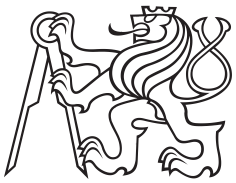
Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Risk-Aware Autonomous Driving via Reinforcement Learning

Bc. Olga Petrova

Supervisor: Mgr. et Mgr. Karel Macek, Ph.D.
Field of study: Cybernetics and Robotics
Subfield: Robotics
January 2019

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Petrova Olga** Personal ID number: **420043**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

Branch of study: **Robotics**

II. Master's thesis details

Master's thesis title in English:

Risk-Aware Autonomous Driving via Reinforcement Learning

Master's thesis title in Czech:

Autonomní řízení a modelování rizik pomocí posilovaného učení

Guidelines:

1. In present literature find at least 2 risk metrics in the field of autonomous driving via reinforcement learning. Discuss motivation behind them and their properties.
2. Propose and implement 1-2 environments for simulation of risky situations in autonomous driving, alternatively use existing environments (eg. MIT or Udacity simulators).
3. Propose and implement 2-3 approaches on the basis of present literature and compare results in simulated environments.

Bibliography / sources:

[1] Javier Garcia, Fernando Fernandez: A Comprehensive Survey on Safe Reinforcement Learning, Universidad Carlos III de Madrid, 2015.
[2] Richard S. Sutton and Andrew G. Barto: Reinforcement Learning: An Introduction, The MIT Press, 2012.
[3] Xin Li, Xin Xu, Lei Zuo: Reinforcement learning based overtaking decision-making for highway autonomous driving, Sixth international conference on intelligent control and information processing, China, 2015.
[4] Buoniu L, Babuka R, de Schutter B, Ernst D: Reinforcement Learning and Dynamic Programming Using Function Approximators, CRC Press, USA, 2010.
[5] Justin Fu, Katie Luo, Sergey Levine: Learning robust rewards with adversarial inverse reinforcement learning, USA, 2017.

Name and workplace of master's thesis supervisor:

Mgr. et Mgr. Karel Macek, Ph.D., DHL Information Services (Europe) s.r.o., Prague

Name and workplace of second master's thesis supervisor or consultant:

prof. Ing. Václav Hlaváč, CSc., Robotic Perception, CIIRC

Date of master's thesis assignment: **10.01.2018** Deadline for master's thesis submission: **08.01.2019**

Assignment valid until: **30.09.2019**

Mgr. et Mgr. Karel Macek, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I want to thank my supervisor for the provided advice and support.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university thesis.

Prague, date ....
signature

Abstract

This thesis explores methods of risk reduction in reinforcement learning applied to an autonomous driving environment. Two risk-reducing approaches are studied closer: policy initialization from expert demonstrations and risk-aware reinforcement learning. For the purpose of policy initialization, two algorithms are compared: (1) Behavioral Cloning and (2) Generative Adversarial Imitation Learning. Both algorithms show their ability to learn a sensible policy from expert's demonstrations, which perform better than baseline random policy. Within the scope of risk-aware reinforcement learning, two algorithms are described and implemented: (1) Q-learning with risk-directed exploration and (2) variance constrained Policy Gradient algorithm. Both algorithms were compared with original risk-neutral versions. The results of the experiments show that Q-learning with risk-directed exploration reduces collision rate and increases return value in comparison with the original version of the algorithm. Policy Gradient algorithm with a variance constraint reduces the variance of the return during the first part of learning and converges to the same values of collision rate and return as ones of the original version.

Keywords: Reinforcement learning, Autonomous driving, Risk, Safety

Supervisor: Mgr. et Mgr. Karel Macek, Ph.D.
DHL IT Services
V Parku 10
Praha 4
14800

Abstrakt

Tato diplomová práce je zaměřena na metody snižování rizika při aplikaci posilovaného učení na úlohu autonomního řízení. K řešení problému se používají dva přístupy: inicializace strategie z expertních příkladů a posilované učení s vnímáním rizika. V rámci studia inicializace strategie věnujeme větší pozornost dvěma metodám: (1) Behavioral Cloning a (2) Generative Adversarial Imitation Learning. Obě metody vykazují schopnost naučit se z expertních příkladů strategii, která je lepší než výchozí náhodná strategie. V rámci posilovaného učení s vnímáním rizika popisujeme dva algoritmy: (1) Q-learning s rizikem řízeným prohledáváním stavového prostoru a (2) Policy Gradient s omezující podmínkou pro rozptyl. Oba algoritmy porovnáváme s originálními verzemi, které jsou vůči riziku neutralní. Výsledky experimentů ukazují, že popsaná varianta algoritmu Q-learning snižuje četnost nehod a zvyšuje hodnotu akumulované odměny v porovnání s originální verzí algoritmu. Zvolená varianta metody Policy Gradient na začátku učení snižuje rozptyl akumulované odměny oproti originální verzi a poté konverguje ke stejné četnosti nehod a hodnotě akumulované odměny.

Klíčová slova: Posilované učení, Autonomní řízení, Riziko, Bezpečnost

Překlad názvu: Autonomní řízení a modelování rizik pomocí posilovaného učení

Contents

Figures

Tables

Chapter 1

Introduction

The task of autonomous driving incorporates many parts each of which demands different approach in the sense of AI algorithms. This work addresses the decision-making process related to driving.

In recent years was made significant progress in understanding and solving tasks of autonomous driving. An interest of the industry creates competition and pushes the field toward better solutions. Usually, it is difficult to design an accurate model of a vehicle and a comprehensive probabilistic model of the environment which could be used in machine learning. The other problem is that the parameters of a vehicle change through time and are heavily dependent on the environment. Reinforcement learning offers a general approach to solving decision problems with or without knowledge of a model.

The real-world environment, in which autonomous car is acting, is naturally stochastic. It means that decision making is always performed under uncertainty. Uncertainty about the outcome of an action serves as a source of risk. The cost of a mistake in autonomous driving may be enormous. That is why researchers are paying increasing attention towards safety and damage avoidance.

Safe reinforcement learning aims to integrate a notion of risk into the process of decision making. It brings possibilities for an agent to make the best decision considering safety parameters.

The purpose of this thesis is to explore and summarize the advantages and disadvantages of different approaches to safe reinforcement learning in autonomous driving and apply chosen algorithms to a particular autonomous driving problem.

The thesis is structured as follows: Chapter 2 describes the problem statement and provides details of current approaches to safe reinforcement learning algorithms used for autonomous driving. Chapter 3 defines the general framework of reinforcement learning and related algorithms. Chapter 4 introduces safe reinforcement learning, the notion of risk and the ways how to incorporate safety into learning. It also provides implementation details of the algorithms

and intuition behind them. Chapter 5 describes experiments and describes the results. Experiment results are discussed in Chapter 6 together with an outlook on further steps for the solution of the stated problem.

Chapter 2

Problem description

2.1 Problem statement

Autonomous driving can be viewed as a task of Artificial Intelligence to drive in a natural driving environment without human input. Autonomous vehicle perceives the environment via various inputs such as LIDAR, GPS, acceleration sensors, and cameras. Control algorithm processes sensor inputs and acts correspondingly in real time. Processed input data may be combined using Sensor Fusion technique [14], which leads to more consistent and filtered inputs. Apart from sensor inputs, an autonomous driving vehicle takes as an input a task from human, e.g., to drive to a specified destination or to park in a parking lot. Input specification, traffic laws, and sensor inputs provide a driving scope for an autonomous vehicle. In this scope, a vehicle still has infinite possibilities of achieving a specified goal. Needless to say that an algorithm must aim for the most optimal way of goal achieving in terms of time, fuel, and safety along with others.

Nowadays, autonomous driving algorithms are developed and used for a wide range of tasks from parking in a parking lot [31] to fully autonomous driving on public roads. Formally, the task can be structured as follows: an input is represented by sensor data and a human-specified goal. An output is a control algorithm making decisions at discrete time steps in real time. Figure 2.1 depicts a stack of tasks which such an algorithm must solve [9]. Modern approaches to autonomous driving include various algorithms for input processing for each step of a control algorithm. The first part includes data acquisition via various sensors. It utilizes standard algorithms for data transfer, storing, and filtering.

The second part of the stack addresses data processing and feature extraction. For example, the problem of localization on the road is addressed with Bayesian Simultaneous Localization and Mapping (SLAM) algorithm [7] and its variations. Computer Vision algorithms are used for object recognition, semantic understanding, and point cloud annotation. They provide scene

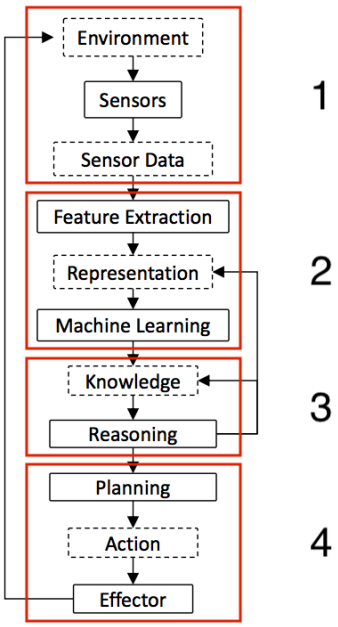


Figure 2.1: Stack of tasks for autonomous driving algorithm

segmentation and labeling which is then used in modeling of the environment and decision making.

The third part of the stack incorporates machine learning and knowledge to provide decisions. For these purposes, one can use probabilistic models or reinforcement learning. These approaches allow dealing with uncertainty in the perception and actuation.

The last part of the pipeline implements decisions made by the previous part. The rate of cycle repetition lies in milliseconds; therefore it is crucial that all parts must be performed in real-time.

2.2 Related work

Reinforcement learning (RL) algorithms are used to train an agent in a simulated or real environment. Recent approaches include applications of model-based and model-free algorithms with function approximators solving control problems in computer game environments and real life.

Sallab et al. [37] developed a deep reinforcement learning framework which incorporates Recurrent Neural Network for hidden state inference and recent work on attention models for focus on relevant information. The framework was tested in an open source 3D car racing simulator called TORCS. Simulation results demonstrate learning of autonomous maneuvering in a scenario of complex road curvatures and simple interaction of other vehicles.

Manuelli et Florence [25] evaluated the ability of several reinforcement learning methods to autonomously drive in an environment with obstacles using input from LIDAR. They have shown, that RL methods work well for the specified problem, but were unable to outperform controllers designed by hand.

Vitelli et Nayebi [41] tested the performance of a Deep Q-Network (DQN) in a 3D VDrift driving simulator environment. They have utilized the approach of Mnih et al. [29] where a DQN with the underlying convolutional neural network was trained with a variant of Q-learning to play Atari games. They have shown that RL algorithms were successful in navigating around a simulated environment and moreover that their greedy agent outperforms hand-designed algorithm.

April, Raphael, Rishi [47] were able to train an RL agent to control the simulated car in JavaScript Racer with the use of Deep Q-Learning. Their agent successfully learned the turning operation, progressively gaining the ability to navigate larger sections of the simulated raceway without crashing. In obstacle avoidance, however, the agent faced challenges which are suspected to be due to insufficient training time.

Algorithms of safe reinforcement learning are used for solving optimal control problem with safety constraints. The scope of research in this area include safe exploration, risk minimization during and after learning, and the design of a robust reward function along with others.

Xiong et al. [45] combined deep reinforcement learning with safety-based control implemented by an artificial potential field and path tracking for autonomous driving. They have used deep deterministic Policy Gradient [23] algorithm to obtain an initial driving policy and then combined it with safety-based control to avoid a collision and drive along the track. They have shown that the resulting algorithm performs well in the TORCS environment.

Fulton and Platzer [11] combined reinforcement learning with formal verification to ensure the safety of a learning agent. They have proved that their approach preserves safety guarantees as well as practical performance benefits provided by reinforcement learning. As an application of the algorithm, they have developed a model of an adaptive cruise control model for an autonomous car.

