CURRICULUM LEARNING FOR ROBOT NAVIGATION IN DYNAMIC ENVIRONMENTS WITH UNCERTAINTIES

BELİRSİZ DİNAMİK ORTAMLARDA ROBOT SEYRÜSEFERİ İÇİN MUFREDATLI ÖĞRENME

DEVRAN DOĞAN

DR. ÖZGÜR ERKENT

Supervisor

Submitted to

Graduate School of Science and Engineering of Hacettepe University

as a Partial Fulfillment to the Requirements

for the Award of the Degree of Master of Science

in Computer Engineering

May 2024

ABSTRACT

CURRICULUM LEARNING FOR ROBOT NAVIGATION IN DYNAMIC ENVIRONMENTS WITH UNCERTAINTIES

Devran DOĞAN

Master of Science, Computer Engineering

Supervisor: Dr. Özgür ERKENT

May 2024, 84 pages

In our study we wanted to see if there is any way we can make the training process of a DRL agent much easier, and optimize the success rate in the given tasks. In order to increase the speed of convergence we adopted curriculum learning techniques. Since the importance of the automated vehicles are increasing day by day, and the capabilities such as target search in unknown environments are gaining more attention, that brings us to the importance of path generation, and the exploration of the environment, when human life is at risk or if humans exist in the environment. As we know in complex real-world applications, safety and risk awareness become unavoidable aspects. We used risk-aware systems in unknown environments for testing the model's robustness in localization and path generation to observe the performance under the situations that are not encountered during training. Systems that are not risk-aware may lead to suboptimal decisions that will lead to failures. These explorations require high computation time. We needed to make improved risk-aware decision making to train a risk-sensitive policy that can have high performance and adaptability to required risk. And can navigate in collision free manner, while acting among static and dynamic obstacles.

DRL algorithms showed their capabilities, in learning also easy to compute reward signals. But they require long training times that makes them limited for real-world applications. Therefore, we used curriculum learning and DRL algorithms to build a goal-oriented model. By doing that we achieved faster convergence, search time for the targets is reduced for the same amount of training episodes. Collision rate is reduced. In the training process we wanted to understand in which order the training becomes really hard. For that reason we injected Gaussian noise to neural network parameters in different forms, we used different environments, delayed the sensory information to see the agents behavior, prediction success and also tested with only static obstacles, with dynamic obstacles, and finally we added both of the obstacles together. Many of the environments were partially observable, we also tested in fully observable environments as well, but we saw that DRL agents can solve these environments easily.

In order to make this study and measure the efficiency, we build a 2D simulation environment. The performance is verified with results of the simulation analysis. We measured the efficiency of the agent, by collecting the total hit ratio metrics. Experiments show the agent with curriculum learning reaches a better success rate, is efficient at control, performs better under noisy conditions, can adapt faster to unknown environments.

Keywords: Deep Reinforcement Learning, Target Search, Risk-sensitive Control, Motion and Path Planning, Autonomous Driving, Uncertainty Quantification, Partially Observable, Automatic Vehicle Control, Q-learning, Obstacle and Collision Avoidance, Unknown Environment Exploration, Risk-aware, Trajectory Planning

ÖZET

BELİRSİZ DİNAMİK ORTAMLARDA ROBOT SEYRÜSEFERİ İÇİN MUFREDATLI ÖĞRENME

Devran DOĞAN

Yüksek Lisans, Bilgisayar Mühendisliği

Danışman: Dr. Özgür ERKENT

May 2024, 66 sayfa

Çalışmamızda, Derin Pekiştirmeli Öğrenme (DRL) ajanlarının eğitim sürecini daha da kolaylaştırmanın ve verilen görevlerde başarı oranını optimize etmenin bir yolunu bulmak istedik. Yakınsama hızını artırmak için müfredat öğrenme tekniklerini benimsedik. Otomatik araçların önemi her geçen gün artmakta ve bilinmeyen ortamlarda hedef arama gibi yetenekler daha fazla ilgi görmektedir. Bu da bizi, insan hayatının risk altında olduğu veya insanların bulunduğu ortamlarda hedef aramanın ve rota oluşturmanın ne kadar önemli olduğuna getiriyor. Bildiğimiz üzere karmaşık gerçek dünya uygulamalarında, güvenlik ve risk farkındalığı kaçınılmaz öneme sahiptir. Risk bilincine sahip sistemleri, bilinmeyen farklı ortamlarda, modelin yer belirlemede ve rota oluşturmada güvenilirliliğini test etmek için kullandık. Ayrıca eğitim sırasında karşılaşılmayan durumlarda modelin performansını görmek için de kullandık. Risk bilincine sahip olmayan sistemler, başarısızlıklara yol açacak optimal olmayan kararların alınmasına yol açabilir. Bu rota oluşturma çabaları yüksek hesaplama süresi gerektirir. Çarpışmasız bir şekilde, statik ve dinamik engeller arasında hareket edebilen, istenilen risk düzeyine, yüksek performans ve uyum yeteneğine sahip riske duyarlı, risk farkındalığının artırıldığı karar verme süreçlerini geliştirmek zorundaydık. DRL algoritmaları, öğrenme ve ödül sinyallerini hesaplama konusunda yeteneklerini gösterdiler.

Ancak, uzun eğitim süreleri gerektirmeleri nedeniyle gerçek dünya uygulamaları için sınırlı uygulama alanına sahipler. Bu nedenle bizde Müfredat öğrenme ve DRL algoritmalarını kullanarak hedef odaklı bir model oluşturduk. Bunu yaparak daha hızlı yakınsama elde ettik, hedef arama süresini aynı miktardaki eğitim sayısı içerisinde azalttık. Çarpışma oranı azaldı. Eğitim sürecinde eğitimin hangi sırayla gerçekten zorlaştığını anlayabilmek adına, farklı yöntemlerle yapay ağ parametrelerine gauss gürültüsü enjekte ettik, farklı ortamlar kullandık, ajanın davranışını, tahmin başarısını görmek için sensör bilgilerini geciktirdik ve sadece statik engeller kullanarak, dinamik engeller kullanarak ve son olarak her iki engeli birden kullanarak sistemi test ettik. Simülasyon ortamlarının çoğu kısmen gözlemlenebilirken, tamamen gözlemlenebilir ortamlarda da sistemi test ettik, ancak DRL ajanlarının bu ortamları kolayca çözebildiğini gördük.

Bu çalışmayı yapabilmek ve verimliliğini ölçmebilmek için 2 boyutlu bir simülasyon ortamı oluşturduk. Performansı, simülasyon sonuçlarının analizi ile doğrulandı. Ajanın verimliliğini toplam çarpma oranı metrikleri ile ölçtük. Deneyler, müfredat öğrenme yöntemi ile eğitilmiş ajanın daha iyi başarı oranına ulaştığını, daha iyi kontrol sağladığını, gürültülü koşullar altında daha iyi performans gösterdiğini ve bilinmeyen ortamlara daha hızlı uyum sağlayabildiğini göstermektedir.

Keywords: Derin Pekiştirmeli Öğrenme, Hedef Arama, Riske Duyarlı Kontrol, Hareket ve Yol Planlama, Otonom Sürüş, Belirsizlik Ölçümü, Kısmen Gözlemlenebilir, Otomatik Araç Kontrolü, Q-öğrenme, Engellerden ve Çarpışmadan Kaçınma, Bilinmeyen Ortam Keşfi, Risk Farkındalığı, Yörünge Planlama

CONTENTS

Page

TABLES

Page

FIGURES

Page

ABBREVIATIONS

DRL	:	**D**eep **R**einforcement **L**earning
DQN	:	**D**eep **Q**- **N**etwork
RL	:	**R**einforcement **L**earning
A3C	:	**A**synchronous **A**dvantage **A**ctor-**C**ritic
MDP	:	**M**arkov **D**ecision **P**rocess
CL	:	**C**urriculum **L**earning
ACL	:	**A**utomated **C**urriculum **L**earning
DDPG	:	**D**eep **D**eterministic **P**olicy **G**radient
TRPO	:	**T**rust **R**egion **P**olicy **O**ptimization
DDQN	:	**D**ouble **D**eep **Q**-**N**etwork
DNN	:	**D**eep **N**eural **N**etwork
SL	:	**S**upervised **L**earning
MPC	:	**M**odel **P**redictive **C**ontrol
LSTM	:	**L**ong **S**hort **T**erm **M**emory
GRU	:	**G**ated **R**ecurrent **U**nit
UCB	:	**U**pper **C**onfidence **B**ound
SPL	:	**S**elf **P**aced Curriculum **L**earning
ICR	:	**I**nstantaneous **C**enter of **R**otation

1. INTRODUCTION

Deep reinforcement learning showed fascinating results when it comes to playing video games [1] not only that also for control problems in robotics [2, 3, 4, 5]. DRL algorithms struggle when there is a delay in the feedback signals or when the rewards are sparse. One of the ways to cope with this problem is to have a custom reward mechanism, designed with prior knowledge, a reward function that guides its agent in its learning process [6]. The other method for dealing with this is to increase the level of difficulty in the environment to make learning more efficient. This approach is called curriculum learning [7, 8]. These approaches consume time, require domain knowledge, and may be hard to apply in complex algorithms. In this study, we have a simple method that changes the degree of difficulty in the training and also testing parts of this study to make the agent take a curriculum learning approach.

The methods in this thesis use a DQN agent, under challenging and changing circumstances. Events can undermine the performance of the algorithm a lot. The idea is to see which order of difficulty makes the agent more robust, and risk-aware and even make it converge faster to increase its performance. When an agent sees some scenes that are easy to complete (e.g. just going to the closer target without dealing with the dynamic object), it becomes clear that the agent's job is to reach the target without colliding with anything else, then the agent learns to control its actions better under challenging situations. That's why the system shows a simple form of automated curriculum learning. To increase the complexity of the problem. We injected noise to create uncertainties. Uncertainty in general curriculum learning scenarios is not considered, because curriculum learning is usually applied to ease the learning process. Our study aims to find more challenging ways for the agent, we also want to find out which is most difficult. We want to see if the curriculum or no-curriculum setting is more robust to the injected noise.

Our goal is to detect an approach that makes the agent easily learn, and adapt, be more robust, and also, good at generalization. Rather than just expecting algorithms to figure out how to behave under changing and challenging environments we applied a simple curriculum

learning approach. We use our method in a custom Gym environment. The only input is sensory information, that gives distance information about the objects in the environment. We used the Vanilla DQN algorithm in our study, but any reinforcement learning algorithm can be used [9]. Future studies can benefit from our findings, for building easily adapting agents under challenging environments, even maybe games like StarCraft. The challenge of mastering StarCraft is still considered unsolved [10]. We expect that our algorithm is going to show a way to train agents in a way that agents' success rate will increase even if the rewards are sparse. If the environment is exceptionally noisy, if the information is delayed, and the agent has to decide on its own.Since our algorithm is simpler, it doesn't store any information, because we automate the position of the destination. That makes our method memory efficient, as we don't need storage of previous states. This study can be found at https://github.com/devran1/Curriculum-Learning-For-Robot-Navigation-In-Dynamic-Environments-With-Uncertainties

The structure of the paper is as follows: We discuss and show previous, and related works in Section II. After explaining what we did in Section III. We demonstrate and give the experimental outcomes under different environments in section IV, and Section V is for discussions and the conclusions.

2. BACKGROUND AND RELATED WORK

This section discusses the state-of-the-art estimation methods, learning methods such as curriculum learning, obstacle avoidance, motion planning, noise injections as adversarial attacks, and risk awareness.

2.1. Deep Reinforcement Learning

RL and DRL methods use environmental feedback (rewards) to learn to choose actions. This environment feedback can come in many different ways, but popular ways of doing this are using images that are directly taken from Atari games [11], first-person shooter games [12, 6, 8], and car racing games [9].

Reinforcement Learning discovers optimal behavior with trial and error interactions with the environment. Q-learning algorithm is a widely used popular RL algorithm [13]. Just like every other RL algorithm, Q-learning has no previous knowledge of the environment, and it learns by interacting with it. Because of its simplicity, Q-learning is used to solve many problems. However, it is not useful if the number of states or actions is too big, as it becomes intractable for problems with a large number of states. Therefore, neural networks are combined with RL to make it more efficient. That further increased the system's generalization ability. If we compare supervised methods with RL, we can see large amounts of labeled data is not necessary, and for that reason RL algorithms are chosen for robotics fields such as path planning [14], and obstacle avoidance [15].

DRL methods can vary in their structure, but mainly they are different versions of the DQN algorithm [11] or actor-critic algorithms such as (A3C) that use parallel actor learning methods [9]. DRL algorithms use neural network structures for function approximation in complex or big RL problems. DRL agents perform sequences of actions "a" in the given states "s" for the given environments, they maximize the total amount of returns also known as rewards "r" [16]. These problems are known as Markov Decision Processes (MDPs), these processes use transition functions to calculate the probability of the next state "s'" from the

current state "s". The crucial need for a DRL algorithm is to have regular and easy-to-obtain rewards that come from the environments to converge to an optimal behavior. If the rewards are not frequent enough, like in Montezuma's Revenge game, DRL algorithms such as DQN and A3C cannot succeed [11, 9]. Montezuma's Revenge game is difficult because it has sparse rewards. That means it is only rewarded when it finishes a task. Therefore, policy generation becomes hard. The games have a complex environment, the agent needs to go to a certain location pick a key, and do other tasks. In our environment we have sparse rewards, agent can only receive rewards when it reaches the destination. This is also similar to a real-world scenario for a robot. In the real-world, robots generally encounter rewards sparsely. This is one of the reasons why we use curriculum learning, to make it easy for the agent to learn the task of reaching the destination. However, compared to Montezuma's Revenge our environment is not that complex.

This sparsity in rewards can be compensated by reward shaping or adjusting the difficulty level of the environment if possible. In the case of reward shaping, a reward function is used by utilizing domain knowledge [6, 17], also changing the level of difficulty considered to be reward shaping [7, 8]. There is another method that deals with the sparsity of the rewards called Power Play, that explores the new unsolved problems while the algorithm is trained to match the difficulty level of the unknown environment [18]. One other related method is hierarchical learning, that designs or controls the policies and divides the policies into sub-policies to match the number of goals for training each policy on each sub-goal [19, 20].

2.1.1. Advantages of Reward Shaping

Reward shaping allows experts to provide additional guidance to the learning agent by designing reward functions that reflect desired behaviors. Providing intermediate rewards can guide the agent toward the desired behavior more quickly compared to sparse reward settings. Reward shaping could introduce uncertainty if the domain knowledge used to shape rewards is imprecise or incomplete. In such cases, the agent might learn suboptimal behaviors based on the flawed reward function.

2.1.2. Advantages of Adjusting Difficulty Levels

By adjusting the difficulty level, developers can control the pace of learning and ensure that the agent is sufficiently challenged without being overwhelmed. That can promote continuous learning. Gradually increasing difficulty levels can facilitate a smooth transition from simple to complex tasks, promoting continuous improvement. If the difficulty level is not appropriately calibrated, it could either lead to the agent getting stuck in suboptimal solutions or facing insurmountable challenges, hindering learning.

