UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Anne Ott

Reinforcement Learning for Autonomous Navigation: A Case Study in Structured Environment

Master's Thesis (30 ECTS)

Supervisors: Amnir Hadachi, PhD.
Shan Wu, MSc

Tartu 2021

Reinforcement Learning for Autonomous Navigation: A Case Study in Structured Environment

Abstract

Over the past years, the field of autonomous driving has known immense progress. Solutions based on machine learning advancements have been used to speed up the development of highly automated driving. Significant progress has been made in solving complex problems using a machine learning's technique deep reinforcement learning. That field deals with agents navigating in an environment to maximize total reward by completing tasks. It has become a robust learning framework now capable of learning complicated behaviours in high dimensional environments. One of the essential aspects of reinforcement algorithms is to ensure safety, as any mistake can have life-threatening consequences.

This thesis aims to provide a standardized environment and an upgraded architecture for training reinforcement learning agents in a structured world. Algorithms based on Trust Region Policy Optimization and Proximal Policy Optimization are used for the training. The results are evaluated by how successfully the agent completes its goal and how well the safety of the agent and its surroundings is preserved. In addition, an evaluation of the proposed models is conducted by comparing them with baseline models.

Keywords:
autonomous driving, machine learning, reinforcement learning, safety

CERCS: P170 Computer science, numerical analysis, systems, control

Stiimulõpe iseseisvas navigeerimises: juhtumiuuring struktureeritud keskkonnas

Lühikokkuvõte: Viimastel aastatel on isejuhtivate sõidukite valdkonnas toimunud suur edasiminek. Kiirendamaks arengut kõrgelt automatiseeritud sõidukite vallas on suurt rolli mänginud masinõppe edusammudel põhinevad lahedused. Märkimisväärne progress keerukate probleemide lahendamisel on toimunud tänu stiimulõppe-nimelisele masinõppe harule. Stiimulõppe põhimõte on see, et keskkonda on paigutatud agent, kelle eesmärk on ümbritsevas keskkonnas tegutseda maksimeerides tegevustest tulenevat preemiat ja täites etteantud ülesandeid. Sellest on saanud võimas tööriist, mis suudab õppida keerukaid käitumismustreid kõrgemõõtmelistes keskkondades. Üks olulistest stiimulõppe aspektidest isejuhtivate sõidukite vallas on ohutus, sest vigadel on eluohtlikud tagajärjed.

Magistritöö eesmärk on pakkuda välja standardiseeritud keskkond ja täiendatud stiimulõppe arhitektuur, kus agendid saaksid õppida struktureeritud maailmas. Agentide treenimiseks kasutatakse algoritme, mis põhinevad Usalduspiirkonna käitumispoliitika optimeerimise algoritmil ja Lähima käitumispoliitika optimeerimise algoritmil. Tulemusi mõõdetakse nii selle järgi, kui edukalt agent täidab etteantud ülesannet, kui ka selle järgi, kui ohutult agent käitub. Lisaks võrreldakse välja pakutud mudeleid lähtemudelitega.

Võtmesõnad:
isejuhtivad sõidukid, masinõpe, stiimulõpe, ohutus

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Acknowledgements

I would like to express my deepest gratitude to my supervisor Dr. Amnir Hadachi for guidance and positive encouragement, and supervisor Shan Wu for technical advice and support. I am grateful for always having been given the right directions and ideas to pursue and help in solving all the issues faced. I would also like to thank Artjom Lind for providing help in technical setups.

I also would like to express my sincere appreciation to my boyfriend for unconditional support and my family for always having belief in my academic abilities.

Contents

1 Introduction

In the past decades, machine learning has gained substantial popularity. It has been helpful in various fields such as speech recognition, computer vision, data analytics, improved understanding of the human genome, etc. One of the key modern technologies that have emerged from machine learning applications is self-driving cars. Such highly automated technologies promise many potential benefits such as greater traffic efficiency, road safety, autonomous navigation, reduced congestion, and decreased carbon emissions. Machine learning algorithms combined with deep learning render autonomous vehicles capable of making decisions in real-time. Data is continuously gathered from the learner's immediate surroundings, and together with the knowledge of previous trips, a set of rules determine how best to proceed. Whether to turn left or right, push the brake or accelerate, algorithms must make decisions within a fraction of seconds. Autonomous driving applications are not limited to cars; some examples of other usages include self-driving robots for delivering food, groceries, or packages and moving around goods in warehouses.

In addition to offering broader access to mobility, autonomous driving can also help to reduce the number of driving-related accidents and crashes. An autonomous vehicle or a package robot is programmed to obey all the road rules; it will not exceed the speed limit. It can detect what humans cannot see, especially at night or in low-light conditions, and react quickly to avoid a collision. One of the most crucial tasks for autonomous vehicles is to plan their motion through traffic without harming themselves or other traffic participants. Therefore, machine learning models need to guarantee everyone's safety.

Since it would be too dangerous and inefficient to train an autonomous vehicle directly in real roads, an approach to safe exploration is to build driving-learning agents that gather knowledge in simulations where the desired behaviour can be learned. One of the machine learning theories relevant for teaching an autonomous driving agent is reinforcement learning (RL). It is a paradigm for learning by trial-and-error inspired by the way humans learn new tasks [1]. The main idea is that an agent learns to interact with the environment by exploring different actions and receiving the next state of the environment, reward and possible penalty. The agent analyses possible outcomes and makes a decision based on the best one and then learns from it.

1.1 Problem statement

To date, most of the existing success in RL has been by agents learning to play games like Atari and AlphaGo. Only in recent years, the application to real-world related problems has increased. However, still many existing environments contain games rather than structured environment problems. Also, much focus is on learning efficient behaviour rather than road safety.

RL contains a broad range of various algorithms. Nevertheless, there is a lack of standardized environments for applying those algorithms and advancing in the development of safe exploration algorithms. In literature, different papers use different simulation environments, different tasks (e.g. lane change, lane keeping, intersections) and evaluation procedures, making it difficult to compare those approaches. This problem is addressed by OpenAI who propose Safety Gym [2]: a set of tools to standardize safe exploration research. However, their proposed environments for a car like robots are very simplistic and have little resemblance to a structured situations.

1.2 Contributions

In this work, we explore RL in autonomous navigation by introducing new functionality to RL models and a structured environment. In order for the RL algorithm to learn to solve a given task, the model needs to be trained in an appropriate environment. Therefore, a new environment design is proposed to resemble a more structured situation where agents can learn to behave in a warehouse-like world. The proposed environment is built upon OpenAI Safety Gym. Another aim for this work is to propose an upgraded model that would lead to a more successful training of the agents. As the proposed environments by OpenAI are rather small and simplistic, the learning process needs to be modified in order for the agent to manage in bigger and more complex environments. This includes changing the reward design and increasing the feedback frequency to the agent. Furthermore, different reinforcement learning algorithms are investigated for finding the best that manage to complete its task and at the same time ensure safety. To our best knowledge, such implementation on Safety Gym has been not done before.

1.3 Road Map

The thesis is composed as follows:

Chapter 2 (Background): This chapter introduces the background of reinforcement learning and its safety-related issues. In addition, it gives an overview of related literature.

Chapter 3 (Methodology): This chapter describes the architecture design of the proposed model. Firstly, the details of the proposed environment are given. Secondly, details about improved model are explained. Lastly, an overview of mathematical background of policy-based reinforcement learning methods is given. This includes details about the algorithms Proximal Policy Optimization and Trust Region Policy Optimization.

Chapter 4 (Results and Analysis): This chapter reveals the results achieved using the proposed methodology. Comparisons between original and proposed models as well as different algorithms are provided. In addition, the description of the evaluation metrics and hyperparameter tuning is provided.

Chapter 5 (Discussions): This chapter shows the conclusion and discussion for the proposed methodology and presents future research perspectives.

2 Background Knowledge and Related Work

2.1 Reinforcement learning

Machine learning is a process whereby a computer program learns from experience to improve its performance at a specified task. Machine learning algorithms are often classified under three broad categories: supervised learning, unsupervised learning and reinforcement learning (RL). Supervised learning algorithms use labelled data for inductive inference like classification or regression, whereas unsupervised learning applies techniques such as clustering or density estimation to unlabelled data. By contrast, in the RL paradigm, an agent learns to improve its performance and choice of actions at a given task by interacting with the surrounding environment. RL agents are not told explicitly how to act; instead, a reward function gives feedback on an agent's performance. [3]

Reinforcement learning assumes an agent situated in an environment that the agent interacts with. At every step of interaction, the agent sees an observation of the state of the world and then decides on an action to take (Figure 1). The environment changes when the agent acts on it but may also vary on its own. The agent also perceives a reward signal from the environment, a number that tells it how good or bad the current world state is. The goal of the agent is to maximize its cumulative reward. The behaviours to complete given tasks are learned through reinforcement learning methods. [4]

RL problems quickly become computationally infeasible in high dimensional spaces and with the growth of task complexity. To solve this problem, RL is combined with deep learning to process sensor information directly, and this is called Deep Reinforcement Learning (DRL). DRL uses neural networks as function approximates for classical RL algorithms. This has led to a significant improvement in RL performance across many different tasks and environment. In this work, reinforcement learning and deep

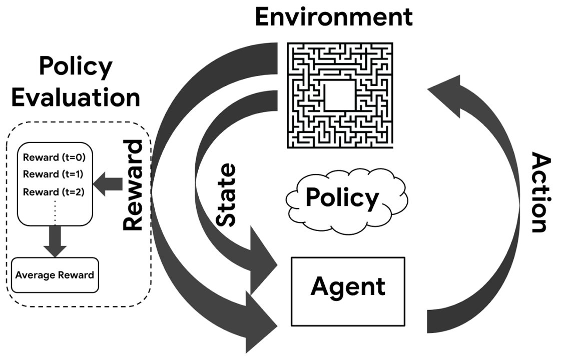


Figure 1. Agent-environment interaction in reinforcement learning [5]

reinforcement learning are used interchangeably because deep reinforcement learning is the major trend of reinforcement learning [6]. The popularity of DRL has increased in fields like complex locomotion, robotics and autonomous driving tasks. It enables scaling of so far intractable problems (e.g. high-dimensional state spaces) [7].

An essential aspect of RL is to have safe algorithms even while learning — like a self-driving car that can learn to avoid accidents without actually having to experience them. The safety issue is a consequence of the trial-and-error nature of RL: agents will sometimes try dangerous behaviours during the learning process [8]. One of the main challenges in reinforcement learning is managing the trade-off between exploration and exploitation. To maximize the cumulative reward, an agent must take advantage of its gathered knowledge by selecting actions known to give high rewards. On the other hand, to discover such beneficial actions, an agent has to take the risk of trying new actions, which may either lead to higher rewards or dangerous situations. In other words, the agent has to exploit what it already knows to obtain higher rewards. At the same time, it has to explore the unknown to discover new beneficial actions for the future. [3]

2.2 Different types of RL algorithms

Mainly, there are three general categories of RL algorithms

1. Value-based methods: Q-learning based approaches such as Deep Q-Networks (DQN), double DQN
2. Policy-based methods: Trust Region Policy Optimization (TRPO), Proximal Policy Optimization (PPO), Deterministic Policy Gradient (DPG) and Constrained Policy Optimization (CPO)
3. Actor-critic methods: Asynchronous Advantage Actor-Critic (A3C), Deep Deterministic Policy Gradient (DDPG), SoftActor Critic (SAC) and Twin Delayed Deep Deterministic Policy Gradients (TD3), Deep Deterministic Actor-Critic (DDAC).

Q-learning is one of the most commonly used value-based RL algorithms. It uses Q-values to iteratively improve the behaviour of the learning agent. Q-values are defined for states and actions, estimating how good is it to take certain action at the given state. An agent starts from a start state and makes a number of transitions from its current state to the next state. At every step of the transition, the agent takes an action, observes a reward from the environment, and then transits to another state. If the agent ends up in one of the terminating states at any point in time then there are no more possible tranisitions. At every time step Q-value is updated using the Bellman equation. The action is chosen by ϵ-greedy policy. With probability $(1 - \epsilon)$, the action with the highest Q-value is chosen and with probability ϵ any random action is chosen.

