{S}$
- Take action $a \in \mathcal{A}$
- Get reward r
- New state $s' \in \mathcal{S}$

Figure 2.1 Illustration of reinforcement learning. The agent performs an action based on the state of the environment and receives a reward based on how good the action is. By taking an action the state of the environment changes as well. [1]

Reinforcement learning can be especially useful in tasks where the best action to take is not necessarily known. A common usage of reinforcement learning is for training agents that can play games [Mnih et al., 2013], since games usually have a built in reward system. The agent can learn through experiences and exploit the reward system to achieve a desired behavior.

For an environment with finite states, the experiences can be recorded on a table [Watkins and Dayan, 1992] and based on these experiences, a policy can be constructed. However if the number of states is too large, the required memory can exceed the available resources. For a continuous state space, a table can not hold all the experiences since virtually infinite states are possible. A solution to this is to use a neural network as a policy that learns the experiences rather than record them on a table [Mnih et al., 2013]. This allows to train environments with large or infinite states. The methods that utilize a deep neural network as the policy are commonly referred to as `Deep Reinforcement Learning`.

2.4 Stochastic Optimization

The main idea of a numerical optimization problem is to optimize an objective function, say $L : \mathbb{R}^d \to \mathbb{R}$ with respect to the decision variable $x \in \mathbb{R}^d$ in a space $\mathcal{X} \subseteq \mathbb{R}^d$. That is,

$$\min_{x \in \mathcal{X}} L(x). \tag{2.5}$$

Solutions to these problems heavily depend on the properties of L and $\mathcal{X}$. That is, under certain optimization friendly conditions such as $L(x)$ being a convex function and $\mathcal{X}$ being a convex set, the optimization problem given by (2.5) can be efficiently solved using thoroughly established convex optimization techniques. These optimization problems become interesting when an uncertainty is introduced into them. For instance, (2.5) can be extended

[1] https://vitalflux.com/reinforcement-learning-real-world-examples/

into a step further by introducing an uncontrolled variable w into the objective function as $L(x, w)$. The idea to deal with such a formulation is to imagine the uncontrolled variable as adversarial and working against us. The solution to this problem is to plan according to the worst case scenario. Robust optimization considers the uncertainty $w \in \mathcal{W}$ with $\mathcal{W}$ denoting the corresponding uncertainty bound set. That is, w is assumed to be equally likely such that it has a uniform distribution over the space $\mathcal{W}$. A robust optimization problem can be formulated as,

$$\min_{x \in \mathcal{X}} \max_{w \in \mathcal{W}} L(x, w), \tag{2.6}$$

where the distribution $\mathbb{P}_w := \mathcal{U}(\mathcal{W})$ is uniform over the space $\mathcal{W}$. In stochastic optimization, the variable w has a known distribution $\mathbb{P}_w \in \mathcal{P}(\mathcal{W})$. Since w is modeled as random, the optimization can be performed over the expected value of the objective function with respect to w as,

$$\min_{x \in \mathcal{X}} \mathbb{E}_{w \sim \mathbb{P}_w}[L(x, w)]. \tag{2.7}$$

Distributionally Robust Optimization

However, to solve (2.7) exactly, the distribution defining the uncertainty w has to be known exactly to compute the expected value of the objective function $L(x, w)$. Assuming a specific functional form for the uncertain random variable w just for the sake of tractability will cause the resulting decision x to impose undue risk. Fortunately, under certain specific assumption on the randomness of the uncertainty w, such unwanted risk can be mitigated with provable and certifiable guarantees using the distributionally robust optimization techniques. Distributionally Robust Optimization (DRO) is a subfield of stochastic optimization. In the case of DRO, the stochastic variable has an unknown distribution $\mathbb{P}_w$ but we believe that $\mathbb{P}_w$ belongs to some set consistent with certain features of distributions such as moments, unimodality etc. This ambiguity in its true distribution makes optimization more complicated. In most DRO problems, it is assumed that some prior information about $\mathbb{P}_w$ is known and is incorporated in an ambiguity set $\mathcal{B}_w \subset \mathcal{P}(\mathcal{W})$ where the ambiguity set is a set that contains probability distributions defined over $\mathcal{W}$. The DRO problem can be written as,

$$\min_{x \in \mathcal{X}} \max_{\mathbb{P} \in \mathcal{B}_w} \mathbb{E}_{w \sim \mathbb{P}}[L(x, w)] \tag{2.8}$$

The goal of this kind of optimization problems is to solve for the worst case distribution in $\mathcal{B}_w$ and find a robust solution. Solving DRO's depends on the properties of the ambiguity set. A comparison between the different optimization paradigms can be seen in Figure 2.2. A common type of ambiguity set

[2] Figure taken from [Shang and You, 2018]

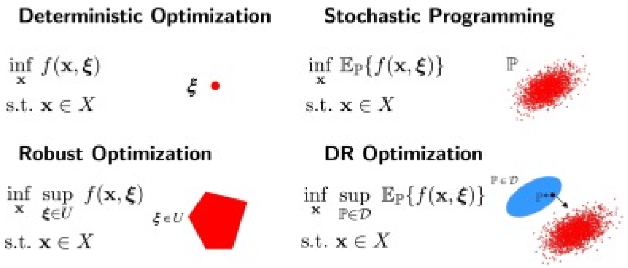

Deterministic Optimization
$\inf_{\mathbf{x}} f(\mathbf{x},\xi)$
s.t. $\mathbf{x} \in X$
ξ
Stochastic Programming
$\inf_{\mathbf{x}} \mathbb{E}_{\mathbb{P}}\{f(\mathbf{x},\xi)\}$
s.t. $\mathbf{x} \in X$
$\mathbb{P}$
Robust Optimization
$\inf_{\mathbf{x}} \sup_{\xi \in U} f(\mathbf{x},\xi)$
s.t. $\mathbf{x} \in X$
$\xi \in U$
DR Optimization
$\inf_{\mathbf{x}} \sup_{\mathbb{P} \in \mathcal{D}} \mathbb{E}_{\mathbb{P}}\{f(\mathbf{x},\xi)\}$
s.t. $\mathbf{x} \in X$
$\mathbb{P} \in \mathcal{D}$

Figure 2.2 Different optimization paradigms.[2]

that is used in current literature is a Wasserstein ambiguity set, which relies on a distance metric between probability distributions called the Wasserstein distance.

Wasserstein Distance

The Wasserstein distance is a distance metric that is defined between probability distributions. It can be used to compute how similar two distributions are. For two distributions $Q, Q' \in \mathcal{P}(\mathcal{W})$, the p-order Wasserstein distance is defined as,

$$W_p(Q, Q') := \left(\inf_{\pi \in \Pi(Q,Q')} \int_{\mathcal{W} \times \mathcal{W}} \|w - w'\|^p \pi(dw, dw') \right)^{\frac{1}{p}}, \tag{2.9}$$

where $\Pi(Q, Q')$ is the set of all joint probability distributions supported on $\mathcal{W} \times \mathcal{W}$, with marginal distributions Q and Q'. This distance metric is also known as the earth mover's distance, where the distributions Q, Q' represent heaps of dirt and Π is the set of all admissible transport plans. The Wasserstein distance represents the minimum required energy to convert one heap of dirt into the other. The minimizing joint distribution π^* to (2.9) is the optimal transport plan that requires the minimum energy for this conversion. The Wasserstein distance is commonly used in fields such as Generative Adversarial Networks (GAN) [Adler and Lunz, 2018] and optimal transport theory. Figure 2.3 shows an illustration of the earth movers distance that is commonly used in optimal transport theory.

Wasserstein Ambiguity Set

The Wasserstein ambiguity set $\mathcal{B}_{w,p}$, is defined around a nominal distribution $\hat{\mathbb{P}}_w \in \mathcal{P}(\mathcal{W})$ such that,

$$\mathcal{B}_{w,p} := \left\{ \mathbb{P} \in \mathcal{P}(\mathcal{W}) \mid W_p\left(\hat{\mathbb{P}}_w, \mathbb{P}\right) \leq \epsilon_w \right\}, \tag{2.10}$$

[3] https://sbl.inria.fr/doc/group__Earth__mover__distance-package.html

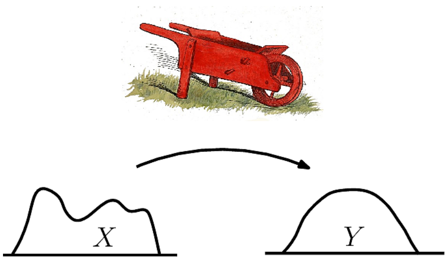


Figure 2.3 Illustration of the earth movers distance. [3]

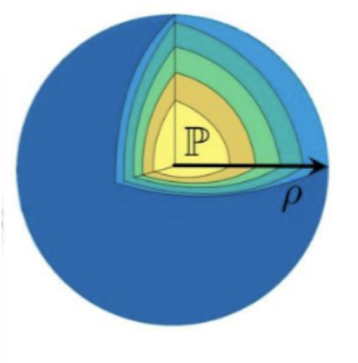


Figure 2.4 A Wasserstein ambiguity set with nominal distribution $\mathbb{P}$ and radius ρ is illustrated here. Any probability distribution within this ball has a Wasserstein distance less than or equal to ρ from the centered nominal distribution $\mathbb{P}$.

where the set $\mathcal{B}_{w,p}$ contains all the distributions supported on $\mathcal{W}$ that have a p$^{\text{th}}$-order Wasserstein distance of at most $\epsilon_w \in \mathbb{R}$ to the nominal distribution $\hat{\mathbb{P}}_w$. This set is commonly referred to as a Wasserstein Ball since it represents a ball in infinite dimensional space with $\hat{\mathbb{P}}_w$ at its center with a radius of ϵ_w. The nominal distribution $\hat{\mathbb{P}}_w$ is usually obtained through some prior knowledge about w. In some cases, $\hat{\mathbb{P}}_w$ can be defined as an empirical distribution that is obtained from samples of w.

Definition 1
For a collection of $N \in \mathbb{N}$ independent samples of w, $\hat{w}_1, \ldots, \hat{w}_N$, an empirical distribution $\hat{\mathbb{P}}_w$ is defined as,

$$\hat{\mathbb{P}}_w := \frac{1}{N}\sum_{i=1}^{N} \delta_{\hat{w}_i}, \tag{2.11}$$

□

where $\delta_{\hat{w}_i}$ is the Dirac delta function. Each sample in this distribution has uniform probability.

The radius ϵ_w is selected such that the true distribution $\mathbb{P}_w$ lies somewhere inside this ball with a probability greater than $1-\beta$. The β parameter will determine the allowed risk factor for a solution. A smaller β will result in a larger radius. Since the radius ϵ_w quantifies the amount of trust (distrust) that we have over the nominal distribution $\hat{\mathbb{P}}_w$, it is chosen such that,

$$\mathbb{P}\left(\mathbb{P}_w \in \mathcal{B}_{w,p}\right) \geq 1-\beta, \quad \beta \in [0,1]. \tag{2.12}$$

Assumption 1

The true distribution $\mathbb{P}_w$ is a light-tailed distribution. That is, $\exists p > 1$ such that,

$$\mathbb{E}_{\mathbb{P}_w}\left[e^{\|w\|^p}\right] = \int_{\mathcal{W}} e^{\|w\|^p} d\mathbb{P}_w(w) < \infty \tag{2.13}$$

□

Assumption 1 is used in [Chen and Paschalidis, 2020] for the following theorem.

Theorem 1

Based on Assumption 1, for an empirical distribution $\hat{\mathbb{P}}_w$, with N atoms, the radius of the Wasserstein ambiguity set is found as,

$$\epsilon_w = \rho\sqrt{\frac{2}{N}\ln\left(\frac{1}{\beta}\right)}, \tag{2.14}$$

□

where $\rho = \mathbf{diam}(\mathbf{supp}(\mathbb{P}_w))$ depends on the true distribution $\mathbb{P}_w$.