2.1.3. Advantages of Hierarchical Learning

By decomposing tasks into sub-goals and training policies for each sub-goal, hierarchical learning simplifies learning in complex environments. Learned sub-policies can be reused or adapted for related tasks, promoting scalability and transfer learning. The hierarchical structure may introduce uncertainty in the coordination and communication between different levels of policies, potentially leading to suboptimal decision-making.

Uncertainty should be introduced carefully, considering the specific characteristics of the environment and the learning algorithm. It can be beneficial when exploring unknown or dynamic environments, as it promotes adaptive learning and robustness. However, excessive uncertainty or poorly managed uncertainty can lead to instability, suboptimal learning, or even failure. Understanding the trade-offs and balancing uncertainty with stability is crucial for effective reinforcement learning. Introducing uncertainty can make the learning process more complex, requiring sophisticated algorithms and strategies to manage it effectively. Excessive uncertainty may lead to unstable learning dynamics, making it difficult for the agent to converge to optimal solutions. Uncertainty may obscure the underlying reasons for the agent's behavior, making it challenging to interpret and debug the learning process. We introduce two types of uncertainty first we add noise in the training and testing parts. Second, we delay input to the DQN algorithm. In both cases, we continue to increase the levels of

uncertainty and expect to see what levels of uncertainty are more challenging and what level of difficulty is enough to challenge the agent.

2.2. Curriculum Learning

Humans are exceptionally good at learning with self-study because the learning difficulty increases as we continue learning. Curriculum learning tries to mimic that. RL methods are a powerful framework for robots to learn from the environment they are in. Nevertheless, using DRL methods in real-world conditions brings a lot of challenges. Curriculum learning methods are used for different reasons, these reasons are starting with improving the performance on a given task set, to using CL to increase the sample efficiency for the performance of the algorithm [21, 22, 23].

2.2.1. Solving harder tasks

If the target task is not achievable directly, CL algorithms can guide the agent in gradually increasing difficulty in solving the complex problem. Recent works involve DRL agents solving mazes from easy to hard order [24], or in robotic control problems [25, 26], video games [27]. Some studies directly try to solve the sparse reward problems with CL [28, 29, 30, 31, 32].

2.2.2. Training general learner agents

These agents are able to solve problems they have not seen before or solve problems in different environments [33]. CL also becomes really useful with the transition between simulation to the real world [34, 35] and also increases the robustness in multi-agent algorithms by Self-Play [36, 37, 38, 39, 40].

2.2.3. Multi-goal agent training

In multi-goal agent settings, the tasks of each agent may vary according to the needs of the problem. Since goal-conditioned policies are generated differently by each agent each robot may have its own set of behaviors that can be further enhanced by the CL algorithm. Multi-goal robotic arm manipulation is investigated in [2, 41, 42, 43, 44, 45, 46] and also multi-goal navigation [47, 48, 49, 50].

2.2.4. Automated Curriculum Learning

One example of an automated curriculum is self-paced curriculum learning (SPL). That is used in supervised learning settings. SPL has a training set $(x_i, y_i)_{i=1}^{D}$, x is the data sample and y is the target value, and D is the size of the data. SPL minimizes the loss in episode e,

$$L(\xi, \omega; e) = \frac{1}{D} \sum_{i=1}^{D} \omega_i l(f_\xi(x_i), y_i) + R(\omega; e) \tag{1}$$

$$R(w; e) = -\lambda(e) \sum_{i=1}^{D} \omega_i \tag{2}$$

Where f_ξ is the network parameter, parameterized by ξ, l represents the loss function, w_i is the set of weights, $R(\omega; e)$ is the regularization term, keeps ω_i away from becoming 0. In the original SPL l_1 norm is used. $\lambda(e)$ is a scheduler, which determines the difficulty of training, and w_i^* is given:

$$\begin{cases} 1 & \text{if } l(f_\xi, y_i) \leq \lambda(e) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

As an intuitive explanation, in each training step, loss is not only used for optimization but also used as a difficulty measure. Where $\lambda(e)$ is a threshold. When training continues $\lambda(e)$

increases and adds more difficult data. However, utilizing the loss function for difficulty measure [51, 52], has limitations. These limitations can be summarized in two aspects. First, loss measures the difference between the truth and the prediction, for that reason it does not represent the difficulty in the data independently from truth values. Secondly, the loss can change per various challenges. Therefore, $\lambda(e)$ needs to be designed for the specific needs of the problem.

Other studies such as [53] worries about the uncertainty in the labels, they call this label noise and utilize curriculum learning for starting the training with highly reliable samples. They believe this can reduce the bias.

In the case of RL, and robotics we can see similar ACL methods. [54] shows ACL on control problem, by demonstrating increasing the performance of robotic grasping. Their study focuses on creating a curriculum over tasks and goals that must be solved. Among other techniques, the reverse curriculum technique [52] is used. In this technique, the agent is resettled to the states that are closer to the goal, similar to our study, but we did not move the agent, we moved the destination closer to the agent, so that the agent and its learning process would not be disturbed, and the agent learns to go to the destination itself. Since resetting to easy/closer states is not general purpose, [34] tackles this problem, saves the history based on the previously stepped states, and calls it "hindsight". When the performance increases, the footprint becomes clearer, and the solution to complex tasks is stored. Recent studies created explicit curriculums for different tasks [33, 48], and storing the successful attempts in RL is also investigated in the case of "self-play" [55], where more than one player played against each other. This technique is applied in GO game [36] and DOTA [56]. For robotic control problems [38] used "self-play" methods to achieve locomotion behavior.

Just like we said before, our algorithm doesn't need to store any information, because we simply automate the position of the destination in three steps. First, it is closer to the agent, it is easy to reach. In the second step, we put it behind the dynamic object, and expect the agent to reach the destination. Agent collides with the dynamic object and learns to avoid the obstacle to reach the destination. In the last part, we put the destination at the right top

corner. The agent learns to avoid colliding with the dynamic object, but this time also needs to learn not to collide with the static obstacle and the boundary of the environment. It is not easy for the agent to go to this tunnel-like area and reach the destination (see Figure 4.1). We measure the agent's behavior, a with hit-success metric.

2.2.5. Entropy-Based Methods

Curriculum learning methods and the MaxDiff RL framework both aim to enhance the learning efficiency and performance of reinforcement learning (RL) agents through distinct mechanisms [57]. The MaxDiff RL framework focuses on overcoming the limitations posed by temporal correlations in RL environments by decorrelating agent experiences, ensuring that data used for learning are as close to Independent and Identically Distributed (I.I.D.) as possible. This approach leverages the statistical mechanics of ergodic processes to achieve better exploration of the state space and more robust learning outcomes, less sensitive to initial conditions and random seeds, thus providing more reliable and consistent learning outcomes. [57]. On the other hand, a novel approach in RL employs Tsallis entropy for regularization, introducing Tsallis Actor-Critic (TAC) methods and a curriculum learning strategy termed TAC with Curricular (TAC2) [58]. TAC2 incorporates curriculum learning by gradually increasing the entropic index q of the Tsallis entropy regularization to balance the exploration-exploitation trade-off more effectively over time. Starting with a lower q to encourage exploration and gradually increasing it to shift towards exploitation, TAC2 dynamically adapts the learning process without the need for an exhaustive search for the optimal q. This integration of curriculum learning principles into entropy regularization offers a unique approach to enhancing sample efficiency and managing the exploration-exploitation trade-off in RL.

The MaxDiff RL framework allows agents to learn effectively from a single episode, enhancing sample efficiency and reducing the need for extensive data [57]. While curriculum learning reduces the complexity of initial tasks and adapts task sequences

based on the agent's performance, MaxDiff RL inherently adjusts the sampling process to reduce correlations, specifically targeting the robustness of learning outcomes across different initializations and random seeds. Similarly, the TAC2 method aims to improve sample efficiency by starting with a more exploratory policy and gradually refining it. By dynamically adjusting the entropic index q, TAC2 achieves performance comparable to TAC with an optimally determined q, but with better sample efficiency. This gradual adjustment helps in systematically managing the exploration-exploitation trade-off, potentially leading to better learning outcomes [58]. Traditional curriculum learning often requires manual design or complex automated systems for task sequencing, whereas TAC2 simplifies this by dynamically adjusting a single parameter, the entropic index q, integrating the exploration-exploitation balance directly into the learning algorithm. This adaptive mechanism allows TAC2 to potentially respond more fluidly to the learning process compared to fixed or manually designed curricula in traditional methods. Both approaches aim to enhance sample efficiency and learning stability, but TAC2's method of entropy regularization offers a unique and straightforward curriculum design.

Overall, Curriculum learning guides the learning process through structured task sequences, whereas MaxDiff RL focuses on improving the statistical properties of agent experiences by decorrelating them. These approaches can be complementary, as combining structured task sequences with decorrelated experience sampling could further enhance RL performance and robustness. Similarly, traditional curriculum learning methods and the TAC2 approach both aim to enhance learning efficiency and performance, but TAC2 leverages entropy regularization and dynamic scheduling of the entropic index as an innovative strategy to achieve these goals. Integrating curriculum learning principles into the entropy regularization framework of TAC2 offers a novel perspective on managing the exploration-exploitation trade-off in RL, providing a unique mechanism to enhance sample efficiency and learning outcomes.

2.2.6. Intrinsic-Based Methods

In [59] they outline two distinct methodologies in reinforcement learning (RL): Curiosity in Hindsight and Autotelic Agents [59]. Curiosity in Hindsight drives exploration by rewarding the agent based on the discrepancy between predicted and realized outcomes, leveraging predictive error to encourage the exploration of novel states. This approach is particularly robust in stochastic environments, helping agents distinguish between predictable and unpredictable elements and avoiding the ”noisy TV” problem where agents get stuck in high-entropy areas. Utilizing hindsight to reassess past experiences, it combines the benefits of curiosity-driven exploration with learning from past actions, creating a versatile and scalable method applicable to various environments. However, balancing intrinsic and extrinsic rewards is complex and task-dependent, with excessive focus on intrinsic rewards potentially leading to inefficient learning. This method enhances performance by ensuring intrinsic rewards diminish in predictable scenarios, thereby effectively guiding exploration in environments with inherent stochasticity. Unlike traditional curriculum learning methods, Curiosity in Hindsight does not explicitly sequence tasks but relies on intrinsic motivation to explore and learn [60]. In contrast, autotelic agents use techniques to automatically shape intrinsic rewards, encouraging exploration based on the agent's ongoing learning dynamics rather than structured task difficulty, thus providing a robust intrinsic motivation framework for effective environmental exploration [60]. Curiosity-driven exploration provides intrinsic rewards based on the agent's uncertainty or prediction error, encouraging exploration of novel states, unlike curriculum learning which structures the learning process through predefined tasks [60]. other methods such as Count-based exploration further incentivize the agent to explore less visited states by providing intrinsic rewards based on state visit counts, contrasting with curriculum learning's approach of starting with easier tasks irrespective of state visit frequency [60, 61].

Curiosity in Hindsight uses intrinsic rewards based on predictive errors to drive exploration, adapting effectively to the environment's stochasticity by distinguishing between predictable

and unpredictable dynamics. In contrast, curriculum learning structures the learning process by adjusting task difficulty and guiding the agent through progressively challenging tasks. While Curiosity in Hindsight is designed to handle stochastic environments, curriculum learning manages complexity by breaking down tasks into simpler, more manageable steps. Both methods aim to improve the scalability and generalization of learning: Curiosity in Hindsight achieves this through modifying intrinsic reward mechanisms, and curriculum learning does so by structuring the learning sequence. Curiosity in Hindsight involves sophisticated modeling to disentangle noise from novelty, requiring more complex implementation compared to the relatively straightforward task sequencing in curriculum learning. Curriculum learning focuses on building upon simpler skills to tackle more complex tasks, enhancing exploration through internally generated rewards, and encouraging the agent to learn from novel experiences. It requires careful task sequence design and may need expert knowledge, whereas intrinsic reward shaping, such as the AIRS framework, necessitates sophisticated mechanisms to balance and adapt intrinsic rewards. While curriculum learning can significantly improve learning efficiency and task generalization, intrinsic reward shaping excels in exploration-heavy tasks and environments with sparse rewards.

In summary, Curiosity in Hindsight focuses on enhancing exploration efficiency in stochastic environments through intrinsic motivation, while curriculum learning improves learning stability and efficiency by structuring the task sequence. Both methods have their strengths and can be complementary in developing robust reinforcement learning agents. In conclusion, while both curriculum learning and intrinsic motivation methods aim to improve learning efficiency, they differ fundamentally in their approach: curriculum learning relies on a structured sequence of tasks to guide learning, whereas intrinsic motivation methods focus on encouraging exploration and learning from the environment without predefined tasks.

2.2.7. Uncertainty-Based Methods

Goal-Conditioned Reinforcement Learning (CQM) and LOGO (Learning Online with Guidance Offline) represent advanced strategies in reinforcement learning designed to enhance exploration and policy optimization in complex environments [62, 63]. CQM leverages a Vector Quantized Variational Autoencoder (VQ-VAE) to condense high-dimensional observations into a manageable goal space, facilitating scalable exploration through curriculum goals that balance uncertainty and temporal distance. This approach enables autonomous goal definition directly from raw observations, bypassing the need for predefined goal spaces and prior knowledge. In contrast, LOGO utilizes Trust Region Policy Optimization (TRPO) to integrate offline demonstration data, generating and refining candidate policies based on state-action pairs rather than explicit rewards. This method ensures guided exploration in sparse reward environments, enhancing scalability in high-dimensional problems. Complementing these approaches, Latent Bayesian Surprise (LBS) focuses on intrinsic rewards by computing Bayesian surprise in the latent space to drive exploration toward uncertain states, avoiding the 'NoisyTV problem' and ensuring computational efficiency [64]. Meanwhile, DIAYN (Diversity is All You Need) promotes diverse skill acquisition by maximizing mutual information between skills and states and minimizing it between skills and actions, fostering extensive state space coverage [65]. Both DIAYN and Curriculum Learning methodologies involve structured exploration, with DIAYN emphasizing diverse skill development and Curriculum Learning progressively increasing task complexity, incorporating expert demonstrations to guide the learning process. These innovative techniques collectively broaden the applicability of reinforcement learning in diverse, uninformed environments, reducing the dependency on detailed prior knowledge and manual goal specification.