Policy Gradient methods are also popular in RL. At each state probability is calculated to determine the action which maximizes the reward. The basic principle uses gradient ascent to follow policies with the steepest increase in rewards (more detailed explanation in section 3.5.1). The idea of using gradient estimates in RL came from Williams in 1992 [9], who proposed the REINFORCE algorithm. Later in 1999, Sutton et al. [10] came up with an idea how to approximate policy function separately from value function and update it according to the gradient of expected reward. They were the first to prove that such policy based function approximations will converge to locally optimal policies. In recent years, the two most outstanding policy-based algorithms have been Trust Region Policy Optimization (TRPO [11]) and Proximal Policy Optimization (PPO [11]) developed by the researches in OpenAI. Mathematics behind the algorithms is discussed in Methodology section 3.4.

TRPO works by preventing the updated policies from deviating too much from previous policies, thus reducing the chance of a bad update. The basic idea is to limit each policy gradient update as measured by the Kullback-Leibler (KL) divergence between the current and the new proposed policy. This method results in monotonic improvements in policy performance. PPO proposed a clipped surrogate objective function by adding a penalty for having a too large policy change. Accordingly, PPO policy optimisation is simpler to implement and has a better sample complexity while ensuring the deviation from the previous the policy is relatively small. The requirement for only needing to calculate first-order gradient also makes PPO easier to implement than TRPO. [3], [7]

The difference between value-based and policy-based methods is essentially a matter of where the burden of optimality resides. Both method types must propose actions and evaluate the following behaviour. Value-based methods focus on evaluating the optimal cumulative reward and have a policy that follows the recommendations, policy-based methods estimate the optimal policy directly, and the value calculation is not necessary [3].

Actor-critic methods are hybrid methods that combine the benefits of policy-based and value-based algorithms. The policy structure that is responsible for selecting actions is known as the actor. The estimated value function criticises the actions made by the actor and is known as the critic. After each action selection, the critic evaluates the new state to determine whether the result of the selected action was better or worse than expected.

This thesis focuses on algorithms Proximal Policy Optimization and Trust Region Policy Optimization, introduced by Schulman et al. [11], [12]. Those are the most notable policy based methods used in DRL. There are a few reasons behind the choice of the algorithms. In autonomous driving, action spaces are continuous. For example, steering can vary from $-90°$ to $90°$ and acceleration can vary from 0 to 100km. This continuous action space will lead to poor performance for value-based methods, because maximum expected future reward is assigned for each possible action at each state. However, in the case of autonomous navigation this leads to an infinite number of possible actions [13]. Furthermore, with policy-based algorithms such as PPO, convergence to an optimal policy is better. When it comes to actor-critic methods, then although they have achieved incredible performance on RL problems such as games, they are likely to be influenced by frequent interaction between the actor and critic and therefore cause instability in the learning process. If a false step is taken at any point, it might have a big impact on the following steps, thus destabilizing the learning [14].

2.3 RL in autonomous driving

Deep RL applied to the self-driving domain is an emergent field. Researchers have tried different RL approaches for autonomous driving using all three categories of RL.

Q-learning has been applied for a variety of tasks. DQN has been proposed to finding an efficient strategy to navigate safely through intersections [15]. Furthermore, DQN has successfully applied [16] for training the agent in a simulator and then transferring it to a real world robot in small city. The agent manages to follow the lane based on a monocular camera input. A combination of Q-learning (DQN) and Actor-Critic (DDAC) has been applied by Sallab et al. [17] to train a race car for complex manoeuvres and simple interactions with other vehicles. The agent successfully learned to keep the lane in challenging curvatures. Similarly, Wang et al. [13] have applied Actor-Critic method

(DDPG) for the same race car. The agent learned to drive fast on straight roads, release the accelerator in curves and preserve safety during the whole journey. Overall, The Open Racing Car Simulator (TORCS) is widely used in autonomous driving experiments [13], [17], [18].

Ye et al. [19] have used a policy-based method (PPO) to create an automated lane change strategy. The effectiveness of the proposed policy is validated by using metrics of task success rate and collision rate. The simulation results demonstrate the lane change manoeuvres can be efficiently learned and executed in a safe, smooth and efficient manner. In addition, PPO has been successfully used for controlling a real vehicle in a parking lot [20], including turning manoeuvres and obstacle avoidance.

Tran et al. [21] have conducted an autonomous vehicle experiment at a non-signalized intersection, which uses PPO (with adaptive KL penalty) and deep RL. Different leading cars (car closest to the intersections) were evaluated. The result showed that autonomous vehicles as leading cars on the intersection outperformed leading human-driven vehicles and all human-driven vehicles. This shows the potential of clearing traffic congestions and making the traffic flow smoother using RL.

All three categories of RL can also be combined together. For example, Chen et al. [22] have designed a framework for a complex urban scenario in autonomous driving using DDQN (value-based method), TD3 (policy-based method) and SAC (actor-critic method). They evaluated their task on challenging roundabouts where the proposed algorithm performed significantly better than the baseline.

2.4 Safety and constrained RL

According to the World Health Organization, every year, the lives of approximately 1.35 million people are cut short as a result of a road traffic crash. Between 20 and 50 million more people suffer non-fatal injuries, with many incurring a disability as a result of their injury [23]. Therefore, autonomous driving applications have a high requirement for safety, and guaranteeing the safety of DRL integrated autonomous driving system is of substantial importance. Deep neural network has been widely known as having poor interpretability. Its complex structure makes it challenging to predict when the agent does not manage to generate a safe policy. The solution proposed in the literature [7] is to combine learned policies with hard constraints. However, balancing between the learned optimal policy and the safety guarantee by hard constraints is non-trivial and requires intensive investigation in the future.

Shalev-Schwartz et al. [24] have also addressed the safety concern and how to ensure accident-free driving. The authors decompose the problem into a composition of a policy for *Desires* (which is to be learned) and trajectory planning with hard constraints (which is not learned). The goal of *Desires* is to enable the comfort of driving, while hard constraints guarantee the safety of driving. The desires are translated into a cost function over driving trajectories. Hard constraints are implemented by finding a trajectory that minimizes the aforementioned cost subject to hard constraints on functional safety.

Originally are both PPO and TRPO unconstraint, meaning that the reward function contains no information about the costs. On the contrary, for constraint algorithms everything about the agent's eventual behaviour is described by the reward function (including penalties). A big problem is that reward design is fundamentally hard. The challenge comes from picking trade-offs between completing objectives and receiving reward and at the same time satisfying safety requirements and not receiving penalties. This is where constrained RL is advantageous. It does not only have a reward function that the agent wants to maximize, but also an environments cost functions that the agent needs to constrain in the objective function. This ensures that agents satisfy safety requirements. Therefore, two versions of both algorithms are present - unconstrained (in this work called TRPO and PPO) and constrained (in this work called TRPO adaptive penalty and PPO with adaptive penalty).

Following the next example (from [8]), the importance of constraint RL can be understood. Consider a self-driving car as the agent. The agent is rewarded for driving from point A to point B as fast as possible. At the same time, we also want the driving behaviour to match traffic rules. In normal RL, the penalty for collision is fixed at the beginning of training and is kept fixed during the whole process. The problem arising is that if the reward for this journey is high enough, the agent may not care whether it gets in lots of collisions as long as it can still complete its trips. It may even bring more reward to drive recklessly and risk those collisions in order to get the reward. In contrast, in constrained RL, an acceptable collision penalty is chosen at the beginning of training, and it is adjusted until the agent meets that requirement. If the car is getting into too many accidents, the penalty is raised until that behaviour is no longer encouraged.

2.5 OpenAI

To maintain fast progress in RL research, it is important that existing works are easily comparable to be able to judge improvements offered by novel methods, which is emphasized by Henderson et al. [25]. RL results are usually difficult to reproduce and are very sensitive to hyperparameter choices. Furthermore, such choices are generally not reported in detail. Therefore, as pointed out by [3], it is essential to have a starting point where the most used algorithms are implemented, documented and well tested.

The algorithms and environments used in this thesis are based on OpenAI, more specifically, a project called Safety Gym. OpenAI describes Safety Gym as a suite of tools for developing AI that respects safety constraints while training and for comparing the "safety" of algorithms and the extent to which those algorithms avoid mistakes while learning [26].

Safety Gym consists of two components, the first one is a module for building new environments (code available in GitHub [2]) and the second one is a benchmark environment to standardize the measurement of the progress (code available in GitHub [27]). Safety Gym introduces environments that require AI agents (Point, Car, Doggo) to navigate cluttered environments to achieve a task (reaching a goal, pressing a button or pushing an object). There are two levels of difficulty, and each time an agent performs an unsafe action, the agent receives a penalty.

3 Methodology

This chapter gives an overview of the upgraded model for autonomous navigation. The overall architecture of the proposed model is presented in Figure 2, where orange denotes improvements of the baseline and green completely new additions. Each following subsection in this chapter will give a detailed overview of those parts. Firstly, the design of the base environment where the agent is situated and its objects are described. Secondly, formulas for reward and cost are explained for different scenarios. Lastly, the mathematical background of PPO, TRPO and their variations is shown. Hyperparameter tuning is presented under chapter 5.

3.1 Base environment

This paper is based on an OpenAI project called Safety Gym [8]. Safety Gym is described as a suite of tools for developing AI that respects safety constraints while training and for comparing the "safety" of algorithms and the extent to which those algorithms avoid mistakes while learning. It consists of two components, the first one is a module for building new environments, and the second one is a benchmark environment to standardize the measurement of the progress.

The aim of the Safety Gym is for an agent to navigate through a cluttered environment to accomplish a task. At the same time, an agent must respect constraints on how it interacts with objects and areas around it. The initial Safety Gym environment supports three tasks - goal, button and push and three agents - point, dog and vehicle. In this paper, the focus is on autonomous navigation and therefore, the agent used is a car. In driving situations, there is usually a defined goal that the car wants to reach. Hence, the chosen task is the goal - the agent must navigate to the destination.

In the original Safety Gym environment, there are different possible physical objects (e.g. boxes, cylinders) that can be used for creating new environments. The proposed