Proof. Based on Assumption 1, a relationship between the parameter β and the radius ϵ_w can be found in [Chen and Paschalidis, 2020],

$$\mathbb{P}\left(W_1\left(\mathbb{P}_w, \hat{\mathbb{P}}_w\right) \geq \epsilon_w\right) \leq \begin{cases} c_1 e^{-c_2 N \epsilon_w^{\max(d,2)}}, & \text{if } \epsilon_w \leq 1 \\ c_1 e^{-c_2 N \epsilon_w^p}, & \text{if } \epsilon_w > 1 \end{cases}, \tag{2.15}$$

where d is the dimension of $w \in \mathbb{R}^d$. In order to obtain $\mathbb{P}(W_1(\mathbb{P}_w, \hat{\mathbb{P}}_w) \leq \epsilon_w) \geq 1-\beta$, the radius has to be selected as follows,

$$\epsilon_w(\beta) = \begin{cases} \left(\frac{\ln\left(c_1\beta^{-1}\right)}{c_2 N}\right)^{1/\max(d,2)}, & \text{if } N \geq \frac{\ln(c_1\beta^{-1})}{c_2} \\ \left(\frac{\ln\left(c_1\beta^{-1}\right)}{c_2 N}\right)^{1/p}, & \text{if } N < \frac{\ln(c_1\beta^{-1})}{c_2} \end{cases}, \tag{2.16}$$

where $c_1, c_2 \in \mathbb{R}$ are constants that depend on d and N. Both approaches depend on some unknown constants and do not make use of available data. Some researchers have proposed to choose the radius ϵ_w without relying on exogenous constants [Zhao and Guan, 2015]. According to the authors, for a discrete nominal distribution $\hat{\mathbb{P}}_w$, the radius of the ambiguity set can be calculated as,

$$\mathbb{P}\left(W_1\left(\mathbb{P}_w, \hat{\mathbb{P}}_w\right) \leq \epsilon_w\right) \geq 1 - \underbrace{e^{\frac{\epsilon_w^2 N}{2\rho^2}}}_{:=\beta}, \tag{2.17}$$

where ρ is the diameter of the support of the true distribution $\mathbb{P}_w$. By solving for ϵ_w we can get,

$$\epsilon_w = \rho\sqrt{\frac{2}{N}\ln\left(\frac{1}{\beta}\right)}. \tag{2.18}$$

$\square$

Tractable Solutions of the DRO

The inner maximization over the ambiguity set in (2.8),

$$\sup_{\mathbb{P}\in\mathcal{B}_{w,p}} \mathbb{E}_{w\sim\mathbb{P}}\left[L(x,w)\right], \tag{2.19}$$

can be very difficult solve. For ease of notation x will be omitted and the objective function is referred as $L(w)$. The primal problem of (2.19) can be written as,

$$v_P = \sup_{\mathbb{P}\in\mathcal{B}_{w,p}} \int_{\mathcal{W}} L(w)\mathbb{P}(dw). \tag{2.20}$$

This is an infinite dimensional optimization problem where finding a tractable solution can be unlikely. For certain specific cases with the ambiguity set $\mathcal{B}_{w,p}$ being a Wasserstein ball centered around a discrete nominal distribution $\hat{\mathbb{P}}_w = \sum_{i=1}^N \delta_{\hat{w}_i}$ with a radius ϵ_w, the problem holds a strong dual solution, as explained in [Gao and Kleywegt, 2016].

$$v_D = \inf_{\lambda\geq 0}\left\{\lambda\epsilon_w^p + \frac{1}{N}\sum_{i=1}^N \sup_{w\in\mathcal{W}}\left[L(w) - \lambda\left\|w - \hat{w}_i\right\|^p\right]\right\} \tag{2.21}$$

The strong duality property causes the primal and the dual solution to be equivalent. The dual problem is easier to work with since instead of optimizing over all the probability distributions in the ambiguity set, the optimization is performed over $\mathcal{W}$, which reduces (2.20) to a finite dimensional optimization problem. However this solution may not be tractable for some $L(\cdot)$. The problem has solutions for specific cases such as where L is a convex function. However for functions that are non-convex, different approximations are available. One solution is to approximate (2.19) by utilizing the Lipschitz constant of $L(\cdot)$ with respect to w.

Definition 2

A Lipschitz continuous function $\phi : \mathbb{R}^d \to \mathbb{R}$ has the property,

$$\|\phi(x) - \phi(y)\| \leq K_\phi \|x - y\|, \quad \forall x, y \in \mathbb{R}^d, x \neq y, \tag{2.22}$$

$\square$

where $\phi(\cdot)$ can change at most by K_ϕ. Trivially, for a function that is not Lipschitz continuous, $K_\phi \to \infty$, which means that the functions growth rate is unbounded.

Theorem 2

The DRO (2.19) has a upper bounded approximation that involves the Lipschitz constant of the objective function $L(\cdot)$ as,

$$\sup_{\mathbb{P} \in \mathcal{B}_{w,p}} \mathbb{E}_{w \sim \mathbb{P}}[L(w)] \leq \mathbb{E}_{w \sim \hat{\mathbb{P}}_w}[L(w)] + \epsilon_w K_L, \tag{2.23}$$

$\square$

where K_L is the Lipschitz constant of the objective function $L(\cdot)$ with respect to w.

The proof of Theorem 2 can be found in [Kuhn et al., 2019] Theorem 5. If the objective function $L(\cdot)$ satisfies to be Lipschitz, then Theorem 2 is trivially satisfied. The upper bounded solution causes the Wasserstein ball radius ϵ_w to be larger than intended. This may be favorable for some problems where a risk averse solution is sought.

3

Problem Statement

In this chapter, we define the problem that this thesis is built around. Specifically, we define how the uncertain robot is modeled and we specify where it lives. After the robot and environment modeling, we describe the risk averse path planning problem statement cast as a Markov Decision Process, where the robot is expected to find a path from a source to destination without colliding with obstacles in a risk averse manner.

3.1 Robot & Environment Model

The robot is modeled as a stochastic discrete time linear time invariant system and it is assumed to move within a bounded environment $\mathcal{X} \subset \mathbb{R}^{n_x}$. There are in total $M \in \mathbb{Z}_+$ obstacles in the environment, each disjoint with the other and they are collectively referred as $\mathcal{O}$ with $|\mathcal{O}| = M$. Further, each obstacle is assumed to be static and circular in shape with constant radius $R_{\mathbf{obs}}^{(i)} > 0$, $\forall i \in \mathcal{O}$. Then, the free space that the robot can traverse namely $\mathcal{X}_{\mathbf{free}} \subset \mathcal{X}$ is given by,

$$\mathcal{X}_{\mathbf{free}} = \mathcal{X} \backslash \mathcal{X}_{\mathbf{obs}}, \quad \text{and} \quad \mathcal{X}_{\mathbf{obs}} := \bigcup_{i=1}^{M} \mathcal{X}_{\mathbf{obs}}^{(i)}, \tag{3.1}$$

where $\mathcal{X}_{\mathbf{obs}}^{(i)} \subset \mathcal{X}$ is the space occupied by the obstacle $i \in \mathcal{O}$. Similar to the obstacles, we define a goal region, $\mathcal{X}_{\mathbf{goal}} \subset \mathcal{X}$, that is both static and circular in shape with constant radius $R_{\mathbf{goal}} > 0$. All the robot states that are inside the region $\mathcal{X}_{\mathbf{goal}}$ are considered to be goal states. The robot is limited to move within the environmental boundaries. Hence, just like the static obstacles, the environmental boundaries are also treated as terminal states. The position of the robot at time $k \in \mathbb{Z}_+$ is formulated as $p_{r,k} \in \mathbb{R}^{n_r}$. The state of the robot at time k is represented as $x_k \in \mathbb{R}^{n_x}$ and it may include the robot's position, velocity and other states of interest so that $n_x \geq n_r$. The robot is controlled through a control input u_k which is selected from $\mathcal{U}$ such that $u_k \in \mathcal{U} \subseteq \mathbb{R}^{n_u}$.

Given the above description, we define the dynamics (evolution) of the robot in $\mathcal{X}$ as,

$$x_{k+1} = Ax_k + Bu_k + w_k, \tag{3.2}$$

where $A \in \mathbb{R}^{n_x \times n_x}$ is the state matrix and $B \in \mathbb{R}^{n_x \times n_u}$ is the input matrix. The robot is subject to a process disturbance $w_k \in \mathbb{R}^{n_x}$. The time invariant true distribution of the process disturbance w_k at any time k namely $\mathbb{P}_w$ is unknown, however it is assumed that a collection of $N \in \mathbb{N}$ independent samples of w_k are available beforehand. That is, an i.i.d sequence $\hat{w}_1, \ldots, \hat{w}_N \in \mathbb{R}^{n_r}$ is assumed to be known in advance. At any time k, the distribution of w_k can be approximated through the following empirical distribution,

$$\hat{\mathbb{P}}_w := \frac{1}{N} \sum_{i=1}^{N} \delta_{\hat{w}_i}, \tag{3.3}$$

where $\delta_{\hat{w}_i}$ is the Dirac delta function. Note that, $\hat{\mathbb{P}_w}$ need not necessarily be the true distribution of the process disturbance w_k. The initial state of the robot is assumed to be random and it is modelled as $x_0 \sim \mathbb{P}_{x_0}(\bar{x}_0, \Sigma_{x_0})$, where $\mathbb{P}_{x_0}$ is assumed to be known with the mean $\bar{x}_0 \in \mathbb{R}^{n_x}$, and the covariance $\Sigma_{x_0} \in \mathbb{R}^{n_x \times n_x}$ also being assumed to be known or estimated from prior experiments. It is clear from the above setting that $\mathbb{P}_{x_k}$ for $k \geq 1$ is not known exactly despite $\mathbb{P}_{x_0}$ being known exactly.

Assumption 2

The set of obstacles and the goal are assumed to be disjoint with each other and further there exists a minimum separation distance $L_{min} > R_{\mathbf{obs}} + R_{\mathbf{goal}}$ between the goal and any of the obstacle regions. That is $\mathcal{X}_{\mathbf{goal}} \cap \mathcal{X}_{\mathbf{obs}} = \emptyset$ and $\forall x_{\mathbf{goal}} \in \mathcal{X}_{\mathbf{goal}}$, $\forall x_{\mathbf{obs}} \in \mathcal{X}_{\mathbf{obs}}$, we see that,

$$\|x_{\mathbf{goal}} - x_{\mathbf{obs}}\| \geq L_{min}. \tag{3.4}$$

□

Main Problem:

The main problems investigated in this thesis are enumerated as follows:

1. Learn the control actions for the robot at all the state space positions such that actions should enable the robot to move towards the goal and avoid the set of static obstacles.

2. Using the learned control actions, find a trajectory, from the initial position $p_{r,0}$ to the goal state $p_g \in \mathcal{X}$ without colliding with any of the obstacles.

3. Learn the control policy and design the trajectory such that the uncertainty due to the random initial condition of the robot states and the available sample based uncertainty modeling of the process noise are taken into account.

An illustration of the environment can be seen in Figure 3.1.

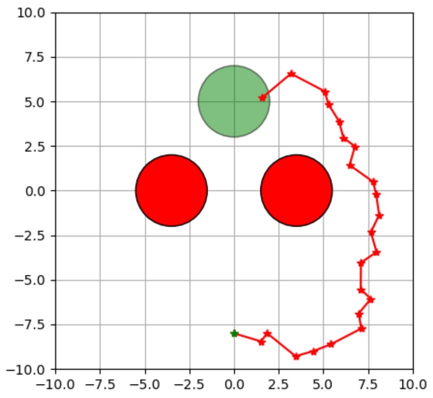


Figure 3.1 Example environment with two obstacles(red) and a goal(green). The red line represents the trajectory of the robot with the initial point of the robot shown with the green marker.

3.2 Markov Decision Process (MDP) Formulation

The above planning problem can be cast as a Markov decision process that consists of the tuple $\langle \mathcal{S}, \mathcal{A}, \mathbb{P}, r \rangle$. Here, $\mathcal{S} \subset \mathbb{R}^{n_\mathcal{S}}$ is the continuous state space, $\mathcal{A} \subset \mathbb{R}^{n_\mathcal{A}}$ is a finite set called the action space with $|\mathcal{A}| \in \mathbb{N}_+ \backslash 0$, $r : \mathcal{S} \to \mathbb{R}$ is the reward function and $\mathbb{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is the state transition probability which defines the probability distribution over the next states. Due to the Markov property of the system, the transition probabilities only depend on previous state and action such that for a s_k and action a_k and the history $h_k := \{s_0, a_0, \ldots, s_k, a_k\}$, we see that $\mathbb{P}(s_{k+1} \mid h_k) = \mathbb{P}(s_{k+1} \mid s_k, a_k), \forall s_{k+1} \in \mathcal{S}$. At step k, the state $s_k \in \mathcal{S}$ contains the state of the robot x_k, the center of the goal p_g, and the centers of the obstacles $p_{\textbf{obs}}^{(i)}$.

$$s_k := \left\{ x_k, p_g, p_{\textbf{obs}}^{(i)} \right\} \in \mathbb{R}^{n_\mathcal{S}}, \quad n_\mathcal{S} = (2+M)n_x, \quad i = 1, \ldots, M. \tag{3.5}$$