Pecka and Svoboda [32] described different approaches to safety in (semi) autonomous robotics with a focus on the exploration of unknown states. They divide approaches into three categories: algorithms from optimal control theory, reinforcement learning algorithms based on state labeling, and algorithms utilizing additional prior knowledge. The work also proposes a new way of state labeling as safe, critical and unsafe states.

Chapter 3

Reinforcement learning

Reinforcement learning (RL) refers to a framework for learning an agent to interact with an environment to maximize some numerical measure, which represents a long-term objective. Reinforcement learning is defined not by characterizing learning methods, but by characterizing a learning problem. Any method that is well suited to solving that problem, is considered a reinforcement learning method [40].

RL algorithms aim to solve the optimal control problem formulated as Markov decision processes (MDP). Markov decision process is a name for a reinforcement learning task that satisfies the Markov property. If the state and action spaces are finite, then it is called a finite Markov Decision Process (finite MDP) [40]. The process of agent–environment interaction in an MDP is depicted in Figure 3.1.

Formally a finite MDP is represented by the tuple (S, A, T, R) where:

- S is a finite set of the environment states s,
- A is a finite set of actions a,
- T is a transition function, $T\colon S \times A \times S \to [0, 1]$, where $T(s, a, s') = p(s'|a, s)$ is the probability that action a in state s will lead to state s',
- R is the reward function, $R\colon S \times A \times S \to \mathbb{R}$, where $R(s, a, s')$ is the reward obtained by executing action a in state s resulting in a transition to the next state s'.

The goal of reinforcement learning is to find a policy π, which defines the learning agent's way of behaving at a given time. Formally it is a mapping from perceived states of the environment to actions to be taken when in those states. A deterministic policy is a mapping from the state space to the action space, which returns an action a for a state s:

$$\pi\colon S \to A \quad \pi(s) = a. \tag{3.1}$$

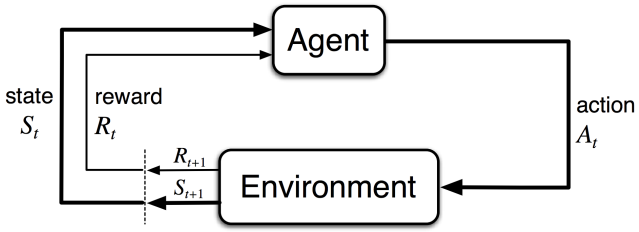


Figure 3.1: The agent–environment interaction in a Markov Decision Process

A stochastic policy is a mapping from the space of state-action pairs to an interval $[0, 1]$, which assigns a probability of taking action a in a state s:

$$\pi \colon S \times A \to [0, 1] \quad \pi(a|s) = p(a|s). \tag{3.2}$$

An optimal policy maximizes the expected value of a future reward and satisfies the Bellmann optimality equation:

$$v_\pi^*(s) = \max_a \sum_{s'} p(s'|s, a) \left[r(s, a, s') + \gamma v_\pi^*(s') \right], \tag{3.3}$$

where $r(s, a, s')$ denotes expected immediate reward on a transition from s to s' under the action a, γ denotes a discount factor and $v_\pi(s)$ denotes a value of a state s under a policy π.

The function $v_\pi(s)$ is referred to as state-value function or simply *value function*. Formally, $v_\pi(s)$ is defined as the expected return when starting in s and following π after that:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right]. \tag{3.4}$$

where $\mathbb{E}_\pi$ denotes the expected value of a random variable given that the agent follows policy π, and t is any time step. The value of the terminal state, if any, is always zero.

Similarly, the action-value function, is the value of taking action a in a state s under a policy π, denoted $q_\pi(s, a)$, is defined as the expected return starting from s, taking action a, and after that following policy π:

$$\begin{aligned} q_\pi(s, a) = & \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] = \\ & \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right]. \end{aligned} \tag{3.5}$$

While learning the agent is estimating true values of $v(s)$ or $q(s, a)$, which are denoted as $V(s)$ and $Q(s, a)$ correspondingly. In this work $V(s)$ is referred to as *value function* and $Q(s, a)$ is referred to as *Q-function*.

In our setting, the agent is represented by a self-driving car, which interacts with the environment at discrete time steps $t = 0, 1, 2, 3...$ At each time step t, the agent receives some representation of the environment state. It can be represented by an image, a depth map, an inner state of a vehicle, or by other appropriate features. Taking action a_{t-1} in state s_{t-1} results in a transition to a new state s_t and receiving a reward r_t.

3.1 Reinforcement learning algorithms

Model-based and model-free algorithms represent two main categories of RL methods. In the literature, model-based algorithms are usually referred to as Dynamic Programming. Model-free algorithms are referred to as reinforcement learning or Neuro-dynamic programming [40].

3.1.1 Model based algorithms

Assuming that we know transition probabilities of MDP $p(s'|s, a)$, expected immediate rewards $r(s, a, s')$ and some initial policy, we can apply an iterative algorithm for policy improvement. At every time step, we update our policy such that it picks the best possible action in a given state with the update rule:

$$\pi'(s) = \arg\max_a \sum_{s'} p(s'|s, a) \left[r(s, a, s') + \gamma v_\pi(s') \right]. \tag{3.6}$$

Once we have improved our policy, we can estimate its value and adjust it once again with the same procedure. This algorithm is called policy iteration, and it falls under the category of model-based algorithms. These algorithms take advantage of the Markov property and estimate a state's value from the value estimates of successor states. This approach is called bootstrapping.

3.1.2 Model free algorithms

Monte Carlo methods fall under model-free approaches as they only require experience to learn from. It is a way of solving an RL problem based on averaging of sample returns. The algorithm is defined for an episodic task only, i.e. finite MDP. The value function is estimated upon the completion of an episode (game). Monte Carlo methods are incremental in an episode-by-episode sense, but not in a step-by-step sense, as value iteration. And because values are estimated as from experience, Monte Carlo methods do not perform bootstrapping.

The value estimation can be performed in an on-policy or off-policy manner. On-policy methods estimate the value of a policy while using it for control, while in off-policy methods policy for estimation and real policy may differ.

Temporal Difference (TD) learning represent a combination of Dynamic Programming and Monte Carlo methods. It learns from the experience like MC, but at the same time performs bootstrapping. An advantage of TD is the fact that it updates value after each step. TD value estimation can also be used in an on-policy or off-policy manner. On-policy TD estimation algorithm is called Sarsa [36]. Off-policy TD is well-known Q-learning [43].

Policy Gradient methods do not require estimation of a value function $v_\pi(s)$ as they rely upon optimizing parametric policies with respect to the expected return.

3.2 Exploration vs exploitation: different strategies

One of the challenges that arise in reinforcement learning is the trade-off between exploration and exploitation. In order to obtain the highest reward from the environment, the agent must take appropriate actions, which were discovered previously. To find such actions, the agent has to try actions it has not taken before. In other words, the agent has to exploit known actions to obtain a reward, but simultaneously it also has to explore space to make better selections in the future. It is essential to find the balance between exploration and exploitation to gain information and simultaneously improve the policy. There are several ways how to address the issue, which are presented in Sections 3.2.1 through 3.2.4.

3.2.1 ϵ-constrained strategy

The action with the highest $q(s, a)$ is selected with probability $1 - \epsilon$ and a random action is selected with probability ϵ (with uniform distribution). A typical parameter value might be $\epsilon = 0.05$, but it highly depends on the environment and specifics of the task.

3.2.2 Softmax strategy

In order to chose the next action, we estimate values of potential next states. Next states represent a set of states to which we arrive by executing correspondent action. Then we use softmax function to convert values of state-action pairs directly into corresponding action probabilities.

$$\pi(a|s) = \frac{e^{q(s,a)}}{\sum_{b \in A} e^{q(b,s)}} \tag{3.7}$$

Softmax function converts values into a discrete probability distribution, from which we can directly sample the actions.

3.2.3 Softmax strategy with varying temperature

In case of strategy with varying temperature, new temperature parameter τ adjusts the amount of randomness in action picking.

$$\pi(a|s) = \frac{e^{\frac{q(s,a)}{\tau}}}{\sum_{b \in A} e^{\frac{q(b,s)}{\tau}}} \tag{3.8}$$

For high temperatures $\tau \to \infty$, all actions have nearly the same probability. The lower the temperature, the more original distribution affect the probability. For a low temperature $\tau \to 0^+$, the probability of the action with the highest expected reward tends to 1. While learning, it is beneficial to decrease the temperature gradually to smoothly transit from exploring policy to exploiting one.

3.2.4 Optimistic initialization

The concept of optimistic initialization is connected with the exploration process. One of the main ideas behind optimistic initialization of value function is to force the agent to explore unknown states. A fundamental advantage of optimistic initialization is that it provides extensive exploration, and as a result of this, it is difficult to miss highly rewarded final states. The policy, which is learned with the optimistic initialization is either optimal or leads to effective learning. The disadvantage of this approach is that it may take much time for the algorithm to get rid of optimism, propagate actual costs of actions, and converge to a final policy [15].

3.3 Design of the reward function

The reward function determines the goal in a reinforcement learning problem. It is defined as a mapping from state-action pair to a real number $R\colon S \times A \times S \to \mathbb{R}$. Sometimes it is given naturally (i.e., a monetary cost or an energy efficiency), but in some domains, it is not trivial to choose the right one suitable for successful learning.

The choice of an appropriate reward function during the design of an MDP abstraction for a real-life problem is a researched issue in RL. A designer usually has an idea about how resulting strategy should look like, and he or she constructs a reward function to emphasize specific goals (i.e., reach the goal and stay safe). One of the biggest threats of the wrong reward function is a reward hacking problem, which can compromise the whole learning [2].

Occasionally the reward function can be easily designed by hand or is given (i.e., in a computer game or a competition). However, in some cases, it is hard to find such a function. A reward function, which makes sense for a human, is not always the best for RL agent training.

The reward function can be sparse in its limit cases, e.g., a single reinforcement signal after completing an episode or dense, as an opposite. The dense reward function assigns rewards for intermediate results. For example, it can be positive signals for completion a sub-level in a multi-level game. Both sparse and dense reward functions are learnable. It means that the agent can successfully learn value function using given rewards. However, sometimes the reward function is too sparse, and the agent can arrive at the local maximum and get stuck there forever. Such a situation occurs when the state with a big reward is difficult to reach.

3.3.1 Inverse reinforcement learning

Sometimes, we want the agent to imitate an expert's behavior in given MDP. In this case, we would like to derive an expert's reward function from a set of observations. Inverse reinforcement learning (IRL) is an algorithm which finds one of the possible expert's reward functions from the demonstrations, represented by state-action tuples [10]. IRL finds a reward function under which the expert's behavior is uniquely optimal. As a result instead of manually designed reward function, we have the optimal reward function suitable for the specific task.

3.4 Reinforcement learning with function approximation

In case of a small environment, we can represent the value function explicitly by a table. Since the number of records in the table growths exponentially with the number of states and actions in a finite MDP, (and potentially it can be infinite), it is necessary to use function approximators.

Many researchers discuss the stability of RL performance combined with function approximators. In principle, RL algorithms are considered to be unstable under function approximators [4, 19]. The reason is that during learning, the estimation error of value function is propagated to change in the policy. The policy change is propagated further to the change in the estimation of a state value. This process may of course rapidly converge to a solution, but also it can diverge and therefore bring no results [22].

Chapter 4

Safe reinforcement learning

4.1 Risk notion

Risk perception and rational behavior were studied since the early beginnings of the economic theory and still are a matter of research nowadays. With the rapid developing of machine learning and autonomous robotics, risk elimination from being a theoretical interest became an essential part of trajectory design and task planning.

Individual agents trained in real environments are exhibited to a natural source of risk due to its stochasticity. Safe algorithms are specially designed to reduce this risk to a reasonable level and to provide a sensibly-behaving predictable agent. The task is usually formalized by assigning a numerical value to the amount of risk. This, however, is not always possible, as we might not have complete or sufficient information about the state.

Economics distinguishes between decisions under risk and decisions under uncertainty [21]. When an agent has a set of actions, whose outcomes are known or can be represented by a probability distribution, it is said, that the agent decides under risk. In decisions under uncertainty, an agent doesn't dispose of prior knowledge about the probability distribution of outcomes. In robotics, these two cases are not studied separately, but we take a look on different methods and describe their approach towards the risk elimination in both cases.