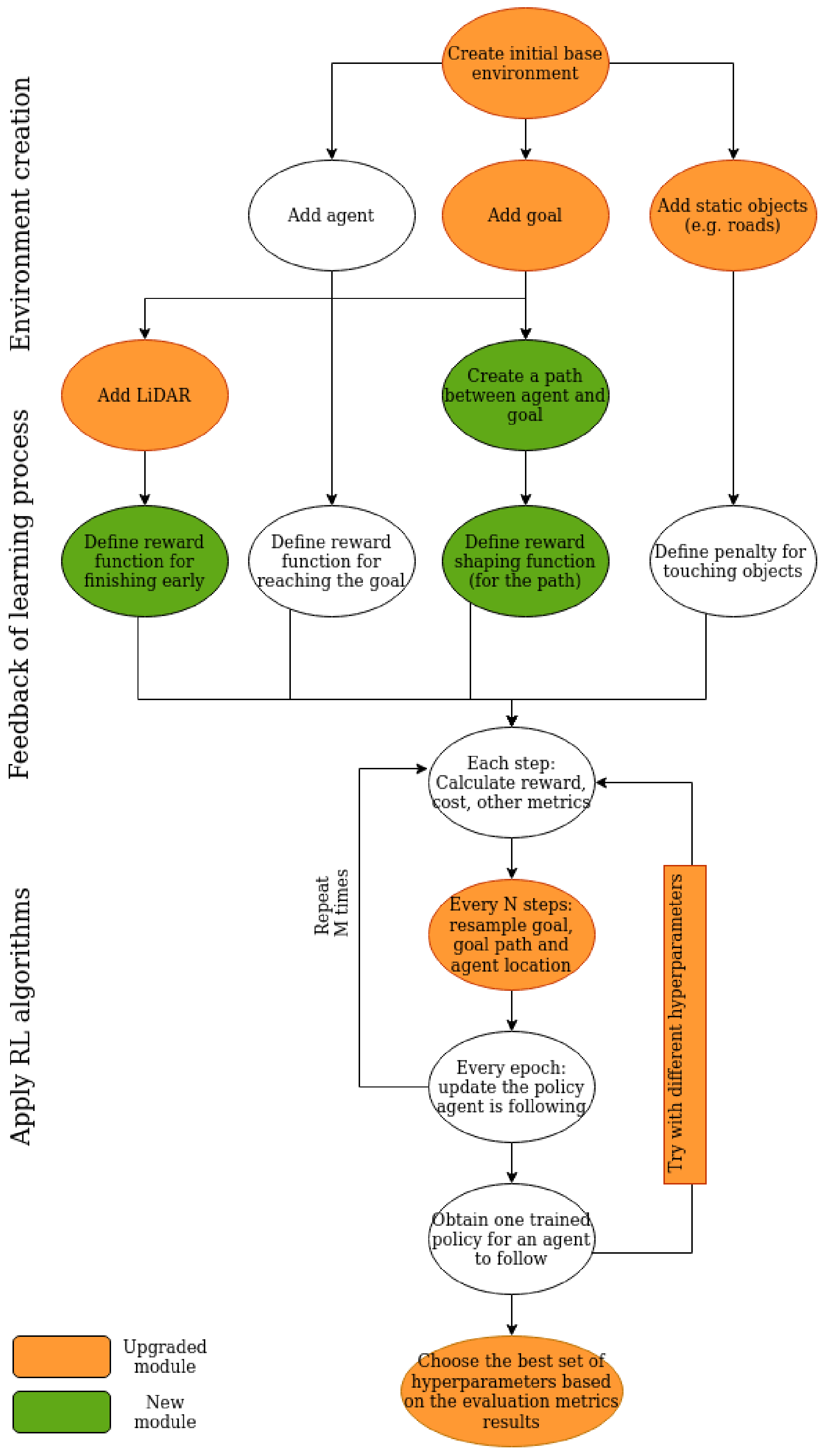

Environment creation
Feedback of learning process
Apply RL algorithms
Create initial base environment
Add agent
Add goal
Add static objects (e.g. roads)
Add LiDAR
Create a path between agent and goal
Define reward function for finishing early
Define reward function for reaching the goal
Define reward shaping function (for the path)
Define penalty for touching objects
Each step: Calculate reward, cost, other metrics
Repeat M times
Every N steps: resample goal, goal path and agent location
Every epoch: update the policy agent is following
Try with different hyperparameters
Obtain one trained policy for an agent to follow
Choose the best set of hyperparameters based on the evaluation metrics results
Upgraded module
New module

Figure 2. Architecture of the proposed model

environment (Figure 3) is built on top of the initial one using those basic object with the aim to resemble a more structured navigation situation. There are four types of objects

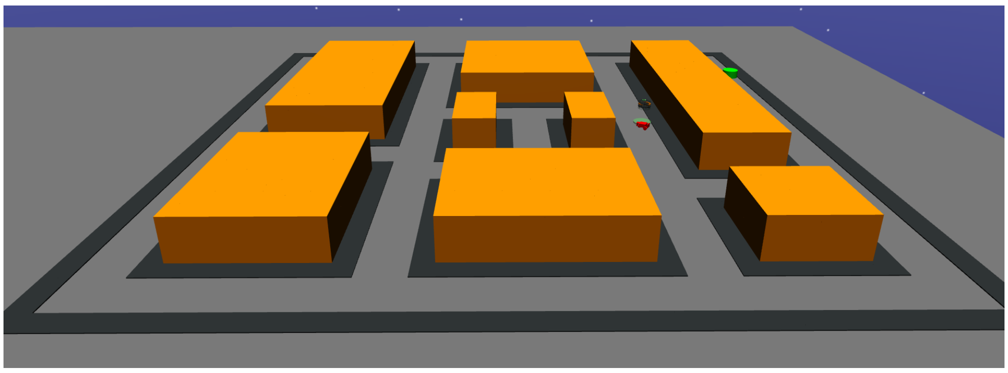


Figure 3. Example of proposed environment

in the new environment. Firstly, immobile objects called pillars are used. These yellow objects are rigid barriers, and the agent is penalized for touching them. They are designed as cubical areas over the environment. Secondly, there are hazards that are dangerous areas to avoid. These dark-grey rectangle planes on the ground are non-physical object, and the agent is penalized for entering and moving on them. Hazardous areas represent the sidewalks, where theoretically cars can drive; however, it is discouraged. Hazards are surrounding all the pillar blocks. Thirdly, the most important part is the goal that the agent navigates to and gets rewarded for such behaviour. This is designed as a green cylinder that can be entered. Lastly, there are green circles in the environment representing optimal route guidance. Those objects can only be placed on the road (light grey areas). At the beginning of the learning process, only one circle of the guidance path leading to the goal is present. If the agent manages to navigate there and touch the circle, then the next one appears and so on. The agent is rewarded for reaching the goal path objects. In addition to this, every next object gives more reward than the previous. Most of the sizes and colours of the objects are changed from the baseline environment. Agent's start location and goal position are chosen randomly at every training episode.

The perception of the surrounding environment is ensured by LiDAR sensor. LiDAR observations are represented visually by "lidar halos" that hovers above the agent (Figure 4). Each layer of the halo represents a different object. For easier comprehension, the layers are colour-coded.

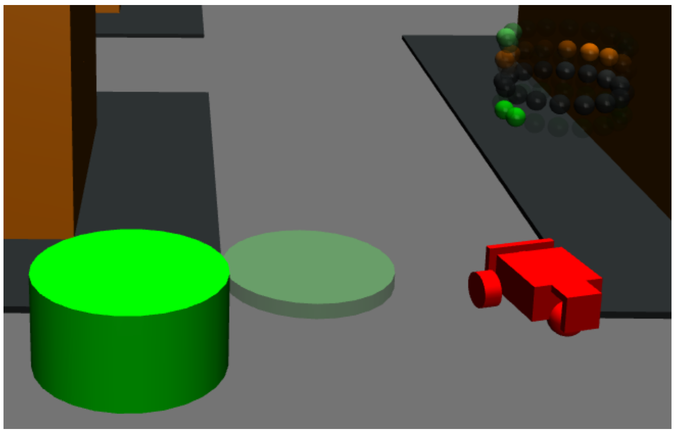


Figure 4. Lidar of the agent

3.2 Path algorithm

Since the path distances in the environment are not very long, a path between the agent's start location and goal location is found using a simple greedy algorithm (algorithm 1). It assumes that all the physical objects (sidewalks, etc) are already in place, and the coordinates are known. The algorithm works as follows. Firstly, it takes the start location and finds its neighbours. Then, it detects which of the neighbours is closest to the goal and checks if the neighbour is situated on the road and has not been visited before. If the conditions are not met, then it moves to the second closest neighbour and so on until the closest point that fits the criteria is found. This location is added to the path. If for some reason, the current location is a dead-end, $None$ is returned. The locations will be displayed to the agent one by one.

Algorithm 1: Path algorithm from start to goal

```
Input: agent's start location, goal location, road coordinates (excluding sidewalks) ;
path = [];
cur_loc = start_loc;
while not cur_loc = goal_loc do
    neigh = get_neighbours(cur_loc);
    next_loc = closest_point_to_goal(neigh);
    if not next_loc then
        return None
    end
    path.append(next_loc);
    cur_loc = next_loc
end
return path
```

3.3 Reward design

In order for the agent to learn the desired behaviour, an appropriate reward design is crucial. Reward function can consist out of many parts, meaning that it does not only include a reward for reaching the end goal. A method called reward shaping can be used to provide additional knowledge to an agent by the addition of an extra reward to the naturally received reward to improve learning speed. This is especially important in bigger environments, where the goal element is far away from the starting point. Reward shaping allows to provide more frequent feedback signal on desired behaviours. However, on many occasions, agents learn to interact with the environment in an unexpected way because the algorithm cleverly finds a new way that maximizes the reward without behaving in a preferred way. Such unanticipated behaviours are described in section 5.

In the proposed model, the agent is given three different types of reward:

1. reward shaping, which is implemented as a path leading to the end goal,
2. reaching the end goal,
3. finishing earlier than the given time.

Firstly, the reward given the whole training process depends on the distance between the agent and the goal/goal path object. Let m be the total number of goal path objects, including the end goal itself and $g_i = (x_{g_i}, y_{g_i}),\ \ i = 1, 2, ..., m$ be the location of the i-th goal path object in an Euclidean plane. Let $a = (x_a, y_a)$ be the location of the agent. The Euclidean distance between the agent's current position and goal path element i is defined as

$$d_i = d(a, g_i) = \sqrt{(x_a - x_{g_i})^2 + (y_a - y_{g_i})^2},\ \ i = 1, 2, ...m. \tag{1}$$

The reward at each interaction with the environment is calculated as follows

$$r_1 = c(d_{i-1} - d_i), \tag{2}$$

where c is a constant that scales the values (in our case $c = 0.8$). Taking the difference between current and previous distance outputs a positive value if the agent moved closer to the object and negative if it moved away. Therefore, this sparse reward encourages the agent to navigate towards the goal path object. Each time the agent reaches the goal path object ($a = g_i$) an additional reward r_{g_i} is given

$$r_2 = r_1 r_{g_i} \tag{3}$$

where r_{g_i} is an incremental reward for reaching i-th element. The goal path objects closer to the goal have higher reward, whereas the circles further away from goal have smaller reward meaning that $\forall i : r_{g_{i-1}} < r_{g_i}$. Without the incremental reward, the agent would not have the motivation to move along the path as it already gets enough reward from reaching the first path object. If the agent reaches the final goal (i.e. $a = g_m$), then an extra reward $r_{g_m} = 1$ is given. This makes sure that reaching the end goal is the main priority.

Besides, it is important to motivate the agent to move as quickly as possible to the final goal. Therefore, an additional function is designed to give extra reward once the agent reaches the goal. Let N be the number of steps given to the agent to complete one path and n the number of steps the agent actually needed. The early finish reward r_3 is calculated as

$$r_3 = c(N - n)l \tag{4}$$

where c is a constant for early reward (in our case 0.0001 because $N - n$ tends to be large), and l is the number of goal path objects in the whole path. It is important to consider the length because shorter paths take fewer steps to complete compared to longer paths.

3.4 Cost design

Cost design is more simplistic compared to reward design. The cost at each interaction with the environment is calculated only when the agent either touches or enters an undesired object (pillars and hazards). If any constraint is violated, then a red circle flashes around the agent to indicate that an accident happened.

Let $h_i = (x_{h_i}, y_{h_i})$ be the location of the i-th hazard object. The cost of staying inside a hazard is calculated as follows

$$C_{hazard} = \begin{cases} c(n - d(a, h_i)), & \text{if agent inside hazard } i \\ 0, & \text{if agent not inside hazard} \end{cases} \tag{5}$$

where $c = 1$ is a constant, $n = 0.25$ is the size of the hazard area, and $d(a, h_i)$ is the Euclidean distance between the agent and the centre of hazard. The cost is calculated when the agent is located inside the hazardous area. Moving towards the centre yields in higher cost, whereas moving away returns less cost. Hence, the agent is encouraged to leave the area.

The cost for pillar objects is easier to calculate, as there is no possibility to enter the object. At every interaction step the cost is either 0 or 1

$$C_{pillar} = \begin{cases} 1, & \text{agent touches the pillar} \\ 0, & \text{agent does not touch the pillar.} \end{cases} \tag{6}$$

At the end of the interaction step final cost is calculated $C = C_{hazard} + C_{pillar}$.

3.5 Mathematical Background of RL Algorithms

The following subchapter focuses on the part under the name "Apply RL algorithms" in Figure 2. It discusses in detail algorithms TRPO and PPO, their variations and mathematical background. In order to understand the algorithms, basic concepts of majorize-maximization algorithm, trust region, natural policy gradient and conjugate gradient are explained first. The main inspiration came from two sources, the original Schulman et al. articles introducing PPO [12] and TRPO [11]. Additional information was gathered from Berkley University's lectures [28] given by Joshua Achiam who is a research scientist in OpenAI and one of the authors of Safety-Gym.

3.5.1 Policy Gradient Methods

Policy Gradient methods (PG) are frequently used algorithms in RL. In PG, a policy π is trained to act based on observations an agent observes in the environment. When an agent follows policy π, it generates a sequence of states, actions, and rewards called the trajectory. Namely, at time t an agent in state s_t takes an action a_t according to the policy π. Based on the behaviour a reward r_t is given.

Definition 1. *A policy is defined as the probability distribution of action a given a state s and denoted as*

$$\pi(A_t = a|S_t = s)$$
$$\forall A_t \in \mathcal{A}(s), S_t \in \mathcal{S}.$$

The objective is to maximize the expected reward while following a policy π. Let θ be a set of parameters (e.g. coefficients, weights, biases in a neural network); then, the policy can be parameterized as π_θ. The total reward for a given trajectory τ is denoted as $r(\tau)$.