Curriculum Learning and LOGO offer distinct strategies for enhancing exploration in reinforcement learning. Curriculum Learning relies on generating intermediate goals and uncertainty-based guidance, often requiring domain knowledge to map observations to a

semantic goal space, which poses challenges in high-dimensional spaces due to the need for manually specified mappings and extensive online interactions. In contrast, LOGO leverages offline demonstration data to guide exploration, reducing the initial burden of exploring vast state-action spaces and eliminating the necessity for predefined intermediate goals. This makes LOGO more scalable and practical in environments where extensive online data gathering is difficult or expensive, as it adapts policies using TRPO and a small amount of offline data. Meanwhile, Latent Bayesian Surprise (LBS) emphasizes intrinsic rewards to drive exploration, motivating the agent to seek states that maximize information gain and improve its understanding of the environment. Unlike Curriculum Learning, which structures the learning process by starting with simpler tasks and gradually increasing complexity, LBS drives exploration based on the model's uncertainty, avoiding frequent model updates by using the latent space for computing Bayesian surprise. Curriculum Learning's computational demands depend on the complexity and number of tasks in the curriculum, whereas LBS maintains computational efficiency by minimizing the need for frequent updates.

Both Goal-Conditioned Reinforcement Learning (CQM) and LOGO represent innovative approaches to tackling the challenges of exploration and learning in complex reinforcement learning environments with sparse rewards. CQM advances traditional curriculum learning by concurrently defining a semantic goal space and suggesting curriculum goals, effectively addressing scalability issues and reducing the need for extensive prior knowledge. It excels in handling high-dimensional observations and maintaining temporal relations between goals, facilitating efficient exploration and control in diverse tasks within complex environments. In contrast, LOGO leverages offline demonstration data to guide initial exploration, enhancing scalability in high-dimensional spaces and providing robust performance guarantees, particularly beneficial in scenarios where data availability is limited and rewards are sparse. Together, these methods illustrate distinct yet complementary strategies for optimizing learning and exploration in reinforcement learning, promising significant advancements in real-world applications where challenges such as sparse rewards and complex environments

are prevalent.

2.3. Obstacle Avoidance

Human tasks can be done with automated robots. This saves manpower, time, and costs. Also, the success rate of the task can be improved. While the mission is carried out, path generation and planning are a must for avoiding collisions or is a must for the avoidance of moving into restricted areas. In recent years, unmanned vehicles, or drones, have started to be used in missions like navigation in uncertain and unknown environments. Examples can be disaster monitoring [66, 67], border control [68, 69], and search and rescue operations [70, 71].

A safe, reliable, collision-free path is necessary to complete the mission. This also involves static obstacles such as walls, trees, and buildings. Therefore, autonomous control needs to improve its operational efficiency, ensure safety, and achieve intelligent control. Studies show that agents need to be more intelligent to understand which actions can achieve better results, shorter, or faster paths, and the way to reach that intelligence is to use learning-based algorithms like RL algorithms [72]. But, earlier studies needed to have a model of the problem, and achieved under-performance when environment information was not obtained.

In [73], a simple DQN algorithm is utilized for building a path-planning algorithm for an uncertain environment. In [74], Q-learning and neural networks are combined with the Adaptive and Random Exploration approach (ARE) to complete navigation tasks and obstacle avoidance. In [11] advantages of DQN are used to solve the difficulties in generalization to reach human level control. [75] Takes the features of the environment as input to the DQN algorithm, and utilizes an end-to-end trainable framework for robotic planning. Then in [76] DQN algorithms had visual input, such as high-dimensional images, and got better results. Also, computer vision based features become popular for training autonomous vehicle algorithms to avoid obstacles [77, 78].

In DRL, deep learning is used as policy, also the noise injections are commonly used to increase the exploration rate of the agent. Noisy Network (Noisy Net) is a great example of this, noise is added to the parameter space and tuned [79]. Moreover, in [80], they used Gaussian noise in the agent's input parameters in both off and on-policy methods such as DQN, DDPG, and TRPO algorithms. However, there were no studies in autonomous driving that investigated the performance of the model with various noise injections. This paper investigates noise injection and its effects on the DQN algorithm for autonomous driving tasks. We used the DQN algorithm for obstacle avoidance and path planning. The agent has a chance to collide on the flight, therefore, it has to avoid the obstacles to reach the destination. For the obstacle avoidance experiments, we used simple distance measurement sensors.

2.4. Noise Injection

Parameter space noise adds the noise directly to the agent's parameters for encouraging exploration. In RL algorithms, noise injection is applied generally to make the agent explore more. Noise added to the parameter space is the way to directly add noise to network parameters. But another reason to apply noise is to measure how robust the DRL method is. Our reason for using noise was to challenge the agent, not only in new environments but also for every step taken in the environment.

In [80] noise is used in DQN, DDPG, and TRPO. In their study, the noise scale is adaptive to the time, also adaptive to the variance in action space. That means the noise levels are tuned and not that high, not challenging enough. Also, the parameter noise surpasses the traditional action space noise, if the tasks have sparse rewards [80]. [79] also utilizes a similar parameter space noise for a better exploration called Noisy Network (Noisy Net). Noisy net is a DRL algorithm that adds parametric noise to its weights. Injected noise is used to tune the RL algorithm, that can replace the hyperparameter tuning. The study applies the noise to DQN, DDQN, and A3C, and proposes improved performance compared to the baselines

[79]. Studies show that noise injection in the DRL algorithm outperforms the agents that don't have noise [78, 17, 81].

2.4.1. Gaussian Noise Injection

Adding noise increases the exploration rate of the algorithm, that's a known phenomenon. The skill of dealing with uncertainty or noise is one of the main powers of RL methods. The uncertainties arise from the unknown situations or unknown environments that are observed as input data. [79] utilizes Noisy Net to increase the exploration, this is injecting noise into the neural network weights. By perturbing the parameter space Noisy Net explores the environment more because this method also injects randomness into the policy. Noisy Net has a noisy layer instead of linear layers, and each weight of the neural network has zero-mean Gaussian noise. The noisy layers are represented as follows:

$$y = (\mu^{\omega} + \sigma^{\omega} \bigodot \varepsilon^{\omega})x + \mu^{b} + \sigma^{b} \bigodot \varepsilon^{b} \tag{4}$$

where ω, is the weight, b is the bias of the neural network, and x is the input to the layer. σ is the Gaussian noise standard deviation, that is learned with the weight. ε are unit Gaussian random variables and $\bigodot$ means element-wise multiplication.

We do not have a noisy layer, we just add noise to the input parameters of the DQN agent. Also, we have noticed more exploration doesn't mean more safety, for that reason, we need to work on risk-awareness. In this study, several of the environments are unknown to the agent, and there is external noise for robustness. Compared to the classic DQN, the noisy method can get a higher success rate.

2.5. Robustness to Adversarial Uncertainty

The lack of robustness in DNN algorithms limits their applications for safety-critical domains, such as collision avoidance. Also, even small changes in the input, such as

adversarial attacks can cause wrong decisions to be taken by DNNs [82, 83]. Moreover, recent studies have shown the danger of adversarial attacks in real-world applications [84, 85], for example, autonomous vehicles can go into another lane. Our work not only considers the DRL algorithms' lack of robustness against input uncertainties but also shows the amount of uncertainty to achieve the robustness.

Our study extends the understanding of efficient neural network robustness analysis [86, 87, 88] for DRL tasks. In RL, techniques of robustness analysis in SL just flag the nominal actions as non-robust if the minimum input is perturbed higher than the threshold given, such as the system's chosen level of uncertainty. These techniques cannot reason for different actions.

The robust DNN algorithms in real-world uncertainties [82] motivated us to study adversarial attacks in the learning tasks of the DQN algorithm. The following part of the study summarizes adversarial attacks in DRL (check [89] for a complete survey).

2.5.1. Adversarial Attacks in DRL

Adversaries act against the learning agent by exploiting its weaknesses in the observation or transition models of the environment. Many of the adversarial attacks are mainly against supervised methods. The same attacks can be applied to RL methods. The study of [90] uses an RL agent to play Atari. Manipulating the input of the bot, [90] shows its performance can decrease by as high as 50%. In this thesis, the adversarial method we used is adding noise to the input parameters of the DQN agent, not only in a constant manner but also we inject increasing levels of noise.

2.5.1.1. Observation Models The techniques for attacking SL networks through small image perturbations [91] can be managed to attack the inputs of the DRL policies. Recent works demonstrate how to specifically use adversarial attacks against a DQN algorithm [90, 92]. Another work shows adversarial and transition perturbations [93].

2.5.1.2. Transition Models These are attacks on an RL agent by changing the physical parameters of the simulator, such as friction, or center of gravity and mass, between training episodes [94, 95]. Other attacks change the transition model between the steps of episodes, such as using disturbance forces [37, 96], or by adding a second agent for a competitive environment. In [97], the second agent manages to visually distract the first agent, rather than applying forces and becomes an observation model adversary technique. Thus, the adversary of a second agent also called the multi-agent games, investigates the adversarial behaviors beyond the scope of this study, we investigate the robustness with frequent and increasing noise injections.

2.5.2. Defense Toward Adversarial Attacks

Many ideas from SL were also used in DRL to build defenses against adversaries, such as training in adversarial environments [81] using or using ensemble models [94, 95]. Furthermore, because adversarial observation models are a form of uncertainty in measurement, there are tight connections between the risk-sensitive [98] and adversarial robustness of DRL. Many of the risk-sensitive DRL models optimize the rewards under the assumptions of environmental stochasticity, rather than optimizing for the expected reward. At the end the policies are more risk-sensitive, robust to stochasticity in the input space, such as sensory noise, but still can fail on hand-crafted adversarial attacks. In other words, modifying the RL algorithms to directly arrange a robust algorithm remains challenging. Instead, in our work we not only train with noise we also test with and without noise. Also we have trained without noise and tested with noise (further details can be found in the Method section of this study).

2.6. Risk Awareness

Autonomous driving and trajectory planning received great progress in recent years [99]. But, existing algorithms and methods are designed for specific scenarios, that are not dealing with uncertainty. Current DRL methods maximize the expectation of total returns

and are oblivious to the risk of catastrophic events. However, instead of utilizing RL for safety-critical robots, focusing on achieving high expected returns makes working on risk and decision-making under uncertainty a crucial challenge.

The robotics field benefits a lot from risk-aware technologies, for example estimation of external disturbances [100, 101], collision estimates [102], and dealing with perception [103]. It is not recent that risk-awareness has become one of the hot topics in the field of autonomous driving [104]. Earlier studies utilized risk-awareness with MPC together for motion planning problems in autonomous driving [105]. It is shown in [106] that risk-awareness can be applied as rule-based specification. In [107], an uncertainty algorithm is built for quantifying risks.

RL-based robot navigation methods are still an active study field because the capability of generalization and robustness is not fully achieved [108], [109]. Several obstacle avoidance, and navigation DRL algorithms also arose to show the risks and uncertainties in the environment. As an example, [110] proposed to use of a neural network to estimate the collisions for obstacle avoidance tasks, using MC-dropout [111] and bootstrapping [112] to predict the uncertainty of the model. Moreover, [113] estimates the increase of uncertainty in novel dynamic scenarios by introducing LSTM, by doing that, they become capable of using the history of the actions. [114] applies a model-free policy for action selection, a GRU approach [115] to estimate the uncertainty in the observations and utilizes the estimated variance to regulate the stochastic policy. But, these algorithms either use MPC for selecting the actions, these methods are still computationally expensive, or they use an additional estimation model to predict the uncertainty.

We are seeing really complex exploration algorithms and they show difficulties they faced with RL algorithms [29]. One of these algorithms uses uncertainty estimation as the measure of novelty, and surprise encourages the agents to search for new policies. These approaches on uncertainty estimation use thresholds such as upper confidence bounds (UCB) [116], also curiosity exploration [29]. Increasing the stochasticity of the model seems to be a popular solution. Even though the limitations still exist for uncertainty estimation methods, such

as generalization. Count-based algorithms like UCB works really well at MDPs but cannot work very well in high dimensional problems. Pseudo counts [28] can reduce this problem. However, it seems like it needs to have an extra density estimation model. Moreover, prediction errors are utilized as rewards, it needs extra modules for auxiliary tasks. As an example, in [29], researchers use neural networks to estimate the difference between the true and the estimated next states.

To increase the safety in autonomous vehicles, researchers assign risk assessments to their RL algorithms [117]. [118] propose a model of combining DRL with a probabilistic-based risk assessment model. They use relative speed and relative distance as the inputs of the neural networks for their DQN algorithm. Collision risk with the environment or other vehicles are used to build a reward function, so they can have driving decisions with minimal risk. [119] shows us a universal risk-aware DQN algorithm for driving through unsignalized occluded intersections. But, we know distance and relative speed are not direct variables that humans use when they are driving, they use their eyes to estimate the risk of colliding with other vehicles and make decisions based on that [120]. For that reason, the risk assessment becomes the input variable of the DRL algorithm, that can have human-like driving capability.

Another way to reach a risk-sensitive RL is by addressing the worst-case scenario of the stochastic return instead of its expectation, but that can cause a conservative policy [121]. Recent studies offer to build a model for the future returns and produce multiple policies with different risk-sensitivities by changing the levels of the risk metric [122]. [122] gathers the stochasticity in collected returns by approximating the variance and the mean of a Gaussian distribution, and distributional RL reconstitutes the distribution of the returns [123, 124, 125]. Distributional RL can produce multiple policies with varying degrees of risk tendencies [124, 126, 127]. Distributional RL is used in autonomous driving under the occluded intersections [128] and mobile-robot navigation problems [129]. These algorithms learn a policy that can have different risk-tendency while training, but they still use a fixed risk-tendency for each task. But, professional drivers can not be as careful during driving in good weather conditions as when driving in stormy weather. To put it another way, the

degree of risk-tendency changes not only as a function of the task but also the real-time feedback of the environment. For reliable intelligent robots it is important to adapt and manage risk-tendency automatically on the fly.

Learning-based methods such as RL can deal with uncertainty, however they have problems with heavy computation and difficulty in generalization. Uncertainty can be achieved with new environments and can be computationally cheap to execute, also a good way to test the generalization. But for challenging the agent we applied external noise to new environments to see the response of the DRL agent. See if we can increase the efficiency of the decision making or find out which order of difficulty is hindering the learning efficiency of the agent. In our method, the risk and uncertainty are easily and efficiently implemented to the DRL. Such as delaying the observation info for risk-awareness, and checking the actions of the agent with the rate of collision is less expensive. Also, DQN is a computationally less expensive algorithm.

Uncertainty and the noise in the parameter space is disruptive for the trade-off between exploration and exploitation. Parameter space is full of information whether the model is partially observable or not. Localization and path planning are necessities of automated navigation in dynamic environments, we also need to track the obstacles to be able to avoid them. If sensor accuracy is limited the task can become very challenging. If there are obstacles that are hidden or blocked by other obstacles problems can increase and become a risk awareness problem. One of the main focuses of these algorithms is to find the shortest and sometimes the fastest trajectory to the target [130, 131, 132], but the other task of the estimation methods is localization [133, 134, 135].