The state s_k is referred as a `terminal state` if $x_k \in \mathcal{X}_{\textbf{obs}} \cup \mathcal{X}_{\textbf{goal}}$ or if $x_k \notin \mathcal{X}$ and as `non-terminal state` otherwise. The dimensions of MDP state s_k depend on the number of obstacles on the environment. Increasing the number of obstacles will cause the dimension of s_k to increase as well [1]. An action $a_k \in \mathcal{A}$ performed at state $s_k \in \mathcal{S}$, will cause a transition to a new state $s_{k+1} \in \mathcal{S}$ with the probability $\mathbb{P}(s_{k+1} \mid s_k, a_k)$. After the transition, a deterministic reward $r_k = r(s_{k+1})$ is obtained based on the state where we land. The actions, $a_k = \pi(s_k)$ are chosen based on a deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$. The set of all admissible control policies is denoted by Π. A sequence $\tau_k^{(\pi)} := (s_k, a_k, s_{k+1}, a_{k+1}, \ldots, s_{K-1}, a_{K-1}, s_K)$ with a terminal state s_K is called a sample path under the policy $\pi \in \Pi$. The cumulative discounted reward for this sample path is,

$$R^\pi(\tau_k) = \sum_{i=0}^{K-k-1} \gamma^i r(s_{i+k+1}), \tag{3.6}$$

where $\gamma \in [0, 1]$ is the discount factor. The discount factor is used in order to take in to account the future rewards. Due to the randomness in the transitions, the value function is defined as the expected value calculated for the discounted returns starting from state s and following policy π. The value function $V : \mathcal{S} \to \mathbb{R}$ defines the expected discounted returns starting from the state s and following policy π. That is,

$$V^\pi(s) := \mathbb{E}[R^\pi(\tau_k \mid s_k = s)]. \tag{3.7}$$

We also define the Q-function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ which represents the expected discounted returns for taking action a at state s and following policy π,

$$Q^\pi(s, a) := \mathbb{E}\left[\sum_{k=0}^{K-1} \gamma^k r(s_{k+1}) \mid s_0 = s, a_0 = a\right]. \tag{3.8}$$

The purpose is to find a policy $\pi \in \Pi$ that maximizes the cumulative discounted rewards for all the states in $\mathcal{S}$. If the Q-function is known for each state and action pair, a policy can be formulated that chooses the best action available at each state as,

$$\pi(s) := \arg\max_{a \in \mathcal{A}} Q^\pi(s, a). \tag{3.9}$$

3.3 Reward Function Design

Typically, the reward function depends on the current state s, current action a and the next state s'. However, we consider rewards that depend on only s',

[1] The increase in dimension of s_k is the price that we need to pay to handle potentially dynamic obstacles.

that is used in [Raajan et al., 2020]. We consider a penalty for both traveling and collision with obstacles. Further, there is an incentive for being in the goal. The reward function $\hat{r} : \mathcal{S} \to \mathbb{R}$ is defined as,

$$\hat{r}(s') = \mathbf{r_{travel}} + \begin{cases} \mathbf{r_{goal}}, & \text{if } s' \in \mathcal{X}_{\mathbf{goal}}, \\ \mathbf{r_{obs}}, & \text{if } s' \in \mathcal{X}_{\mathbf{obs}} \text{ or } s' \notin \mathcal{X}. \end{cases} \tag{3.10}$$

where $\mathbf{r_{travel}}$ is the travel penalty, $\mathbf{r_{goal}}$ is the reward for reaching the goal, and $\mathbf{r_{obs}}$ is the penalty for obstacle collision or leaving the environment. The surface plot of the reward function $\hat{r}(\cdot)$ can be seen in Figure 3.2. The

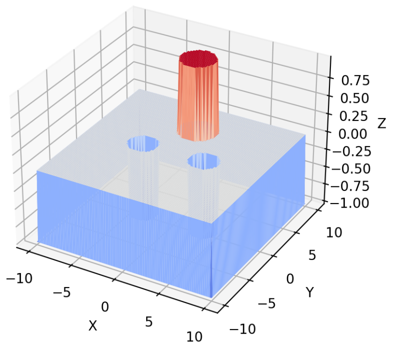


Figure 3.2 Discontinuous reward function for an environment with 2 obstacles (pits) and a goal (peak) and $\mathbf{r_{travel}} = -0.001$, $\mathbf{r_{goal}} = 1$, $\mathbf{r_{obs}} = -1$.

discontinuity in the reward function $\hat{r}(\cdot)$ causes its Lipschitz constant $K_{\hat{r}} \to \infty$, which will be important in further chapters. For this reason, it has to be approximated by a Lipschitz continuous function. The continuous reward function can be separated into 4 parts namely, 1) constant travel cost, 2) goal reward, 3) obstacle collision penalty, and 4) penalty for travelling outside the environment. For the obstacles and the goal, the radial step function can be approximated with a tanh function with the shape,

$$f(x) = \frac{A_r}{2}\left(1 + \tanh\left(\frac{R_{\mathbf{rad}} - \|p_r - p\|_2}{\delta}\right)\right), \tag{3.11}$$

where $p_r \in \mathbb{R}^{n_r}$ denotes the position of the robot, $p \in \mathbb{R}^{n_r}$ denotes either the center of the obstacle or that of the goal, $R_{\mathbf{rad}} \in \mathbb{R}_{>0}$ is the radius of the obstacle or the goal, $A_r \in \mathbb{R}$ is the gain and $\delta \in \mathbb{R}_{>0}$ is the slope. By using the distance to the border of the obstacle/goal, the resulting function becomes a radially symmetric hyperbolic tangent function. A smaller δ will result in a steeper surface near the border. A similar structure can be used for the borders that takes the distance to the borders across all the position dimensions, with the limits of the environment being $[\underline{p}, \overline{p}]$ and $\underline{p}, \overline{p} \in \mathbb{R}^{n_r}$. Then, the continuous approximation of the reward function in (3.10) can be written as,

$$\begin{aligned} r(s') = {} & \mathbf{r_{travel}} \\ & + \frac{\mathbf{r_{obs}}}{2} \sum_{j=1}^{n_r} \left(2 + \tanh\left(\frac{\underline{p}(j) - p_r(j)}{\delta_1} \right) + \tanh\left(\frac{p_r(j) - \overline{p}(j)}{\delta_2} \right) \right) \\ & + \frac{\mathbf{r_{goal}}}{2} \left(1 + \tanh\left(\frac{R_{\mathbf{goal}} - \|p_r - p_g\|_2}{\delta_3} \right) \right) \\ & + \sum_{i=1}^{M} \frac{\mathbf{r_{obs}}}{2} \left(1 + \tanh\left(\frac{R^{(i)}_{\mathbf{obs}} - \left\| p_r - p^{(i)}_{obs} \right\|_2}{\delta_4} \right) \right). \end{aligned} \tag{3.12}$$

For simplicity, we let $\delta_i = \delta, \forall i = 1, \ldots, 4$. However, this is a tuning parameter that one can choose to shape the reward function appropriately. The continuous reward function can be seen in Figure 3.3. Figure 3.4 shows a

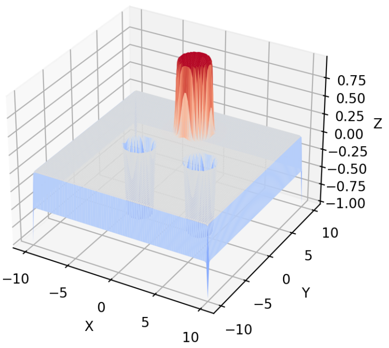


Figure 3.3 Continuous reward function for an environment with 2 obstacles (pits) and a goal (peak) and $\mathbf{r_{travel}} = -0.001$, $\mathbf{r_{goal}} = 1$, $\mathbf{r_{obs}} = -1$.

comparison between the reward functions $\hat{r}(\cdot)$ and $r(\cdot)$.

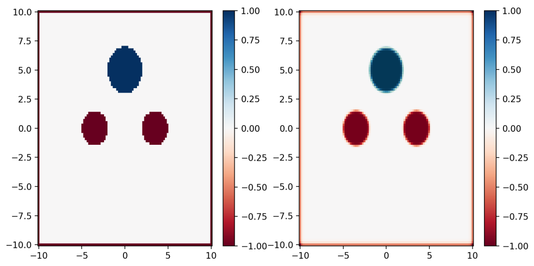


Figure 3.4 Comparison between the discontinuous and the continuous reward function.

4

Method

In this chapter we explain our solution to the problem given in Chapter 3. We will go over the Q-Learning theory and implementations before we show our proposed solution Distributionally Robust Q-Learning.

4.1 Q-Learning (DQN)

The Q-values represent the expected discounted returns if action a is taken at state s. If the Q-values for a system is known, a policy π can be used to maximize the expected returns. In order to estimate the Q-values, the standard temporal difference learning based Q-Learning procedure is usually employed, [Watkins and Dayan, 1992]. Q-learning is a reinforcement learning method that makes use of interactions between the agent (robot) and the environment. The purpose is to learn the relationship between the actions and the expected rewards by utilizing past experiences. From now on we will drop the superscript π on $Q^{\pi}(s,a)$ for brevity of the notation. Similarly we will denote the current state s_k as s, the next state s_{k+1} as s' and the action a_k as a. Q-Learning makes use of the Bellman operator $\mathcal{T} : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$, which is defined as,

$$\mathcal{T}Q(s,a) := \mathbb{E}_{s'}\left[r(s') + \gamma \max_{a'\in\mathcal{A}} Q(s',a')\right], \tag{4.1}$$

where the expectation is over the next states s', which come from the transition probability $\mathbb{P}_{s'} := \mathbb{P}(s' \mid s,a)$, where s and a are not included in the notation for sake of notation.

Definition 3

The operator F is a α-contraction with respect to some norm $\|.\|$ and some argument function X if

$$\forall X, \bar{X} : \quad \|FX - F\bar{X}\| \leq \alpha \|X - \bar{X}\| \tag{4.2}$$

□

LEMMA 1
The Bellman operator $\mathcal{T}$ defined in (4.1) is a γ-contraction such that for two Q-functions Q_1 and Q_2,

$$\|\mathcal{T}Q_1 - \mathcal{T}Q_2\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty. \tag{4.3}$$

□

Proof. For two Q-functions $Q_1, Q_2 : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}$, the infinite norm $\|\cdot\|_\infty$ is defined as,

$$\|Q_1 - Q_2\|_\infty = \max_{s\in\mathcal{S}, a\in\mathcal{A}} |Q_1(s,a) - Q_2(s,a)|. \tag{4.4}$$

The infinite norm for the Bellman operator can be written as,

$$\|\mathcal{T}Q_1 - \mathcal{T}Q_2\|_\infty = \max_{s\in\mathcal{S}, a\in\mathcal{A}} |\mathcal{T}Q_1(s,a) - \mathcal{T}Q_2(s,a)|. \tag{4.5}$$

By substituting the definition of the Bellman operator in (4.1) it can be proven that,

$$\begin{aligned} \mathcal{T}Q(s,a) &= \mathbb{E}_{s'}[r(s' + \gamma \max_{a'\in\mathcal{A}} Q(s',a')] \\ &= \int \left(r(s') + \gamma \max_{a'\in\mathcal{A}} Q(s',a') \right) \mathbb{P}(s' \mid s,a) ds' \end{aligned} \tag{4.6}$$

$$\begin{aligned} &\max_{s\in\mathcal{S}, a\in\mathcal{A}} |\mathcal{T}Q_1(s,a) - \mathcal{T}Q_2(s,a)| \\ &= \max_{s\in\mathcal{S}, a\in\mathcal{A}} \left| \gamma \int \left(\max_{a'\in\mathcal{A}} Q_1(s',a') - \max_{a'\in\mathcal{A}} Q_2(s',a') \right) \mathbb{P}(s' \mid s,a) ds' \right| \\ &\leq \gamma \max_{s'\in\mathcal{S}} \left| \max_{a'\in\mathcal{A}} Q_1(s',a') - \max_{a'\in\mathcal{A}} Q_2(s',a') \right| \\ &\leq \gamma \max_{s'\in\mathcal{S}, a'\in\mathcal{A}} |Q_1(s',a') - Q_2(s',a')| = \gamma \|Q_1 - Q_2\|_\infty. \end{aligned} \tag{4.7}$$

□

Since the Bellman operator is a contraction operator, the Q-values will converge to an optimal value $Q^*(s,a)$. At the optimal Q-values $Q^*(s,a)$, the Bellman operator converges to a fixed point.

$$\mathcal{T}Q^*(s,a) = Q^*(s,a), \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \tag{4.8}$$