When we want to minimize risk, it is necessary to adopt a method of its quantification. A natural approach is to define a mapping from the set of all possible distributions over return G (implicitly defined in Equation 3.4) to a real value. The financial theory developed many examples of risk metrics, which are used in practice, e.g., standard deviation, value at risk (VaR), expected shortfall (CVaR). On the contrary, in the field of robotics, this subject did not draw such attention due to the complexity of its problems.

We denote the finite set of possible outcomes as Ω. By P we denote a

probability mass function that assigns probabilities $P(\omega)$ to outcomes $\omega \in \Omega$. Consider a cost function $Z\colon \Omega \to \mathbb{R}$ that assigns costs $Z(\omega)$ to outcomes. The reward Z is then a random variable. Let $\mathcal{Z}$ denote the set of all random variables on Ω. A risk metric is a mapping which maps cost random variable to a real number:

$$\rho\colon \mathcal{Z} \to \mathbb{R}. \tag{4.1}$$

In reinforcement learning problems we usually work with return G instead of cost Z, but we can show that $Z = -G$:

$$Z_t = \sum_{k=0}^{\infty} -\gamma^k R_{t+k+1} = -\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = -G_t. \tag{4.2}$$

Risk metric effectively tells us how much the distribution $Z(\omega)$ can be potentially dangerous. By introducing a change to the policy we can shape the cost distribution, which allows us to suppress the risk (in terms of a metric). The original maximization task of the expected value of the reward is transformed to a constrained problem:

$$\pi(s) = \arg\max_a \; q_\pi(a, s) \quad s.t. \quad \rho < \alpha, \tag{4.3}$$

where α represents some threshold, below which we want to suppress the risk.

In principle, there exist infinitely many ways how to define a risk metric. An only small subset, however, corresponds to a natural concept of risk. In order to describe the completeness of a metric, the economic theory developed a number of properties, which describe a coherent metric.

Many attempts to find a comprehensive risk metric were made primarily in financial theory field [28]. An attempt to develop a coherent risk metric for robotics was made by Majumdar and Pavone [26]. They parallel the effort in the finance community on developing axioms that any rational metric of risk [3] in a robotics application should satisfy. They state that a sensible risk metric should fulfill axioms A0-A5:

A0 Monetary rewards. The costs Z are expressed in monetary terms (tangible and interpretable values).

A1 Monotonicity. Let $Z,\ Z' \in \mathcal{Z}$ be two cost random variables. Suppose $Z(\omega) \leq Z'(\omega)$ for all $\omega \in \Omega$. Then $\rho(Z) \leq \rho(Z')$. If a random reward Z' is greater or equal to Z no matter what random outcome occurs, then Z' is at least as risky as Z.

A2 Translation invariance. Let $Z \in \mathcal{Z}$ and $c \in \mathbb{Z}$. Then $\rho(Z + c) = \rho(Z) + c$. If costs are increased by a deterministic amount c, then the corresponding risk is increased by the same amount.

A3 Positive homogeneity. Let $Z \in \mathcal{Z}$ and $\beta \in \mathbb{R}$ and $\beta \geq 0$. Then $\rho(\beta Z) = \beta\rho(Z)$. If costs are scaled by a deterministic non-negative amount β, then the corresponding risk is scaled by the same amount.

A4 Subadditivity. Let Z, $Z' \in \mathcal{Z}$. Then $\rho(Z + Z') \leq \rho(Z) + \rho(Z'))$. Diversification of actions may decrease risk.

A5 Comonotone additivity. Suppose Z and Z' are comonotone, i.e. $(Z(\omega) - Z(\omega')(Z'(\omega) - Z'(\omega')) \geq 0$, for all $(\omega \times \omega') \in \Omega \times \Omega$. Then $\rho(Z + Z') = \rho(Z) + \rho(Z'))$. If two costs fall and rise together, there is no benefit from diversifying.

A6 Law invariance. Suppose Z and Z' are identically distributed. Then $\rho(Z) = \rho(Z')$. If two tasks have the same distribution of costs, then their correspondent risks are identical.

The most commonly used risk metrics in robotics are the expected cost and worst-case metrics [26]. The expected cost corresponds to risk neutrality while the worst-case assessment corresponds to extreme risk aversion. Majumdar and Pavone [26] provide a list of popular risk metrics with correspondent axioms they satisfy. These risk metrics are listed in Table 4.1.

Risk metric	Axioms
Conditional Value at Risk (CVaR)	A1–A6
Expected Cost	A1–A6
Worst case	A1–A6
Mean–Variance	A6
Entropic risk	A1, A2, A6
Value at Risk (VaR)	A1–A3, A5, A6
Standard semi-deviation	A1 –A4, A6

Table 4.1: Axioms satisfied by popular risk metrics.

4.2 Safety in reinforcement learning

Machine learning community researched many issues related to risk in reinforcement learning. The range of research includes safe exploration, solution robustness, and risk-sensitivity, we review these in detail below.

Amodei et al. provided a profound overview of research in the area of AI safety and suggested directions for further research [2]. Their primary interest is directed towards a problem of accidents in machine learning systems. They studied each part of RL-agent design learning process and traced major safety-related threats.

They distinguish five sources of accidents:

1. **Unsafe Exploration**: possible permanent damage of agent during environment exploration.

   The unsafe exploration has roots in the necessity of environment examination versus lack of information about it. There are several ways how to address this issue, but first, we should divide it into two problems. The problem of risk connected with initial exploration when an agent decides under uncertainty can be solved only by providing an agent with some information about the environment. It can be done by directed exploration or by expert's demonstrations. Further exploration can be secured by a teacher or some metric which assign a risk value for an action.

2. **Reward Hacking**: agent gaming its reward function.

   The problem of reward hacking is related to the agent's ability to exploit reward function in order to obtain high reward with minimum means. For example, if we have a vacuum cleaning robot and we reward it for achieving an environment free of messes, it might disable its vision so that it does not find any messes. It reminds us of the Goodhart's law: "When a measure becomes a target, it ceases to be a good measure.". As a consequence we should be aware, that reward hacking is a natural behavior of the agent. Authors propose several methods on how to address this issue, description of which is out of the scope of this thesis.

3. **Unscalable Oversight**: an inability of an agent to respect aspects of the objective that are too expensive to be frequently evaluated during training.

   Unscalable oversight problem arises because of the high computational complexity of reward. The issue is connected to reward function engineering. When an objective is too complex, the agent should use some proxy objectives, which together effectively approximate the original one. Authors mention the most promising algorithm for addressing this issue: semi-supervised reinforcement learning. According to the paper "...it resembles ordinary reinforcement learning except that the agent can only see its reward on a small fraction of the timesteps or episodes".

4. **Negative Side Effects**: unwanted damage or alteration of surrounding during normal agent functioning.

   Negative Side Effects represent an unwanted change of environment which was caused by agent behavior. Reward function usually states a direct goal, not specifying how to accomplish it. It can happen, that most efficient trajectory involves unrelated change or partial destruction of the environment. It is impracticable to penalize all unwanted state-action pairs manually, therefore proposed approaches are aimed towards impact regularization in general.

5. **Distributional shift**: an inability of an agent to behave robustly in a slightly different environment.

The distributional shift is related to situations when the environment is different from the one which was used during learning. This problem is especially relevant when an agent is trained in some simulator and then is deployed to the real environment. Learned model parameters become inapplicable and agent cannot function properly. The problem can be addressed by using more robust models or combinations of them. Potentially related areas include change and anomaly detection, hypothesis testing and transfer learning.

- Safe RL
  - Optimization Criterion
    - Worst Case Criterion
    - Risk Sensitive Criterion
    - **Constrained Criterion**
      - Castro et al. (2012)
    - Other Optimization Criteria
  - Exploration Process
    - External knowledge
      - Providing Initial Knowledge
      - **Deriving a Policy from Demonstrations**
      - Teacher Advice
    - **Risk Directed Exploration**
      - Law (2005)

Table 4.2: Overview of the approaches for safe reinforcement learning used in this thesis

Designing a truly risk-aware policy is not a matter of a single algorithm. The effort should be directed towards the elimination of all sources by targeting roots of the problems.

Garcia et Fernandez, in comprehensive survey on safe reinforcement learning [12] provide an overview of methods to risk approach in robotics applications. Authors created a clear structure of recently used methods and provided insights of key features of each approach. The survey serves as a digest of safe leaning methods, we do not describe each of them, but utilize the structure to show the methods we are interested in. The structure is depicted in Table 4.2.

Taking into consideration specific sources of risk in autonomous driving, we focus on one particular problem: safe exploration. The safe exploration can be disentangled to two sub-problems: policy initialization and safety

constrained reinforcement learning. The topics are described separately in Sections 4.3 and 4.4. Section 4.3 describes three algorithms which can be used for policy initialization, namely behavioral cloning, inverse reinforcement learning, and Generative Adversarial Imitation Learning.

4.3 Policy initialization

Reinforcement learning is a general framework for solving an MDP problem. The algorithm in the default setting assumes immortality of agent besides all. When we deal with tangible agents interacting with the real environment, an assumption of immortality and thus repeatability of experience is violated. This fact represents a source of risk for the agent and environment. It is especially relevant at the beginning of learning when a strategy is practically random, and state-space is not explored. Simply speaking, an agent does not know *how* and *where* to go. This problem can be mitigated using the proper policy initialization and safe space exploration.

The policy initialization can be addressed by either Behavioral Cloning or inverse reinforcement learning. Both algorithms assume the same problem setup and both produce a strategy as output, but inverse reinforcement learning also provides a reward function. Problem setup is the following: given set of training examples from an expert, we have to design a policy which imitates the expert's behavior. Training examples consist of state-action tuples.

4.3.1 Behavioral Cloning

Behavioral Cloning is a supervised learning technique which teaches an agent to act in the same way as an expert. It is formulated as a standard discriminative machine learning problem. Given a set of trajectories and model structure it seeks optimal parameters for the policy function by solving an optimization problem (4.4) with respect to a loss function [49]:

$$\theta^* = \arg\min_{\theta} \sum_{\mathcal{P}_E} \sum_{t=1}^{T} \ell\left(a_t, \pi_\theta\left(s_t\right)\right), \tag{4.4}$$

where

ℓ is a loss function (e.g. mean squared error or cross-entropy),
a_t is an expert's action in time t,
π_θ is a parametrized policy,
θ are the parameters of the policy.

Implicitly it infers posterior probability $p(a|s)$ distribution of actions given states. Examples of an application on real problems can be found here [33, 38].

4.3.2 Inverse reinforcement learning

Inverse reinforcement learning (IRL) seeks not only for a valid strategy π but also for a reward function under which the behavior of an expert is optimal. The task is formulated as a saddle-point optimization problem: it simultaneously optimizes agent's policy by minimizing the loss in the inner loop and finds a cost (reward) function by solving a maximization problem in the outer loop.

As a specific example of the IRL algorithm, we adopt maximum causal entropy IRL [50], which fits a cost function from a family of functions $\mathcal{C}$ with the optimization problem:

$$\max_{c \in \mathcal{C}} \left(\left(\min_{\pi \in \Pi} -H(\pi) + \mathbb{E}_{\pi} \left[c(s,a) \right] \right) - \mathbb{E}_{\pi_E} \left[c(s,a) \right] \right), \tag{4.5}$$

where

$c(s,a)$ is a cost function $c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$,
$H(\pi)$ is γ-discounted causal entropy $H(\pi) \triangleq \mathbb{E}_{\pi}[-\log \pi(a|s)]$,
π_E is the expert's policy.

Maximum causal entropy IRL seeks a cost function $c \in \mathcal{C}$ that assigns low cost to the expert policy and high cost to other policies, thereby allowing the expert policy to be found via a certain reinforcement learning procedure:

$$RL(c) = \arg\min_{\pi \in \Pi} -H(\pi) + \mathbb{E}_{\pi} \left[c(s,a) \right], \tag{4.6}$$

which maps a cost function to high-entropy policies that minimize the expected cumulative cost.