Without a doubt, the rate of uncertainty becomes really important in high dimensional environments. How to benefit from uncertainty and uncertainty rate is a growing field of study. Exploration of parameter space is studied a lot in the literature. [79, 80] gained improvements in terms of performance with little changes in the original algorithms. Comprehensive studies in [136] shows that eliminating the Noisy Net from Rainbow DQN results in a considerable performance drop in some of the Atari games. But, directly adding

diagonal Gaussian noise to the parameters is not good enough, because this method ignores the correlation of the weights.

Other noise exploration strategies are the evolution strategies [137]. Evolution models establish the population by injecting noise to the parameters of the model, that explores the environment efficiently. Nevertheless, evolution algorithms do not reflect on the structure of the agent, causing poor sample efficiency. However, this is not the subject of this study.

In this study, the parametrization of the noise is built based on the Gaussian noise injection. This part of the study focuses on how resilient the curriculum algorithm is. Evaluation tool is the success rate of the training and testing. It is understood that the total reward is higher when the noise is around %30. Similar results found in other studies [138], Among other reasons we wanted to make the environments harder for the agent, higher levels of noise-injection can make the environment really hard for the agent. Not only that, we delayed the observations and we didn't give the sensory information of the dynamic obstacle, when the dynamic obstacle is behind a static obstacle.

Our review of existing literature reveals significant advancements in DRL, curriculum learning, obstacle avoidance, noise injection, and robustness to adversarial uncertainty. Reward shaping, adjusting difficulty levels, and hierarchical learning are crucial in DRL. Curriculum learning is particularly beneficial for solving complex tasks and training agents to generalize well. The challenges associated with obstacle avoidance and noise injection are pivotal to our study, as they underscore the need for resilient algorithms in uncertain environments. This background sets the stage for our proposed method, which aims to enhance the robustness and adaptability of DRL agents.

3. PROPOSED METHOD

3.1. Kinematic Model

Kinematics defines the rules of motion of objects. It only considers position, velocity, and acceleration. These equations describe how the state variables change over time. In a kinematic model, these equations are mostly derived from basic principles of motion, such as the equations of uniform acceleration for one-dimensional motion. Kinematic models may apply constraints that limit the motion of the system. These constraints could arise from physical limitations, such as obstacles in the environment, or geometric constraints, such as the range of motion of a joint in a robotic arm. Kinematic models are widely used in robotics, vehicle dynamics, and animation. In robotics, for instance, kinematic models are essential for planning and controlling the motion of robot arms or mobile robots. In vehicle dynamics, kinematic models can be used to predict the motion of vehicles under different driving conditions.

3.1.1. Linear movement

Linear movement refers to motion along a straight path, where an object or point moves from one position to another without deviating from that path. It is characterized by constant direction and speed. In linear movement, the object moves in a straight line from one point to another. The direction of motion remains constant throughout the movement. Linear movements can occur at a constant speed, where the object covers equal distances in equal intervals of time. Alternatively, it can occur at variable speeds, where the object's velocity changes over time. If the velocity is constant, the object moves with uniform linear motion. If the velocity varies, the object experiences non-uniform linear motion.

Acceleration: Acceleration is the rate of change of velocity over time. Positive acceleration occurs when the object speeds up, negative acceleration (deceleration) occurs when the object

slows down, and zero acceleration occurs when the object maintains a constant velocity. Our agent moves with uniform acceleration.

To set our vehicle in motion, instead of changing its position we increase the acceleration, then, apply the acceleration into the velocity, and subsequently, apply the velocity to determine the position. However, the movement is time-dependent, we need to add delta time to our formula.

$$\nu = a.dt \tag{5}$$

y component of the acceleration is zero, because the vehicle doesn't accelerate sideways.

3.1.2. Steering

Steering is a little bit complicated. First, we will assume the car can be approximated with the kinematic bicycle model. When the front wheels are steered, the vehicle will drive in a circular motion. A car with steering angle α will follow a circular path, around the instantaneous center of rotation (ICR). Where we show in Figure 3.1 with letter "C". Chassis of the car is represented with the perpendicular lines of the intersecting tires. We show this tire to tire distance $\overline{AB}$ with the letter "L".

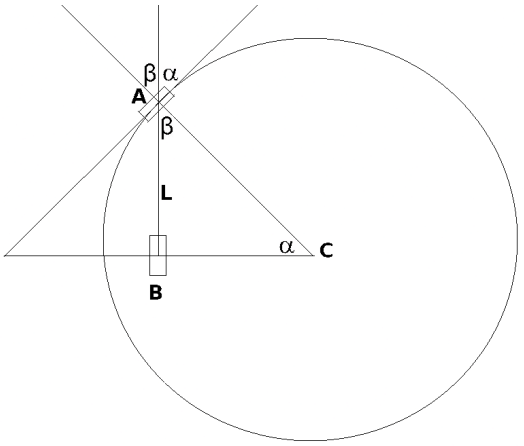

Figure 3.1 Car mechanics

Bicycle model, lets us imagine that the vehicle only has two wheels, that are located at the center of the vehicle. The radius of the circle formed by the front wheel equals the length of the $\overline{AC}$ line on the triangle $\overline{ABC}$, formed by the three points. Angle of the wheel α is also equal to the steering angle of the vehicle. Now we can calculate the turning radius of the vehicle, by dividing chassis length by the sine of steering angle.

$$R = L/\sin(\alpha) \tag{6}$$

Similar to the position, we do not change the vehicle's angle directly. To determine how fast the vehicle is rotating. Therefore, we have to find the angular velocity.

$$\omega = \nu_x/R \tag{7}$$

Again, we do not use ν_y because the car doesn't accelerate in y direction. We also make ω equal to 0 when the steering angle is 0. We integrate position angle and position as follows. p, denotes position and θ, denotes the angle.

$$p = \nu.dt \tag{8}$$

$$\theta = \omega.dt \tag{9}$$

That's for the movement of the vehicle.

3.2. DQN

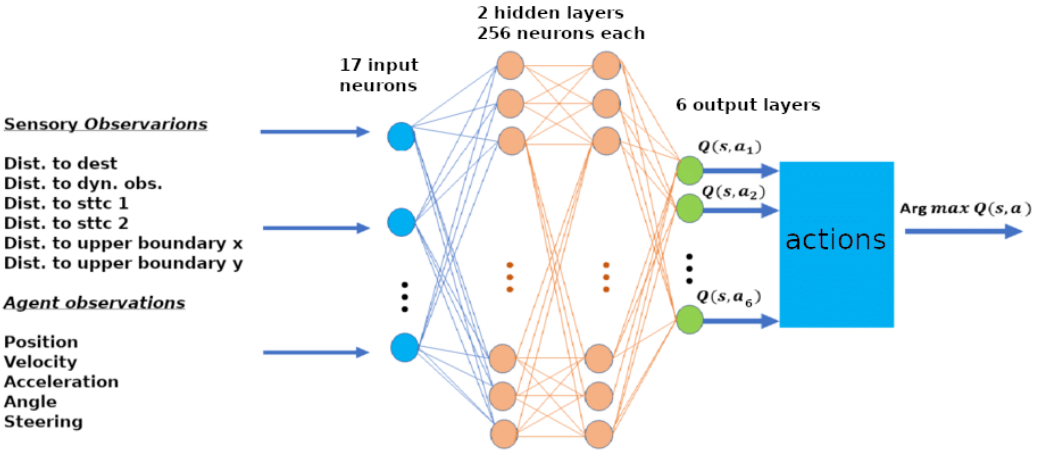

Figure 3.2 DQN structure

We know from the kinematic model that the agent needs to have five values to control its movements. These values are position, velocity, acceleration, angle, and steering. All these values are given as an input to the neural network. We also needed sensory information to observe the environment, for that reason six sensory measures were added to the system, and given as an input value. These measures are the including distance to the destination, distance to the dynamic obstacle, distance to the first static obstacle, distance to the second obstacle, distance to the upper x direction boundary of the environment, and finally distance to the upper y direction. Lower environment boundaries aren't added because distance is calculated simply x, and y directions of the objects are subtracted from the x, and y value of the vehicle, and the lower boundary was equal to 0. Therefore, adding it would be just be adding the x and y values of the vehicle, and that is the position of the vehicle, which already exists in the given input.

Agent takes 6 different actions, going right, going left, accelerating forward, accelerating backwards, brake for forward action, and brake for backward acceleration. Learning rate of the agent is 0.01, gamma parameter is 0.99, epsilon is decreasing linearly and the end parameter is 0.01, and the batch size is 64.

3.2.1. Observation values

3.2.1.1. Position values Position values include x, and y values, velocity also has ν_x, and ν_y values, acceleration only one dimensional value it is either 0, 10 or -20. The negative value of the acceleration is used as a break in the simulation. Angle value is one value in degrees, which is the amount of how much the wheel is turned by steering. Steering is also one value in degrees, maximum steering is 90 degrees.

3.2.1.2. Sensory values Sensory values include distance to destination, dynamic obstacle, first static obstacle, and second static obstacle has x, and y values. But distance to x upper boundary only has x, and distance to y upper boundary only has y values. At the end, the total size of the input becomes 17.

3.3. Gauss Noise Injection

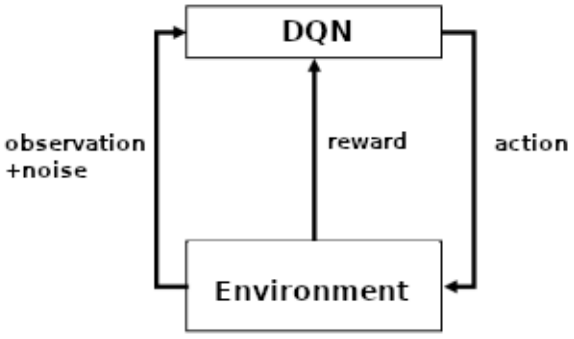

Figure 3.3 Noise Injection

Noise injection refers to the deliberate addition of random or stochastic elements into a system or process. This technique is commonly used in various fields such as signal processing, machine learning, control systems, and simulations to introduce randomness and mimic real-world variability or uncertainty

Equation 10 is the general formula of noise addition, $n(x, y)$ represents the Gaussian noise here. Probability density of Gaussian distribution is given in the Equation 11.

$$g(x, y) = f(x, y) + n(x, y) \tag{10}$$

The primary purpose of noise injection is to enhance the robustness, generalization, or realism of a system or model. By introducing random variations, noise injection helps prevent overfitting in machine learning models, improves the accuracy of simulations by accounting for uncertainties, and enables systems to adapt to unpredictable environments.

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(\frac{(x-\mu)^2}{2\sigma^2})} \tag{11}$$

In Eqn. 11, μ represents the mean, σ represents the standard deviation, and the σ^2 is called the variance. The Gauss function has reached its maximum at the mean value, and it radiates with the standard deviation.

We implemented the mean based on a condition. If the shape of the vector is a 1D vector, the mean of the vector is set to the value of the vector. Otherwise, it calculates the mean along the first axis of the 2D array. We again implemented the standard deviation with the same condition. If the shape of the vector is a 1D vector, standard deviation is set to 0. Otherwise, it calculates the standard deviation along the first axis of the 2D array.

The effects of noise injection depend on factors such as the type and intensity of noise, the characteristics of the system or model, and the specific objectives of the application. In some cases, noise injection may improve performance by promoting adaptability and resilience, while in others, it may introduce undesired artifacts or reduce the precision of

results. Overall, noise injection is a versatile technique used to introduce randomness, more exploration and uncertainty into systems and models, enabling them to better capture the complexities of real-world scenarios and improve their performance in challenging environments.

In our proposed method, We combine a kinematic model with deep Q-network (DQN) algorithms. The kinematic model accurately describes the vehicle's linear movement and steering mechanics, which are essential for realistic simulation. The DQN algorithm leverages sensory inputs to navigate and avoid obstacles, with Gaussian noise injection simulating uncertainty. This noise injection introduces a challenging training environment, aligning with our objective of improving robustness and adaptability through curriculum learning. This method addresses the challenges identified in the introduction and is supported by the literature reviewed in the background section.

4. EXPERIMENTAL RESULTS & ANALYSIS

We designed an environment to test our DQN algorithm. However, our environment had sparse rewards, and it seemed the agent was challenged to reach the destination. We realized it would be so much easier for the agent to reach a closer target. That would be the easy way to train the agent. Reaching a closer target would teach the agent that its purpose is to reach the target. Our simple curriculum learning setting can be seen in Figure 4.1.

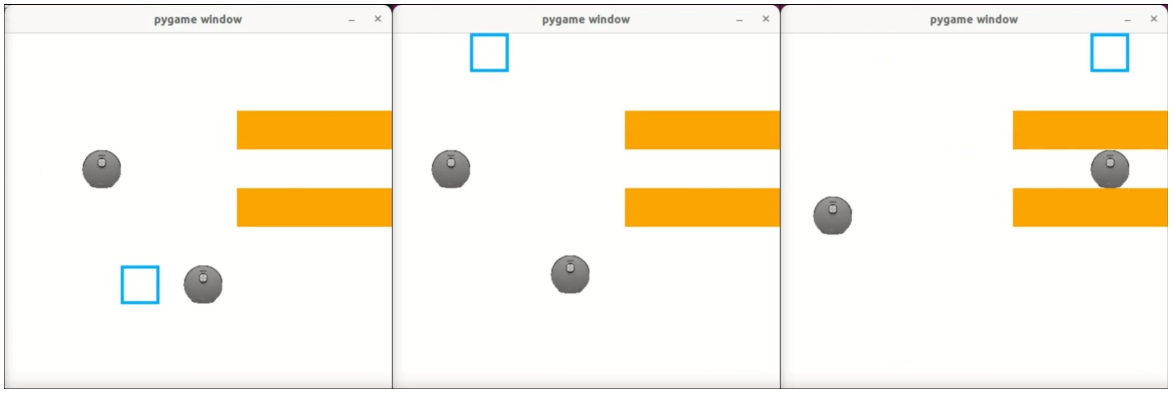

Figure 4.1 Curriculum learning, 3 destination locations. The gray circular objects are dynamic obstacles and the agent. Yellow blocks are the static blocks. Blue frame is the destination. Dynamic object, is the one that is among the two yellow static blocks. Dynamic block can only move from right to left.

Curriculum learning made training the agent so much easier for us. Our curriculum learning had three steps. First the destination was really close to the agent. In 10 or 15 steps the agent would reach the destination. For about 3000 episodes it would learn to reach the closer target. Then, we would remove it to the other side of the dynamic obstacle for another 3000 episodes. Our dynamic obstacle moves in really simple terms, in a training environment it moves from right to left, except in the reverse environment. In the reverse environment dynamic obstacle moves from left to right. There are static obstacles that block the agent's view. When the dynamic obstacle is inside this tunnel of static obstacles, the agent's sensors cannot receive information about the dynamic object. We wanted to achieve

a risk aware agent, the agent would take wide turns to avoid getting hit by the dynamic obstacle even though it didn't exist in some of the challenging environments. In the third part, the destination is placed not only behind the dynamic obstacle but also to the corner of the environment, for 4000 episodes. That created a tunnel-like environment. On one side a static obstacle, and on the other environment boundary was making it really hard to achieve the destination. Each time that agent collided with an environment object the game was terminated.