The optimal point for the Q-values can be found by iteratively applying the bellman operator $\mathcal{T}$ since it is a contraction. At the optimal point, the Q-function will represent the expected cumulative rewards for taking an action a

at state s. The Q-values can be updated by interacting with the environment and gathering experiences in the form of $\langle s, a, r, s' \rangle$. The simplest implementation of Q-Learning [Watkins and Dayan, 1992] uses a table to record the Q-values. However this becomes infeasible if the state space is continuous or the number of states is simply too large. A method that can handle continuous states as well as discrete states called Deep Q-Learning (DQN) [Mnih et al., 2013], estimates the Q-values by using a nonlinear function such as a deep neural network. Specifically, it utilizes two neural networks namely: i) Q-network $Q(s, a; \theta)$ and ii) the target network $Q(s, a; \theta^{-})$, where θ, θ^{-} are the weights of the networks. The Q-network estimates the Q-values for the state-action pairs and is used as the policy. The target network is utilized when generating the targets in the training process. The two networks have the same architecture and every Γ steps, the weights of the Q-network are copied to the target network. The reasoning behind this is to have a stable target during training. If the targets are generated by the Q-network, the targets can become non-stationary and can cause problems with the methods stability. The networks have $n_{\mathcal{S}}$ inputs and $n_{\mathcal{A}}$ outputs such that the Q-values for each action is calculated in one pass. At each step, the state s, the action taken a, the reward r, and the next state s' is stored as an experience in a memory buffer $\mathcal{M}$. The buffer has a fixed size and once its full, the oldest experience is replaced by the new one. At each step, the Q-network is trained with experience replay, where a random batch of experiences are sampled from the buffer $\mathcal{M}$ and with the Bellman equation, the targets are calculated. For a sampled experience $\langle s_j, a_j, r_j, s'_j \rangle$, the target y_j is calculated as,

$$y_j = \begin{cases} r_j + \gamma \max_{a' \in \mathcal{A}} Q(s'_j, a'; \theta^{-}), & \text{if } s'_j \text{ is non-terminal,} \\ r_j, & \text{if } s'_j \text{ is terminal.} \end{cases} \tag{4.9}$$

After the targets have been calculated for a batch of N_{batch} experiences, the loss is computed as,

$$L = \frac{1}{N_{batch}} \sum_{j=1}^{N_{batch}} \left(Q\left(s_j, a_j; \theta\right) - y_j \right)^2 \tag{4.10}$$

The Q-network is trained on L using backpropagation to reduce the loss. This process is repeated for many steps until the Q-network converges. During training, the actions are selected with an ϵ-greedy policy, where the action will be selected randomly from $\mathcal{A}$ with probability $\epsilon \in [0, 1]$, or from the current policy $Q(s, a; \theta)$ with probability $1 - \epsilon$. The reasoning behind this is to force the agent to explore the state space, rather than exploiting the current policy. The selection of ϵ is an important matter since it determines whether the agent will focus more on exploration or exploitation. A common practice in reinforcement learning is to explore in the beginning of the training, and focus more on exploiting in the late stages of training with some small $\epsilon > 0$. The pseudocode for Deep Q-Learning can be seen in Algorithm 1.

Algorithm 1 Deep Q-learning with Experience Replay

```
Initialize replay memory $\mathcal{M}$ to capacity $N_{mem}$
Initialize action-value function Q with random weights $\theta$
Set $\theta^- \leftarrow \theta$
for episode= $1 : N_{ep}$ do
    Initialize state $s \in \mathcal{S}$
    for $step = 0 : N_{step}$ do
        With probability $\epsilon$ select a random action from $\mathcal{A}$, otherwise select
        $a = \arg\max_{a'} Q(s, a'; \theta)$
        Execute action $a$ and observe reward $r$ and state $s'$
        Store transition $\langle s, a, r, s' \rangle$ in $\mathcal{M}$
        Set $s \leftarrow s'$
        for $j = 1 : N_{batch}$ do
            Uniformly sample transition $\langle s_j, a_j, r_j, s'_j \rangle$ from $\mathcal{M}$
            Set $y_j = \begin{cases} r_j & \text{for terminal } s'_j \\ r_j + \gamma \max_{a'} Q(s'_j, a'; \theta^-) & \text{for non-terminal } s'_j \end{cases}$
        Compute loss $L = \frac{1}{N_{batch}} \sum_{j=1}^{N_{batch}} \left(Q\left(s_j, a_j; \theta\right) - y_j\right)^2$
        Perform a gradient descent step on $L$
        Set $\theta^- = \theta$ every $\Gamma$ steps
```

4.2 Distributionally Robust Q-Learning (DRDQN)

In this section, the Q-function defined in section 4.1 will be formulated as a DRO problem. As mentioned in section 3.1, the process disturbance w comes from an unknown distribution $\mathbb{P}_w$. This causes the transitions between the states to be stochastic. An action a taken at state s will cause a transition to state s' with distribution $\mathbb{P}_{s'} := \mathbb{P}(s' \mid s, a)$. In order to achieve a risk averse solution to the MDP in section 3.2, an ambiguity set can be constructed for the unknown transition distribution $\mathbb{P}_{s'}$. Since we have access to N samples of w, given a state s_k and an action a_k, an empirical distribution can be computed by using the samples of w.

The Wasserstein Ambiguity Set

Given a robot state $x_k \in \mathcal{X}$ and an input $a_k \in \mathcal{A}$, an empirical distribution for x_{k+1} is given by,

$$\hat{\mathbb{P}}_{x_{k+1}} := \frac{1}{N}\sum_{i=1}^{N} \delta_{\hat{x}_{k+1}^{(i)}} = \frac{1}{N}\sum_{i=1}^{N} \delta_{Ax_k + Ba_k + \hat{w}_i}. \tag{4.11}$$

By knowing the state of the robot x_k, the full state s_k can be obtained by using the positions of the goal and obstacles of the current environment (which do not depend on the position of the robot), since these stay constant during an episode. We refer to the samples of s' obtained from (4.11) as $\hat{s}'^{(i)}$ for $i = 1, \ldots, N$.

$$\hat{\mathbb{P}}_{s'} = \frac{1}{N}\sum_{i=1}^{N} \delta_{\hat{s}'(i)} \tag{4.12}$$

When an action a is performed while in state s, the nominal distribution for the center of the Wasserstein ball will be $\hat{\mathbb{P}}_{s'}$, and the worst case transition will be coming from a distribution that is inside this ball. We define the ambiguity set as $\mathcal{B}_{s,a}$,

$$\mathcal{B}_{s,a} := \left\{ \mathbb{P} \in \mathcal{P}(\mathcal{S}) \mid W_1\left(\hat{\mathbb{P}}_{s'}, \mathbb{P}\right) \leq \epsilon_{s'} \right\}, \tag{4.13}$$

where $\hat{\mathbb{P}}_{s'}$ is the empirical distribution of the transition given s, a, which is computed using the empirical distribution $\hat{\mathbb{P}}_w$. The Wasserstein ball radius $\epsilon_{s'}$ is chosen such that the true distribution $\mathbb{P}_{s'}$ lies within this Wasserstein ball with probability greater than $1 - \beta$. The β parameter will determine the allowed risk factor for the solution. A smaller β will result in a larger radius which causes the generated policy to be much more risk averse and vice-versa. Since, the radius $\epsilon_{s'}$ quantifies the amount of trust(distrust) that we have over $\hat{\mathbb{P}}_{s'}$, it is chosen such that,

$$\mathbb{P}(\mathbb{P}_{s'} \in \mathcal{B}_{s,a}) \geq 1 - \beta, \quad \beta \in [0, 1]. \tag{4.14}$$

As shown in Theorem 1, and given Assumption 1, the Wasserstein radius $\epsilon_{s'}$ that satisfies (4.14) as,

$$\epsilon_{s'} = \rho\sqrt{\frac{2}{N}\ln\left(\frac{1}{\beta}\right)}, \tag{4.15}$$

where $\rho = \mathbf{diam}(\mathbf{supp}(\hat{\mathbb{P}}_{s'}))$. Although according to Theorem 1, ρ is calculated with respect to the true distribution $\mathbb{P}_{s'}$, since this is unknown to us, we chose to approximate it with the nominal distribution $\hat{\mathbb{P}}_{s'}$.

Approximated Solution to the Wasserstein Distributionally Robust Q-Learning Problem

In order to achieve a risk averse policy, we can modify the Bellman operator in (4.1) such that it gives the worst case expected returns by turning it into a DRO problem over the ambiguity set defined in (4.13). We define the distributionally robust Bellman operator $\hat{\mathcal{T}} : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$ to represent the worst case expected returns, so that risk can be incorporated into the Q-values. That is,

$$\hat{\mathcal{T}}Q(s,a) := \inf_{\mathbb{P}\in\mathcal{B}_{s,a}} \mathbb{E}_{s'\sim\mathbb{P}}[h(s')], \quad \text{where,} \tag{4.16}$$

$$h(s') := \underbrace{r(s')}_{:=h_r(s')} + \underbrace{\gamma \max_{a'\in\mathcal{A}} Q(s',a')}_{:=h_Q(s')}. \tag{4.17}$$

Since the Q-function is approximated using a neural network with hidden layers and non-linear activation functions, $h(s')$ turns out to be a non-convex function of states. Since an exact solution to the infinite dimensional problem (4.16) using duality theory (2.21) is difficult to find when the objective function is non-convex, we resort to the Lipschitz constant based approximation in Theorem 2.

Lemma 2
The Lipschitz approximation in Theorem 2 has an equivalent solution for the case when the objective function h is to be minimized and this results in a lower bound for (4.16), where $\epsilon_{s'}$ becomes larger in practice and the solution can become more risk averse. That is,

$$\inf_{\mathbb{P}\in\mathcal{B}_{s,a}} \mathbb{E}_{s'\sim\mathbb{P}}[h(s')] \geq \mathbb{E}_{s'\sim\hat{\mathbb{P}}_{s'}}[h(s')] - \epsilon_{s'} K_h, \tag{4.18}$$

□

where K_h is the Lipschitz constant of $h(\cdot)$.

Proof. The Lipschitz approximation in Theorem 2 can be converted into a minimization problem by switching the objective function $h(\cdot)$ with $-h(\cdot)$ and multiplying the result with -1 as,

$$\inf_{\mathbb{P}\in\mathcal{B}_{s,a}} \mathbb{E}_{s'\sim\mathbb{P}}[h(s')] = -\sup_{\mathbb{P}\in\mathcal{B}_{s,a}} \mathbb{E}_{s'\sim\mathbb{P}}[-h(s')]. \tag{4.19}$$

If we substitute $h(\cdot)$ with $-h(\cdot)$ in the maximization problem, the Lipschitz approximation becomes,

$$\begin{aligned} &\sup_{\mathbb{P}\in\mathcal{B}_{s,a}} \mathbb{E}_{s'\sim\mathbb{P}}[-h(s')] \leq \mathbb{E}_{s'\sim\hat{\mathbb{P}}_{s'}}[-h(s')] + \epsilon_{s'} K_{-h} \\ \iff &-\sup_{\mathbb{P}\in\mathcal{B}_{s,a}} \mathbb{E}_{s'\sim\mathbb{P}}[-h(s')] \geq -\mathbb{E}_{s'\sim\hat{\mathbb{P}}_{s'}}[-h(s')] - \epsilon_{s'} K_{-h}, \end{aligned} \tag{4.20}$$

where K_{-h} is the Lipschitz constant of $-h(\cdot)$. By using (4.18), a minimization problem can be written as,

$$\begin{aligned}\iff \inf_{\mathbb{P}\in\mathcal{B}_{s,a}} \mathbb{E}_{s'\sim\mathbb{P}}[h(s')] &\geq -\mathbb{E}_{s'\sim\hat{\mathbb{P}}_{s'}}[-h(s')] - \epsilon_{s'}K_{-h} \\ \iff \inf_{\mathbb{P}\in\mathcal{B}_{s,a}} \mathbb{E}_{s'\sim\mathbb{P}}[h(s')] &\geq \mathbb{E}_{s'\sim\hat{\mathbb{P}}_{s'}}[h(s')] - \epsilon_{s'}K_{-h}. \qquad (4.21)\end{aligned}$$

The Lipschitz constant K_{-h} is equivalent to K_h since,

$$\underbrace{\|-h(x)-(-h(y))\|}_{=\|h(x)-h(y)\|} \leq K_{-h}\|x-y\|, \quad \forall x,y\in\mathcal{S}, x\neq y. \qquad (4.22)$$

Hence,

$$\inf_{\mathbb{P}\in\mathcal{B}_{s,a}} \mathbb{E}_{s'\sim\mathbb{P}}[h(s')] \geq \mathbb{E}_{s'\sim\hat{\mathbb{P}}_{s'}}[h(s')] - \epsilon_{s'}K_h. \qquad (4.23)$$

□

Calculating the Lipschitz Constant K_h of $h(s')$

The Lipschitz constant for $h_r(s')$ and $h_Q(s')$ can be calculated or estimated independently and then combined to get the Lipschitz constant of $h(s')$.