Definition 2. *An objective of RL is to maximize the expected reward following a parametrized policy*

$$.J(\theta) = \mathbb{E}_{\pi_\theta}\big[r(\tau)\big]$$

Policy gradient algorithms try to solve the optimization problem for a policy

$$\max_\theta \; J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\Big[\sum_{t=0}^{\infty}\gamma^t r_t\Big] \tag{7}$$

by taking stochastic gradient ascent on the policy parameters θ, using the policy gradient g

$$g = \nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t\nabla_\theta \log\pi_\theta(a_t|s_t)A^{\pi_\theta}(s_t,a_t)\right], \tag{8}$$

where π_θ is a stochastic policy, $A^{\pi_\theta}(s_t, a_t)$ is an estimator of the advantage function, γ is the discount factor and s_t and a_t are state and action at timestep t, respectively. The intuition behind this formula is that we wish to increase the probability of actions that are better than average and decrease the probability of action worse than the average.

Discount factor $\gamma \in (0, 1)$ describes the present value of the future rewards because the agent cares more about rewards received sooner rather than later. In other words, it defines up to what extent future rewards influence the return in timestep t.

The estimator of the advantage function $A^\pi(s_t, a_t)$ estimates the value of the selected action at timestep t. In order to compute $A^\pi(s_t, a_t)$ two values are needed - discounted sum of rewards and baseline estimate. Discounted sum or rewards $Q^\pi(s,a)$ is the value of state action pair when policy π is followed

$$Q^\pi(s,a) = \mathbb{E}_{\tau\sim\pi_\theta}\Big[\sum_{t=0}^{\infty} r_t | s_0 = s, a_0 = a\Big].$$

The baseline estimate, also called value function $V^\pi(s)$ is the average return an agent gets if it starts in state s and then acts according to policy π for the rest of its life

$$V^\pi(s) = \mathbb{E}_{\tau\sim\pi_\theta}\Big[\sum_{t=0}^{\infty} r_t | s_0 = s\Big].$$

This value is frequently updated based on what the agent has experienced in the environment. Those two combined give

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s).$$

Advantage function value describes how much better was the action taken ($Q^\pi(s,a)$) compared to what would normally be expected ($V^\pi(s)$) to happen in the state s. A positive value indicates that the given action was better than average and the probability of taking a given action increases, the same logic applies to the negative value. In practice, the most used technique for computing good estimates of the advantage function is Generalized Advantage Estimation (GAE) described in the paper [29].

As stated before, a policy π_θ is optimized for the maximum expected discounted rewards. Nevertheless, few aspects decrease PG performance. PG computes the steepest ascent direction for the rewards and updates the policy towards that direction. However, an assumption for this method is that the surface is flat. When the surface has high curvature, then the algorithm can choose bad moves. While large steps lead to a disaster, too small steps make the model learn very slowly.

The second issue with PG is that finding a proper learning rate is complex because it is very sensitive to the surface. Therefore, PG suffers from convergence problems.

Thirdly, the whole trajectory is sampled for just one policy update and can not be updated on every timestep. Making an update every time step would be not sample efficient even for toy experiments. Therefore, it is too expensive for more complex simulations.

Theoretically, it is possible to ensure that any policy update always improves the expected rewards using Majorize-Maximization (MM) algorithm. The MM algorithm does it by iteratively maximizing a lower bound function and using it to approximate the expected reward. It guarantees that the objective function ($J(\theta)$) is non-decreasing. MM algorithm converges to the optimal policy.

3.5.2 Trust region

Two main optimization methods are trust region and line search. Gradient descent falls under the line search - first, the descending direction is determined and next, a step is taken towards that direction.

Trust region has a different principle as its main goal is to make sure the new policy and old policy are not too far from each other. First, the maximum step size that is wanted to be explored is determined. Secondly, an optimal point within this trust region is located. Let δ be the initial maximum step size. The following maximization problem is obtained

$$\begin{gathered} \max_{s \in \mathbb{R}^n} \ m_k(s) \\ s.t. \ ||s|| \leq \delta \end{gathered} \tag{9}$$

where m is an approximation to the original objective function. The aim is to find the optimal s for m inside the radius δ.

The radius δ of the can be changed during runtime according to the curvature of the surface. The trust region can be expanded when the divergence of the new and current policy is small and shrunk when the divergence is large.

3.5.3 Optimization problem

In RL, we want to optimize the new policy π'. The objective function that should be optimized is following

$$\text{maximize}_{\pi'} \;\; J(\pi') = \text{maximize}_{\pi'} \;\; J(\pi') - J(\pi), \tag{10}$$

where π' denotes the new policy, π the old policy, $J(\pi')$ is the expected reward for using the new policy and $J(\pi)$ is the expected reward for the old policy. The equation holds because $J(\pi)$ is a constant.

The relationship of expected returns can be described with relative policy performance identity:

$$\text{for any policies } \;\pi', \pi : \;\; J(\pi') - J(\pi) = \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi}(s_t, a_t) \right], \tag{11}$$

where γ is a discount factor $\gamma \in (0, 1)$, A^{π} is an advantage function, s is state and a is an action. The advantage function is used instead of expected reward, since it helps to decrease the variance of the estimation. If the baseline is independent of our policy parameter, then the usages of A^{π} results in the same optimal policy. Proof of equation 11 can be found in the original TRPO paper [11] appendix A.

As a next step, a lower bound function is needed for the MM algorithm. This function consists of two parts. The first part is a function $\mathcal{L}$ which is s the surrogate advantage, a measure of how policy π' performs relative to the old policy π using data from the old policy. $\mathcal{L}$ is defined using our objective function

$$\begin{aligned} \mathcal{L}_{\pi} \;\; (\pi') &= \frac{1}{1-\gamma} \mathbb{E}_{s \sim d^{\pi}, a \sim \pi} \;\; \left[\frac{\pi'(a|s)}{\pi(a|s)} A^{\pi}(s, a) \right] \\ &= \mathbb{E}_{\tau \sim \pi} \;\; \left[\sum_{t=0}^{\infty} \gamma^t \frac{\pi'(a_t|s_t)}{\pi(a_t|s_t)} A^{\pi}(s_t, a_t) \right] \end{aligned} \tag{12}$$

where d is the discounted future state distribution

$$d^{\pi}(s) = (1-\gamma) \sum_{t=0}^{\infty} P(s_t = s|\pi). \tag{13}$$

On the other hand, the relationship between the expected rewards can be rewritten as

$$\begin{aligned} J(\pi') - J(\pi) &\overset{eq.11}{=} \mathbb{E}_{\tau\sim\pi'}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi}(s_t,a_t)\right] \\ &= \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi'},a\sim\pi'}\ \left[A^{\pi}(s,a)\right] \\ &= \frac{1}{1-\gamma}\mathbb{E}_{s\sim \color{red}{d^{\pi'}},a\sim\pi}\ \left[\frac{\pi'(a|s)}{\pi(a|s)}A^{\pi}(s,a)\right] \end{aligned} \tag{14}$$

where geometric series rule is used for γ.

Note that equations 12 and 14 are almost identical, there is only one issue (marked in red) that the state distributions are different. However, if two policies are close, then it can be approximated $d^{\pi'} \approx d^{\pi}$, meaning that we can change $s \sim d^{\pi'}$ to $s \sim d^{\pi}$ and obtain $\mathcal{L}$

$$J(\pi') - J(\pi) \overset{eq.14}{\approx} \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi},a\sim\pi}\ \left[\frac{\pi'(a|s)}{\pi(a|s)}A^{\pi}(s,a)\right] \overset{eq.12}{=} \mathcal{L}_{\pi}\ (\pi') \tag{15}$$

The second part of the lower bound ensures the closeness of π' and π that we assumed in equation 15. It is expressed using Kullback-Leibler divergence and looks like the following $\sqrt{\mathbb{E}_{s\sim d^{\pi}}[D_{KL}(\pi'||\pi)[s]]}$. Namely, KL-divergence is a measure of the difference between two distributions of data p and q (Figure 5) and is defined as

$$D_{KL}(P||Q) = \sum_{x=1}^{N} P(x)\log\frac{P(x)}{Q(x)}.$$

KL divergebce between two policies is

$$D_{KL}(\pi'||\pi)[s] = \sum_{a\in\mathcal{A}} \pi'(a|s)\log\frac{\pi'(a|s)}{\pi(a|s)}. \tag{16}$$

Therefore, if policies are close in KL-divergence, then the approximation in the equation 15 is good.

Putting together parts one and two, the final lower bound is

$$\begin{gathered} J(\pi') - J(\pi) \geq \mathcal{L}_{\pi}\ (\pi') - C\sqrt{\mathbb{E}_{s\sim d^{\pi}}[D_{KL}(\pi'||\pi)[s]]}, \\ \text{where } C \propto \frac{\epsilon\gamma}{(1-\gamma)^2}. \end{gathered} \tag{17}$$

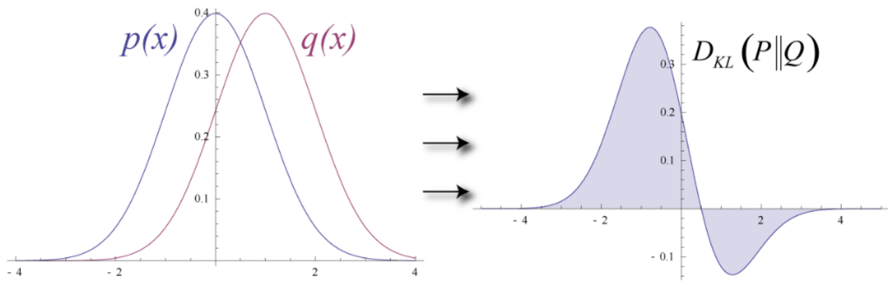


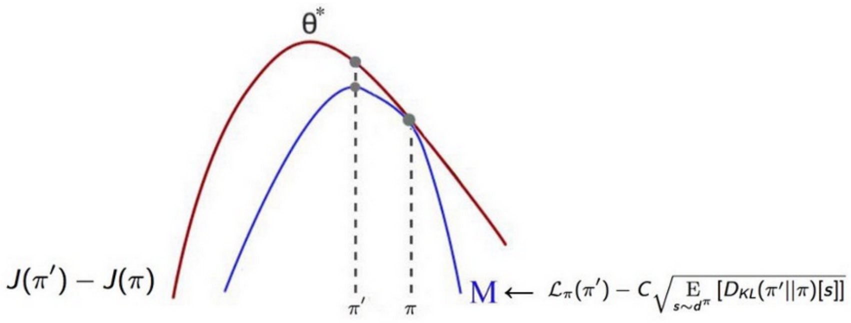
Figure 5. Illustration of the Kullback-Leibler divergence for two normal distributions [30]

More detailed proof for why the inequality holds is provided in the Schulman TRPO paper [11].

In summary, the final function that is maximized by the MM algorithm is

$$\text{maximize}\ \mathcal{L}_{\pi}\ \ (\pi') - C\sqrt{\mathbb{E}_{s\sim d^{\pi}}[D_{KL}(\pi'||\pi)[s]]}) \tag{18}$$

and is referred to as KL penalized. If we maximize this equation with respect to π', then we are guaranteed to monotonically improve π based on the majorize-maximize algorithm (Figure 6). Therefore, the new policy is always better than the old one.



θ^*
$J(\pi') - J(\pi)$
π'
π
$M \leftarrow \mathcal{L}_{\pi}(\pi') - C\sqrt{\mathop{\mathrm{E}}_{s\sim d^{\pi}}[D_{KL}(\pi'||\pi)[s]]}$

Figure 6. Lower bound function in MM [31]

In practice, if we used the penalty coefficient C recommended, then the step sizes would be very small. One possibility to increase the step size is to use a trust region constraint -

constrain the KL-divergence between the new and old policy. Using Lagrangian Duality, the equation 18 can be formulated using trust region constraint discussed in section 3.5.1

$$\begin{aligned} \text{maximize } & \mathcal{L}_{\pi}(\pi') \\ s.t. \; & \mathbb{E}_{s\sim d^{\pi}}\big[D_{KL}(\pi'||\pi)[s]\big] \leq \delta \end{aligned} \tag{19}$$

This is easier to solve than equation 18 and therefore is more preferred in algorithm's implementations.