4.3.3 Generative Adversarial Imitation Learning

Both BC and IRL have their advantages and downsides. Behavioral Cloning succeeds only with a large amount of data. IRL learns a policy which prioritizes entire trajectories over others [35]. IRL algorithm is very computationally demanding, requiring to solve RL in the inner loop. It makes this algorithm hard to scale for large environments. Ho and Ermon [17] came with a solution to this. They formulated a dual problem to IRL, which finds a proper policy without explicitly finding a reward function [17]. Their imitation learning

algorithm finds a policy close to an expert's in terms of occupancy measures om. The optimization problem is formulated as follows:

$$\min_{\pi} d_{\psi}(om_{\pi}, om_{E}) - H(\pi), \tag{4.7}$$

where

d_{ψ} is a regularization function,
om_{π} is an occupancy measure of agent's policy (4.8),
om_{E} is an occupancy measure of expert's policy (4.8).

$$om_{\pi} \colon C \times A \to \mathbb{R} \quad om_{\pi} = \pi(a|s) \sum_{t=0}^{\infty} \gamma^{t} P(s_t = s|\pi) \tag{4.8}$$

Occupancy measure (4.8) can be understood as a discounted joint probability of state-action pair. Regularization function d_{ψ} smoothly penalizes violations in the difference between the occupancy measures of agent and expert. Authors have shown, that there exists a one-to-one correspondence between policy and occupancy measure, which means the closer agent's occupancy measure is to expert's, the more similar the policies are. The definition of occupancy measure allows us to write $\mathbb{E}_{\pi}[c(s,a)] = \sum_{s,a} om_{\pi}(s,a)c(s,a)$.

The optimization procedure is implemented as a generative adversarial network. The algorithm simulates a zero-sum game between discriminator network D and generative model network G. The model generates data, and its job is to be as much as possible close to actual data distribution. The job of D is to distinguish between the distribution of data generated by G and the actual data distribution. When D cannot distinguish generated data from the real data, then the generator has managed to learn actual data distribution. In our case, G is generating om_{π} and its job to find a proper policy π, such that its occupancy measure resembles the expert's one well. Pseudocode describing the algorithm is listed in Algorithm 1.

Empirical mean $\hat{\mathbb{E}}_{\tau}[\cdot]$ is computed using occupancy measure derived from sampled trajectories:

$$\hat{\mathbb{E}}_{\tau}[f(s,a)] = \sum_{s,a} \hat{om}_{\tau}(s,a) f(s,a). \tag{4.9}$$

4.4 Risk-aware reinforcement learning

After the initialization step, the policy may serve as a base behavior for the agent. The agent was learned only from expert demonstrations, which means that in some regions of state space policy is undefined (or assigned

Algorithm 1 Generative Adversarial Imitation Learning

Input Expert trajectories τ_E generated by expert's policy π_E
Initial policy and discriminator parameters θ_0, ω_0
Output optimized policy parameters θ^*

1: **for** $i = 0, 1, 2 \dots$ **do**
2: Sample trajectories τ_i using policy π
3: Update the discriminator parameters ω with the cross entropy loss gradient

$$\hat{\mathbb{E}}_{\tau_i} \left[\nabla_\omega \log(D_w(s,a))\right] + \hat{\mathbb{E}}_{\tau_E} \left[\nabla_\omega \log(1 - D_w(s,a))\right]$$

4: Update the discriminator parameters θ with cost function $\log(D_w(s,a))$

$$\hat{\mathbb{E}}_{\tau_i} \left[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)\right] - \lambda \nabla_\theta H(\pi_\theta)$$

$$\text{where } Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i} \left[\nabla_\theta \log(D_w(s,a))|s_0 = \bar{s}, a_0 = \bar{a}\right]$$

5: **end for**
6: **return** θ

to random). It is clear, that for a successful operating in a high-dimensional environment as a roadway, the bare policy imitation is insufficient. It is necessary to continue learning through safe exploration of space.

This section provides an overview of risk-aware reinforcement learning algorithms with a brief comment on their advantages and applicability. Algorithms, which are used later in work, are described in more details in Sections 4.4.2 and 4.4.3.

4.4.1 Overview of risk-aware reinforcement learning algorithms

Hans et al. [16] came with a notion of safety, which is concerned with states or transitions that can lead to damage. They introduced the concept of a safety function for determining a state safety degree and a backup policy that is able to lead the controlled system from a critical state back to a safe one. The safety function is learned from collected exploration data. It estimates the minimal reward r_{min} that would be observed when executing an action and afterward following the backup policy (min-reward estimation). For this purpose, min-reward samples (s, a, r_{min}), which depend on the backup policy π_b, are collected during exploration and used to estimate a min-reward function $R^{\pi_b}_{min}(s,a)$. The backup policy is activated when the agent reaches an unknown state during exploration and thus is unable to choose a safe action. A good choice of such a policy for known problems is a dynamic-based

controller or rule-based control. In case we do not have access to one, it has to be learned from observation data. The policy is obtained by solving an altered Bellman optimality equation that does not maximize the expected sum of rewards, but the minimal reward to come. They successfully evaluated their approach on a simplified simulation of a gas turbine. The experiments showed that the exploration using their technique was safe and covered large parts of the state space.

Moldovan et Abbeel [30] developed an approach based on an idea of a safe space: an ergodic part of the state space. From any state of ergodic space, the agent can get to another state, which makes it safe. An opposite to it is a set of states from where the agent cannot return, e.g. crash. They improved the R-max algorithm by introducing a safety constraint. According to the original work [5], in R-max, the agent always maintains a complete, but possibly inaccurate model of its environment and acts based on the optimal policy derived from this model. The model is initialized in an optimistic fashion: all actions in all states return the maximal possible reward. During execution, it is updated based on the agent's observations. Moldovan et Abbeel additionally restricted the space of eligible policies to those that preserve ergodicity of space with some user-specified probability δ, called the safety level. Their experiments in a Martian terrain exploration problem show that their method is able to explore better than classical exploration methods.

Garcia et Fernandez [13] developed a Policy Improvement through Safe Reinforcement Learning (PI-SRL) algorithm for state space exploration which takes risk into consideration. The risk of an action with respect to policy π is defined as the probability that the state sequence generated by the execution of policy π, terminates in an error state. Agent explores the state space by executing actions which slightly deviate from the main policy. Their experiments demonstrate the effectiveness of the algorithm in four different continuous domains: the car parking problem, pole-balancing, helicopter hovering, and business management (SIMBA).

Yu et Haskell [48] developed a family of simulation-based algorithms to solve approximately large-scale risk-aware MDPs. They defined a risk-to-go function J^π for any given policy π as (4.10).

$$J^\pi(s_0) \triangleq c(s_0, a_0) + \rho(\gamma c(s_1, a_1) + \rho(\gamma c(s_2, a_2) + \dots)), \tag{4.10}$$

where each ρ is a one-step coherent conditional risk measure and $c(s, a)$ is cost resulting from taking action a in stase s. The risk-to-go function is estimated through simulation-based approximate value iteration. They validated their algorithm on an optimal maintaining problem. They showed, that their algorithm was able to reduce chance of reaching the bad state and incurring a large cost.

4.4.2 Policy Gradient with variance constraint

Algorithm 2 Policy Gradient with variance constraint

Input MDP, Variance threshold b
Output Parametric policy π_θ

1: **while** θ not converged **do**
2: Sample episodes τ_i using policy π_θ
3: **for** each episode τ_k $\{s_0, a_0, r_0, \dots s_T, a_T, r_T\}$ **do**
4: Update estimation of return and variance

$$\widetilde{G}_{k+1} = \widetilde{G}_k + \alpha_k(R^k - \widetilde{G}_k)$$

$$\widetilde{\sigma}^2_{k+1} = \widetilde{\sigma}^2_k + \alpha_k((R^k)^2 - \widetilde{G}^2_k - \widetilde{\sigma}^2_k)$$

5: Compute variance penalty, update policy parameters

$$\zeta_t = \begin{cases} \lambda g'(\widetilde{\sigma}^2_k - b)((R^k)^2 - 2\widetilde{G}_k) & \text{if } \sigma^2_t > b \\ 0 \quad \text{if } \sigma^2_t < b \end{cases}$$

$$\theta_{k+1} = \theta_k + \beta_k(R^k - \zeta_t)z^k$$

6: **end for**
7: **end while**
8: **return** π_θ

Castro et al. [6] developed a policy-based algorithm with variance related risk criteria. They've introduced a new formula for the variance of the cost-to-go in episodic tasks. The algorithm is based on a return variance estimation and subsequent update of policy using an estimated gradient. The problem is stated as a constrained optimization problem (4.11).

$$\max_\theta V(s_0) \quad \text{s.t.} \quad \widetilde{\sigma}^2 [G(s_0)] \leq b, \tag{4.11}$$

where s_0 is a starting state, $V(s_0)$ is the value of state s_0, $\widetilde{\sigma}^2$ is the variance of the accumulated reward, and θ is parameters of policy. Here we consider G to be an undiscounted reward, equivalently we can say that $\gamma = 1$.

The simulation based algorithm for the constrained optimization problem (4.11) is described by pseudocode in Algorithm 2.

Where $\widetilde{G}$ is an estimation of return, R^k is the cumulative reward along trajectory k, $\widetilde{\sigma}^2$ is an estimation of return variance, $z^k \triangleq \nabla \log P(s^k)$ is the trajectory likelihood ratio derivative, α_k and β_k are positive step sizes, and $g(x) = (\max(0, x))^2$ is the penalty function.

The pseudocode describing the algorithm is stated in Algorithm 2. Castro

Algorithm 3 Q-learning with risk-directed exploration

Input MDP, Empirical model of environment $T_e : S \times A \times S \to [0, 1]$
Risk metric ρ, Reward-risk mixing proportion λ
Learning rate α, Discount rate γ, Number of steps n
Output Q-function $Q : S \times A \to \mathbb{R}$

1: Initialize $Q : S \times A \to \mathbb{R}$ arbitrary
2: **while** Q not converged **do**
3: Initialize episode $s_0 \sim p(s_0)$
4: **while** s is not terminal **do**
5: Estimate the n-step return distribution $G(s, a)$ using empirical model T_e
6: Compute risk-adjusted utility for all action in current state

$$U(s, a) = \lambda\rho(G(s, a)) + (1 - \lambda)Q(s, a)$$

7: Sample action, obtain a new state and a reward from environment

$$a \sim U(s, a) \quad s' = T(s, a) \quad r = R(s, a, s')$$

8: Update Q-function

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a'))$$

9: Update empirical model T_e

$$n(s, a, s') = n(s, a, s') + 1$$

10: **end while**
11: **end while**
12: **return** Q

et al. provide a proof that it converges almost surely to a locally optimal point of the corresponding objective function. They have demonstrated the applicability of the algorithm in a portfolio planning problem; namely they showed the ability of the algorithm to reduce the variance of the return distribution.

4.4.3 Q-learning with risk-directed exploration

Yang and Qiu [46] proposed a risk measure based on expected utility and entropy. Given a state, the measure of risk for a particular action is the weighted sum of the entropy and normalized expected reward of that action. Law [24] used this metric for risk-adjusted utility computation as a part of risk-sensitive Q-learning algorithm.

Basic Q-learning is an off-policy reinforcement learning algorithm. While learning, a Q-function is estimated through interaction with environment. Q-function is updated after every action execution (step) with an estimation of a one-step return.

Law's risk-sensitive algorithm estimates an one-step or two-step return distribution using empirical (learned) model of the environment. Using the return distribution, an entropy-based risk metric is applied for risk-adjusted utility computation. This risk-adjusted utility is then used for action selection. The actions are selected with the probability proportional to their utility. She has shown, that the algorithm was able to reduce risk during exploration.

The approach of Law can be extended, to create an improved version of the algorithm. This work presents such an enhanced version, it incorporates three improvement over the original version:

1. Q-function approximation by a neural network
2. adjustable number of steps for the estimation of the return distribution
3. ability to work with an arbitrary risk metric

Pseudocode describing this algorithm is stated in Algorithm 3. The user can specify the number of steps for the return distribution estimation and a risk metric to work with. The algorithm can run in a model-based mode (with provided empirical model) of in a model-free mode.