LEMMA 3
Let $f_i:\mathbb{R}^n\to\mathbb{R}, i=1,\dots,N$ be Lipschitz continuous with constants K_{f_i}. Then the Lipschitz constant of the function $g(x)=\sum_{i=1}^N f_i(x)$ is,

$$K_g = \sum_{i=1}^N K_{f_i} \qquad (4.24)$$

□

Proof. We will prove for the case $N=2$ and the result for $N>2$ follows similarly. For the two Lipschitz continuous functions f_1, f_2,

$$\begin{aligned}\|f_1(x)-f_1(y)\| &\leq K_1\|x-y\|, \quad \forall x,y\in\mathbb{R}^n, x\neq y \\ \|f_2(x)-f_2(y)\| &\leq K_2\|x-y\|, \quad \forall x,y\in\mathbb{R}^n, x\neq y \qquad (4.25)\end{aligned}$$

To find the Lipschitz constant of the function $g=f_1+f_2$, we see that,

$$\begin{aligned}\|g(x)-g(y)\| &= \|f_1(x)+f_2(x)-f_1(y)-f_2(y)\| \\ &\leq \|f_1(x)-f_1(y)\| + \|f_2(x)-f_2(y)\| \\ &\leq \underbrace{(K_1+K_2)}_{:=K_g}\|x-y\|, \quad \forall x,y\in\mathbb{R}^n, x\neq y \qquad (4.26)\end{aligned}$$

□

Given Assumption 2 and the reward function being a sum of tanh functions, its Lipschitz constant can be inferred by looking at the individual tanh functions.

LEMMA 4
Let $A_r \in \mathbb{R}$, $\delta \in \mathbb{R}_{>0}$. Then the Lipschitz constant of the scalar function $f : \mathbb{R} \to \mathbb{R}$ with the shape,

$$f(x) = \frac{A_r}{2}\left(1 + \tanh\left(\frac{x}{\delta}\right)\right), \tag{4.27}$$

□

is $K_f = \frac{|A_r|}{2\delta}$.

Proof. For the given scalar functional, its Lipschitz constant corresponds to the maximum magnitude of its slope.

$$\begin{aligned} K_f &= \sup_{x\in\mathbb{R}} \left|\frac{d}{dx} f(x)\right| \\ \frac{d}{dx} f(x) &= \frac{A_r}{2\delta} \underbrace{\left(1 - \tanh^2\left(\frac{x}{\delta}\right)\right)}_{\geq 0} \\ \left|\frac{d}{dx} f(x)\right| &= \frac{|A_r|}{2\delta}\left(1 - \tanh^2\left(\frac{x}{\delta}\right)\right) \\ \frac{d}{dx}\left|\frac{d}{dx} f(x)\right| &= -\frac{|A_r|}{\delta^2} \underbrace{\left(1 - \tanh^2\left(\frac{x}{\delta}\right)\right)}_{\to 0 \text{ as } x \to \pm\infty} \tanh\left(\frac{x}{\delta}\right) \end{aligned} \tag{4.28}$$

By solving for $\frac{d}{dx}\left|\frac{d}{dx} f(x)\right| = 0$, it can be seen that the maximum slope occurs when $x \to \pm\infty$ or $x = 0$. Thus the Lipschitz constant can be found as,

$$K_f = \left|\frac{d}{dx} f(0)\right| = \frac{A_r}{2\delta} \tag{4.29}$$

□

LEMMA 5
Let $A_r \in \mathbb{R}$, $\delta \in \mathbb{R}_{>0}$, $R \in \mathbb{R}_{>0}$ and $p, g \in \mathbb{R}^2$. Then the Lipschitz constant of the function $f : \mathbb{R}^2 \to \mathbb{R}$ with the shape,

$$f(p) = \frac{A_r}{2}\left(1 + \tanh\left(\frac{R - \|p - g\|_2}{\delta}\right)\right), \tag{4.30}$$

□

is $K_f = \frac{|A_r|}{2\delta}$.

Proof. Assume $p = [x, y]$ and $g = [g_x, g_y]$ and $f(p) = \frac{A_r}{2}(1 + \tanh(h(p)))$, where

$$h(p) = \frac{R - \|p - g\|_2}{\delta} \tag{4.31}$$

The maximum slope can be computed by using the absolute value of the partial derivatives, since f is radially symmetric.

$$\begin{aligned}
\frac{\partial f}{\partial x} &= \frac{\partial f}{\partial h}\frac{\partial h}{\partial x}\\
\frac{\partial f}{\partial h} &= \frac{A_r}{2}\left(1-\tanh^2(h)\right)\\
\frac{\partial h}{\partial x} &= -\frac{1}{\delta}2(x-g_x)\frac{1}{2}\frac{1}{\|p-g\|_2}\\
\frac{\partial f}{\partial x} &= \frac{A_r}{2}\underbrace{\left(1-\tanh^2\left(\frac{R-\|p-g\|_2}{\delta}\right)\right)}_{\geq 0}\left(-\frac{1}{\delta}\right)\frac{1}{\|p-g\|_2}(x-g_x)\\
\left|\frac{\partial f}{\partial x}\right| &= \frac{|A_r|}{2\delta}\left(1-\tanh^2\left(\frac{R-\|p-g\|_2}{\delta}\right)\right)\frac{1}{\|p-g\|_2}|x-g_x| \qquad (4.32)
\end{aligned}$$

As shown in the proof of Lemma 4, the maximum slope occurs at $\tanh(0)$, which corresponds to $\|p-g\|_2 = R$. The points that satisfy this create a circle at the border of the obstacle/goal. For any point on this circle such as $|x-g_x| = R$ and $y = g_y$, the magnitude of the slope becomes $\frac{|A_r|}{2\delta}$. Thus the Lipschitz constant of the function f is $K_f = \frac{|A_r|}{2\delta}$. □

THEOREM 3
Given Assumption 2, the Lipschitz constant of the reward function $h_r(\cdot)$ given by (3.12) is,

$$K_r = \frac{\max\{|\mathbf{r_{goal}}|, |\mathbf{r_{obs}}|\}}{2\delta}. \qquad (4.33)$$

□

Proof. Based on Assumption 2, the individual terms that contribute to the total reward function r in (3.12) do not interfere with each other. Then, it follows that the Lipschitz constant of r is equivalent to the maximum of the Lipschitz constants of the individual terms. □

The second part of the objective function $h(s')$ given by $h_Q(s')$ contains the Q-function which is approximated by a neural network. The neural network takes state s as an input and returns the Q-values for each action. There are several methods available to estimate the Lipschitz constant of a deep neural network with ReLU activations such as FastLip [Weng et al., 2018a], LipSDP [Fazlyab et al., 2019], CLEVER [Weng et al., 2018b]. As shown in [Jordan and Dimakis, 2020], a comparison between different methods is available and can be seen in Figure 4.1 and 4.2. Although an exact computation

	LipMIP*	LipLP*	LipSDP	SeqLip	CLEVER	LiPOpt	FastLip
Local/Global	Local	Local	Global	Global	Local	Local	Local
Guarantee	Exact	Upper	Upper	Heuristic	Heuristic	Upper	Upper
ℓ_p-Norms	ℓ_1,ℓ_∞ only	ℓ_1,ℓ_∞ only	ℓ_2 only	ℓ_p	ℓ_p	ℓ_p	ℓ_p
Activation	ReLU only	ReLU only	ReLU, Diff	ReLU only	ReLU, Diff	Diff only	ReLU only

Figure 4.1 Comparison between different Lipschitz estimation methods.

Method	Time (s)	Relative Error
RandomLB	0.334±0.019	−41.96%
CLEVER	20.574±4.320	−36.97%
LipMIP	69.187±70.114	0.00%
LipLP	0.226±0.023	+39.39%
FastLip	0.002±0.000	+63.41%
LipSDP	20.570±2.753	+113.92%
SeqLip	0.022±0.005	+119.53%
NaiveUB	0.000±0.000	+212.68%

Figure 4.2 Accuracy and computation time comparisons between the different Lipschitz estimation methods.

of the Lipschitz constant would be ideal, as seen in Figure 4.2 LipMIP* takes much longer to complete this calculation. Another constraint for selecting the method for the Lipschitz approximation is that we are defining the Wasserstein distance (2.9) with the 2-norm $\|\cdot\|$. When comparing the methods that can support the 2-norm, the relative errors and times are taken in to consideration. From the available methods the best option is using LipSDP [Fazlyab et al., 2019], which gives us an upper bound for the Lipschitz constant of the deep neural network. The LipSDP package uses the MATLAB engine for python together with CVX and MOSEK[1] to approximate the Lipschitz constant of a dense neural network by using the weights of the network. In order to find the Lipschitz constant for each output respectively, we give the algorithm the weights that contribute to the chosen output. The Lipschitz constant of h_Q can be computed by combining the Lipschitz constant for each output.

[1] A student license was obtained in order to use MOSEK.

Lemma 6
Let $f_i : \mathbb{R}^n \to \mathbb{R}, i = 1 \dots, N$ be Lipschitz continuous with constants K_{f_i}. Then the Lipschitz constant of the function $g(x) = \max\left(\{f_i(x)\}_{i=1}^{N}\right)$ is,

$$K_g = \max\{K_{f_1}, \dots, K_{f_N}\}. \tag{4.34}$$

□

Proof. We will prove for the case $N = 2$ and the result for $N > 2$ follows similarly. For the function $g = \max\{f_1, f_2\}$, $f_1, f_2 : \mathbb{R}^n \to \mathbb{R}$, where f_1 and f_2 are Lipschitz continuous with constants K_{f_1}, K_{f_2}, the Lipschitz constants can be defined by,

$$\|\nabla f_1\| \leq K_{f_1}, \quad \|\nabla f_2\| \leq K_{f_2}, \quad \forall x \in \mathbb{R}^n. \tag{4.35}$$

The gradient of g is,

$$\|\nabla g\| = \begin{cases} \|\nabla f_1\|, & \text{if } f_1(x) > f_2(x) \\ \|\nabla f_2\|, & \text{if } f_2(x) > f_1(x) \end{cases} \leq \max\{\|\nabla f_1\|, \|\nabla f_2\|\}. \tag{4.36}$$

Thus $K_g = \max\{K_{f_1}, K_{f_2}\}$. For a function that is the maximum of N Lipschitz continuous functions, this process can be applied inductively to find,

$$K_g = \max\{K_{f_1}, \dots, K_{f_2}\}. \tag{4.37}$$

□

Theorem 4
Given that $h_Q(s') = \gamma \max\{Q(s', a_1), \dots, Q(s', a_n)\}$, where $n = |\mathcal{A}|$, and the network has the upper bounded Lipschitz constants $K_{a_i}, \forall a_i \in \mathcal{A}$, the Lipschitz constant of $h_Q(s')$ is,

$$K_Q = \gamma \max\left\{K_{a_1}, \dots, K_{a_n}\right\}. \tag{4.38}$$

□

Proof. The proof follows by the direct application of Lemma 6 on $h_Q(s')$.□

Having now found the Lipschitz constants of both $h_r(s')$ and $h_Q(s')$, the following theorem establishes the Lipschitz constant for the objective function $h(s')$.

THEOREM 5
The upper bound of the Lipschitz constant of $h(s')$ defined in (4.17) is given by,

$$K_h \leq \frac{\max\{|\mathbf{r}_{\mathbf{goal}}|, |\mathbf{r}_{\mathbf{obs}}|\}}{2\delta} + \gamma \max\{K_{a_1}, \dots, K_{a_n}\}, \quad n = |\mathcal{A}|. \tag{4.39}$$

□

Proof. Using (4.17), the upper bound for the Lipschitz constant of $h(s')$ can be computed by using Lemma 3. The result is an upper bound due to K_Q being an upper bound to $h_Q(s')$. □

Since the approximation to (4.16) is available, we can train a model similar to the DQN method by generating the targets with the distributionally robust Bellman operator $\hat{\mathcal{T}}$. However when inspecting (4.9), it can be seen that there is a separate case for terminal states. This discontinuity can not be applied to the distributionally robust Q-Learning method since it would violate the Lipschitz continuity condition. Removing the terminal state condition in (4.9) has the possibility to cause the robot to ignore the obstacles and go straight to the goal, since the rewards from the goal spread out due to γ and can move through the obstacles and cancel out the collision penalties. Here we propose a solution to this problem.

Null Policy $\hat{\pi}$

We propose a null action $\hat{a} := \mathbf{0} \in \mathbb{R}^{n_{\mathcal{A}}}$ and define a null action space $\hat{\mathcal{A}} := \mathcal{A} \cup \hat{a}$. The null action is equivalent to taking no action. We define the null policy $\hat{\pi}$ as,

$$\hat{\pi}(s) = \begin{cases} \arg\max_{a \in \mathcal{A}} Q(s, a), & \text{if } s \text{ is non-terminal,} \\ \hat{a}, & \text{if } s \text{ is terminal.} \end{cases} \tag{4.40}$$

The null policy causes the robot to remain in the obstacle or goal after a terminal state is reached. This causes the next state Q-values to also include the penalty/reward. Similarly to DQN, the targets for an experience $\langle s_j, a_j, r_j, s'_j \rangle$ can be computed as,

$$y_j = \mathbb{E}_{s' \sim \hat{\mathbb{P}}_{s'}}[r(s' + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta^-)] - \epsilon_{s'} K_h. \tag{4.41}$$

The pseudocode for the distributionally robust DQN can be seen in Algorithm 2.