Now the optimization problems are defined. However, solving them is a more complex task. TRPO algorithm consists out of two parts - natural policy gradient and conjugate gradient method, which will be introduced in order to explain the TRPO algorithm.

3.5.4 Natural Policy Gradient

Let $\bar{D}_{KL}(\pi||\pi') = \mathbb{E}_{s\sim d^{\pi}}\big[D_{KL}(\pi'||\pi)\big]$. Natural Policy Gradient solves the following optimization problem

$$\begin{aligned} \pi_{k+1} = \text{argmax}_{\pi'} \; & \mathcal{L}_{\pi_k}(\pi') \\ s.t. \; & \bar{D}_{KL}(\pi'||\pi) \leq \delta. \end{aligned} \tag{20}$$

Using Taylor's expansion, both terms can be approximated up to the second-order

$$\begin{aligned} \mathcal{L}_{\theta_k} &\approx g^T(\theta-\theta_k) & g &= \nabla_{\theta}\mathcal{L}_{\theta_k}(\theta)|_{\theta_k} \\ \bar{D}_{KL}(\pi'||\pi) &\approx \frac{1}{2}(\theta-\theta_k)^T H(\theta-\theta_k) & H &= \nabla^2_{\theta}\bar{D}_{KL}(\theta||\theta_k)|_{\theta_k} \end{aligned} \tag{21}$$

where g is the policy gradient, H is a measure of the policy's curvature relative to the model parameter θ and ∇ stands for the gradient. Note that KL-divergence Hessian H is equal to a special matrix called the Fisher information matrix.

Now the problem in the equation 20 can be rewritten using previous approximations as

$$\begin{aligned} \theta_{k+1} = \; & \arg\max_{\theta} \; g^T(\theta-\theta_k) \\ \text{s.t.} \; & \frac{1}{2}(\theta-\theta_k)^T H(\theta-\theta_k) \leq \delta. \end{aligned} \tag{22}$$

This quadratic equation can be solved analytically

$$\theta_{k+1} = \theta_k + \underbrace{\sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g}_{\text{natural policy gradient}} . \tag{23}$$

The natural policy gradient direction $H^{-1}g$ is covariant and maps the changes wanted in the policy into corresponding parameter space. That is, it points in the same direction regardless of the parametrization used to compute it, which is not true for regular policy gradient. Therefore, the natural policy gradient is better than the regular policy gradient. The pseudocode for NPG is described under the algorithm 2.

Algorithm 2: Natural Policy Gradient, source [32]

```
1 Input: initial policy parameters $\theta_0$ ;
2 for k=0,1,2,... do
3     Collect set of trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$ ;
4     Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm (e.g. GAE) ;
5     Form sample estimates for policy gradient $\hat{g}_k$ (using advantage estimates) ;
6     Form sample estimates for KL-divergence Hessian/Fisher Information Matrix
        $\hat{H}_k$ ;
7     Compute Natural Policy Gradient update: $\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{\hat{g}_k^T \hat{H}_k^{-1} \hat{g}_k}} \hat{H}_k^{-1} \hat{g}_k$
8 end
```

An important fact is that NPG is a second-order optimization method and therefore is computationally expensive. More precisely, calculating the inverse of Hessian H is an expensive operation and numerically unstable. This issue is solved by PPO that applies the NPG concept with complexity closer to the first-order optimization method and is discussed later.

3.5.5 Conjugate Gradient

Conjugate Gradient can be used for solving quadratic objective function (like in equation 22) more efficiently. The concept is similar to gradient ascent but can be done in fewer iterations. If the model has N parameters, then the optimal point can be found with less or equal to N steps. The main idea being used is that the next search direction must be orthogonal (conjugate) to all previous directions at every iteration step. Truncated natural policy gradient (TNPG) uses the conjugate method (row 7 in algorithm 2), which gets rid of the need to compute H^{-1}.

Algorithm 3: Truncated Natural Policy Gradient, source [32]

```
1  Input: initial policy parameters $\theta_0$ ;
2  for k=0,1,2,... do
3      Collect set of trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$ ;
4      Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm ;
5      Form sample estimates for policy gradient $\hat{g}_k$ (using advantage estimates) ;
6      Form sample estimates for KL-divergence Hessian/Fisher Information Matrix
         $\hat{H}_k$ ;
7      Use conjugate gradient with $n_{cg}$ iteration to obtain $x_k \approx \hat{H}_k^{-1}\hat{g}_k$ ;
8      Estimate proposed step $\Delta_k \approx \sqrt{\frac{2\delta}{x_k^T \hat{H}_k^{-1} x_k}} x_k$ ;
9      $\theta_{k+1} = \theta_k + \Delta_k$
10 end
```

3.5.6 Trust Region Policy Optimization

Finally, combining together conjugate gradient and natural policy gradient, TRPO can be formulated. Nonetheless, there are still two problems with the NPG update. Firstly, due to the approximation made in equation 21, the KL-divergence constraint may be violated. Secondly, NPG might not be robust to trust region size, and at some iterations, δ may be too large. Both may lead to performance degradation.

The solution to first problem is to require improvement in objective function in consecutive steps ($\mathcal{L}_{\theta_k}(\theta_{k+1}) \geq 0$). Second issue can be solved with enforcing KL-constraint ($\bar{D}_{KL}(\theta_{k+1}||\theta_k) \leq \delta$). Line search (algorithm 4) ensures the imporvement of the objective function and satisfcation of the KL divergence constraint. If the verification fails, the NPG is decayed by factor α $(0 < \alpha < 1)$ until the requirements are met.

Algorithm 4: Line search for TRPO, source [32]

```
1 Compute proposed policy step $\Delta_k = \sqrt{\frac{2\delta}{\hat{g}_k^T \hat{H}_k^{-1} \hat{g}_k}} \hat{g}_k$ ;
2 for k=0,1,2,..., L do
3     Compute proposed update $\theta = \theta_k + \alpha^j \Delta_k$ ;
4     if $\mathcal{L}_{\theta_k}(\theta) \geq 0$ and $\bar{D}_{KL}(\theta||\theta_k) \leq \delta$ then
5         accept the update and set $\theta_{k+1} = \theta_k + \alpha^j \Delta_k$ ;
6         break ;
7     else
8 end
```

TRPO combines the line search and TNPG. The final algorithm is described under algorithm 5, where the addition to the algorithm 3 is marked in red.

Algorithm 5: TRPO, source [32]

```
1 Input: initial policy parameters $\theta_0$ for $k$=0,1,2,... do
2     Collect set of trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$ ;
3     Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm;
4     Form sample estimates for policy gradient $\hat{g}_k$ (using advantage estimates) ;
5     Form sample estimates for KL-divergence Hessian/Fisher Information Matrix
        $\hat{H}_k$ ;
6     Use conjugate gradient with $n_{cg}$ iteration to obtain $x_k \approx \hat{H}_k^{-1}\hat{g}_k$ ;
7     Estimate proposed step $\Delta_k \approx \sqrt{\frac{2\delta}{x_k^T \hat{H}_k^{-1} x_k}} x_k$ ;
8     Perform backtracking line search with exponential decay to obtain final update
        $\theta_{k+1} = \theta_k + \alpha^j \Delta_k$
9 end
```

3.5.7 Proximal Policy Optimization

A problem arising with natural policy gradient is that it uses second-order derivatives (calculation of Hessian H in equation 21), which calculation is computationally expensive. As most real-world problems have high computational complexity, NPG is not scalable for such issues. Alternatively, PPO does not calculate NPG and solves the problem by approximately enforcing the KL constraint. The constraint is formulated inside of the reward function as a penalty. Therefore, the first-order optimizer like gradient descent can be used to solve our problem. Although the constraint may be violated from time to time, the damage is way smaller, and the computation is much cheaper. There are two variants of PPO: PPO with Clipped Objective and PPO with Adaptive KL Penalty.

PPO with Adaptive KL Penalty

As the name says, Adaptive KL Penalty uses penalty on KL divergence to adapt the penalty coefficient so a target KL value d_{targ} is met at each policy update. Meaning that the objective function is penalized if the new policy differs from the old policy . The new objective function is formulated the following way

$$\text{maximize } \mathcal{L}_{\pi}\ (\pi') - \beta \bar{D}_{KL}(\pi' || \pi) \tag{24}$$

where β controls the weight of the penalty. KL-divergence between the old and new policy is measured with d as

$$d = \bar{D}_{KL}(\pi' || \pi) \tag{25}$$

Based on d, the value of β is dynamically adjusted. If it is higher than the target value, β is shrunk. Whereas, if it is below the target, the trust region is expanded

$$\beta_{k+1} = \begin{cases} \beta_k/2, & d < d_{targ}/1.5 \\ 2\beta_k, & d > 1.5 d_{targ}. \end{cases} \tag{26}$$

The algorithm is not sensitive to the heuristically made parameter choice (1.5 and 2). The starting value of β is a hyperparameter that might require tuning, but in practice, the choice has not much significance as the algorithm quickly adjusts it. The pseudocode of PPO with Adaptive KL Penalty is given under algorithm 6.

The policy update is computed using stochastic gradient descent (SGD). For SGD optimization Adaptive Moment Estimation (Adam) [33] is used. Adam is an optimization algorithm that can be used instead of the classical SGD to update network weights based on the training data. It is an efficient method for large problems involving a lot of data or parameters and does not require much memory.

Algorithm 6: PPO with Adaptive KL Penalty, source [34]

```
1  Input: initial policy parameters $\theta_0$, initial KL penalty $\beta_0$, target KL-divergence $d_{targ}$;
2  for k=0,1,2,... do
3      Collect set of trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$ ;
4      Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm;
5      Compute policy update $\theta_{k+1} = \arg\max_\theta \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta||\theta_k)$ by taking K
         steps of minibatch SGD (via Adam);
6      if $d \geq 1.5 d_{targ}$ then
7          $\beta_{k+1} = 2\beta_k$
8      end
9      if $d \leq 1.5/d_{targ}$ then
10         $\beta_{k+1} = \beta_k/2$
11     end
12 end
```

PPO with Clipped Objective

An alternative to Adaptive KL Penalty is PPO with Clipped Objective (CLIP). In practice, this has been proven to perform better than the adaptive KL penalty.

As a reminder from TRPO equation 15, TRPO uses a ratio to measure the difference between two policies. Denote it with $r_t(\pi') = \frac{\pi'(a_t|s_t)}{\pi(a_t|s_t)}$, so $r_t(\pi) = 1$. Namely, new policy is evaluated with samples collected from the old policy.

A very similar objective function is used by CLIP. The idea of CLIP is to modify the objective function so that it penalizes changes that move $r_t(\pi')$ away from 1. Therefore, the new objective function is constructed to clip the estimated advantage function $\hat{A}_t$ if the new policy is not close to the old policy. The proposed objective function looks like the following

$$\mathcal{L}^{CLIP}(\pi') = \mathbb{E}_t\Big[\min(\underbrace{r_t(\pi')\hat{A}_t}_{\text{unclipped}}, \underbrace{\text{clip}(r_t(\pi'), 1-\epsilon, 1+\epsilon)\hat{A}_t}_{\text{clipped}})\Big], \tag{27}$$

where epsilon is hyperparameter, usually $\epsilon \in \{0.1, 0.2, 0.3\}$.

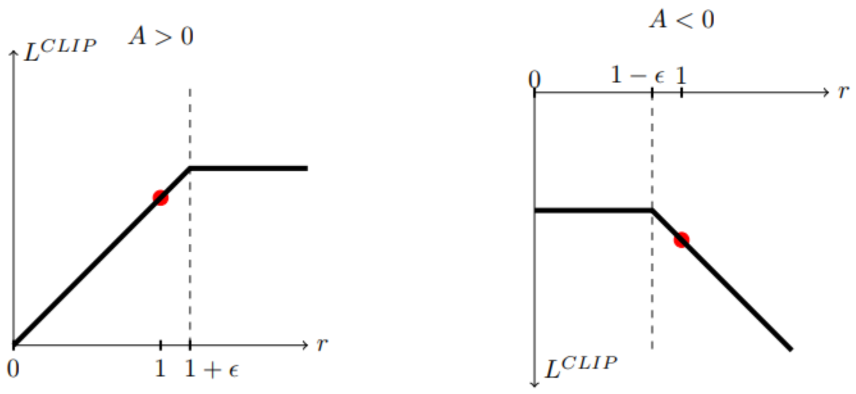


Figure 7. Clipped surrogate objective

The term $\text{clip}(r_t(\pi'), 1-\epsilon, 1+\epsilon)\hat{A}_t$ makes sure that if r_t falls outside of the interval $[1-\epsilon, 1+\epsilon]$ then the advantage function will be clipped. Additionally, taking the minimum of the clipped and unclipped objective makes $\mathcal{L}^{CLIP}$ a lower bound of the unclipped objective $r_t(\pi')\hat{A}_t$. All this means that the new policy does not benefit by going far away from the old policy. Figure 7 shows a single timestep t in $\mathcal{L}^{CLIP}$. The red dot on the figure shows the starting point for the optimization where $r = 1$. If the advatage function is positive $A > 0$, then the advatage function is clipped at $1+\epsilon$, meaning that there is a limit to how much the objective function can increase. Similarly, if the advantage function is negative $A < 0$, then the advantage function is clipped at $1-\epsilon$. It should be noted that $\mathcal{L}^{CLIP} = \mathcal{L}^{TRPO}$ if $r = 1$.