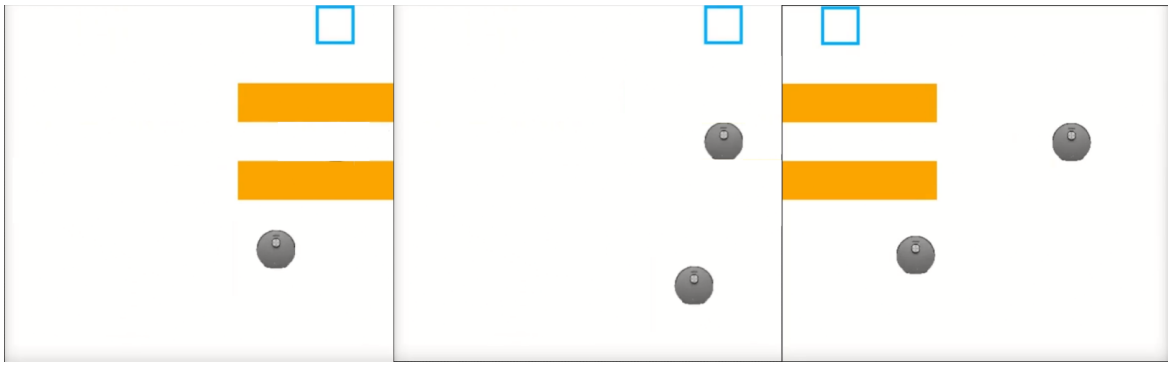

Figure 4.2 Different environments, first environment is static environment, second environment, without-walls environment, and the third environment is the reverse environment.

In order to see which environment and order of difficulty would be the hardest, we trained and tested on different environments (see Figure 4.2). In order to analyze the outcomes, we run each test 3 times, and write the average value inside the tables. We trained in a static environment that only had static obstacles, dynamic objects that didn't exist. At this point, we were also testing if the DQN agent was acting properly. If it was going to the destination. In this environment we didn't expect to have risk awareness. Only achievements were static obstacle avoidance and autonomous driving. First row of Table 4.1 shows training in the static environment and testing in different environments. After training in the same static environment we tested on the without walls environment. This time, static obstacles were gone, and only dynamic obstacle existed.

That wasn't the only way to test agents' performance, therefore, we injected Gaussian noise. In the following tables we see from 0 to up to 40% percent noise injection during the testing. Tables show the noise levels applied in the testing part. We added 0%, 10%, 20%, 30%, and 40% noise to the system. These noises were not implemented in the training session before, it was only the testing that involved the noise. Later in the study we also implemented the noise to the training sessions, also we added 50% in the training, not in the testing part. In the testing environments, we implemented the noise in different time schedules. First of all in the first 30 episodes of 100 episodes we gave a fixed amount of Gaussian noise. We implemented 30% Gaussian noise, in our experiments, and realized above 30% was decreasing the system's performance quite noticeably. Second, we added noise for the last 30 episodes of the testing, and at last, we implemented dynamically increasing noise, that we called step noise in the tables. That way the noise is increased during the game until it reaches 45%. In each episode it would increase with a fixed rate when the test reached 100 episode noise would reach 45%, and the game would finish.

Training in a normal environment. Normal environment was our first designed environment, that includes dynamic obstacles and the two static environments, noise levels still implemented. After training our agent in a normal environment, we removed the dynamic obstacle and tested. We expected to see, risk-aware agent. That took wide turns and didn't get closer to the tunnel of static environments. To our surprise the generated path didn't really change much. Agent, just continued to take the similar paths, only the difference was that the agent waited some amount of time to realize that the dynamic obstacle was not coming from the tunnel anymore.

After seeing agent's performance in similarly shaped environments. We wanted to reverse the environment structure, to see if what the agent learned is general or not. We introduced a reverse environment. Reverse environment has both the dynamic and the static obstacles. Here, the position of the static and dynamic obstacles moves to the left side of the environment, and the destination is moved to the left top corner of the environment.

We still were not content about the risk-aware behavior of the agent. For that reason,

we delayed the sensory information given to the agent. Agents observations consist of its position, velocity, angle, acceleration, and steering. Aside from that, we used distance measures as sensory information. Distance information includes the distance to destination, distance to dynamic obstacles, distance to static obstacles and distance to the environment boundaries. When the dynamic obstacle is inside the static obstacles we do not send any real time measures, we simply send 0 to the network.

In the game, an agent moves and we count its moves, when we delay the information we do it based on this count. We call these environments prediction environments. We delay the information on different counts. From 5 to 30 count delays. Prediction (5 times) means, the data is delayed for 5 counts of action. Earlier environments such as static and without walls environments are fully observable. When we continue to experiment and change environments, they become partially observable. Static environment, without walls environment are fully observable. Reverse environment, when the dynamic obstacle is blocked, and we added noise, and delayed information, environments become partially observable.

Table names indicate the training environment, and the test environments are given in the first column of the tables. We trained on the environments that is explained in the table names, then used their pretrained weights, to test on the test environments. We have done the same experiments in the curriculum, and in no-curriculum settings.

Our first experiment starts with the static environment (see Table 4.1), we wanted to see the success of the DQN without adding too much complexity. Agent is trained on the static environment, then we tested it not only in the static environment, but also on the without walls environment. Almost lost 50% of success rate, like a coin toss, agents learn to go to the destination, but not learn to avoid the dynamic obstacle. We measure the success rate, with hit-metrics. 100 episode was given, and the number of times the agent reaches the destination is considered as success rate. We run the each environment 3 times, and took the round value of average success rate. When we move from the static environment to the normal environment we see the results are a little bit better, perhaps not removing

static obstacles adds a little bit of familiarity to the new environment. That makes the agent perform slightly better. Risk-ware environments are the same environments, with the static environment and with the normal environment for that reason we did not show the results in the tables again. Reverse environment proves that it is not easy for the agent to adapt to such differences. Success rate drops to average 21%, different environments seem to be a big challenge.

Prediction environment is the same environment as the normal environment. Just the sensory information is delayed. When ”times” increases, that means the agent has to move more to gain information, while doing that it has to predict. Prediction environment success rates show that this environment is a great deal of a challenge, it turns out even if the surrounding is familiar, the agent heavily relies on the sensory information, and when it is late the success rate drops to 12%. It can be seen that the success rate decreases too fast, when the delayed time increases.

If we compare the curriculum setting with no-curriculum setting. We see in curriculum learning when we train on the static environment and test it on the static environment the success rate is higher compared to the no-curriculum setting. But when we test the same agent on the normal environment we see that no-curriculum results are better than the curriculum setting. Reverse environment is worse in the no-curriculum setting. But when the noise is applied in the first and last episodes no-curriculum shows better results, and increasing noise success rates are similar, but no-curriculum setting is slightly better. The success of the predictions are almost the same in both cases. It turns out that no-curriculum setting is affected by the noise easily, and the dynamic obstacle becomes a really big challenge.

Test env.	**0% noise**	**10% noise**	**20% noise**	**30% noise**	**40% noise**	**noise first**	**noise last**	**step noise**
static (no dyn.)	99	97	93	89	85	95	91	78
without walls	51	48	48	45	32	45	38	40
normal (dyn.,sttc.)	54	55	47	42	33	57	56	50
risk-aware(no dyn.)	..	..	..	same	env.	with	the	static
risk-aware(dyn.)	..	..	..	same	env.	with	the	normal
reverse	21	15	12	7	5	10	13	4
prediction(5 times)	12	11	9	6	3	4	5	1
prediction(10 times)	8	5	5	3	1	2	2	0
prediction(15 times)	1	0	0	0	0	0	0	0
prediction(20 times)	0	1	0	0	0	1	0	0
prediction(25 times)	0	0	0	0	0	0	0	0
prediction(30 times)	0	0	0	0	0	0	0	0

Table 4.1 Curriculum: Trained on the static environment.

Test env.	**0% noise**	**10% noise**	**20% noise**	**30% noise**	**40% noise**	**noise first**	**noise last**	**step noise**
static (no dyn.)	96	93	92	84	72	94	94	76
without-walls	51	47	44	42	29	43	44	37
normal (dyn.,sttc.)	67	60	56	48	42	61	66	45
risk-aware(no dyn.)	..	..	..	same	env.	with	the	static
risk-aware(dyn.)	..	..	..	same	env.	with	the	normal
reverse	19	16	11	9	5	16	17	5
prediction(5 times)	11	10	6	2	0	9	10	0
prediction(10 times)	9	4	1	0	0	2	6	0
prediction(15 times)	0	0	0	0	0	0	0	0
prediction(20 times)	0	0	0	0	0	0	0	0
prediction(25 times)	0	0	0	0	0	0	0	0
prediction(30 times)	0	0	0	0	0	0	0	0

Table 4.2 No Curriculum: Trained on the static environment.

In Table 4.3, we no longer have static environment, because the before we trained with it and used its weights and tested the agent on other environments. Now our training environment is without-walls environment. We used the without-walls weights to test in other environments. Trained on the without walls environment, that only has the dynamic obstacle, for that reason, we expected to have better success rates, when we added the walls (tested in normal environment) results weren't that bad. After training in the "without walls" environment, we tested on the "normal environment". When we trained with curriculum learning it was not hard for the agent to adapt to the changing environment. Adding walls after training with the dynamic obstacle didn't seem to be that challenging. Around 10% success rate was lost. Risk aware environment without the dynamic obstacle, is just the without walls environment. Therefore, we are seeing higher success rates that are similar to the results of the without walls environment. Risk aware environment, with the dynamic obstacle is the normal environment. We already tested in that environment. Therefore table 4.3 doesn't

have the results. After training in the without walls environment, and testing in reverse environment, showed that it was really challenging for the agent.

If we compare curriculum setting with the no-curriculum setting, we are seeing that the results of curriculum learning is higher than the no-curriculum learning, except for the predictions. In no-curriculum setting, prediction success rates are higher than the curriculum setting. This is opposite of what we expected, because training with curriculum learning had the highest success rates. One explanation could be that in the training part the agent memorized the positions of moving the destination, and in the testing part the destination was fixed to the corner, and the agent was still searching for these locations. Perhaps in order of difficulty. Therefore until it found the real destination position it just collided with the environment objects.

Test env.	**0% noise**	**10% noise**	**20% noise**	**30% noise**	**40% noise**	**noise first**	**noise last**	**step noise**
without walls	97	93	90	88	76	94	88	78
normal(dyn.,sttc.)	88	85	79	68	61	85	83	75
risk-aware (no dyn.)	93	93	93	81	76	90	84	81
risk-aware (dyn.)	..	..	..	same	env.	with	the	normal
reverse	11	11	9	7	4	8	9	0
prediction (5 times)	56	51	48	43	35	49	49	43
prediction (10 times)	46	46	39	34	27	35	39	33
prediction (15 times)	31	29	25	23	19	23	27	19
prediction (20 times)	15	11	9	5	1	9	11	1
prediction (25 times)	2	1	0	0	0	0	0	0
prediction (30 times)	0	0	0	0	0	0	0	0

Table 4.3 Curriculum: Trained on the without-walls environment.

Test env.	**0% noise**	**10% noise**	**20% noise**	**30% noise**	**40% noise**	**noise first**	**noise last**	**step noise**
without-walls	92	92	89	82	69	91	93	72
normal (dyn., sttc)	78	76	70	67	62	70	75	65
risk-aware(no dyn.)	75	75	70	65	63	71	74	66
risk-aware(dyn.)	..	..	..	same	env.	with	the	normal
reverse	13	12	9	5	3	9	11	1
prediction(5 times)	69	58	50	45	33	56	67	36
prediction(10 times)	57	51	47	38	33	52	53	37
prediction(15 times)	42	39	34	29	21	37	37	24
prediction(20 times)	23	19	14	8	1	16	19	0
prediction(25 times)	11	9	6	2	0	5	6	0
prediction(30 times)	2	0	0	0	0	0	0	0

Table 4.4 No Curriculum: Trained on without-walls environment.

Normal environment: We can see the normal environment in Figure 4.1, for curriculum learning setting. When trained and tested on the normal environment curriculum setting is almost the same, when we test both of the systems in a risk-aware setting we see until the noise level 20% they show similar results. After noise level 20% curriculum shows better outcome. When the noise is increasing the curriculum learning shows better results. For tests that are done in the reverse environment, no-curriculum shows better results. In prediction they show almost the same success rates, except when the predictions are 20 times. No-curriculum shows a better success rate when the predictions are 20 times delayed. In a random position situation, curriculum setting doesn't show big changes in the success rate, but no-curriculum setting success rates decrease with the increasing level of noise.

Test env.	**0% noise**	**10% noise**	**20% noise**	**30% noise**	**40% noise**	**noise first**	**noise last**	**step noise**
normal (dyn.,sttc)	97	96	90	85	75	92	95	89
risk-aware(no dyn.)	96	94	91	84	79	89	92	87
risk-aware(dyn.)	97	96	92	80	77	90	92	85
reverse	82	79	78	62	56	75	74	67
prediction(5 times)	81	78	75	68	53	74	76	69
prediction(10 times)	74	70	64	59	46	69	73	66
prediction(15 times)	61	58	54	45	36	54	60	45
prediction(20 times)	51	48	46	32	26	38	45	35
prediction(25 times)	36	34	29	15	9	21	24	19
prediction(30 times)	12	12	9	1	0	11	4	7
random from a list (no pred.)	57	58	55	48	36	53	43	34

Table 4.5 Curriculum: Trained on the normal environment.

No-curriculum setting is the one where we didn't move the destination closer to the agent to create easy to hard learning positions for the destination. When the same experiments are done in the no-curriculum setting the results are showing that the curriculum setting is showing higher success rates when trained on noisy environments, that also can be seen in Table 4.5, and Table 4.6.

Test env.	**0% noise**	**10% noise**	**20% noise**	**30% noise**	**40% noise**	**noise first**	**noise last**	**step noise**
normal (dyn., sttc)	94	90	90	83	70	89	93	87
risk-aware(no dyn.)	97	93	90	76	63	88	95	65
risk-aware(dyn.)	93	90	87	79	57	85	89	60
reverse	87	83	78	67	50	75	85	55
prediction(5 times)	84	80	74	65	51	72	83	53
prediction(10 times)	76	72	69	61	47	69	74	49
prediction(15 times)	63	56	48	38	27	51	60	35
prediction(20 times)	53	49	46	38	28	49	50	23
prediction(25 times)	40	36	33	26	11	37	38	16
prediction(30 times)	17	11	9	4	0	13	15	7
random from a list (no pred.)	55	52	49	37	31	50	54	34

Table 4.6 No Curriculum: Trained on the normal environment.