Algorithm 2 Lipschitz Approximated Wasserstein Distributionally Robust DQN

```
Require: Disturbance samples $\hat{w}_1, \dots \hat{w}_N$, Learning rate $\eta$, Max episodes $N_{ep}$, Episode Length $N_{step}$, Batch size $N_{batch}$
  Initialize replay memory $\mathcal{M} \leftarrow \emptyset$
  Initialize network weights $\theta, \theta^-$
  Estimate Lipschitz constant of network $\theta^-$
  Compute $\epsilon_s$ by (4.15)
  for episode $= 1 : N_{ep}$ do
      Initialize $s \in \mathcal{S}$
      for $k = 1 : N_{step}$ do
          Select action $a$ with $\epsilon$-greedy policy $\hat{\pi}$
          Observe next state $s'$ and reward $r$
          Append experience $(s, a, r, s')$ to $\mathcal{M}$
          Initialize loss $L \leftarrow 0$
          for $j = 1 : N_{batch}$ do
              Sample experience $(s_j, a_j, r_j, s'_j)$ from $\mathcal{M}$
              Compute nominal distribution $\hat{\mathbb{P}}_{s'}$ given $s_j, a_j$ by (4.11)
              Approximate target $y_j$ by (4.41) with the target network
              Accumulate loss $L \leftarrow L + (y_j - Q(s_j, a_j; \theta))^2$
          Compute loss mean $L \leftarrow \frac{L}{N_{batch}}$
          Update weights $\theta$ by loss $L$ with backpropagation
          Set $\theta^- \leftarrow \theta$ and compute Lipschitz constant of network $\theta^-$ every $\Gamma$ steps
```

5

Numerical Results

In this chapter we present our experiments with the different models that were discussed in previous chapters, and the results that we got. This chapter consists of three sections namely, 5.1 Simulation Parameters, where we define the the simulation parameters that were used in the experiments, 5.2 Results, where we present the obtained results for different methods, 5.3 Implementation Details, where we go over how these simulations and methods were implemented.

5.1 Simulation Parameters

We consider the robot to be moving in an environment $\mathcal{X} \subset \mathbb{R}^2$ with the limits being $[-10, 10]^2$ in both dimensions. There are in total two obstacles and a goal region with equal radius namely, $R_{\mathbf{goal}} = R_{\mathbf{obs}}^{(i)} = 2$, for $i = 1, 2$. The robot moves within $\mathcal{X}$ according to the following dynamics,

$$x_{k+1} = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{A} x_k + \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{B} u_k + w_k. \tag{5.1}$$

The state of the robot x_k represents the position of the robot in $\mathcal{X} \subset \mathbb{R}^2$. The process disturbance $w_k \in \mathbb{R}^2$ shifts the position of the robot by a random amount in each axis. For simulation purposes we considered 10^4 samples w that were sampled from distributions with zero mean and covariance being equal to $0.15 I_2$. The action space $\mathcal{A}$ consists of $|\mathcal{A}| = 8$ actions where each action is a $\mathbb{R}^2$ vector with unit norm, that represents a step that can be taken in one of the 8 equally spaced radial directions. The robot takes a step in the specified direction for each action and the process disturbance shifts the position of the robot by a random amount. The reward function has the constants $\mathbf{r_{travel}} = -0.001$, $\mathbf{r_{goal}} = 1$ and $\mathbf{r_{obs}} = -1$. The steepness of the tanh functions is chosen as $\delta = 0.1$.

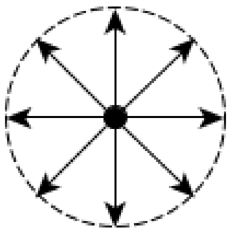


Figure 5.1 The available actions are equally spaced vectors on the unit circle where each correspond to taking a step in that specific direction.

5.2 Results

We have trained two versions of our model, one with the Wasserstein radius $\epsilon_{s'} = 0$ and another with $\epsilon_{s'} = 0.067$ which was obtained by using $N = 10^4$ samples and setting the risk factor $\beta = 0.1$.

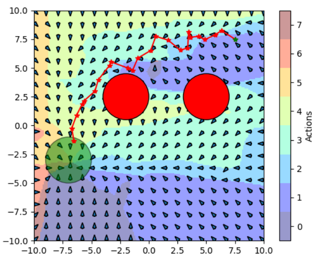
Table 5.1 The mean and the standard deviation of the total rewards with different noise covariances corresponding to different training models are tabulated here.

		Noise covariance					
		$\Sigma_w = 0_2$		$\Sigma_w = 0.15 I_2$		$\Sigma_w = 0.3 I_2$	
		Travel Reward		Travel Reward		Travel Reward	
Models	$\epsilon_{s'}$	Mean	Std	Mean	Std	Mean	Std
DQN	N/A	0.636	0.545	0.662	0.514	0.627	0.573
DRDQN	0	0.850	0.334	0.811	0.401	0.749	0.510
DRDQN	0.067	0.829	0.356	0.811	0.387	0.756	0.489

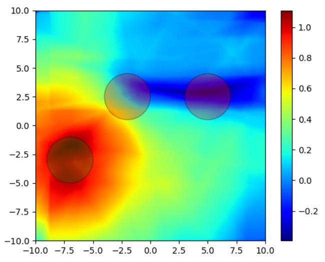
Table 5.2 The percentage of trajectories that reached the goal, resulted in collision and those that did neither are tabulated here for different noise covariances corresponding to different training models.

		Reached Goal			Resulted in Collision			Wandering Around		
		$\Sigma_w(\times I_2)$			$\Sigma_w(\times I_2)$			$\Sigma_w(\times I_2)$		
Models	$\epsilon_{s'}$	0	0.15	0.3	0	0.15	0.3	0	0.15	0.3
DQN	N/A	76.7%	82.2%	81.4%	1.5%	3.9%	6.9%	21.8%	13.9%	11.7%
DRDQN	0	97.9%	96.4%	93.1%	0.7%	3.1%	6.5%	1.4%	0.5%	0.4%
DRDQN	0.067	95.3%	95.7%	93%	0.5%	2.4%	5.5%	4.2%	1.9%	1.5%

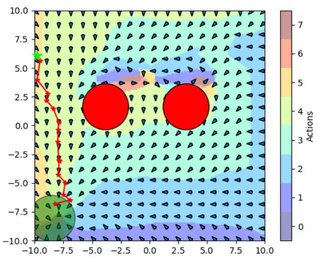
The hyperparameter details of the training results are made available Table 5.3. The resulting policy and the state values for the trained models can be seen in Figure 5.2. The arrows represent the action that the policy gives at the respective robot position and goal/obstacle positions. The heatmap represents the same values with color but in higher resolution to better understand the decision boundaries. The figures on the right side of Figure 5.2 represent the value of each state s which can be computed by $\max_{a \in \mathcal{A}} Q(s, a)$. It can be seen that the rewards propagate from the goal and the obstacles. Further, the resulting learned policy restricts the robot moving between the obstacles and there exists a boundary around the obstacles. When compared with the DQN model, our solution is more risk averse since it does not prefer to get close to the obstacles while the DQN policy is allowing some risk. This difference is due to the fact that DQN learns the expected rewards, while the DRDQN learns the worst case expected rewards making it to be a more risk averse solution. Due to the noise samples used in DRDQN model with $\epsilon_{s'}$ calculated using (4.15), learning the policy occurs in less steps compared to the DQN model. However DRDQN can take more time since the computational load is higher for calculating the targets. The DRDQN with $\epsilon_{s'} = 0$ is virtually the same as DQN as it learns faster since it calculates the expected values in (4.1) more accurately compared to that of DQN which uses only one sample. The models have been evaluated by running 10^5 episodes each with random goal/obstacle configurations, for three different noise distributions. As seen in Table 5.1 both versions of DRDQN have a higher average total reward compared to DQN with lower variances. Also in Table 5.2, the percentage of trajectories that have reached the goal, collided with an obstacle or border or have not reached the goal or collided, has been provided. It can be inferred that as the covariance of the noise increases, the DRDQN model is able to maintain a low collision rate due to the worst case approximations. In some scenarios, the policy can have loops where the robot can get stuck. Since the training is done with process noise, the robot has a higher chance to break free and this can delay the policy learning to fix this issue. However this can be rectified by training the models for a larger number of steps.



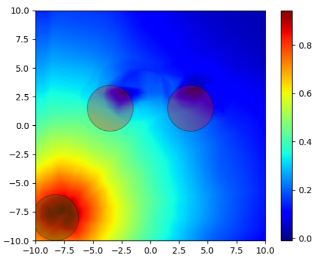
(a) DQN policy



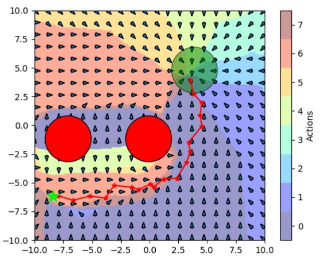
(b) DQN values



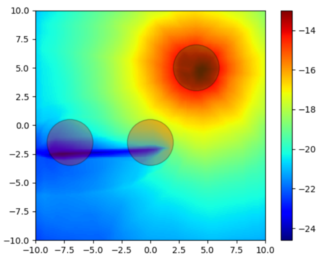
(c) DRDQN ($\epsilon_s' = 0$) Policy



(d) DRDQN ($\epsilon_{s'} = 0$) Values

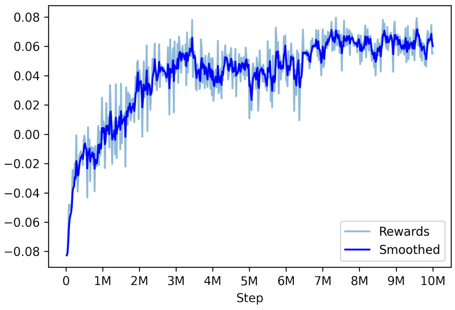


(e) DRDQN policy with $\epsilon_{s'} = 0.067$.

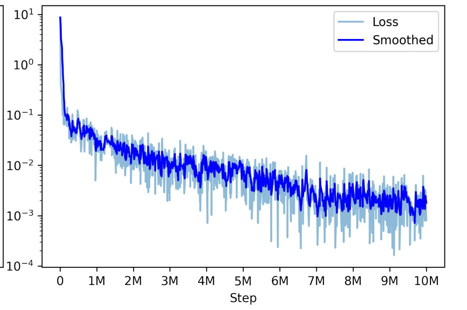


(f) DRDQN values with $\epsilon_{s'} = 0.067$.

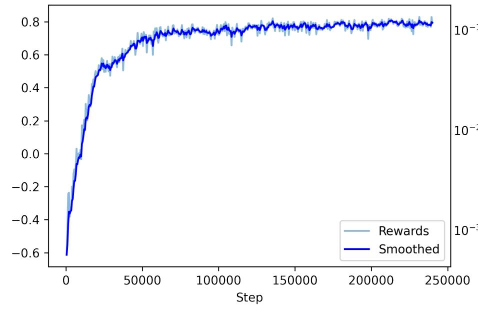
Figure 5.2 The result of training DQN model and DRDQN model with $\epsilon_{s'} = 0.067$ is shown here. Also, the solution trajectories from a starting position in green star to the goal region in green color avoiding both the red color obstacles are shown in both cases. The plot on the left shows the learned control policies and the one on right depicts the learned Q values. Clearly, the DRDQN model exhibits more risk averse behaviour given that it minimizes the worst case loss function.



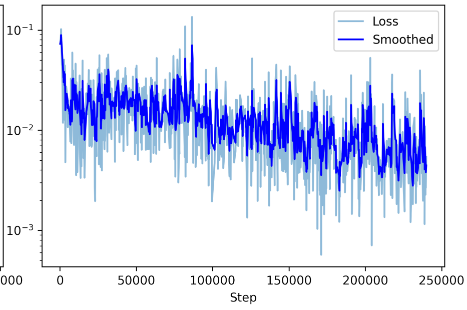
(a)



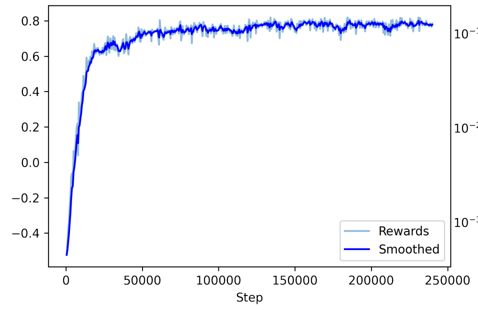
(b)



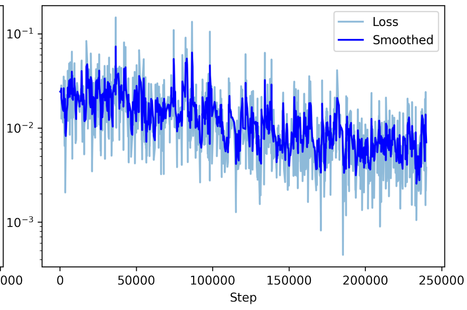
(c)



(d)

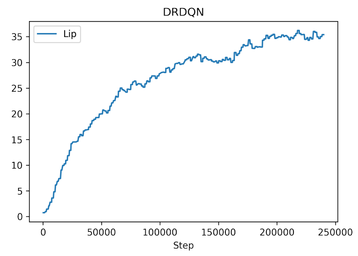


(e)

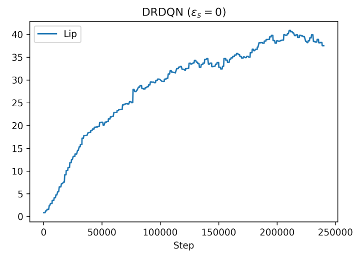


(f)

Figure 5.3 Total rewards during evaluations and losses for the trained models are shown on the left and right plots respectively. The first, second and the third rows correspond to the DQN, DRDQN (with $\epsilon_{s'} = 0$), and DRDQN (with $\epsilon_{s'} = 0.067$) respectively.