4.4.4 Comment on described safe reinforcement learning approaches

The approach of Hans et al. [16] is general and can be applied to any domain, but it does not guarantee, that learned backup policy is itself safe and complete for problems with large state space. Adapted R-max algorithm [5] is a useful framework for solving model-free MDP, but due to the fact that it uses linear programming for an exact solution of MDP, it is not scalable for larger problems. PI-SRL algorithm [13] cannot guarantee complete safety of the agent because of the randomized policy, but developed risk metric can reduce risk during exploration. Approaches of Castro et al. [6] and Yu et Haskell [48] are scalable to environments with big state and action space due to usage of iterative algorithms and estimation techniques. Further in work, we perform experiments to validate variance-constrained Policy Gradient algorithm developed by Castro et al. and an improved version of Law's Q-learning with risk-directed exploration.

Chapter 5

Experiments

This chapter presents the results of experiments conducted for all selected safe reinforcement learning algorithms. Each experiment section includes specification of algorithm parameters and description of the reward function.

For all algorithms used in the experiments, an own implementation was developed based on the results from [17, 24, 6]. The source code of algorithms is available in GitHub project repository **risk-aware-rl**[1]. The project includes the implementation of an MDP with an autonomous driving environment.

The project was implemented in Python with the usage of third-party packages, including Tensorflow [1] for optimization procedures and PyGame [39] for MDP environment. A full list of requirements is stored in the root of the project repository. The code is compatible with Python version 3.6.

Each experiment described in this chapter can be reproduced with a corresponding script located in the project repository. Each script outputs a trained model and performance metrics for the experiment. The usage examples are provided in the project's README file. Additionally, the project contains a training set for policy initialization algorithms and utility scripts for data generation.

5.1 Environment specification

One of the typical environment for autonomous driving algorithms testing is a two- or a multi-lane road viewed from above. The objective of an agent (self-driving car) is, e.g., to switch the lane, to overtake as many cars as possible or to reach a goal. Such an environment is stochastic due to a limited field of view and possibly non-deterministic behavior of other drivers. A two-lane road with cars represented by rectangles as depicted in Figure 5.1, is an example of such environment. It is used throughout the experiments.

[1] https://github.com/petroolg/risk-aware-rl

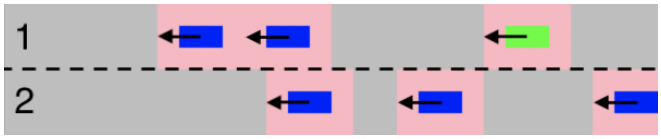


Figure 5.1: Environment used for experiments

The environment is represented by a two-lane roadway where the first lane is slower than the second lane. Agent's car is colored green, non-agent cars are colored blue. Cars displacement is generated randomly, but agent always starts at the first lane. Each car is surrounded by a safety margin, colored red on the Figure. The agent can access the events of the environment, e.g., collision with other car or the fact it was overtaken by a faster car from the second lane. Further in text cars from the first lane are referred to as "slow" and cars from the second lane are referred to as "fast". The reward function is constructed using events from the environment. In total there are 7 possible events which can occur in the environment. These include:

e_1 collision of agent's car with with a non-agent car
e_2 agent overtakes a fast car
e_3 agent is overtaken by a fast car
e_4 agent overtakes a slow car
e_5 agent is overtaken by a slow car
e_6 safe collision of agent's car with a non-agent car
e_7 none of above happened

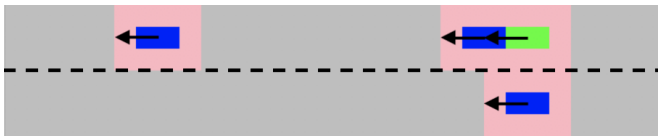


Figure 5.2: Collision of agent and a non-agent car

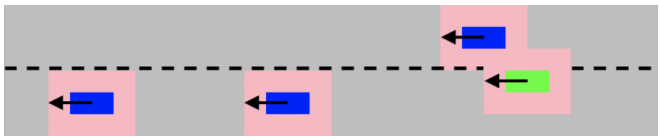


Figure 5.3: Safe collision of agent and a non-agent car

Further in the text, the reward function is represented by a vector with entries corresponding to a specific action. A state is represented by 30×6 single channel image and agent's velocity. Action space is discrete and is represented by Cartesian product of two spaces: $\text{Acc}_y = \{-1, 0 + 1\}$ and $\text{Shift}_x = \{-1, 0, +1\}$. Acc_y set defines acceleration in y axis (along the road), Shift_x defines shift in x axis (athwart the road).

Event	e_1	e_2	e_3	e_4	e_5	e_6	e_7
Reward	-10	2	-2	1	-1	-0.05	0

Table 5.1: Reward function used for policy initialization algorithms

The length of an episode is limited to 50 frames for policy initialization algorithms and 30 frames for risk-aware reinforcement learning. The goal of the agent is to overtake as many cars as possible and to avoid an accident. At each step, agent can turn left or right, speed up or slow down or do nothing.

5.2 Policy initialization algorithms

For purposes of policy initialization, two algorithms were used: Behavioral Cloning and Generative Adversarial Imitation Learning. As a baseline for these algorithms serves random policy function, which assigns a random action independently on a state. Experiment specifications and results are to be found in subsequent Sections 5.2.1 and 5.2.2. The goal was to compare the performance of two algorithms on the same problem.

Performance of algorithms for policy initialization is evaluated with four measures:

1. Return distribution
2. Return-per-step distribution
3. Percentage of episodes ended with an accident (Collision rate)
4. Episode length distribution

Table 5.1 describes a reward function used for policy initialization algorithms. Training set consists of 350 expert's trajectories generated by manual input in simulator resulting in approximately 24,000 state-action pairs. Table 5.2 shows the numbers of state-action pairs grouped by action, sorted in descending order. We can see, that training set is not balanced, e.g., "do nothing" action class is over-represented.

5.2.1 Behavioral cloning

Input: State-action pairs $(s_0, a_0), (s_1, a_1), \ldots (s_n, a_n)$
Output: Policy function $\pi\colon S \times A \to [0, 1]$

The policy function is represented by a 3-layer neural network with sigmoid activation function. The network takes as an input a state represented by

Action	N
do nothing	16068
decelerate	2225
turn left	1575
accelerate	1470
turn left, accelerate	1312
turn right	720
turn right, accelerate	596
turn left, decelerate	93
turn right, decelerate	3

Table 5.2: Distribution of action classes in the training set

		Training set	Random policy	BC	GAIL
Return per step	μ	0.116	-1.241	-0.553	-0.406
	M	0.09	-0.91	0.10	-0.16
Collision rate		0.0%	98.5%	22.5%	51.5%
Return	μ	6.29	-11.28	2.35	-3.36
	M	6.00	-11.00	4.95	-6.05
Episode length	μ	68.74	14.42	41.89	35.38
	M	68.00	12.00	50.00	47.50

Table 5.3: Performance metrics for policy initialization algorithms

flattened game frame and outputs a categorical distribution with probabilities of specific actions. The loss function is weighted to address an issue of non-balanced training set.

5.2.2 Generative adversarial imitation learning

Input: MDP, State-action pairs $(s_0, a_0), (s_1, a_1), \ldots (s_n, a_n)$
Output: Policy function $\pi\colon S \times A \to [0, 1]$

The policy function is represented by a 3-layer neural network with sigmoid activation function. Discriminator function is represented by a 2-layer neural network with sigmoid activation function. Pseudocode describing the algorithm is listed in Section 4.3.3, Algorithm 1. Policy function takes as an input a state represented by flattened game frame and outputs a categorical distribution with probabilities of specific actions. The discriminator function takes as an input a pair of state and action and outputs a score, which tells whether it was an expert's behavior or agent's.

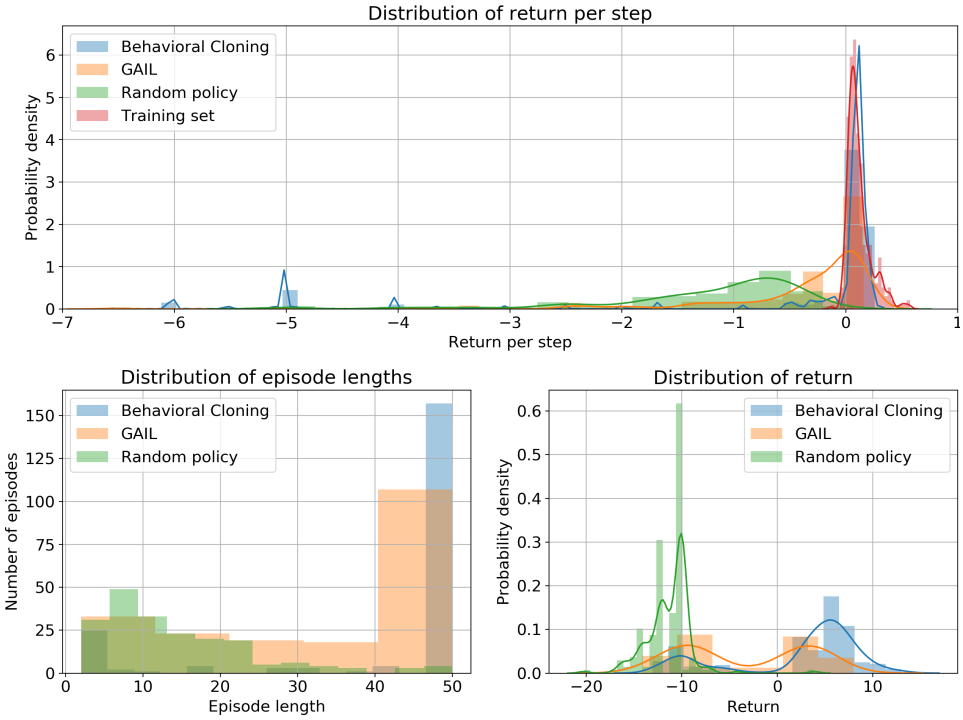


Figure 5.4: Distributions of return per step, return and episode lengths for policy initialization algorithm

5.2.3 Experiment results

The policies generated by both algorithms were evaluated on a test set containing 200 games with random settings. Figures 5.4 and 5.5 depict main results for policy initialization experiments.

Figure 5.4 shows distributions of return per ster, episode length and return for the baseline random policy, the training set and policies of both initialization algorithms. Table 5.3 contains numeric description of these distributions, where μ is expected value and M is median. The table also includes collision rates measured on train sets for all algorithms.

From the graphs in Figure 5.4 and Table 5.3, we can see that both learned policies outperform the random one. As is evident from the graph, the distribution of return per step for BC policy is very similar to the train set distribution, except for small peaks in negative part of returns. The distribution of return per step for GAIL policy is more spread, but it does not have such peaks. However, BC policy is indubitably better than GAIL policy. It has lower collision rate (22.5% compared to 51.5%) and significantly higher expected return (2.35 compared to -3.36). Median and expected value of episode lengths does not differ significantly.