Now we added noise to the training. We implemented the noise starting from the normal environment. It turns out when we added 20% noise, the success rate of the curriculum algorithm increased relatively less than 5% compared to the no-curriculum setting. We not only add noise in the training part, but we also keep adding noise in the testing. When we add 30% noise in the training, we see that the success rate of curriculum and no-curriculum settings almost drops 70%, and in no-curriculum settings we see that the success rate drops significantly after 20% noise is added in the test. After adding 50% noise levels in the tests, we saw the performance level of the algorithms in both settings drops to worrying levels. For that reason we didn't go beyond 40% noise in the tests.

At this point we introduce another challenge. Just because we were moving the destination closer to the agent's location we didn't want to give the agent a random position, each time when it started, it started from the same location. Later, in order to add randomness in the

agent's location, we created a list of locations and chose random locations for the agent. When randomness applied we didn't apply the prediction, because we wanted to analyze the difficulty levels separately. For that reason we only applied the noise, and it turns out when randomness is added, the success rate is reduced by almost 45%. Randomness is applied with respect to the destination positions. Area 1 has its own set of random positions that consist of locations closer to the destination. Area 2 is the location when the destination moves behind the dynamic obstacle, area 2 also includes the positions of area 1, and area 3 involves both area 2, and area 1 locations. (see Figure 4.3)

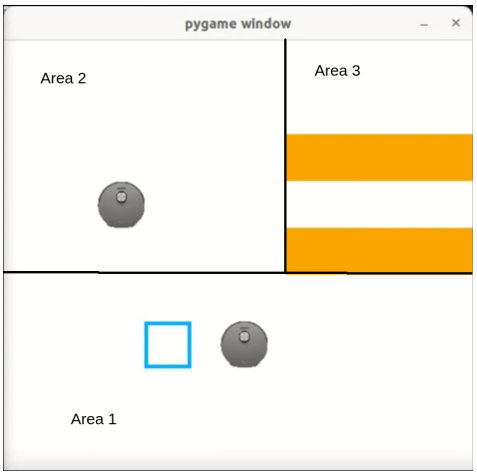

Figure 4.3 Areas of randomness

In Table 4.7 when we add noise to the training we see that curriculum learning has a significant difference in the success rate. If the training noise is 20%, testing and training on the normal environment shows that for curriculum learning the success rate is almost around 90% and no-curriculum is around 80%. When there is no dynamic obstacle in the risk aware setting, the success rate of the curriculum is around 95% but in no-curriculum learning it changes and decreases with the noise added to the test. In the no-curriculum setting when the noise added to the last episodes success rate is the highest second is the firstly added noise, and the last highest success rate belongs to the increasing noise setting. The Same pattern can be seen in the curriculum setting. Prediction values don't change that much, in both settings.

Test env.	**0% noise**	**10% noise**	**20% noise**	**30% noise**	**40% noise**	**noise first**	**noise last**	**step noise**
normal (dyn., sttc.)	95	93	94	85	79	90	93	85
risk-aware(no dyn.)	97	95	95	89	80	91	94	90
risk-aware(dyn.)	94	93	93	82	76	89	93	87
reverse	81	80	81	73	64	78	76	66
prediction(5 times)	78	79	77	70	62	78	75	67
prediction(10 times)	74	77	73	63	54	73	75	64
prediction(15 times)	59	58	58	50	44	56	60	43
prediction(20 times)	49	49	45	36	27	40	44	29
prediction(25 times)	35	32	28	13	7	29	30	19
prediction(30 times)	10	8	9	4	1	11	9	5
random from a list (no pred.)	63	61	57	53	50	60	65	44

Table 4.7 Curriculum: Trained on the normal environment + 0.20 noise.

Test env.	**0% noise**	**10% noise**	**20% noise**	**30% noise**	**40% noise**	**noise first**	**noise last**	**step noise**
normal (dyn.,sttc.)	89	87	83	77	67	86	87	73
risk-aware(no dyn.)	90	85	80	75	69	84	89	70
risk-aware(dyn.)	87	84	80	76	66	82	86	78
reverse	75	73	69	59	54	74	74	55
prediction(5 times)	75	70	67	63	55	72	75	63
prediction(10 times)	70	68	63	57	48	70	71	50
prediction(15 times)	54	50	45	38	29	49	53	31
prediction(20 times)	44	40	33	27	18	40	43	21
prediction(25 times)	34	29	27	20	12	30	32	18
prediction(30 times)	11	7	5	2	0	8	9	0
random from a list (no pred.)	60	56	52	46	38	57	59	40

Table 4.8 No Curriculum: Trained on the normal +0.20 noise environment.

If the training noise is 30% we see that no-curriculum learning test results are decreasing to 30%. In curriculum setting success rates are around 65%. Increasing noise is affecting the agents performance a lot especially in no-curriculum settings. In no-curriculum success rates starting with 70% but decreasing quickly with increasing noise in the test. In the reverse environment curriculum method is around 60%, but no-curriculum is around 55% and keeps decreasing. When the noise is 30% and the success rate of the no-curriculum method decreases below 40%. Same similarity of success rate decrease can be seen in prediction values, and the random environment. After training with noise curriculum setting shows resilience against noise in the test environment.

Test env.	**0% noise**	**10% noise**	**20% noise**	**30% noise**	**40% noise**	**noise first**	**noise last**	**step noise**
normal (dyn.,sttc.)	75	68	63	70	65	65	71	61
risk-aware(no dyn.)	76	74	71	65	61	65	70	57
risk-aware(dyn.)	73	73	69	62	57	66	73	55
reverse	65	67	62	54	49	60	68	47
prediction(5 times)	68	66	62	56	48	65	71	48
prediction(10 times)	65	61	61	55	47	57	63	44
prediction(15 times)	48	47	47	36	28	45	44	38
prediction(20 times)	36	38	35	27	20	32	37	19
prediction(25 times)	24	21	25	19	13	21	26	14
prediction(30 times)	9	5	5	1	0	5	8	2
random from a list (no pred.)	53	51	51	47	37	49	51	38

Table 4.9 Curriculum: Trained on the normal environment + 0.30 noise.

Test env.	**0% noise**	**10% noise**	**20% noise**	**30% noise**	**40% noise**	**noise first**	**noise last**	**step noise**
normal (dyn., sttc.)	72	66	59	49	33	65	71	45
risk-aware(no dyn.)	71	68	61	55	34	69	70	50
risk-aware(dyn.)	69	66	56	46	27	65	68	35
reverse	58	53	47	37	24	52	57	33
prediction(5 times)	62	56	45	35	23	57	61	33
prediction(10 times)	57	53	48	36	26	55	56	32
prediction(15 times)	43	41	37	31	28	40	42	29
prediction(20 times)	31	27	21	17	11	28	31	15
prediction(25 times)	19	17	13	3	0	16	18	2
prediction(30 times)	0	0	0	0	0	0	0	0
random from a list (no pred.)	49	41	37	25	13	40	46	23

Table 4.10 No Curriculum: Trained on the normal +0.30 noise environment.

When the noise levels are 40% (see Table 4.11) in the training, success rates of the both models are not that different. Curriculum setting shows slightly better results. Prediction rates in the curriculum setting is better after the 15 times prediction, no-curriculum success rates reaches to 0 mostly. In curriculum setting the success rate is around 30% and decreases with the increasing level of difficulty of prediction. Random environment values are similar, and the curriculum shows slightly better results.

Test env.	**0% noise**	**10% noise**	**20% noise**	**30% noise**	**40% noise**	**noise first**	**noise last**	**step noise**
normal (dyn.,stcc.)	53	51	49	43	38	50	52	29
risk-aware(no dyn.)	56	53	53	44	39	51	54	37
risk-aware(dyn.)	51	52	50	45	41	48	50	32
reverse	35	33	28	23	19	30	32	26
prediction(5 times)	44	39	39	33	27	37	39	26
prediction(10 times)	40	35	34	30	25	35	35	28
prediction(15 times)	36	33	30	28	21	31	33	23
prediction(20 times)	35	35	32	21	18	31	34	23
prediction(25 times)	16	14	11	8	4	11	13	6
prediction(30 times)	3	2	2	1	0	1	1	0
random from a list (no pred.)	45	40	38	35	29	39	43	31

Table 4.11 Curriculum: Trained on the normal environment + 0.40 noise.

Test env.	**0% noise**	**10% noise**	**20% noise**	**30% noise**	**40% noise**	**noise first**	**noise last**	**step noise**
normal (dyn., sttc.)	49	41	35	26	13	42	48	20
risk-aware(no dyn.)	50	45	37	26	14	44	46	22
risk-aware(dyn.)	48	39	34	23	15	40	44	20
reverse	31	26	19	9	6	27	30	7
prediction(5 times)	39	31	25	13	7	34	37	10
prediction(10 times)	32	28	21	15	5	30	31	12
prediction(15 times)	25	19	13	8	0	20	22	3
prediction(20 times)	14	9	3	1	0	10	8	0
prediction(25 times)	5	1	0	0	0	0	0	0
prediction(30 times)	0	0	0	0	0	0	0	0
random from a list (no pred.)	40	32	32	25	19	33	37	20

Table 4.12 No Curriculum: Trained on the normal +0.40 noise environment.

When the training noise level reaches 50% (see the Table 4.13) we see that the curriculum method shows better results, especially when the noise level is 40%, and in increasing noise setting no-curriculum method reaches 0 success rate. When the prediction is 5 times and noise is 30% we see no-curriculum prediction success values reaches to 0. For curriculum setting we start to see zeros when the noise is 40%. In a random setting the curriculum shows slightly better results. 50% training noise and 40% testing noise adds up to 90% noise in total, at this point many of these success rates can be achieved by chance.

Test env.	**0% noise**	**10% noise**	**20% noise**	**30% noise**	**40% noise**	**noise first**	**noise last**	**step noise**
normal (dyn., sttc)	35	31	25	12	2	29	30	11
risk-aware(no dyn.)	41	36	25	13	1	33	38	15
risk-aware(dyn.)	33	25	19	10	0	29	32	8
reverse	23	11	8	4	0	18	20	7
prediction(5 times)	27	23	18	7	0	20	24	8
prediction(10 times)	24	16	10	3	0	19	17	4
prediction(15 times)	15	9	7	0	0	13	11	2
prediction(20 times)	9	4	0	0	0	6	7	0
prediction(25 times)	6	1	0	0	0	2	3	0
prediction(30 times)	0	0	0	0	0	0	0	0
random from a list (no pred.)	21	17	14	10	0	15	16	6

Table 4.13 Curriculum: Trained on the normal environment + 0.50 noise.

Test env.	**0% noise**	**10% noise**	**20% noise**	**30% noise**	**40% noise**	**noise first**	**noise last**	**step noise**
normal (dyn., sttc.)	30	27	15	7	0	28	29	2
risk-aware(no dyn.)	31	19	11	3	0	20	30	0
risk-aware(dyn.)	27	15	9	1	0	19	25	0
reverse	22	11	7	1	0	17	20	0
prediction(5 times)	19	13	5	0	0	12	18	0
prediction(10 times)	13	7	1	0	0	9	11	0
prediction(15 times)	4	1	0	0	0	0	0	0
prediction(20 times)	0	0	0	0	0	0	0	0
prediction(25 times)	0	0	0	0	0	0	0	0
prediction(30 times)	0	0	0	0	0	0	0	0
random from a list (no pred.)	15	11	9	5	0	9	13	0

Table 4.14 No Curriculum: Trained on the normal +0.50 noise environment.

This Figure 4.4 shows the summary of the tables. Each column shows no-curriculum setting values subtracted from curriculum setting values formula 12 to make this table. x means curriculum setting values from the table, x_{no} means no-curriculum setting.

$$x - x_{no} \tag{12}$$

Columns, table 4.1-table 4.2 , table 4.3 - table4.4, table 4.5 - table 4.6, table 4.7 - table 4.8, table 4.9 - table 4.10, table 4.11 - table 4.12, table 4.13 - table 4.14 represents in order.

We show the added training noise starting from 4th column, that represents the table4.7-table4.8. If the bar is high, that means the curriculum is showing better performance than the no-curriculum setting or no-curriculum setting is showing bad performance.

4.0.1. 1st Column

In the first column we are seeing the biggest success in the static environment happening when we add noise 40%, but when we change the test environment we are seeing that when the noise level is 40% success rate is similar to 30%. That means noise is easy to adapt to. But, when the environment changes, that challenges the curriculum setting more. In both the static and the without walls test environment we can see when we apply the noise last, no-curriculum shows better results.

In the normal environment, we can see no-curriculum shows better results except for the step noise setting. That means curriculum setting only shows better performance when it is trained with the noise. If noise is not added, it shows worse performance than the no-curriculum setting.

In risk aware 1 environment (without dynamic) is the same as the static environment. So the graphs are the same. and risk aware 2 (with dynamic) is the same with the normal environment.

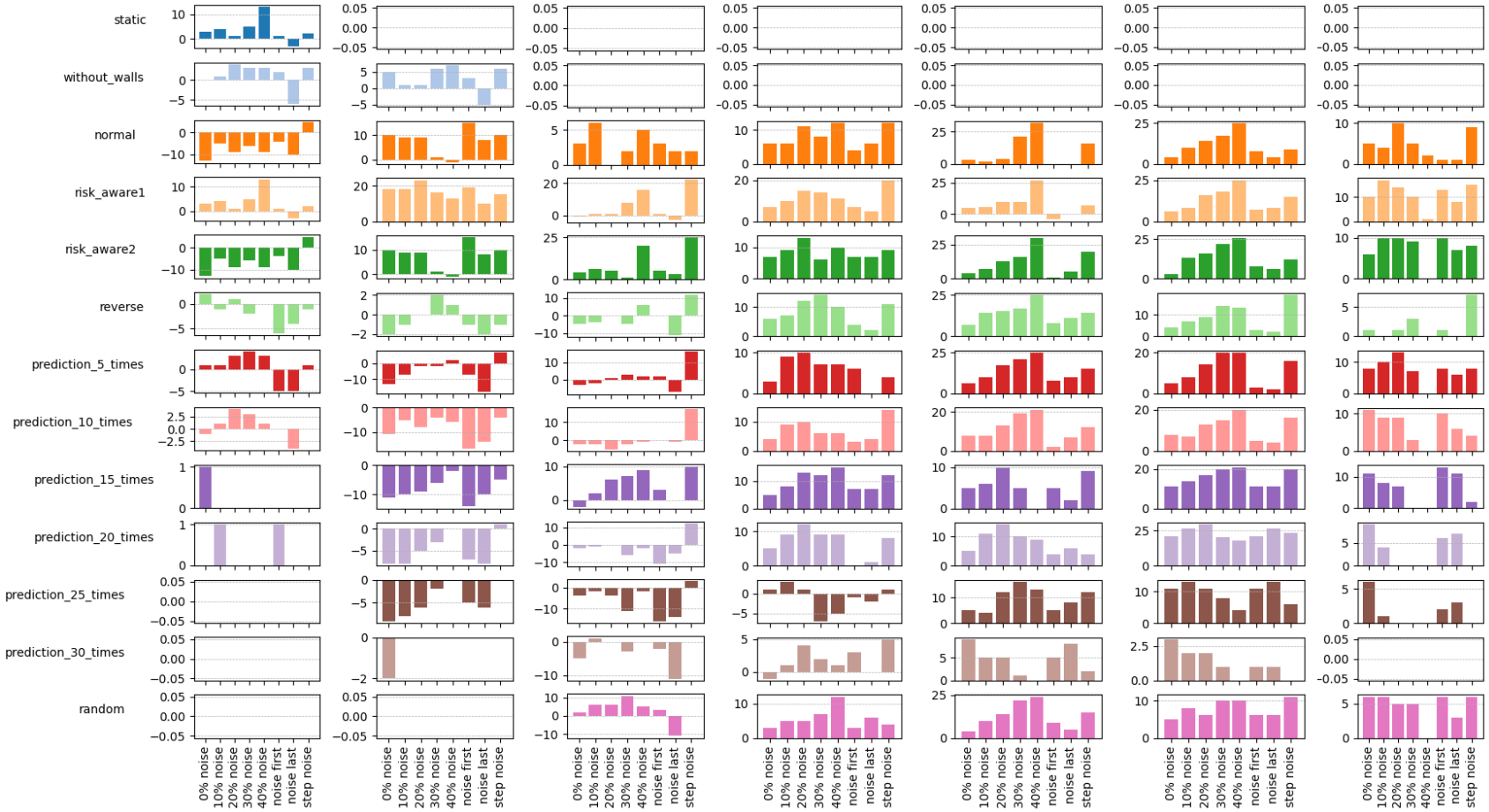

Figure 4.4 Performance gain of curriculum method. $x - x_{no}$ graphs.