(a) Lipschitz constant for DRDQN ($\epsilon_{s'} = 0.067$).



(b) Lipschitz constant for DRDQN ($\epsilon_{s'} = 0$).

Figure 5.4 The Lipschitz constant of the target network $Q(s, a; \theta^-)$ during training for both DRDQN models.

5.3 Implementation Details

The episodes are limited to 50 steps in order to prevent infinite loops during training. During an episode the positions of the goal and obstacles remain static. At the start of each episode, the positions of the obstacles are randomly sampled from the environment $\mathcal{X}$ while considering Assumption 2. After the obstacles, the goal is sampled using the free space such that no overlap occurs with the obstacles or the borders of the environment. Finally the initial position of the robot is selected randomly from the available free space such that the distance between the goal and initial position is at most λ. The reason for this is that since in early stages of training, it is difficult for the robot to end up in the goal by making taking random actions due to the exploration phase. By setting the initial position close to the goal, the agent has a higher chance of experiencing a goal state. As the training continues, we increase λ such that the goal can be sampled further away. By starting with an easy problem we allow the agent to learn faster and progressively make the problem more difficult so in the end the robot has a higher chance to learn a good and safe policy. This method is also utilized in [Raajan et al., 2020]. We use a dense neural network with $n_{\mathcal{S}} = 8$ inputs that correspond to the current states to approximate the Q-values. The network has two hidden layers with 150 neurons each that have ReLU activation functions. The network has $8 + 1$ outputs where each output corresponds to the Q-value for the state-action pair. We use prioritized experience replay for both DQN and our method. The hyperparameters α and β' used in prioritized experience replay are the recommended parameters given in [Schaul et al., 2015], where $\alpha = 0.7$ and $\beta' = 0.5 \rightarrow 1$ is linearly increased during training. The dueling layer is only used for DQN. During training, collisions do not end the simulation in order to allow the robot to explore further and gain experiences that reach the goal. This does not change anything for the experience replay part since terminal states are handled separately. For all the models the probability of taking a random action ϵ is reduced from 1 to 0.1 linearly for the first 3/4 of training and kept at 0.1 for the remaining episodes. All the simulations are implemented in Python and use open source libraries such as NumPy, Pandas and Matplotlib. The neural networks and the training process was implemented using PyTorch and the PyTorch Lightning API. The Lipschitz estimation in LipSDP uses the Matlab engine for Python together with CVX and MOSEK, which require a license to use. For this project, a student license was obtained. All the code and resources used for this project together with instructions to use it can be found in `https://github.com/CemAlpturk/Distributionally_Robust_RL`. The simulations were performed on a Dell R530 with 2 Xeon E5-2620 6-core 12-thread CPU's and 132GB of RAM. The simulation time can be improved by utilizing a GPU during training.

Table 5.3 Hyperparameters used in the simulation results are tabulated here.

Hyperparameters	DQN	DRDQN ($\epsilon_{s'} = 0$)	DRDQN ($\epsilon_{s'} = 0.067$)
γ	0.9	0.9	0.9
η	10^{-4}	10^{-4}	10^{-4}
Steps per episode	50	50	50
Total steps	10^7	2.4×10^5	2.4×10^5
N_{batch}	32	32	32
β	N/A	N/A	0.1
N (samples)	N/A	10^4	10^4
Γ	5000	1500	1500
$\lvert\mathcal{M}\rvert$	5000	5000	5000
ϵ	$1 \to 0.1$	$1 \to 0.1$	$1 \to 0.1$

Dueling Layer and Prioritized Experience Replay

To increase the performance of the algorithm some additions can be made such as Dueling architecture [Wang et al., 2015], and prioritized experience replay [Schaul et al., 2015]. The architecture of the networks consists of hidden layers followed by a dueling layer which consists of the value and advantage streams. These streams are combined to compute the Q-values. The logic of the dueling layer is to estimate the value of a state and the advantage of the state action pair.

$$Q(s,a) = V(s) + A(s,a) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A(s,a') \tag{5.2}$$

The reason for using this layer is that, although the Q-values represent the expected returns, knowing the exact values has no significance for the policy, since the action that maximizes the Q-values is used. The advantage of each actions will represent the returns with respect to the other actions, which will be easier to estimate. These values can be combined with the value stream to compute the Q-values. As mentioned in [Raajan et al., 2020], the dueling layer improves the predictions if the Q-values and prevents different actions having similar Q-values, which can cause the agent to get stuck in loops. Figure 5.5 shows an illustration of a simple duelling architecture.

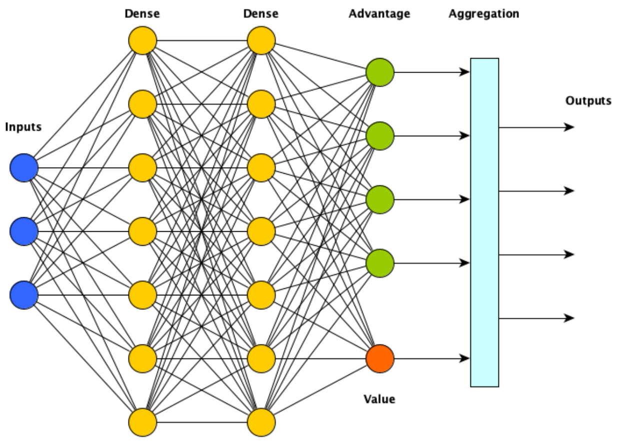


Figure 5.5 Duelling architecture illustration.

In reinforcement learning problems where the goal states are rare, the agent may not be able to find these states and may end up with poor performance. Prioritized experience replay tries to tackle this problem by assigning a priority to each experience in the memory buffer. For each experience $< s_j, a_j, r_j, s'_j >$, a priority p_j is assigned as such,

$$p_j = \begin{cases} 1 & \text{if } \mathcal{M} = \emptyset \\ \max_{i<j} p_i & \text{else} \end{cases}$$

These priorities are updated based on the td-error that occurs during training. This results in experiences with large errors to be prioritized more thus having a higher chance to be sampled during training. This way rare events can be experienced more by the agent.

The probability of sampling experience j, $P(j)$ is found as,

$$P(j) = \frac{p_j^\alpha}{\sum_k p_k^\alpha}$$

where p_j is the priority of the experience j and the exponent $\alpha \in \mathbb{R}$ determines how much prioritization is used. For the case $\alpha = 0$, sampling becomes uniform. The priorities for each experience are kept in a separate data structure called a SumTree, which is a binary tree whose leaf nodes hold the priorities for each experience. Each node in this data structure holds the sum of the values of its children. The root node will hold the sum of all the leaf nodes. An illustration of a SumTree data structure is shown in Figure 5.6. The SumTree can be used to perform weighted sampling from a set. The SumTree allows to sample from a set with N elements, with $O(\log(N))$ instead of $O(N)$ complexity based on their priorities. When the memory buffer $\mathcal{M}$ becomes large, repeated weighted sampling can cause serious overhead.

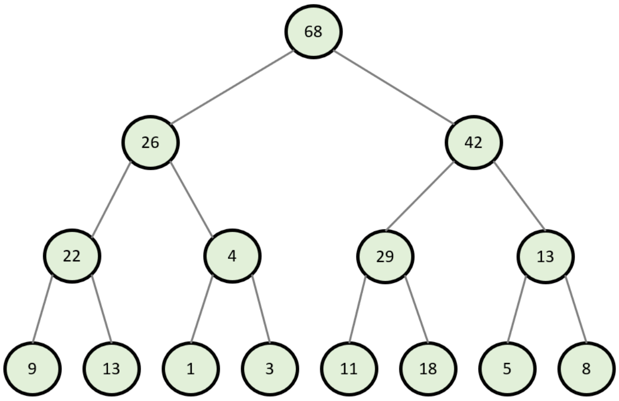


Figure 5.6 Example of a SumTree data structure. The leaf nodes hold the priority of each experience.[1]

Since the sampling is not uniform, a bias is introduced. The bias can be taken care of by weighing the samples accordingly with importance sampling. The weight for the transition j is computed as,

$$w_j = \left(\frac{1}{N} \cdot \frac{1}{P(j)} \right)^{\beta'},$$

where N is the number of elements in $\mathcal{M}$ and $\beta' \in \mathbb{R}$. The weights are normalized by $1/\max_i w_i$. The pseudocode for the prioritized experience replay can be seen in Algorithm 3.

[1] https://www.fcodelabs.com/2019/03/18/Sum-Tree-Introduction/

Algorithm 3 DQN with Prioritized Experience Replay

```
Initialize replay memory $\mathcal{M}$ to capacity $N_{mem}$ with minibatch size $N_{batch}$
Learning rate $\eta$
Initialize action-value function Q with random weights $\theta$
Set $\theta^- \leftarrow \theta$
**for** episode$= 1 : N_{ep}$ **do**
    Initialize state $s \in \mathcal{S}$
    **for** $step = 1 : N_{step}$ **do**
        With probability $\epsilon$ select a random action $a$, otherwise select $a = \arg\max_{a'} Q(s, a'; \theta)$
        Execute action $a$ and observe reward $r$ and state $s'$
        Store transition $\langle s, a, r, s' \rangle$ in $\mathcal{M}$ with maximal priority $p = \max_i p_i$
        Initialize weight change $\Delta \leftarrow 0$
        **for** $j = 1 : N_{batch}$ **do**
            Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
            Compute weights $w_j = (N_{mem}.P(j))^{-\beta'} / \max_i w_i$
            Set $y_j = \begin{cases} r_j, & \text{for terminal } s'_j, \\ r_j + \gamma \max_{a'} Q(s'_j, a'; \theta^-), & \text{for non-terminal } s'_j. \end{cases}$
            Compute td-error $\zeta_j = y_j - Q(s_j, a_j; \theta)$
            Update transition priority $p_j \leftarrow |\zeta_j|$
            Accumulate weight change $\Delta \leftarrow \Delta + w_j \zeta_j \nabla_\theta Q(s_j, a_j; \theta)$
        Update weights $\theta \leftarrow \theta + \eta\Delta$
        Set $\theta^- \leftarrow \theta$ every $\Gamma$ steps
```

6

Discussion

In this chapter we discuss the results that we have obtained through our simulations. This chapter consists of seven sections namely, 6.1 Reward Function Selection, where we discuss about the process of selecting and tuning the reward function, 6.2 Linear Models, where present a potential solution to the DRO using linear regression, 6.3 Lipschitz Approximation, where we discuss the stability and accuracy of our solution to the DRO, 6.4 Terminal States, where we go over our reasoning behind the null policy, 6.5 Limitations, where we explain the limitations of our solution, finalized with the 6.6 Conclusion and 6.7 Future Research sections.

6.1 Reward Function Selection

In reinforcement learning, the most crucial part is engineering a good reward function. Unlike in supervised learning, where we have access to the 'correct' labels, we have to generate them ourselves. The reward function should encourage the model to perform a certain task. In cases where the reward function does not represent the goal well enough, the model may come up with a solution or policy that is totally different than expected. One of the biggest problems we experienced was to tune the reward function such that it found a safe path to the goal region. In some of our early experiments, where the travel penalty was relatively higher, the model ended up deciding that it was better to end the simulation as fast as possible whether by going to the goal or colliding with the nearest obstacle, rather than trying to find a safe path to the goal. It took a lot of trial and error until a good reward function was found.

In the beginning of this project we approximated the reward function with Gaussians rather than tanh functions, and designed it such that the the region near the obstacle/goal had non-zero reward in order to encourage the robot to move towards the goal or steer clear of the regions near the obstacles. This caused the regions near the goal to have positive rewards, and resulted in the robot moving towards the goal but it never actually went inside the goal region. Since stepping inside the goal would end the simulation, the agent would receive a one time reward. However moving close to the obstacle without entering meant that the simulation would continue and the cumulative reward it received would be higher. The agent exploited this mechanic and resulted in a poor policy. For this reason we decided to use radial tanh functions to approximate the reward function.