Figure 5.5 depicts visualization of policies for the train set and both initialization algorithms. States (represented by images) are embedded in a

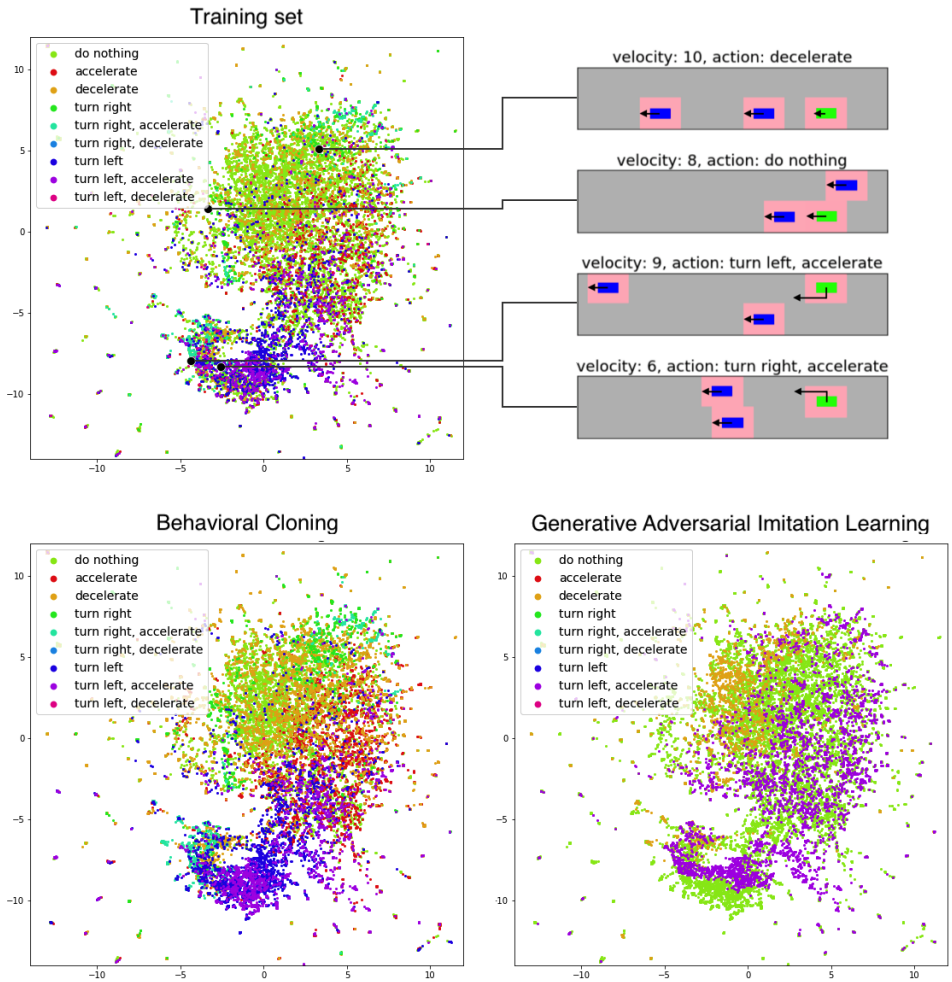


Figure 5.5: Visualization of policies generated by different initialisation algorithms

lower dimension using UMAP algorithm [27] and colored by an action which agent or teacher takes in this state. The embedding places similar states near each other in low dimension forming clusters of states in which agent takes the same action. Visualization contains four "zoomed in" states with correspondent points in low dimension.

From Figure 5.5, we can see that BC policy mimics training policy very well. It chooses the same actions in most states. This explains a high similarity between performance metrics of training set and BC policy. We can also see, that GAIL agent is using only a subset of actions in its strategy, omitting the rest – namely, it uses a subset of three actions: "do nothing", "turn left, accelerate" and "decelerate". Such behavior can be explained by partial mode collapse of generator network and training set structure. The generator learned to output a number of the most frequent actions (as we see from Table 5.2). Indirect approach of imitation expert's occupancy measure lead to learning the most essential part of behavior and ignoring less significant parts.

5.3 Q-learning with risk-directed exploration

Q-learning with risk-directed exploration is a variant of well-known model-based Q-learning algorithm, which was described in Section 4.4.3. Model-based Q-learning assumes knowing a transition function of MDP. In our case we do not have underlying dynamic model, therefore the transition function should be estimated through simulations. There are several ways how a transition function can be learned through interaction with MDP. In our experiments, it was represented by experienced transitions in the form of a dictionary. For each state-action pair, the dictionary holds an empirical distribution over possible next states. The dictionary was constructed during 100,000 episodes of random policy execution.

Using such empirical model and a reward function we are able to estimate one-step return distribution for action a in state s. We can extend this approach to calculate multi-step return distribution for arbitrary number of steps, since we know the probabilities of taking actions in subsequent states. In the experiments, a 3-step return distribution was used.

The pseudocode describing the algorithm is listed in Section 4.4.3, Algorithm 3. The algorithm accepts an arbitrary risk metric which makes it very flexible. Different risk metrics plugged in the algorithm help to emphasize aspects which may be more important for the specific task.

Two specific risk metrics were used in experiments: Conditional Value at Risk [34] and Entropy-based metric [46]. Experiments specifications and results are described in Sections 5.3.1 and 5.3.2. The goal of experiments was to compare model-free Q-learning, model-based Q-learning and Q-learning with risk-directed exploration.

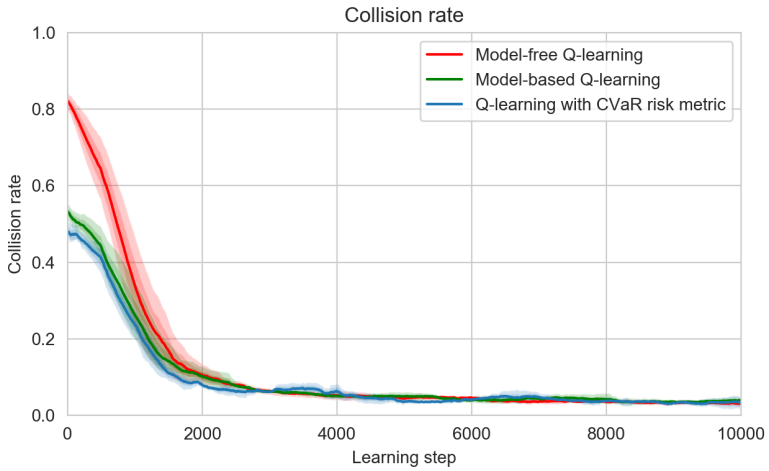
The performance of algorithms for risk-directed exploration is evaluated with two measures:

1. Return distribution
2. Percentage of episodes ended with an accident (Collision rate)

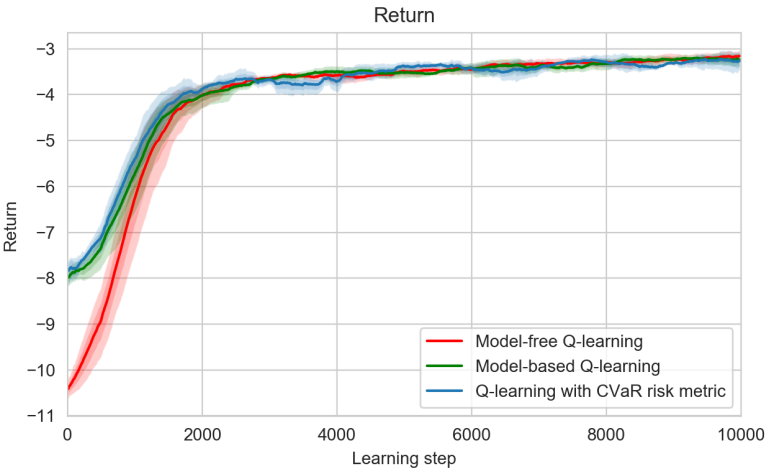
Table 5.4 describes a reward function used for Q-learning with risk-directed exploration. In both experiments, Q-function is represented by 3-layer neural network with leaky ReLU activation function. The network takes as an input a state-action pair, where state is represented by flattened game frame. It outputs a real number for given pair. The learning was limited to 10,000 steps.

Event	e_1	e_2	e_3	e_4	e_5	e_6	e_7
Reward	-10	1.0	-1.0	0.5	-0.5	-1.0	0

Table 5.4: Reward function used for Q-learning with CVaR risk metric



(a) : Evolution of collision rate



(b) : Evolution of return value

Figure 5.6: Evolution of performance metrics during learning

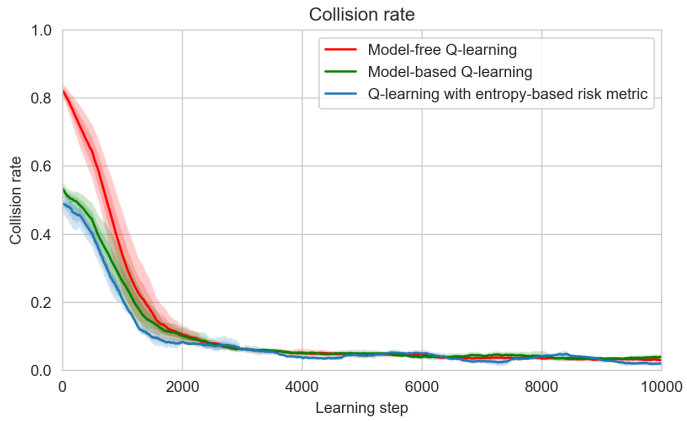
5.3.1 Q-learning with CVaR risk metric

Input: MDP, Empirical model of environment $T_e \colon S \times A \times S \to [0, 1]$, CVaR risk metric
Output: Q-function $Q \colon S \times A \to \mathbb{R}$

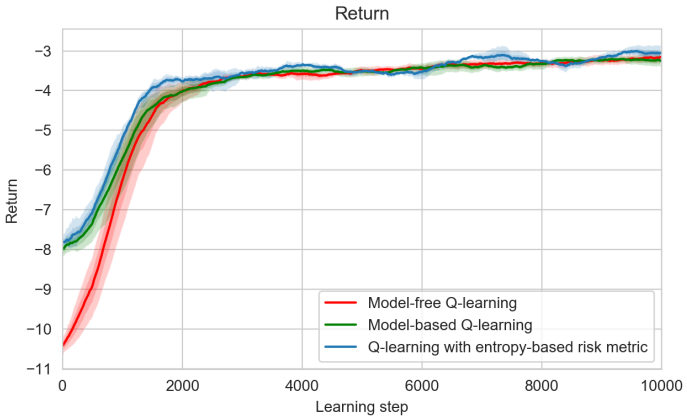
Yu et Haskell [48] define CVaR_α as follows:

$$\text{CVaR}_\alpha(G) \triangleq \inf_{\eta \in \mathbb{R}} \left\{ \eta + \frac{1}{1-\alpha} \hat{\mathbb{E}} \left[(G - \eta)_+ \right] \right\}. \tag{5.1}$$

CVaR_α is the expected value of return in the worst $\alpha\%$ of cases. Conditional Value at Risk estimates the risk of taking action a in state s and following



(a) : Evolution of collision rate



(b) : Evolution of return value

Figure 5.7: Evolution of performance metrics during learning

exploitative policy after that. It estimates the return in a pessimistic way, which in theory leads to more conservative actions.

In this experiment α is a hyper-parameter which was tuned using grid-search technique.

5.3.2 Q-learning with entropy-based risk metric

Input: MDP, Empirical model of environment $T_e \colon S \times A \times S \to [0, 1]$, Entropy-based risk metric
Output: Q-function $Q \colon S \times A \to \mathbb{R}$

Yang and Qui [46] define their entropy-based metric (in this text referred to as entropy risk, ER) as follows:

$$\mathrm{ER}(G) = \beta H(G) - (1-\beta)\frac{\mathbb{E}\left[G\right]}{\max_{a\in\mathcal{A}_s}\left|\mathbb{E}\left[G\right]\right|}. \tag{5.2}$$

where $H(G)$ is the entropy of return distribution, $\mathbb{E}\left[G\right]$ is its expected value, and β is a weighting coefficient. Entropy-based risk metric penalizes return distributions with high entropy and comparably low expected value. The high-entropy return indicates high uncertainty about future return, which helps to avoid risky actions.

In this experiment, β is a hyper-parameter which was tuned using grid-search technique.

5.3.3 Experiment results

The evolution of performance metrics during learning is shown in Figures 5.6 and 5.7. The figures contain metric values for model-free, model-based and risk-sensitive variant of Q-learning. The graphs show the evolution of the expected value and corresponding 1σ and 2σ confidence intervals for metric distributions. The expected value and confidence intervals were computed from the results of several runs of the algorithms (10 for model-based, 8 for model-free, and 4 for risk-aware Q-learning).