Pseudocode for PPO with Clipped Objective is given under algorithm 7.

Algorithm 7: PPO with Clipped Objective source [34]

```
1 Input: initial policy parameters $\theta_0$, clipping threshold $\epsilon$;
2 for k=0,1,2,... do
3     Collect set of trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$ ;
4     Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm;
5     Compute policy update
          $$\theta_{k+1} = \text{arg max}_\theta \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$
      by taking $K$ steps of minibatch stochastic gradient descent (via Adam) where
          $$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathbb{E}_t\Big[\text{min}(r_t(\theta)\hat{A}_t^{\pi_k}, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t^{\pi_k})\Big]$$
```

4 Experimental Results and Analysis

This chapter intends to present conducted experiments and results obtained. Used evaluation metrics are introduced, as well as training setup and the process of hyperparameter tuning. All measured metrics with confidence intervals are reported, and a comparison between the baseline OpenAI model and the proposed model is provided.

4.1 Evaluation metrics

To evaluate the performance of the current policy, at each step of the training, various metrics are measured. The main three metrics used to describe the progress are Average Episode Reward, Average Episode Cost and Cost Rate. To explain the formulas calculating the metrics mentioned above, we need to define and explain some adopted notations. The training process consists of o epochs, each epoch consists of n episodes, and each episode contains m interactions with the environment.

$$\underbrace{\underbrace{1, 2, 3, 4, \dots}_{episode_1}\underbrace{\dots, \mathrm{i}, \dots}_{episode_2} \dots \underbrace{\dots}_{episode_k} \dots \underbrace{\dots, \mathrm{m}}_{episode_n}}_{\mathrm{epoch}_1}, \dots, \underbrace{\underbrace{1, 2, 3, 4, \dots}_{episode_1}\underbrace{\dots, \mathrm{i}, \dots}_{episode_2} \dots \underbrace{\dots}_{episode_k} \dots \underbrace{\dots, \mathrm{m}}_{episode_n}}_{\mathrm{epoch}_\mathrm{o}}$$

Note that n is not fixed and varies at every epoch as it depends on how quickly the agent completes its task. On the contrary, m and o are fixed at the beginning of training.

Let $j_{kr}(\theta)$ be the reward of the k-th episode, $j_{kc}(\theta)$ be the cost of the k-th episode and $j_{ic}(\theta)$ be the cost of the i-th environment interaction. Average Episode Reward $J_r(\theta)$ is calculated based on the objective function of the optimization problem that should be maximized and is done so at each epoch

$$J_r(\theta) = \frac{1}{n} \sum_{k=1}^{n} j_{kr}(\theta). \tag{28}$$

Average Episode Cost $J_c(\theta)$ is the quantity aimed to constrain. An average is taken over all the sums of costs, and it describes how many non-desired acts the agent performed on average. The Average Episode Cost of the epoch is therefore

$$J_c(\theta) = \frac{1}{n}\sum_{k=1}^{n} j_{kc}(\theta). \tag{29}$$

Cost Rate ρ_c differs mainly because it is calculated over the whole training process ($N = mo$ environment interactions)

$$\rho_c = \frac{1}{N}\sum_{i=1}^{N} j_{ic}(\theta). \tag{30}$$

Similarly to average cost, the cost rate is minimized by the algorithms. The reason behind using this metric to evaluate performance lies in few properties. It describes the overall safety - a lower cost rate means fewer unsafe things happened throughout the training. Furthermore, it is more intuitive to allow comparisons between training runs of unequal length.

4.2 Training Results

In the methodology section, four different RL algorithms were described

- TRPO with adaptive penalty (TRPO-AP) (equation 18)
- TRPO (equation 19)
- PPO with adaptive penalty (PPO-AP) (equation 24)
- PPO (equation 27).

The unconstrained RL algorithms' (TRPO, PPO) reward function contains no information about the cost. Those algorithms tend to receive high rewards by taking unsafe actions. On the other hand, constrained RL algorithms (TRPO-AP, PPO-AP) attain lower levels of rewards and correspondingly maintain desired levels of costs. In literature, unconstraint RL algorithms seem to have been more popular, the reason might be that they give better results regarding the reward.

4.2.1 Training setup

All experiments use separate feedforward MLP policy and value networks of size $(256, 256)$ with tanh activations. Experiments for the agent used batch sizes of 100 000 environment interaction steps, and the agents were trained for $6\ 000\ 000$ steps for hyperparameter tuning and $10\ 000\ 000$ steps for the best combination of hyperparameters.

All the code is written in Python3 and is available on GitHub [35], [36] . The training of the model was conducted on the following machine: Intel(R) Core(TM) i7-9800X CPU @ 3.80GHz, 62 GB RAM, 16 CPUs, Ubuntu 18.04.5 LTS.

4.2.2 Hyperparameter tuning

Hyperparameter tuning plays a significant role in the overall performance of machine learning (ML) models. Good choice of parameters ensures that the algorithm can learn well on its own during the learning process. Reinforcement learning requires numerous predefined parameters. These parameters are defined at the beginning of the training and are not changed during the training (e.g. learning rate). Selection of the parameters has a strong impact on the learning process, environment interaction and also the required learning time [37]. What makes the search for the best parameters complex is that hyperparameter tuning is a very time-consuming process.

Generally, in ML, finding the best combination of hyperparameters can be done either by random search or grid search. Random search sets up a grid of hyperparameter values and selects random combinations to train the model. Using such grid allows controlling the number of parameter combinations that are attempted explicitly. However, random search suffers from the lack of guidance and may not represent all the range of the parameters. Grid search is a traditional approach to find suitable hyperparameters. A reasonable subset of values is defined for each hyperparameter. By contrast to random search where only some combinations are explored, each value combination is tried in a grid search.

Finally, the combination achieving the best results is used in the actual learning task. This method is easy to implement and a widely used approach for optimising parameters [37]. Grid search obtains more accurate results; therefore, we use it to determine the best hyperparameters in this work. In order to narrow down the suitable parameter value ranges, optimal values from existing papers [12], [26] and [38] were used as a guideline.

For TRPO, three different parameters were tuned, namely lidar range, value function optimizer learning rate and target Kullback-Leibler divergence δ. In TRPO-AP, the same hyperparameters are used; only δ is excluded as it is not needed in the adaptive penalty algorithm. For PPO, four different parameters were chosen. Similarly to TRPO, both lidar range and value function learning rate were tuned. In addition, optimal values for policy learning rate and clip ratio are searched. In the PPO-AP algorithm, clip ratio is not used and therefore, it is omitted from tuning. The rest of the parameter values were taken from the Safety Gym baseline repository.

Before proceeding to the best combinations, the chosen parameters are introduced. Lidar range defines how far the lidar sensor perceives objects. KL divergence δ describes how big of a difference we think is appropriate between new and old policies after an update. Clip ratio is a PPO specific hyperparameter for clipping in the policy objective. It measures how far can the new policy go from the old policy while still improving the objective function.

Choosing the best learning rate is a challenging task because values too small make the training process long, whereas values too large may result in learning a sub-optimal set of weights too fast or an unstable training process. Therefore, value function learning rate and policy learning rate were chosen as tunable parameters. The policy learning rate is the learning rate for policy optimizer Adam. It determines the proportion that weights are updated. Larger values result in faster initial learning before the rate is updated, and smaller values slow learning right down during training. Value function $V^{\pi}(s)$ (described in the Methodology section) can not be computed exactly and needs to be approximated. Generally, it is done with a neural network, $V^{\phi}(s_t)$, which is updated concurrently with the policy. Concurrent updates make sure that the value network always approximates the value function of the most recent policy. The method for learning V^{ϕ} is implemented as a minimization of mean-squared error. This is done with one or more gradient descent steps starting from the previous value parameters ϕ^{k-1}. The learning rate used in gradient descent is one of the tunable parameters chosen for this work.

For determining the best hyperparameter combination, all three metrics need to be taken into account. The policy should achieve high Average Reward and, at the same time, low Average Cost and Cost Rate values. Problematic is that the metrics are measured in very different scales, for example Average Reward usually takes values $J_r \in [0, 30]$, Average Cost $J_c \in [0, 10000]$ and Cost rate $\rho_c \in [0, 1]$. For this reason, all metrics are scaled to a range $\widetilde{J_r}, \widetilde{J_c}, \widetilde{\rho_c} \in [0, 1]$. Scaled metrics preseve the distance between different values and at the same time make the metrics comaparable. For $\widetilde{J_c}$ and $\widetilde{\rho_c}$ values close to 0 are most optimal (as then the cost is minimized), whereas for $\widetilde{J_r}$ it is 1 (reward is maxmimized). To unify the metrics even further, the inverse of cost related metrics is taken $\widetilde{J_c}' = 1 - \widetilde{J_c}, \widetilde{\rho_c}' = 1 - \widetilde{\rho_c}$. Now, it can be treated as a maximization problem, and a score of goodness for each final policy can be calculated

$$score = \gamma_1 \widetilde{J_r} + \gamma_2 \widetilde{J_c}' + \gamma_3 \widetilde{\rho_c}', \;\; score \in [0, 1]$$

where all the weights are equal $\gamma_1 = \gamma_2 = \gamma_3 = \frac{1}{3}$. Hyperparameter combination yielding the highest score is chosen as the best for each algorithm. In order to calculate the values of all the three metrics last five epochs of training were averaged to reduce noise. The results obtained through grid search are displayed in the table 4.2.2.

Hyperparameter	TRPO	TRPO-AP	PPO	PPO-AP
Lidar range	1	Exp*	5	5
Value learning rate (Adam)	0.0005	0.002	0.002	0.002
Target KL divergence δ	0.01	-	-	-
Policy learning rate (Adam)	-	-	0.0001	0.0003
Clip ratio	-	-	0.3	-

* The formula for exponential lidar he formula used is $e^{\text{-distance to the object}}$ and is the suggested in the Safety Gym project. Exponential lidar range is described as closeness or inverse distance. So if the object is close e.g. with the distance to agent 0.1, then the sensor perceives its closeness to be $e^{-0.1} = 0.9$. When the object is far for example the distance to agent is 0.9, then the sensor perceives its closeness to be $e^{-0.9} = 0.4$.

4.2.3 Bootstrapping

In general, RL algorithms are evaluated with figures and tables of average rewards and maximum reward achieved over a fixed number of steps. However, Henderson et al. [25] have pointed out that RL algorithms' stability is not guaranteed and simply presenting the average returns is not enough for a fair comparison. As the number of trials and random seeds is usually left unmentioned, the reported average returns can be misleading. The authors recommend combining the presented results with confidence intervals, to provide a clearer picture of the algorithm's performance.

Henderson et al. have proposed to evaluate performance by using bootstrapping. It is a popular method to gain insight into a population distribution from a small sample, including calculating the sample mean and confidence intervals. Bootstrapping does not assume any underlying distribution of data, unlike the traditional approach for confidence intervals that expects the data to be normally distributed. Therefore, we do not have to worry about any assumptions of the data acquired during the model training.