6.2 Linear Models

Finding a tractable solution for the DRO in (4.16) is complex for non-convex objective functions such as h. However for convex functions tractable formulations exist [Chen and Paschalidis, 2020]. We have tried to approximate the Q-function with linear regression using polynomial features in order to take advantage of these solutions, but the model was not powerful enough to find an admissible path to the goal.

6.3 Lipschitz Approximation

One of the main problems we encountered in the Lipschitz approximation was the stability of our method. For a reward function with a large Lipschitz constant, the computed targets became large in magnitude, which caused the weights of the networks to become large as well. This caused the Lipschitz constant of the networks to increase, which in turn caused the targets to increase again. This created a positive feedback loop where in the end the Lipschitz constant of the networks would blow up and LipSDP would fail to give a result. We overcame this issue by using smaller rewards and initializing the network weights to small values in order to keep the Lipschitz constant of the objective function small. With our final results it can be seen that the Lipschitz constant of the network converges.

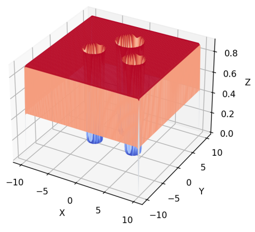
It is also important to note that the Lipschitz approximation in Lemma 2 is a lower bound to the DRO. With an upper bound, the results could be interpreted much better since we could have an idea of how good the approximation is.

6.4 Terminal States

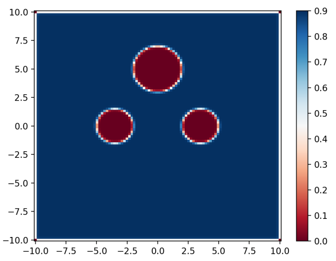
In order to remove the discontinuation in (4.9) in the DRDQN method, we have initially tried to ignore the terminal state case, where each state was treated as a non-terminal state during training. This resulted in the rewards from the goal spreading out and going through the obstacles, which caused the robot to ignore the obstacles and try to go towards the goal in a straight line.

Another solution we tried was to find a continuous approximation of (4.9) for the DRDQN method, by changing γ with respect to s', such that near terminal states we would get $\gamma = 0$ and $\gamma = 0.9$ in non-terminal states. This was done by using the same approximation that we performed for the reward function (3.12) where $\mathbf{r_{travel}} = 0.9$ and $\mathbf{r_{goal}} = \mathbf{r_{obs}} = -0.9$. This way we found a continuous transition for the discontinuity in the states. However this method did not prove to work and resulted in the robot trying to end the simulation as fast as possible by either going to the goal or going to the nearest obstacle. The resulting policy can be seen in Figure 6.2. The reason for this behavior is due to the worst case expected cumulative rewards is much lower for non-terminal states, and the robot prefers to end the simulation since the goal reward or the collision penalty is very close when compared to the Q-values of non-terminal states. We believe that this method could be improved by tuning the reward function. Another problem with this method was that it was unstable, where the Lipschitz constant of the network kept growing exponentially and did not converge, which eventually caused numerical errors with LipSDP.

For these reasons we have implemented the null policy, so that when the robot enters a terminal state, it can not get out of it, which causes the cumulative penalty to increase, resulting in a risk-averse behavior.



(a)



(b)

Figure 6.1 Continuous approximation for γ, where at terminal states $\gamma = 0$ and $\gamma = 0.9$ at non-terminal states.

6.5 Limitations

The Wasserstein distributionally robust deep Q-Learning problem posed in section 4.2 is an infinite dimensional optimization problem and tractable so-

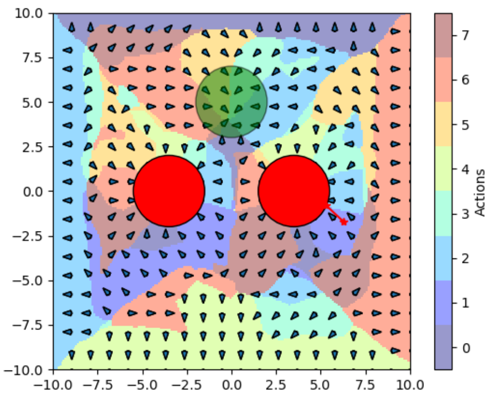


Figure 6.2 Resulting policy from the dynamic γ implementation. The robot tries to move towards the closest terminal state regardless of it being in an obstacle or goal.

lutions using duality theory can be found only when the objective function is convex. However, in our case, the objective function being the neural network based approximation of the Q-function, turns out to be non-convex. This limits our options and leads us to look for tractable approximations like the Lipschitz or the duality based solutions under some special cases.

The state definition of the MDP can be considered to be not realistic since the agent knowns the position of the robot and the goal/obstacles with absolute precision. Initially the state was defined similar to [Raajan et al., 2020] where the robot had distance sensors pointing in specific directions that would give the distance of the closest object in that direction, which is a more realistic approach to a path planning problem. However, since this distance sensor is a discontinuous function of the position of the robot, this caused problems in calculating the Lipschitz of the objective function h. Due to this reason we decided not to use this state definition.

6.6 Conclusion

We proposed a risk averse path planning using approximated Wasserstein distributionally robust deep Q-Learning approach. Through a carefully designed reward function, we showed how to learn a safe control policy for a robot with uncertain dynamics in an environment. Our numerical simulation results demonstrated our proposed approach. Our model is able to find a safe path in real time for a configuration of the environment, given that the model has been trained beforehand.

6.7 Future Research

Although we have chosen relatively simple robot dynamics, our proposed method can be applied to more complicated dynamics, linear or even non-linear. The project could be taken further by implementing it on a non-linear system such as a robot that has steering dynamics.

The states of the system can be improved to consider dynamic obstacles as well as obstacles that have different shapes than circles such as convex polygons as long as the reward function is designed appropriately. Another improvement to the states can be to model sensors on the robot in order to have a more realistic representation of the environment, rather than giving the model too much information.

For this project we have chosen a fixed δ for the obstacles and goal in the reward function. This could be improved by utilizing δ such that it is based on the covariance of the disturbance samples. This could determine how wide the region of influence is for each object in the environment.

Bibliography

Adler, J. and S. Lunz (2018). “Banach wasserstein gan”. In: Bengio, S. et al. (Eds.). *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc.

Aoude, G. S., B. D. Luders, J. M. Joseph, N. Roy, and J. P. How (2013). “Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns”. *Autonomous Robots* **35**:1, pp. 51–76.

Chen, R. and I. Paschalidis (2020). *Distributionally Robust Learning*. Vol. 4. ISBN: 978-1-68083-773-5. DOI: 10.1561/2400000026.

Fazlyab, M., A. Robey, H. Hassani, M. Morari, and G. J. Pappas (2019). “Efficient and accurate estimation of lipschitz constants for deep neural networks”. URL: http://arxiv.org/abs/1906.04893.

Gao, R. and A. J. Kleywegt (2016). “Distributionally robust stochastic optimization with wasserstein distance”. URL: http://arxiv.org/abs/1604.02199.

Jordan, M. and A. G. Dimakis (2020). “Exactly computing the local lipschitz constant of relu networks”. URL: http://arxiv.org/abs/2003.01219.

Kandel, A. and S. J. Moura (2020). “Safe Wasserstein constrained deep q-learning”. *arXiv preprint arXiv:2002.03016.*

Kuhn, D., P. M. Esfahani, V. A. Nguyen, and S. Shafieezadeh-Abadeh (2019). “Wasserstein distributionally robust optimization: theory and applications in machine learning”. DOI: 10.48550/ARXIV.1908.08729. URL: https://arxiv.org/abs/1908.08729.

Lathrop, P., B. Boardman, and S. Martınez (2021). “Distributionally safe path planning: wasserstein safe rrt”. *IEEE Robotics and Automation Letters* **7**:1, pp. 430–437.

Luders, B., M. Kothari, and J. How (2010). “Chance constrained rrt for probabilistic robustness to environmental uncertainty”. In: *AIAA guidance, navigation, and control conference*, p. 8160.

Majumdar, A. and M. Pavone (2020). “How should a robot assess risk? towards an axiomatic theory of risk in robotics”. In: *Robotics Research*. Springer, pp. 75–84.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller (2013). “Playing atari with deep reinforcement learning”. *CoRR* **abs/1312.5602**. arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

Raajan, J., P. V. Srihari, J. P. Satya, B. Bhikkaji, and R. Pasumarthy (2020). “Real time path planning of robot using deep reinforcement learning”. In: vol. 53. Elsevier B.V., pp. 15602–15607. DOI: 10.1016/j.ifacol.2020.12.2494.

Renganathan, V., S. Safaoui, A. Kothari, B. Gravell, I. Shames, and T. Summers (2022). “Risk bounded nonlinear robot motion planning with integrated perception & control”. *arXiv preprint arXiv:2201.01483*.

Renganathan, V., I. Shames, and T. H. Summers (2020). “Towards integrated perception and motion planning with distributionally robust risk constraints”. *IFAC-PapersOnLine* **53**:2, pp. 15530–15536.

Safaoui, S., B. J. Gravell, V. Renganathan, and T. H. Summers (2021). “Risk-averse rrt planning with nonlinear steering and tracking controllers for nonlinear robotic systems under uncertainty”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3681–3688. DOI: 10.1109/IROS51168.2021.9636834.

Schaul, T., J. Quan, I. Antonoglou, and D. Silver (2015). “Prioritized experience replay”. DOI: 10.48550/ARXIV.1511.05952. URL: https://arxiv.org/abs/1511.05952.

Shang, C. and F. You (2018). “Distributionally robust optimization for planning and scheduling under uncertainty”. *Computers Chemical Engineering* **110**, pp. 53–68. ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2017.12.002. URL: https://www.sciencedirect.com/science/article/pii/S009813541730426X.

Subramani, D. N. and P. F. Lermusiaux (2019). “Risk-optimal path planning in stochastic dynamic environments”. *Computer Methods in Applied Mechanics and Engineering* **353**, pp. 391–415.

Wang, Z., N. de Freitas, and M. Lanctot (2015). “Dueling network architectures for deep reinforcement learning”. *CoRR* **abs/1511.06581**. arXiv: 1511.06581. URL: http://arxiv.org/abs/1511.06581.

Watkins, C. J. C. H. and P. Dayan (1992). “Q-learning”. *Machine Learning* **8**:3, pp. 279–292. DOI: 10.1007/BF00992698.

Weng, T.-W., H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, D. Boning, I. S. Dhillon, and L. Daniel (2018a). *Towards fast computation of certified robustness for relu networks*. arXiv: 1804.09699 [stat.ML].

Weng, T.-W., H. Zhang, P.-Y. Chen, J. Yi, D. Su, Y. Gao, C.-J. Hsieh, and L. Daniel (2018b). *Evaluating the robustness of neural networks: an extreme value theory approach.* DOI: 10.48550/ARXIV.1801.10578. URL: https://arxiv.org/abs/1801.10578.

Xiao, X., J. Dufek, and R. Murphy (2019). "Explicit-risk-aware path planning with reward maximization". *arXiv preprint arXiv:1903.03187.*

Xie, J., Z. Shao, Y. Li, Y. Guan, and J. Tan (2019). "Deep reinforcement learning with optimized reward functions for robotic trajectory planning". *IEEE Access* **7**, pp. 105669–105679.

Yan, C., X. Xiang, and C. Wang (2020). "Towards real-time path planning through deep reinforcement learning for a uav in dynamic environments". *Journal of Intelligent & Robotic Systems* **98**:2, pp. 297–309.

Zhao, C. and Y. Guan (2015). "Data-driven risk-averse two-stage stochastic program with ζ-structure probability metrics". *Available on Optimization Online* **2**:5, pp. 1–40.

Lund University **Department of Automatic Control** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER'S THESIS *Date of issue* June 2022 *Document Number* TFRT-6166
Author(s) Cem Alptürk	*Supervisor* Venkatraman Renganathan, Dept. of Automatic Control, Lund University, Sweden Anders Rantzer, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle

Risk Averse Path Planning Using Lipschitz Approximated Wasserstein Distributionally Robust Deep Q-Learning

Abstract

We investigate the problem of risk averse robot path planning using the deep reinforcement learning and distributionally robust optimization perspectives. Our problem formulation involves modelling the robot as a stochastic linear dynamical system, assuming that a collection of process noise samples is available. We cast the risk averse motion planning problem as a Markov decision process and propose a continuous reward function design that explicitly takes into account the risk of collision with obstacles while encouraging the robot's motion towards the goal. We learn the risk-averse robot control actions through Lipschitz approximated Wasserstein distributionally robust deep Q-Learning to hedge against the noise uncertainty. The learned control actions result in a safe and risk averse trajectory from the source to the goal, avoiding all the obstacles. Various supporting numerical simulations are presented to demonstrate our proposed approach.

Keywords		
Classification system and/or index terms (if any)		
Supplementary bibliographical information		
ISSN and key title 0280-5316		*ISBN*
Language English	*Number of pages* 1-57	*Recipient's notes*
Security classification		

http://www.control.lth.se/publications/