In the reverse environment when we add noise to the test, the success difference is varying, which doesn't seem to be a good indication of the performance of the model. We see that when we apply the noise in different schedules, no-curriculum setting is always better.

Prediction environment (5 times): when the noise is 30% curriculum reaches its peak performance. it shows better results in step noise setting, first and last noise setting curriculum setting shows worse performance then the no-curriculum.

Prediction environment (10 times): When there is noise in the testing similar results can be seen. But the first noise setting and step noise gains are the same. only performs bad when the noise is applied last time. Prediction (15,20) only shows good performance when the noise is smaller than 10%, and first noise setting.

4.0.2. 2nd Column

We no longer use the static environment as a training setting therefore it is empty. The without walls of the first column and the without walls of the second column environments bar graphs seem similar. Again, the last noise setting shows bad results in the curriculum setting.

Normal environment: This time we are seeing better curriculum results. 30% and 40% settings are showing really small development. Compared to the first column normal environment bar graph, second column graph shows better results. Curriculum learning setting is more challenging than the no-curriculum setting if the environment has unknown objects such as dynamic obstacles. Also, that means noise is less challenging then the unknown environments.

Risk aware 1 (without dynamic): shows great bars, that means it is easy for the curriculum learning setting.

Risk aware 2 (with dynamics): Graphs are the same with the normal environment.

Reverse environment: Until the noise is bigger than the 20% curriculum setting shows worse results. It shows bad results with different noise schedules. that means, Unknown environment + noise settings are going to give really bad outcomes. That means the curriculum cannot adapt when both the environment and noise is changing.

Prediction environment (5 times): Only good when noise is 40%, first and last noise settings are bad, only increasing step noise is showing a good success rate.

Prediction environment (10 and 15 times): Showing bad results. only prediction (20 times) step noise a little bit of improvement, and prediction (30 times) 0% noise shows good results. Prediction environments are challenging the curriculum settings more than any other environment. Curriculum setting doesn't have any success over no-curriculum setting.

4.0.3. 3rd Column

This time, we start the training with the normal environment. We show better results compared to the no-curriculum setting.

Risk aware 1 (without dyn.): This time going from more challenging environment to less challenging environment makes the curriculum learning setting perform almost similar to no-curriculum setting.

Risk aware 2: is the same with the normal environment. It shows better than risk aware 1 but compared to earlier columns it is really bad. Except, when the noise is 40%, then the result is good.

Reverse environment: Just like earlier columns, changing with every noise setting.

The prediction environment (5 and 10 times) are similar. Prediction (15 times) curriculum setting shows better results. Other levels of prediction just like others don't show better results than the curriculum setting.

Random environment: only last noise shows bad curriculum success rates.

4.0.4. 4th Column

Including this column and later columns have noise in the training setting.

Normal environment: When there is noise curriculum setting shows outstanding results.

Risk aware 1 environment: Easy environment, shows better results Risk aware 2 environment: Same with normal, shows similar results with normal, risk awareness at this point is not a challenging task for the curriculum agent. Compare to earlier changing environments and prediction environment it is not that challenging. Even the reverse environment results are showing improvement

Prediction (5, 10, 15, and 20 times) are similar and better than no-curriculum setting. Results of prediction (25, and 20 times) are not that good.

Random environment: showing great compared to curriculum setting.

4.0.5. 5th Column

Normal environment: 'hen there is noise only in the 5Th column normal environment first and last noise values shows 0 improvement over no-curriculum setting. Same observation can be made for risk aware 1, and 2 environments.

In risk aware 2 environment, we see that success rate is increasing when the noise increases, but it just shows the no-curriculum setting is performing bad when the noise increases. Same can be said for the first, last, and step noise. Similar structures can be seen in the reverse environment. However, the difference value reaches to 25. That means that the environment challenges the no-curriculum setting a lot.

Prediction (5, and 10 times) are almost the same. Difference value is reaching to as 25, in (15 times) we see 40% noise success levels are the same, so bar size is 0. Prediction (20, 25, and 30 times) positive and good, not as good as prediction (5, and 10 times).

Random environment: Difference value reaches to 25, that shows no-curriculum setting is really challenged here.

Risk aware environment: The difference value reaches 25. That means it is really challenging for the no-curriculum setting.

4.0.6. 6th and 7th Column

We can see the reverse environment is not showing that great success differences anymore. 40% noise level successes are almost the same. Prediction (15 times and later) are showing less and less success rates.

But random values are showing higher success rates.

The experimental results demonstrate the effectiveness of our proposed method under various conditions. We conducted tests with different levels of noise and curriculum settings, showing that agents trained with a curriculum approach perform better and are more resilient to noise compared to those trained without a curriculum. These findings support our initial hypothesis that curriculum learning, even with noise, can enhance the agent's robustness and adaptability. This section ties back to the goals stated in the introduction and the challenges discussed in the background, validating our approach.

5. CONCLUSION

In our experiments, we evaluated the performance of the DQN agent in environments with varying complexities and challenges, using both curriculum learning and no-curriculum learning approaches. Our findings demonstrate that curriculum learning significantly improves the agent's ability to adapt and perform in complex and dynamic environments, compared to training in a no-curriculum setting.

Curriculum learning facilitated the agent's understanding of its goal by starting with simpler tasks and gradually increasing the complexity. This approach enabled the agent to achieve higher success rates, particularly in challenging environments with dynamic obstacles. In the static environment, the success rate was higher for curriculum learning compared to the no-curriculum setting. However, when tested in a normal environment, no-curriculum learning showed slightly better results, likely due to the agent memorizing specific positions during training.

When noise was introduced during training, the curriculum learning approach consistently showed higher success rates across different levels of noise compared to no-curriculum learning. This indicates that curriculum learning helps the agent develop a more robust policy that can handle environmental uncertainties. With noise levels up to 30%, curriculum learning maintained a relatively high success rate, while the no-curriculum approach saw significant performance drops. At 40% noise, both methods showed decreased success rates, but curriculum learning still performed better overall.

Introducing delayed sensory information (prediction environments) revealed the agent's heavy reliance on real-time data. Success rates dropped sharply as the delay increased, with both curriculum and no-curriculum settings showing reduced performance. However, curriculum learning still maintained slightly better results, indicating some level of improved predictive capabilities.

Training in a dynamic environment with curriculum learning helped the agent develop risk-aware behaviors, such as avoiding dynamic obstacles effectively. In

reverse environments, where the positions of obstacles and goals were changed, the curriculum-trained agent struggled but still outperformed the no-curriculum-trained agent, highlighting the benefits of structured learning in developing adaptable strategies.

When the agent's starting position was randomized, the success rate decreased significantly, especially in no-curriculum settings. This further underscores the importance of curriculum learning in helping the agent generalize its policy to handle diverse starting conditions and environments.

Our study demonstrates that curriculum learning is a powerful strategy for training reinforcement learning agents in complex and dynamic environments. By progressively increasing task difficulty, curriculum learning helps agents develop more robust and adaptable policies. This approach not only improves performance in familiar environments but also enhances the agent's ability to handle new and unforeseen challenges, including high noise levels and delayed sensory information. Future work could explore further enhancements to curriculum learning, such as adaptive curricula that dynamically adjust the difficulty based on the agent's performance, to further improve training efficiency and robustness.

In our thesis, we wanted to provide an order of difficulty for training the agent. When we analyze the data, our findings for the difficulty of the noise can be understood based on the nature of how noise is introduced and its impact on the agent's performance. The difficulty of the noise injections from easiest to hardest follows:

Test env.	**static env**	**without walls env**	**normal env**	**normal + 20% env**	**normal + 30% env**	**normal + 40% env**	**normal + 50% env**
static	0.036	0	0	0	0	0	0
without-walls	0.029	0.034	0	0	0	0	0
normal (dyn., sttc)	**-0.129**	0.098	0.032	0.091	0.145	0.249	0.211
risk-aware (no dyn.)	0.036	0.191	0.063	0.122	0.113	0.266	0.436
risk-aware (dyn.)	**-0.129**	0.098	0.097	0.096	0.182	0.287	0.385
reverse	**-0.126**	**-0.068**	**-0.012**	0.110	0.235	0.314	0.143
prediction(5 t)	0.059	**-0.107**	0.021	0.078	0.231	0.310	0.472
prediction(10 t)	0.154	**-0.231**	0.008	0.101	0.199	0.336	0.559
prediction(15 t)	1.000	**-0.342**	0.085	0.185	0.126	0.532	0.912
prediction(20 t)	1.000	**-0.613**	**-0.047**	0.166	0.258	0.803	1.000
prediction(25 t)	0	**-12.00**	**-0.267**	**-0.047**	0.460	0.928	1.000
prediction(30 t)	0	0	**-0.357**	0.263	1.000	1.000	0
random from a list (no pred.)	0	0	0.057	0.099	0.273	0.207	0.374

Table 5.1 Level of difficulty.

We see the first column is the test environment, and the following columns are the training columns. Black bold color indicates the lowest numbers, that means curriculum learning is showing bad performance compared to the no-curriculum setting, that also shows what is challenging the agent. Blue color indicates the lowest success rate when it is not a negative number, which again means what is challenging the agent, and red color is the highest success rate, that means it is easiest for the agent. We used formula 13 to make this table. x

means curriculum setting values from the table, x_{no} means no-curriculum setting values.

$$(x - x_{no}) / \sum x \tag{13}$$

We took the difference between curriculum and no-curriculum then, for the same environment, (for example normal environment we sum all the values in the row, 0%, 10%, 20%, 30%, 40% noise-first, noise-last, step noise) summed all that numbers then divided it with sum of the numbers in the curriculum tables for the same environment, so that we can assess the difficulty level of the environment. Shortly "Total difference / total success in the given environment for the curriculum setting."

Table 4.15 shows, trained on static and tested on without walls environment challenges the agent more than the without walls environment 0.029. When we train the agent on the static environment, and test it on the normal environment it becomes really challenging for the curriculum agent **-0.129**.

Then we see on the same test environment (row), if we train with normal environment and test with normal environment curriculum agent is challenged, the reason is there is no noise in the environment and if curriculum agent is not trained with noise it is under-performing 0.032. Training on the normal environment with 40% noise seems easy for the agent 0.249

Trained on the static environment tested on the risk-aware (no-dyn) environment shows 0.036 which means it is challenging for the agent. But, there is no noise therefore the curriculum is under-performing. When we train on the normal environment with 50% noise, and test on the risk-aware (no dyn) environment we see it is not challenging the agent anymore 0.436.

When we train on the static environment and test on the risk-aware (dyn) environment we see that the curriculum shows worse than the no-curriculum method. Because the value is negative **-0.129**. No noise is added to the training, dynamic obstacle weren't existed in the training, that makes the environment an unknown environment. Training with noise plays a really big role in our curriculum setting. Also changing environment to unknown environment is really challenging for the agent.

When we train on the normal + 20% noise environment and test on the risk-aware (dyn) environment that is the normal environment. We realize it becomes really challenging for the agent. We do not see which situation is challenging because the noise is added to the training and also the risk aware environment (dyn) is the same environment as the normal environment. We do not know what is challenging here 0.096. But when we look at the value 0.385, it means that situation training with 50% noise and testing in the risk-aware (dyn) environment is not challenging.

We know that when we increase the noise, no-curriculum shows bad results. Therefore, that may not be a direct success of the curriculum algorithm.

Trained on static environment, and tested on the reverse environment, shows **-0.126** low success rate. That means it is really challenging for the curriculum environment. First it doesn't seem like it is showing worse results than the risk-aware (dyn) environment. But when we look at the without walls trained agent we see again that the reverse environment shows **-0.068** value. Also normal environment trained agents are also showing **-0.012** negative values. That means it is really challenging for the agent. There is no noise in the training parts, but it is clearly shown. When we train in the normal+ 20% noise environment, and test it on the reverse environment. We can see it is not negative this time, but it is the lowest value 0.110, which means environment is challenging the agent, and we reach the peak success 0.314 when trained on normal + 40% noise environment That is telling us that

success level of the agent, can be increased with the injected noise.

When we look at the prediction test environments we see training on without walls environment and delaying the information (5 times) is challenging until the prediction is (25 times), there is no noise, also prediction environment is normal environment and we didn't add wall when we trained the agent, that means agent is colliding with the static obstacles (**-0.107,-0.231,-0.342,-0.613,-12.00**).

When we train the agent on the normal environment and test on the prediction environment (5 to 15 times) we see (0.021,0.008,0.085) values. That shows the agent is challenged, these numbers are so close to being a negative values. When we test on the prediction (20 to 30 times), we reach the negative values (**-0.047,-0.267,-0.357**). This time, we train on the same environment and test it on the same environment except we delay the information. That shows delaying is really challenging and the agents are relying on sensory information a lot.

When we train the agent on the normal+20% noise environment prediction environment is no longer a big challenge. Only predictions (20 to 30 times) are challenging. 0.166 is small but not really close to becoming negative, **-0.047** is negative, but prediction (30 times) is 0.263 not negative.

Trained on normal +30% noise shows 0.460 as the smallest number. But, this number is the biggest blue number compared to all the other blue numbers. Highest number is 1.000 for the prediction (30 times). When agents reach the destination really small amounts of times such as less than 10 times. Calculating, subtracting and dividing similar and same values cannot tell us a lot. For that reason seeing 1.000 doesn't tell us anything.