From the graphs we can clearly see that Q-learning with risk-directed exploration and plain model-based Q-learning outperform model-free Q-learning in terms of collision rate and return. This is an unsurprising result, as model (even incomplete) provides essential information for Q-function estimation ant thus helps to avoid dangerous states. But most importantly, Q-learning with risk-directed exploration has lower collision rates and higher return in comparison with model-based algorithm.

	CVaR	ER
Collision rate	-2.4%	-2.9%
Return	+3.5%	+5.9%

Table 5.5: Average improvement of performance metrics for Q-learning with risk-directed exploration

During first 3,000 learning steps, the risk-aware Q-learning shows lower collision rate and higher return than the model-based and model-free versions. After 3,000th step, all three algorithms converge to the same values and continue to improve with the same pace. Table 5.5 shows average improvement of performance metrics during first 3,000 learning steps for Q-learning with risk-directed exploration in comparison with model-based Q-learning. From the table, we can see that usage of entropy-based risk metric lead to slightly better performance than CVaR. From this, we can conclude, that it is more suitable for this task.

5.3.4 Policy Gradient with variance constraint

Input: MDP, Variance threshold b
Output: Policy function $\pi\colon S \times A \to [0, 1]$

To solve a constrained optimization problem stated in Equation 4.3, an implementation of Policy Gradient algorithm was developed, namely the technique of simulation-based optimization. The algorithm estimates the expected value and variance of the return from several episodes. Policy parameters are updated using an estimated gradient with variance penalty.

The algorithm optimizes policy function directly without estimation of the value function. This may be considered to be more convenient for the control algorithm designer, because it eliminates the need of constructing a policy from the output of the value function.

The pseudocode describing the algorithm is listed in Section 4.4.2, Algorithm 2. The algorithm comprises a modified version of REINFORCE [44] algorithm with additional estimation of the return variance, which is minimized as a measure of risk.

Performance of Policy Gradient algorithm with variance constraint is evaluated with four measures:

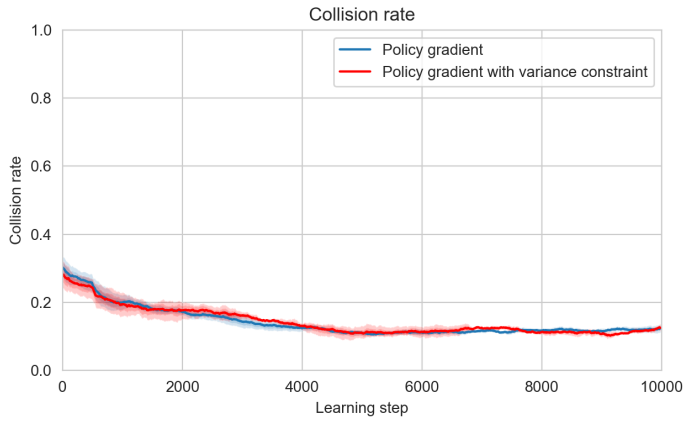
1. Return distribution
2. Percentage of episodes ended with an accident (Collision rate)
3. Variance of the return

Event	e_1	e_2	e_3	e_4	e_5	e_6	e_7
Reward	-10	1.0	-1.0	0.5	-0.5	-1.0	0

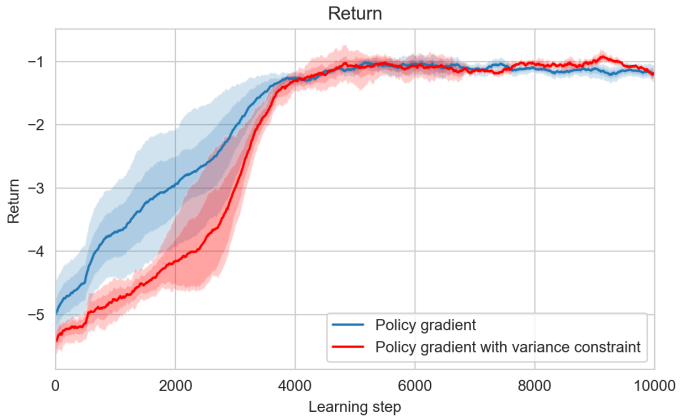
Table 5.6: Reward function used for Policy Gradient algorithm with variance constraint

Table 5.6 describes the reward function used during experiments. The reward function has the same structure as the one used in Section 5.3. The policy function is represented by a 3-layer neural network with leaky ReLU activation function. The network takes as an input a state represented by the flattened game frame and outputs a categorical distribution with probabilities of specific actions. The learning was limited to 10,000 steps.

In this experiment, the Variance threshold b is a hyper-parameter which was tuned using grid-search technique.



(a) : Evolution of collision rate



(b) : Evolution of return value

Figure 5.8: Evolution of performance metrics during learning

5.4 Experiment results

Figures 5.8 and 5.9 show the evolution of performance metrics while learning. For each learning step the graphs contain the expected value and corresponding 1σ and 2σ confidence intervals for metric distribution. The expected value and confidence intervals were computed from the results of several runs of the algorithms (8 for unconstrained and 4 for constrained versions).

From the graph in Figure 5.9, we can see that policy iteration algorithm with variance constraint reduces the variance of return during the first part of learning (up till the ∼3,000th step) and converges to the same value after the 4,000th step.

Graphs in Figures 5.8a and 5.8b show the evolution of the collision rate and return while learning. The collision rate of both algorithms is effectively identical during the whole learning. The return, on the contrary, is lower by

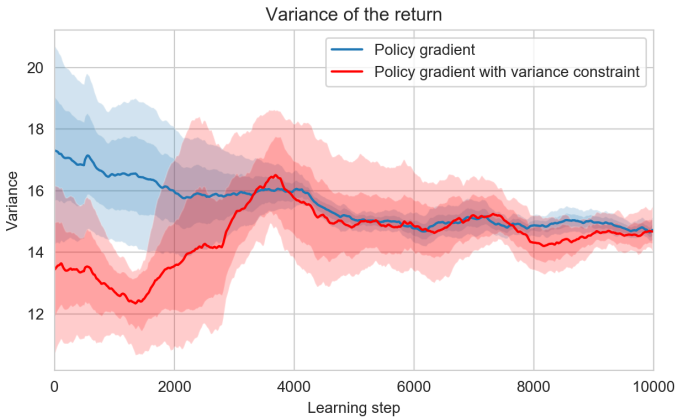


Figure 5.9: Evolution of return variance

about 0.88 points in average for the constrained algorithm during the first part of learning. From these two facts, we can conclude that the constrained algorithm prefers more passive policies. It overtakes less frequently, which in theory should lead to a safer policy, but due to the lack of experience, its policy still has the same collision rate. The results of the experiment point out that the return variance might not be the best risk metric for this task since its minimization does not affect the collision rate.

Chapter 6

Discussion

This chapter discusses key findings emerged from the results of experiments described in Chapter 5 and ideas for future work.

6.1 Experiment results: key findings

For the purpose of policy initialization, two algorithms were presented: Behavioral Cloning (BC) and Generative Adversarial Imitation Learning (GAIL). Both algorithms performed better than the baseline random policy. Policy generated by BC algorithm performed better and learned faster than the one generated by GAIL. However, it had a stronger tendency to overfit on the training set. Figure 5.5 in Section 4.3 show a high similarity between the training set policy and the one learned with BC. The loss function for BC algorithm was weighted to address an unbalanced training dataset.

The issue of overfitting and the unbalanced training set should have been addressed by GAIL framework. The training, however, resulted in a policy with a reduced action set (some were omitted by the policy, e.g. "turn right" action). The reasoning behind this behavior is discussed in Section 5.2.3. Generally, we can draw a conclusion, that GAIL framework needs a greater amount of data in comparison to BC, and additionally, for successful learning, the training set must cover state space densely. GAIL framework suits best for the environments with a specific starting state (or a finite set) and lower stochasticity due to the computationally demanding process of occupancy measure computation.

The risk-aware Q-learning algorithm described in Section 4.4.3 reduces the risk through modification of the reward function. The results of the experiments showed that risk-aware version of Q-learning reduced collision rate and improved the return value during the first part of the learning and achieved same performance in comparison with the original version of the algorithm. It proves that risk-aware Q-learning does not suffer from reduced exploration.

A major downside of Q-learning with risk-directed exploration is that it demands a model of environment. For larger environments, it is practically impossible to construct an explicit transition dictionary, even though dimensionality reduction and approximation techniques can be used.

The Policy Gradient algorithm with variance constraint described in Section 5.1 optimizes policy directly minimizing the variance of the return. The result of the experiment showed that variance minimization does not affect the collision rate, which indicates that variance is not the best choice of the risk metric for the autonomous driving problem. The variance minimization reduces exploration at the beginning of the learning process, which results in lower return. However, constrained algorithm gradually improves its policy and achieves the same results at the end of the learning.

6.2 Future work

Described algorithms can be used in real problems to make reinforcement learning safer and potentially reduce costs of learning. Those algorithms which interact with the environment during learning should be used in combination with a built-in rule-based safety system to ensure elimination of collisions and agent destruction during learning. Signals obtained from the internal safety system may be used as a part of reward function together with other reinforcements.

The deep reinforcement learning is a very flexible and powerful framework for automatic control. Presented algorithms reduce risk originating from the stochasticity of the environment and need to explore previously unvisited regions. However, the nature of the algorithm brings up a safety vulnerability by itself, namely an inability to interpret its actions. The issue can be addressed with *safety cases* [42]. In a safety case, we provide a set of environment's conditions under which a specific output of the control algorithm is desired and then test if the learned problem model ensures this output under *all considered environment conditions*. Ehlers [8] developed an algorithm for formal verification of a neural network with piece-wise linear activation functions. This approach enables to test network decisions against a set of safety rules.

Another interesting area of research not discussed in this work is the design of the reward function to prevent *reward hacking* and avoid *negative side effects*, the problems mentioned in Section 4.2. The reward function is strongly related to safety and reward hacking is an issue the designer should think about. Amodei, Olah et al. [2] provide an overview of algorithms and approaches to tackle this problem. One of the most interesting research direction of reward function design is *empowerment*. The empowerment is defined as the maximum possible mutual information between the agent's potential future actions and its potential future state (or equivalently, the Shannon capacity

of the channel between the agent's actions and the environment) [20]. The empowerment is often maximized (rather than minimized) as a source of intrinsic reward. It is possible to combine two sources of reinforcement in the reward function, or to optimize the policy with respect to the empowerment.

The safe reinforcement learning is impossible without *prior knowledge* of the environment or building an *internal belief* about it. Recent applications of the actor-learner algorithm to multiplayer games include inference of environment state by Recurrent Neural Network [18]. In this setting, reinforcement learning solves the control problem in the Partially Observable MDP, where states are not observed directly. This enables to incorporate another source of risk into the problem.

For truly reliable and safe RL algorithms it is essential to approach *all sources of potential risk* on different levels starting from signal acquisition to decision making under uncertainty. Without a doubt, the field of safe RL has a significant amount of interesting open problems which are being or will be addressed by researchers in the future.

Chapter 7

Conclusion

This work explores methods of risk reduction in reinforcement learning (RL) applied to an autonomous driving environment. Two risk reduction approaches were studied closer, namely: policy initialization techniques and risk-aware reinforcement learning.

For the purpose of policy initialization, two algorithms were implemented and compared: Behavioral Cloning (BC) and Generative Adversarial Imitation Learning (GAIL).

The experiment results showed that both policy initialization algorithms performed better than the baseline random policy. The policy generated by BC algorithm showed significantly better results in comparison to GAIL policy: it had a higher expected value of the return and a lower collision rate. However, the policy learned with GAIL framework had a slightly higher expected value of the return per step. It implies that GAIL policy performed slightly better for unseen states at the beginning of an episode but failed to maintain its performance afterward. The results indicate that both algorithms can provide a sensible initial policy, which can be used during subsequent learning.

Within the scope of risk-aware RL, two algorithms were described and implemented: Q-learning with risk-directed exploration and Policy Gradient algorithm with variance constraint.

The results of the experiments demonstrated that Q-learning with risk-directed exploration had a lower collision rate and higher return during the first part of the learning in comparison with both model-free and model-based versions of Q-learning. At the end of the learning, it achieved equal performance with both algorithms. These results prove that risk-aware Q-learning decreases the risk during exploration.