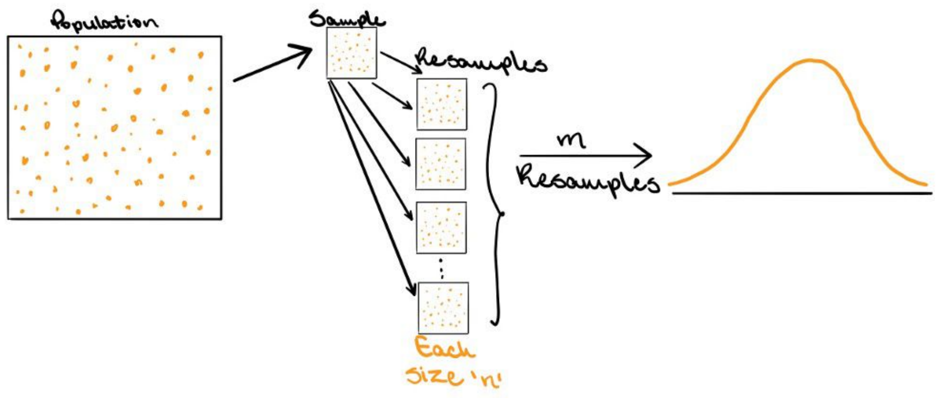


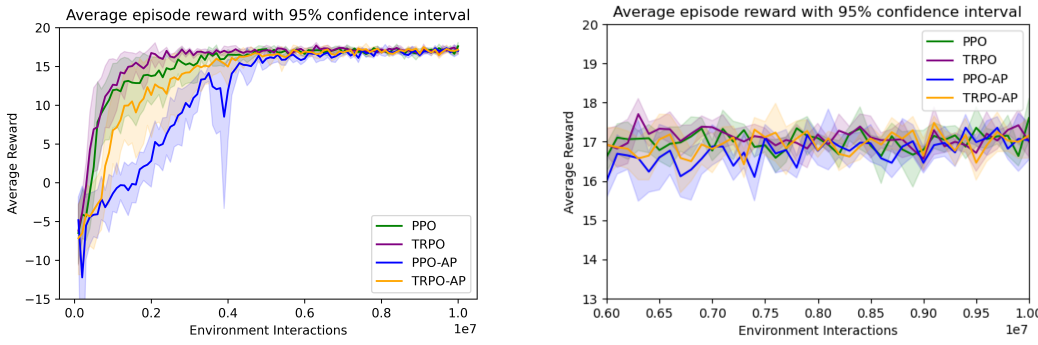
Figure 8. Bootstrap sampling with replacement by Trist'n Joseph

Bootstrapping is done by repeatedly taking small samples, calculating the desired statistic (which in our case is Average Reward/Average Cost/Cost Rate), and taking the mean of the calculated statistics (Figure 8). The process can be summarized the with the following steps

1. Choose a number of bootstrap samples to perform m
2. Choose a sample size n
3. For each bootstrap sample m
   (a) Draw a sample **with replacement** with the chosen size n
   (b) Calculate the statistic on the sample
4. Calculate the mean of the m calculated sample statistics
5. Calculate confidence intervals as 2.5th and 97.5th percentile of the bootstrap samples, this ensures that 95% of the time, the true parameter value falls between mentioned percentiles.

4.2.4 Training results

The final models with four different algorithms are trained with the best combination of hyperparameters. In order to obtain more accurate results, the model is trained 5 times with identical set of hyperparameters, but 5 different random seeds and 10 000 000 environment interaction steps. In order to have more credible results, bootstrapping ($m = 1000, n = 20$) is used to find mean and confidence intervals for all three performance metrics.

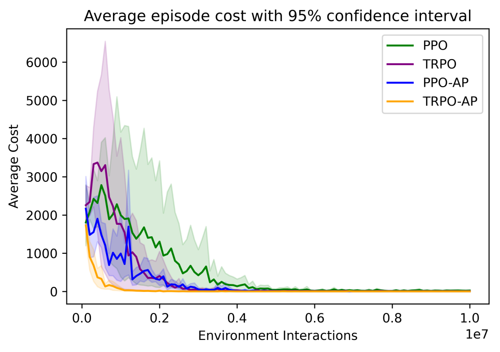


(a) Whole training process (b) End of the training process

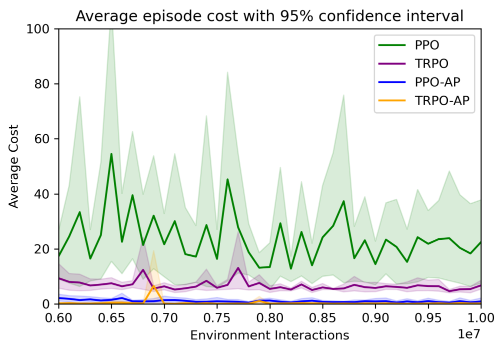
Figure 9. Average Reward with 95% confidence intervals

Figure 9 describes the Average Reward for four different models with 95% confidence intervals. Unconstrained algorithms learn to maximize the reward very quickly, TRPO after ~2 million steps and PPO after ~4 million. However, for constraint algorithms it takes more time to converge to optimal. PPO-AP is unstable in the first half of the training, but later manages to converge to the same level as the other algorithms. For TRPO-AP it also takes around ~5 million interactions, however, the learning process is more stable. At the end of the training, all algorithms reach the same amount of reward. Therefore, there is no difference which algorithm to choose if there is enough computational power.

On Figure 10 average penalties received among algorithms are presented. Unconstrained



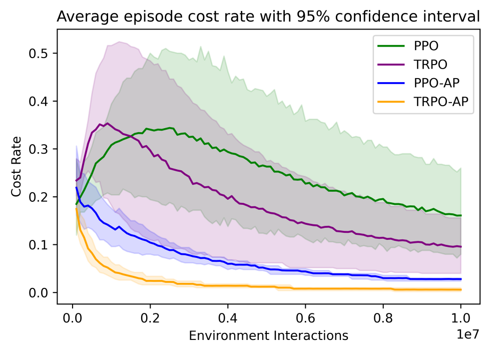
(a) Whole training process



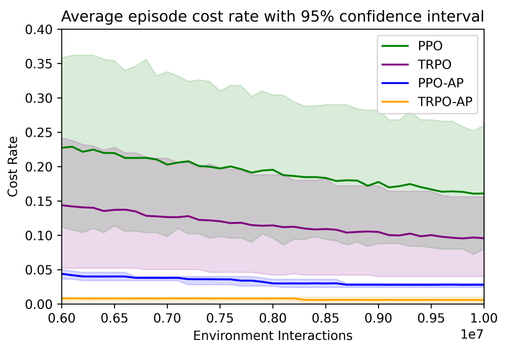
(b) End of the training process

Figure 10. Average Cost with 95% confidence intervals

algorithms act dangerously during the beginning of the training. Furthermore, the wide confidence intervals of TRPO and PPO indicate the agents continue reckless behaviour during the whole process. On the contrary, constrained algorithms learn to behave safely after few interactions with the environment. The reason behind is that the penalty coefficient is increased when non-safe behaviours occur often. In the second half of the training both TRPO-AP and PPO-AP maitain equally low costs with very little variance. TRPO-AP shows especially quick convergence to 0. Altogether, PPO has the unsafest behaviour.



(a) Whole training process



(b) End of the training process

Figure 11. Cost Rate with 95% confidence intervals

Finally, Figure 11 demonstrates Cost Rate values which show similar trends to Average Cost. Clearly, the best Cost Rate is achieved with TRPO-AP. Both constraint algorithms have not only the lowest values, but also the narrowest confidence interval, indicating the stability of following the safety rules. The small increase in the beginning of the training for PPO and TRPO shows the exploration phase - agents try out different and possibly dangerous behaviours, to learn quicker actions leading to the highest rewards. This explains, why unconstraint algorithms managed to achieve high rewards early in the training process.

4.3 Evaluation Results

A question arises whether the modifications done on the environment and the model were necessary at all. Therefore, comparison of original and proposed models is provided. Perhaps, the baseline model trained in the original environment presented by Safety Gym might easily adapt in the proposed environment such that creating a new environment was redundant. Or possibly, model trained in proposed environment might be able to generalize well and adapt to original environment's requirements. Therefore, we test how an agent, who initially is trained in one RL environment, performs in another environment. Both environments have the same task to reach the goal; however, the safety requirements are different.

4.3.1 Evaluation in original environment

In this section, we measure how well different models manage in the original environment. Firstly, to have a baseline for the comparison, one model for each algorithm was trained following the original Safety Gym [8] implementation with its suggested parameters and training length (hereafter referred as "original model"). An example of the original environment's settings is displayed on Figure 12. According to the paper, the default world size (area where objects can be placed) is a 8×8 grid and maximum number of environment interactions per one episode is 10 000. Secondly, one model for each algorithm was trained on the proposed environment using the best combination of hyperparameters (hereafter referred as "proposed model"). Next, both original and proposed models were evaluated over 100 episodes (to reduce the impact of randomly generated start and end positions), each episode maximum 10 000 interactions with the environment. In total 8 different models were evaluated. During the testing, Cost Rate over the 100 episodes, as well as Average Cost, Average Reward per episode were measured.

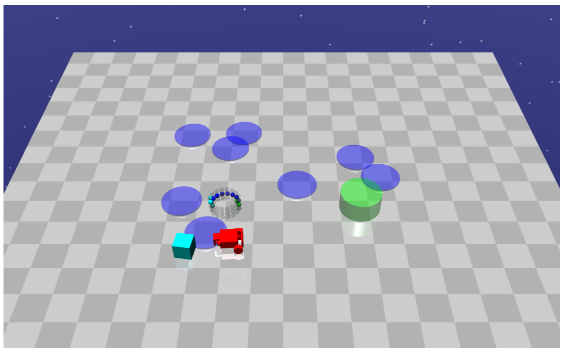


Figure 12. Original environment

The results for the experiments are presented in tables 1, 2, 3 and 4. Tables show that original model receives higher rewards (around 8 points better) than the proposed model. One of the reason behind it is that the modified model expects the roads to be straight and to have a path helping to find the goal object. When it does not observe goal with it's lidar, it continues to drive straight or making small turns. Such behaviours do not lead to success in the original environment which has very random layout. However, when it comes to cost, proposed model maintains the same average cost or even in some cases better (PPO, TRPO-AP) than the baseline. The comparison of the agents' performance in the original environment shows that even though the proposed models receive slightly less reward, they manage to ensure the same safety.

	Avg Reward	Avg Cost	Cost Rate
Original model (baseline)	28.8	110.2	0.11
Proposed model	19.0	68.2	0.068

Table 1. PPO

	Avg Reward	Avg Cost	Cost Rate
Original model (baseline)	29.8	93.8	0.094
Proposed model	22.5	96.7	0.089

Table 2. TRPO

	Avg Reward	Avg Cost	Cost Rate
Original model (baseline)	20.6	25.6	0.026
Proposed model	12.1	32.4	0.032

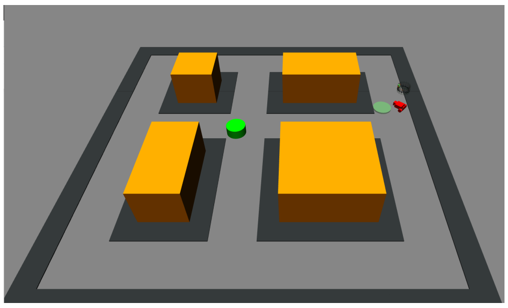
Table 3. PPO-AP

	Avg Reward	Avg Cost	Cost Rate
Original model (baseline)	15.7	19.9	0.02
Proposed model	8.6	18	0.018

Table 4. TRPO-AP

4.3.2 Evaluation in proposed environment

Previously, it was evaluated how model trained in the proposed environment preforms in the original one. In this section, exactly the opposite is done - showing both models' performance in the proposed environment. Furthermore, the testing containes two different environments, a small and a big world (Figure 13). A small environment is rather easy having grid size of 18×18 and 6 roads, whereas a big environment is twice as large having grid size 36×36 and 10 roads. In smaller one the agent might find the goal with enough exploration and luck whereas bigger world requires following the navigation circles.


(a) Small world

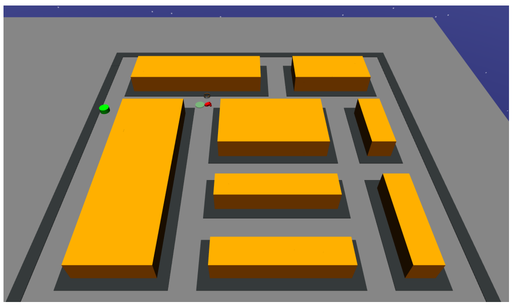

(b) Big world

Figure 13. Different testing environments

Similarly to previous subchapter, the same testing procedure on 100 episodes was followed. The comparison of metrics for different algorithms in various settings is given in tables 5, 6, 7 and 8.