The policy Gradient algorithm with variance constraint reduced the variance of return during the first part of the learning process and converged to the same values of collision rate and achieved return compared to the unconstrained algorithm. The experiment results implicate that variance might not be the

best choice of risk metric for the particular autonomous driving problem since its minimization did not affect the collision rate.

The main contributions of this thesis are: (1) openly available implementation of selected policy initialization and risk-aware RL algorithms, (2) improvement of Q-learning with risk-directed exploration by adding a neural network function approximator and additional modifiable parameters.

Achieved results show that some of described risk-aware reinforcement learning methods are capable of reducing risk and providing similar or the same results in terms of return and collision rate after the learning phase.

Bibliography

[1] Abadi, Martín et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] Amodei, Dario et al. "Concrete Problems in AI Safety". In: *CoRR* abs/1606.06565 (2016).

[3] Artzner, Philippe et al. "Coherent Measures of Risk". In: *Mathematical Finance* 9.3 (1999), pp. 203–228.

[4] Boyan, Justin A. and Moore, Andrew W. "Generalization in Reinforcement Learning: Safely Approximating the Value Function". In: *Advances in Neural Information Processing Systems 7*. Ed. by G. Tesauro, D. S. Touretzky, and T. K. Leen. MIT Press, 1995, pp. 369–376. URL: http://papers.nips.cc/paper/1018-generalization-in-reinforcement-learning-safely-approximating-the-value-function.pdf.

[5] Brafman, Ronen I. and Tennenholtz, Moshe. "R-max - a General Polynomial Time Algorithm for Near-optimal Reinforcement Learning". In: *J. Mach. Learn. Res.* 3 (Mar. 2003), pp. 213–231. ISSN: 1532-4435. DOI: 10.1162/153244303765208377. URL: https://doi.org/10.1162/153244303765208377.

[6] Castro, Dotan Di, Tamar, Aviv, and Mannor, Shie. "Policy Gradients with Variance Related Risk Criteria". In: (June 2012). eprint: 1206.6404. URL: https://arxiv.org/pdf/1206.6404.

[7] Durrant-Whyte, Hugh and Bailey, Tim. "Simultaneous localization and mapping: part I". In: *IEEE Robotics Automation Magazine* 13.2 (2006), pp. 99–110. ISSN: 1070-9932. DOI: 10.1109/MRA.2006.1638022.

[8] Ehlers, Rüdiger. "Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks". In: *ATVA*. 2017.

[9] Fridman, Lex. *Deep Learning for Self-Driving Cars Lecture Notes*. 2018. URL: https://selfdrivingcars.mit.edu/.

[10] Fu, Justin, Luo, Katie, and Levine, Sergey. "Learning Robust Rewards with Adversarial Inverse Reinforcement Learning". In: (Oct. 2017). eprint: 1710.11248. URL: https://arxiv.org/pdf/1710.11248.

[11] Fulton, Nathan and Platzer, André. *Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning*. 2018. URL: https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17376.

[12] García, Javier and Fernández, Fernando. "A Comprehensive Survey on Safe Reinforcement Learning". In: *J. Mach. Learn. Res.* 16.1 (Jan. 2015), pp. 1437–1480. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2789272.2886795.

[13] García, Javier and Fernández, Fernando. "Safe Exploration of State and Action Spaces in Reinforcement Learning". In: *CoRR* abs/1402.0560 (2014). arXiv: 1402.0560. URL: http://arxiv.org/abs/1402.0560.

[14] Girardin, Guillaume et al. *Sensors and Data Management for Autonomous Vehicles*. Tech. rep. Yole Development, 2015.

[15] Grześ, Marek and Kudenko, Daniel. "Improving Optimistic Exploration in Model-free Reinforcement Learning". In: *Proceedings of the 9th International Conference on Adaptive and Natural Computing Algorithms*. ICANNGA'09. Kuopio, Finland: Springer-Verlag, 2009, pp. 360–369. ISBN: 3-642-04920-6, 978-3-642-04920-0. URL: http://dl.acm.org/citation.cfm?id=1813739.1813779.

[16] Hans, Alexander et al. "Safe exploration for reinforcement learning." In: *ESANN*. 2008, pp. 143–148.

[17] Ho, Jonathan and Ermon, Stefano. "Generative Adversarial Imitation Learning". In: (June 2016). eprint: 1606.03476. URL: https://arxiv.org/pdf/1606.03476.

[18] Jaderberg, Max et al. "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning". In: *CoRR* abs/1807.01281 (2018).

[19] Kakade, Sham and Langford, John. "Approximately Optimal Approximate Reinforcement Learning". In: *Proceedings of the Nineteenth International Conference on Machine Learning*. ICML '02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 267–274. ISBN: 1-55860-873-7. URL: http://dl.acm.org/citation.cfm?id=645531.656005.

[20] Klyubin, Alexander, Polani, Daniel, and Nehaniv, Chrystopher. "Empowerment: a universal agent-centric measure of control". In: *2005 IEEE Congress on Evolutionary Computation*. Vol. 1. 2005, 128–135 Vol.1. DOI: 10.1109/CEC.2005.1554676.

[21] Knight, Frank H. *Risk, Uncertainty, and Profit: Chapter 1: The Place of Profit and Uncertainty in Economic Theory - Scholar's Choice Edition*. Bibliolife DBA of Bibilio Bazaar II LLC, 2015. ISBN: 9781296048273. URL: https://books.google.cz/books?id=qL9ZrgEACAAJ.

[22] Kober, Jens, Bagnell, J. Andrew, and Peters, Jan. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274. ISSN: 1741-3176. DOI: 10.1177/0278364913495721. URL: http://dx.doi.org/10.1177/0278364913495721.

[23] Lillicrap, Timothy P. et al. "Continuous control with deep reinforcement learning". In: *CoRR* abs/1509.02971 (2015).

[24] L.M. Law, Edith. "Risk-directed Exploration in Reinforcement Learning". MA thesis. Montreal, Quebec: McGill University, Feb. 2005.

[25] Lucas, Manuelli and Pete, Florence. *Reinforcement Learning for Autonomous Driving Obstacle Avoidance using LIDAR*. Tech. rep. Massachusetts Institute of Technology, 2015.

[26] Majumdar, Anirudha and Pavone, Marco. "How Should a Robot Assess Risk? Towards an Axiomatic Theory of Risk in Robotics". In: *CoRR* abs/1710.11040 (2017). arXiv: 1710.11040. URL: http://arxiv.org/abs/1710.11040.

[27] McInnes, Leland and Healy, John. "Umap: Uniform manifold approximation and projection for dimension reduction". In: *arXiv preprint arXiv:1802.03426* (2018).

[28] McNeil, Alexander J., Embrechts, Paul, and Frey, Rudiger. *Quantitative risk management : concepts, techniques and tools*. Princeton, NJ: Princeton University Press, 2015. ISBN: 978-0691166278.

[29] Mnih, Volodymyr et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[30] Moldovan, Teodor Mihai and Abbeel, Pieter. "Safe Exploration in Markov Decision Processes". In: *CoRR* abs/1205.4810 (2012). arXiv: 1205.4810. URL: http://arxiv.org/abs/1205.4810.

[31] Paromtchik, Igor and Laugier, Christian. "Motion generation and control for parking an autonomous vehicle". In: *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*. Vol. 4. IEEE. 1996, pp. 3117–3122.

[32] Pecka, Martin and Svoboda, Tomás. "Safe Exploration Techniques for Reinforcement Learning - An Overview". In: *MESAS*. 2014.

[33] Pomerleau, Dean A. "ALVINN: An Autonomous Land Vehicle in a Neural Network". In: *Advances in Neural Information Processing Systems 1*. Ed. by D. S. Touretzky. Morgan-Kaufmann, 1989, pp. 305–313. URL: http://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network.pdf.

[34] Rockafellar, R. Tyrrell and Uryasev, Stanislav. "Optimization of Conditional Value-at-Risk". In: *Journal of Risk* 2 (2000), pp. 21–41.

[35] Ross, Stephane and Bagnell, Drew. "Efficient Reductions for Imitation Learning". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010, pp. 661–668. URL: http://proceedings.mlr.press/v9/ross10a.html.

[36] Rummery, Gavin A. and Niranjan, Mahesan. *On-Line Q-Learning Using Connectionist Systems*. Tech. rep. Engineering Department, Cambridge University, 1994.

[37] Sallab, Ahmad El et al. "Deep Reinforcement Learning framework for Autonomous Driving". In: (Apr. 2017). eprint: 1704.02532. URL: https://arxiv.org/pdf/1704.02532.

[38] Sammut, Claude et al. "Learning to Fly". In: *Proceedings of the Ninth International Conference on Machine Learning*. Ed. by Morgan Kaufmann. 1992, pp. 385–393.

[39] Shinners, Pete et al. *PyGame*. http://pygame.org/. 2011.

[40] Sutton, Richard S. and Barto, Andrew G. *Reinforcement Learning: An Introduction*. MIT Press, 2018. ISBN: 9780262039246.

[41] Vitelli, Matt and Nayebi, Aran. *CARMA: A Deep Reinforcement Learning Approach to Autonomous Driving*. Tech. rep. Stanford University, 2016.

[42] Wagner, Michael and Koopman, Philip. *A Philosophy for Developing Trust in Self- Driving Cars*. 2015.

[43] Watkins, Christopher John Cornish Hellaby. "Learning from Delayed Rewards". PhD thesis. Cambridge, UK: King's College, 1989. URL: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

[44] Williams, Ronald J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8.3 (1992), pp. 229–256. ISSN: 1573-0565. DOI: 10.1007/BF00992696. URL: https://doi.org/10.1007/BF00992696.

[45] Xiong, Xi et al. "Combining Deep Reinforcement Learning and Safety Based Control for Autonomous Driving". In: *arXiv preprint arXiv:1612.00147* (2016).

[46] Yang, Jiping and Qiu, Wanhua. "A measure of risk and a decision-making model based on expected utility and entropy". In: *European Journal of Operational Research* 164 (2005), pp. 792–799.

[47] Yu, April, Palefsky-Smith, Raphael, and Bedi, Rishi. "Deep reinforcement learning for simulated autonomous vehicle control". In: *Course Project Reports: Winter 2016 (CS231n: Convolutional Neural Networks for Visual Recognition)* (2016), pp. 1–7.

[48] Yu, Pengqian, Haskell, William B., and Xu, Huan. "Approximate Value Iteration for Risk-aware Markov Decision Processes". In: *CoRR* abs/1701.01290 (2017). arXiv: 1701.01290. URL: http://arxiv.org/abs/1701.01290.

[49] Zhan, Eric et al. "Generative Multi-Agent Behavioral Cloning". In: (Mar. 2018). eprint: 1803.07612. URL: https://arxiv.org/pdf/1803.07612.

[50] Ziebart, Brian D. et al. "Maximum entropy inverse reinforcement learning". English. In: *Proceedings of the National Conference on Artificial Intelligence*. Vol. 3. Dec. 2008, pp. 1433–1438. ISBN: 9781577353683.

Appendix A

CD contents

The enclosed CD contains a copy of this thesis and the source code of implemented algorithms. The scripts are located in the project directory **risk-aware-rl**. The project also contains an implementation of an autonomous driving environment in **MDP** subfolder.

```
/
├── thesis.pdf
└── risk-aware-rl ..... Project directory with the source code.
    ├── requirements.txt
    ├── README.txt
    ├── MDP ..... Autonomous driving environment.
    │   ├── environment.py
    │   ├── model.py
    │   └── reward.json
    ├── policy_initialization ..... Policy initialization algorithms.
    │   ├── BC_agent.py
    │   └── GAIL_agent.py
    ├── risk_aware_rl ..... Risk-aware RL algorithms.
    │   ├── policy_gradient_agent.py
    │   └── q_learning_agent.py
    └── utils ..... Utility scripts.
        ├── record_trajectories.py
        ├── replay_trajectory.py
        └── train_model.py
```