	Small environment			Big environment		
	Average Reward	Average Cost	Cost Rate	Average Reward	Average Cost	Cost Rate
Original model (baseline)	2.81	3147	0.468	0.82	4055	0.451
Proposed model	17.8	3.8	0.011	33.2	6.6	0.013

Table 5. PPO

Results for both small and big environments show that the original model fails to complete its tasks regardless the used algorithm. For small environment, the rewards are slightly

	Small environment			Big environment		
	Average Reward	Average Cost	Cost Rate	Average Reward	Average Cost	Cost Rate
Original model (baseline)	3.5	4161	0.63	0.579	5056	0.582
Proposed model	17.6	3.6	0.012	34.4	2.9	0.005

Table 6. TRPO

	Small environment			Big environment		
	Average Reward	Average Cost	Cost Rate	Average Reward	Average Cost	Cost Rate
Original model (baseline)	2.79	303.3	0.045	-3.99	685.9	0.075
Proposed model	17.6	0	0	38	0	0

Table 7. PPO-AP

	Small environment			Big environment		
	Average Reward	Average Cost	Cost Rate	Average Reward	Average Cost	Cost Rate
Original model (baseline)	2.16	136	0.02	-1.75	489	0.052
Proposed model	15.2	0	0	34.8	0	0

Table 8. TRPO-AP

above 0, whereas for big environment reward drop below 0 in worst cases. On the other hand, the proposed model receives average reward over 15 in small world and over 33

in big world with each algorithm. Looking at the cost, the differences between original and proposed models are immense. Unconstrained algorithms (TRPO, PPO) receive enormous average cost around $3000 - 5000$ for original model, whereas for proposed model this range is between 2.9 and 6.6. Constrained algorithms do even better - in the testing phase with proposed model 0 unsafe actions occurred. Original constrained models had lower costs than with the unconstrained algorithms, however, it still remained above 130.

The reason behind original model's bad performance lies in the lack of lidars. As the neural network is trained without a lidar to observe the goal path objects, then in the testing it is not possible to include it as the dimensions would not match. So the baseline model cannot perceive them with lidar, however, it still receives reward entering and driving towards goal path objects.

The results show, that the proposed environment changes were necessary for the agent to learn to drive in a structured environment. Only using the models coming from the original environment proposed in the Safety-Gym paper are not enough for the agent to obtain this kind of behaviour. Therefore, the design of the new environment, including the navigation help, extra lidars, new reward and the penalty are all very important given the desired task. Furthermore, as proposed model succeeded in the original environment, then it was not overfitted (e.g. learn to drive only straight or make 90 degree turns) as it was able to generalize well enough and maintain the safety.

5 Discussion

This chapter will present conclusions for this thesis and discussion of unexpected behaviour encountered. Furthermore, future work and perspectives are described.

5.1 Unexpected behaviour

A critical part of reinforcement learning is the reward architecture. To achieve the results presented in the previous sub-chapters, different reward designs were tested. Not all the implementations resulted in the agent performing the task in the intended way. Here is presented one example of what can go wrong with the reward function design.

In the development process, one of the main issues in the agent's behaviour was its struggle with moving straight. Not only did it always start drifting to either the right or left side and ended up crashing the sidewalks, but it also was circling around its own z-axis. To address this problem, an additional reward function for upright position was designed to reward the agent for moving straightforward. First, with small extra rewards, it did not seem to impact the behaviour, and the reward for an upright position was increased. However, the cleverness of the algorithm was underestimated. The agent learned to circle around one spot, as turning in a small angle earned more reward than trying to reach the goal. The same behaviour has also been noted in the literature [39].

5.2 Conclusion

This thesis investigates reinforcement learning techniques for autonomous navigation. The main focus was on applying four policy-based RL algorithms for an agent to learn to behave safely and successfully in a warehouse looking environment. We designed a suitable structured environment, proposed modifications for the existing model, used different algorithms for training, applied hyperparameter tuning, and measured the progress both from aspects of successfully completing the task and avoiding dangerous actions.

The evaluation of both baseline and proposed models for each four algorithms (PPO, TRPO, PPO-AP, TRPO-AP) was carried out. The results showed that the settings proposed by the baseline Safety Gym project are not enough to produce a model that is capable of successfully interacting with more complex situations. On the contrary, our proposed models managed to generalize well and gave satisfactory results in both the baseline environment as well as the proposed environment. The constrained RL algorithms (TRPO-AP, PPO-AP) performed the best regarding safety. Namely, they managed to ensure 100% safety during testing and therefore would suit the best for usage in autonomous driving. However, the training time to learn the desired behaviour is shorter for unconstrained algorithms (TRPO, PPO).

5.3 Future Work

As future work, different direction can be explored further. The main emphasis should be on improving the proposed environment by adding more physical and dynamic objects. In order to make it more realistic, a way to integrate more complex objects like pedestrians or traffic lights should be found.

In addition, using a broader range of values for hyperparameter tuning should be considered to get the most optimal model. Performance of other policy-based reinforcement learning methods like Deterministic Policy Gradient or Constrained Policy Optimization could also be added to the comparison of models. Furthermore, ensemble methods like boosting or bagging could be extended to reinforcement learning as they have a rich theoretical foundation in supervised learning. They can be used for learning the value function and thus enhance the learning speed and final performance of the models.

References

[1] Nathan Benaich. *6 areas of AI and machine learning to watch closely*. Jan. 2017. URL: https://medium.com/@NathanBenaich/6-areas-of-artificial-intelligence-to-watch-closely-673d590aa8aa.

[2] Alex Ray, Joshua Achiam, and Dario Amodei. *Safety Gym*. https://github.com/openai/safety-gym. 2019.

[3] B Ravi Kiran et al. "Deep reinforcement learning for autonomous driving: A survey". In: *IEEE Transactions on Intelligent Transportation Systems* (2021).

[4] OpenAI. *Part 1: Key Concepts in RL*. 2018. URL: https://spinningup.openai.com/en/latest/spinningup/rl%5C_intro.html.

[5] Google AI Blog. *Off-Policy Estimation for Infinite-Horizon Reinforcement Learning*. Apr. 2020. URL: https://ai.googleblog.com/2020/04/off-policy-estimation-for-infinite.html.

[6] S. Chen et al. "Stabilization Approaches for Reinforcement Learning-Based End-to-End Autonomous Driving". In: *IEEE Transactions on Vehicular Technology* 69.5 (2020), pp. 4740–4750. DOI: 10.1109/TVT.2020.2979493.

[7] Zeyu Zhu and Huijing Zhao. *A Survey of Deep RL and IL for Autonomous Driving Policy Learning*. 2021. arXiv: 2101.01993 [cs.RO].

[8] Amodei Dario Achiam Joshua Ray Alex. *Safety Gym*. Nov. 2019. URL: https://openai.com/blog/safety-gym.

[9] Ronald J. Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Mach. Learn.* 8.3–4 (May 1992), pp. 229–256. ISSN: 0885-6125. DOI: 10.1007/BF00992696. URL: https://doi.org/10.1007/BF00992696.

[10] Richard S Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 2000. URL: https://

proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[11] John Schulman et al. *Trust Region Policy Optimization*. 2017. arXiv: 1502.05477 [cs.LG].

[12] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[13] Sen Wang, Daoyuan Jia, and Xinshuo Weng. *Deep Reinforcement Learning for Autonomous Driving*. 2019. arXiv: 1811.11329 [cs.CV].

[14] Abhishek Gupta et al. "Policy-Gradient and Actor-Critic Based State Representation Learning for Safe Driving of Autonomous Vehicles". In: *Sensors* 20.21 (2020). ISSN: 1424-8220. DOI: 10.3390/s20215991. URL: https://www.mdpi.com/1424-8220/20/21/5991.

[15] David Isele et al. *Navigating Occluded Intersections with Autonomous Vehicles using Deep Reinforcement Learning*. 2018. arXiv: 1705.01196 [cs.AI].

[16] Peter Almasi, Robert Moni, and Balint Gyires-Toth. "Robust Reinforcement Learning-based Autonomous Driving Agent for Simulation and Real World". In: *2020 International Joint Conference on Neural Networks (IJCNN)* (July 2020). DOI: 10.1109/ijcnn48605.2020.9207497. URL: http://dx.doi.org/10.1109/IJCNN48605.2020.9207497.

[17] Ahmad EL Sallab et al. "Deep reinforcement learning framework for autonomous driving". In: *Electronic Imaging* 2017.19 (2017), pp. 70–76.

[18] Xinlei Pan et al. *Virtual to Real Reinforcement Learning for Autonomous Driving*. 2017. arXiv: 1704.03952 [cs.AI].

[19] Fei Ye et al. *Automated Lane Change Strategy using Proximal Policy Optimization-based Deep Reinforcement Learning*. 2020. arXiv: 2002.02667 [cs.LG].

[20] Andreas Folkers, Matthias Rick, and Christof Buskens. "Controlling an Autonomous Vehicle with Deep Reinforcement Learning". In: *2019 IEEE Intelligent*

Vehicles Symposium (IV) (June 2019). DOI: 10.1109/ivs.2019.8814124. URL: http://dx.doi.org/10.1109/IVS.2019.8814124.

[21] Duy Quang Tran and Sang-Hoon Bae. "Proximal Policy Optimization Through a Deep Reinforcement Learning Framework for Multiple Autonomous Vehicles at a Non-Signalized Intersection". In: *Applied Sciences* 10.16 (2020). ISSN: 2076-3417. DOI: 10.3390/app10165722. URL: https://www.mdpi.com/2076-3417/10/16/5722.

[22] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. *Model-free Deep Reinforcement Learning for Urban Autonomous Driving*. 2019. arXiv: 1904.09503 [cs.LG].

[23] *Road traffic injuries*. Feb. 2020. URL: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries.

[24] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. *Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving*. 2016. arXiv: 1610.03295 [cs.AI].

[25] Peter Henderson et al. *Deep Reinforcement Learning that Matters*. 2019. arXiv: 1709.06560 [cs.LG].

[26] Alex Ray, Joshua Achiam, and Dario Amodei. "Benchmarking safe exploration in deep reinforcement learning". In: *arXiv preprint arXiv:1910.01708* (2019).

[27] Alex Ray, Joshua Achiam, and Dario Amodei. *Safety Starter agents*. https://github.com/openai/safety-starter-agents. 2019.

[28] Joshua Achiam. *Advanced Policy Gradient Methods*. Oct. 2017. URL: http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_13_advanced_pg.pdf.

[29] John Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2018. arXiv: 1506.02438 [cs.LG].

[30] Wikipedia. *Kullback–Leibler divergence*. URL: https://en.wikipedia.org/wiki/Kullback%5C%E2%5C%80%5C%93Leibler_divergence#/media/File:KL-Gauss-Example.png.

[31] Jonathan Hui. *RL — Trust Region Policy Optimization (TRPO) Explained*. Oct. 2018. URL: https://jonathan-hui.medium.com/rl-trust-region-policy-optimization-trpo-explained-a6ee04eeeee9.

[32] OpenAI. *Trust Region Policy Optimization*. 2018. URL: https://spinningup.openai.com/en/latest/algorithms/trpo.html.

[33] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[34] Jonathan Hui. *RL — Proximal Policy Optimization (PPO) Explained*. Sept. 2018. URL: https://jonathan-hui.medium.com/rl-proximal-policy-optimization-ppo-explained-77f014ec3f12.

[35] Anne Ott. *Safety gym*. https://github.com/anneott/safety-gym. 2021.

[36] Anne Ott. *Safety starter agents*. https://github.com/anneott/safety-starter-agents. 2021.

[37] Roman Liessner et al. "Hyperparameter Optimization for Deep Reinforcement Learning in Vehicle Energy Management". In: Feb. 2019. DOI: 10.5220/0007364701340144.

[38] Logan Engstrom et al. *Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO*. 2020. arXiv: 2005.12729 [cs.LG].

[39] Jette Randlov and Preben Alstrøm. "Learning to Drive a Bicycle Using Reinforcement Learning and Shaping." In: Jan. 1998, pp. 463–471.

Appendix

II. Licence