Training on the normal environment and testing on the random environment shows us, 0.057, it is a really small number which makes it one of the easiest challenges out there. Also, training on the normal + 50% noise environment shows 0.374 value which means it is not a big challenging environment.

When we try to give an order we will look at the value of lowest numbers which indicates how bad the curriculum is against the no-curriculum setting. Also, we look at how low the highest number is, that measure indicates the easiness of the challenge for the curriculum learning agent. So according to given information we can give the order from easy to hard environment:

static: 0.036 only the number.

without walls: 0.029 lowest, 0.034 highest.

normal: **-0.129** lowest, 0.032, 0.249 highest.

risk aware (no dyn): 0.036 lowest, 0.436 highest.

random: 0.057 lowest, 0.374 highest.

risk aware (dyn): **-0.129** lowest, 0.096, 0.385 highest.

reverse: **-0.126** lowest, **-0.068**, **-0.012**, 0.110, 0.314 highest.

prediction (5 times): **-0.107** lowest, 0.021, 0.472 highest.

prediction (10times): **-0.231** lowest, 0.008, 0.559 highest.

prediction(15 times): **-0.342** lowest, 0.085, 1.000 highest.

prediction(20 times): **-0.613** lowest, **-0.047**, 0.166, 1.000 highest.

prediction(25 times): **-12.00** lowest, **-0.267**, **-0.047**, 0.460, 1.000 highest.

prediction(30 times): **-0.357** lowest, 0.263, 1.000 highest.

When we analyze the noise difficulty levels from the Figure 4.4 we see increasing noise is easier for curriculum setting to adapt, it is always the highest in the figure 4.4. Then noise-first shows highest bars, and mainly the smallest bars are in the last-noise setting. Except in the 5th column it is always changing, and in many of the prediction environments curriculum difference values for the noises is always changing. For that reason reaching a conclusion for the difficulty levels of the noise is not easy.

Now, when we attempt to give an order for the environments, we realize, partially observable environments are generally harder than fully observable environments. Fully Observable Environments generally show higher performance metrics, indicating easier navigation and decision-making. The agent has access to the complete state of the environment at all times. This makes decision-making more straightforward as all relevant information is available. Partially Observable Environments on the other hand, typically exhibit lower performance metrics due to the additional complexity in inferring the state of the environment. The agent has limited information about the state of the environment. This adds complexity to the decision-making process as the agent must infer or predict the missing information. In our experiments, Fully Observable Environments are Static Environment, Without Walls Environment, Risk-aware Environment (without dynamic obstacles). Partially Observable Environments are Normal Environment, Risk-aware Environment (with dynamic obstacles) (just normal environment.), Reverse Environment, Prediction Environment, Random from a list environment.

If we make a performance analysis, by comparing the success rates from the tables we can see partially observable environments are generally harder than fully observable environments. The main reason is that agents in partially observable environments need to deal with uncertainty and incomplete information, making the tasks more challenging. This is reflected in the performance metrics from the tables, where we observe lower performance in partially observable environments compared to fully observable ones. Possible reasons:

Limited Information causes Uncertainty, and ambiguity. In a partially observable environment, the agent does not have access to complete information about the state of the environment. This limitation means the agent must make decisions based on incomplete data, increasing the complexity of planning and decision-making. Because the agent cannot see the entire environment, it must deal with uncertainty and ambiguity regarding the state of the world. The agent must infer or estimate the missing information, which can lead to suboptimal decisions if the inferences are incorrect, or delayed. In our prediction environments, sensory information is delayed. Therefore, the agent can not make the connections between the results of its actions, making it harder to learn from feedback. The agent must develop strategies to correlate actions with delayed outcomes, which is more challenging than when immediate feedback is available.

Increased Complexity in Strategy: To effectively operate in a partially observable environment, the agent often needs more sophisticated strategies, such as maintaining a belief state (a probability distribution over possible states) or using memory to keep track of past observations. This adds computational complexity to the agent's algorithms such as Exploration vs. Exploitation Trade-off. The agent must balance exploration (gathering more information about the environment) and exploitation (using the current information to make the best decisions). In partially observable environments, this trade-off is more pronounced because exploring can be risky without knowing the full context. These factors collectively make decision-making and learning in partially observable environments more challenging than in fully observable environments.

In conclusion, our study reaffirms that curriculum learning, combined with noise injection, significantly improves an agent's robustness and generalization in uncertain environments. These findings have important implications for future research in DRL and robotics. Our proposed method can be applied to more complex tasks and different algorithms, paving the way for further advancements. The conclusion ties back to the introduction and background, illustrating how our study contributes to the field by addressing the key challenges and advancing the understanding of DRL in noisy environments.

In our future work, we would like to extend our study, with more difficulty, and with more complex environments. In order to have more control on our environments, we designed them ourselves. The task we gave to our agent was autonomous driving. We want to extend this study, for different tasks, and make it less dependent on the domain of the task. For future work we should try existing challenging environments such as ViZDoom, and Starcraft II environments. Try new adversarial attack mechanisms, such as adding a second agent that tries to trick the main agent. Increase the types of noise such as random noise, systematic noise, environmental noise, and also the noise in the DQN algorithm. The DQN algorithm is the simplest of the DRL algorithms, for that reason, we should add other DRL methods to our further study.

REFERENCES

[1] N. Justesen, P. Bontrager, J. Togelius, and S. Risi, "Deep Learning for Video Game Playing," *arXiv.org*, 2017. https://arxiv.org/abs/1708.07902 (accessed Apr. 29, 2024).

[2] M. Andrychowicz *et al.*, "Hindsight Experience Replay," *arXiv.org*, 2017. https://arxiv.org/abs/1707.01495 (accessed Apr. 29, 2024).

[3] Y. Chebotar, K. Hausman, M. Zhang, G. Sukhatme, S. Schaal, and S. Levine, "Combining Model-Based and Model-Free Updates for Trajectory-Centric Reinforcement Learning," *arXiv.org*, 2017. https://arxiv.org/abs/1703.03078 (accessed Apr. 29, 2024).

[4] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous Deep Q-Learning with Model-based Acceleration," *arXiv.org*, 2016. https://arxiv.org/abs/1603.00748 (accessed Apr. 29, 2024).

[5] P. Mirowski *et al.*, "Learning to Navigate in Complex Environments," *arXiv.org*, 2016. https://arxiv.org/abs/1611.03673 (accessed Apr. 29, 2024).

[6] G. Lample and Chaplot, Devendra Singh, "Playing FPS Games with Deep Reinforcement Learning," *arXiv.org*, 2016. https://arxiv.org/abs/1609.05521 (accessed Apr. 29, 2024).

[7] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, 2009, doi: https://doi.org/10.1145/1553374.1553380.

[8] Y. Wu and Y. Tian, "Published as a conference paper at ICLR 2017 Training agent for first-person shooter game with actor-critic curriculum learning." Available: https://openreview.net/pdf?id=Hk3mPK5gg

[9] V. Mnih *et al.*, "Asynchronous Methods for Deep Reinforcement Learning," *arXiv.org*, 2016. https://arxiv.org/abs/1602.01783 (accessed Apr. 29, 2024).

[10] O. Vinyals *et al.*, "StarCraft II: A New Challenge for Reinforcement Learning," *arXiv.org*, 2017. https://arxiv.org/abs/1708.04782 (accessed Apr. 29, 2024).

[11] Volodymyr Mnih *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: https://doi.org/10.1038/nature14236.

[12] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, “ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning,” *arXiv.org*, 2016. https://arxiv.org/abs/1605.02097 (accessed Apr. 29, 2024).

[13] T. Zhang, X. Huo, S. Chen, B. Yang, and G. Zhang, “Hybrid Path Planning of A Quadrotor UAV Based on Q-Learning Algorithm,” Jul. 2018, doi: https://doi.org/10.23919/chicc.2018.8482604.

[14] Y. Li, S. Zhang, F. Ye, T. Jiang, and Y. Li, “A UAV Path Planning Method Based on Deep Reinforcement Learning,” Jul. 2020, doi: https://doi.org/10.23919/usnc/ursi49741.2020.9321625.

[15] I. Yoon, Malik Aqeel Anwar, R. V. Joshi, Titash Rakshit, and Arijit Raychowdhury, “Hierarchical Memory System With STT-MRAM and SRAM to Support Transfer and Real-Time Reinforcement Learning in Autonomous Drones,” *IEEE journal on emerging and selected topics in circuits and systems*, vol. 9, no. 3, pp. 485–497, Sep. 2019, doi: https://doi.org/10.1109/jetcas.2019.2932285.

[16] Richard S. Sutton and Andrew G. Barto, “Reinforcement learning: An introduction,” MIT press, 2018.

[17] R. Mangannavar and G. Srinivasaraghavan,“Learning Agents with Prioritization and Parameter Noise in Continuous State and Action Space”, 2019, Accessed: Apr. 29, 2024. [Online]. Available: https://openreview.net/pdf?id=ryGiYoAqt7

[18] J. Schmidhuber, “POWERPLAY: Training an Increasingly General Problem Solver by Continually Searching for the Simplest Still Unsolvable Problem,” *arXiv.org*, 2024. https://arxiv.org/abs/1112.5309 (accessed Apr. 29, 2024).

[19] M. M. Botvinick, Y. Niv, and A. C. Barto, “Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective,” *Cognition*, vol. 113, no. 3, pp. 262–280, Dec. 2009, doi: https://doi.org/10.1016/j.cognition.2008.08.011.

[20] T. D. Kulkarni, K. R. Narasimhan, A. Saeedi, and J. B. Tenenbaum, “Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation,” *arXiv.org*, 2016. https://arxiv.org/abs/1604.06057 (accessed Apr. 29, 2024).

[21] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized Experience Replay,” *arXiv.org*, 2015. https://arxiv.org/abs/1511.05952 (accessed Apr. 29, 2024).

[22] D. Horgan *et al.*, “Distributed Prioritized Experience Replay,” *arXiv.org*, 2018. https://arxiv.org/abs/1803.00933 (accessed Apr. 29, 2024).

[23] Yannis Flet-Berliac and P. Preux, “Samples are not all useful: Denoising policy gradient updates using variance,” *arXiv.org*, 2019. https://www.semanticscholar.org/paper/Samples-are-not-all-useful%3A-Denoising-policy-using-Flet-Berliac-Preux/e86828a6c61d9cb5ecdc83cf1793c19e0d6661d2 (accessed Apr. 29, 2024).

[24] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman, “Teacher-Student Curriculum Learning,” *arXiv.org*, 2017. https://arxiv.org/abs/1707.00183 (accessed Apr. 29, 2024).

[25] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, “Reverse Curriculum Generation for Reinforcement Learning,” *arXiv.org*, 2017. https://arxiv.org/abs/1707.05300 (accessed Apr. 29, 2024).

[26] B. Ivanovic, J. Harrison, A. Sharma, M. Chen, and M. Pavone, “BaRC: Backward Reachability Curriculum for Robotic Reinforcement Learning,” *arXiv.org*, 2018. https://arxiv.org/abs/1806.06161 (accessed Apr. 29, 2024).

[27] T. Salimans and R. Chen, “Learning Montezuma’s Revenge from a Single Demonstration,” *arXiv.org*, 2018. https://arxiv.org/abs/1812.03381 (accessed Apr. 29, 2024).

[28] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying Count-Based Exploration and Intrinsic Motivation," *arXiv.org*, 2016. https://arxiv.org/abs/1606.01868 (accessed Apr. 29, 2024).

[29] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven Exploration by Self-supervised Prediction," *arXiv.org*, 2017. https://arxiv.org/abs/1705.05363 (accessed Apr. 29, 2024).

[30] P. Shyam, W. Jaśkowski, and F. Gomez, "Model-Based Active Exploration," *arXiv.org*, 2018. https://arxiv.org/abs/1810.12162 (accessed Apr. 29, 2024).

[31] D. Pathak, D. Gandhi, and A. Gupta, "Self-Supervised Exploration via Disagreement," *arXiv.org*, 2019. https://arxiv.org/abs/1906.04161 (accessed Apr. 29, 2024).

[32] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by Random Network Distillation," *arXiv.org*, 2018. https://arxiv.org/abs/1810.12894 (accessed Apr. 29, 2024).

[33] R. Portelas, C. Colas, K. Hofmann, and P.-Y. Oudeyer, "Teacher algorithms for curriculum learning of Deep RL in continuously parameterized environments," *arXiv.org*, 2019. https://arxiv.org/abs/1910.07224 (accessed Apr. 29, 2024).

[34] OpenAI *et al.*, "Solving Rubik's Cube with a Robot Hand," *arXiv.org*, 2019. https://arxiv.org/abs/1910.07113 (accessed Apr. 29, 2024).

[35] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull, "Active Domain Randomization," *arXiv.org*, 2019. https://arxiv.org/abs/1904.04762 (accessed Apr. 29, 2024).

[36] D. Silver *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017, doi: https://doi.org/10.1038/nature24270.

[37] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust Adversarial Reinforcement Learning," *arXiv.org*, 2017. https://arxiv.org/abs/1703.02702 (accessed Apr. 29, 2024).

[38] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, “Emergent Complexity via Multi-Agent Competition,” *arXiv.org*, 2017. https://arxiv.org/abs/1710.03748 (accessed Apr. 29, 2024).

[39] B. Baker *et al.*, “Emergent Tool Use From Multi-Agent Autocurricula,” *arXiv.org*, 2019. https://arxiv.org/abs/1909.07528 (accessed Apr. 29, 2024).

[40] Oriol Vinyals *et al.*, “Grandmaster level in StarCraft II using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, pp. 350–354, Oct. 2019, doi: https://doi.org/10.1038/s41586-019-1724-z.

[41] R. Zhao and V. Tresp, “Energy-Based Hindsight Experience Prioritization,” *arXiv.org*, 2018. https://arxiv.org/abs/1810.01363 (accessed Apr. 29, 2024).

[42] P. Fournier, O. Sigaud, M. Chetouani, and P.-Y. Oudeyer, “Accuracy-based Curriculum Learning in Deep Reinforcement Learning,” *arXiv.org*, 2018. https://arxiv.org/abs/1806.09614 (accessed Apr. 29, 2024).

[43] M. Eppe, S. Magg, and S. Wermter, “Curriculum goal masking for continuous deep reinforcement learning,” *arXiv.org*, 2018. https://arxiv.org/abs/1809.06146 (accessed Apr. 29, 2024).

[44] R. Zhao and V. Tresp, “Curiosity-Driven Experience Prioritization via Density Estimation,” *arXiv.org*, 2019. https://arxiv.org/abs/1902.08039 (accessed Apr. 29, 2024).

[45] M. Fang, T. Zhou, Y. Du, L. Han, and Z. Zhang, “Curriculum-guided Hindsight Experience Replay.” Accessed: Apr. 29, 2024. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/83715fd4755b33f9c3958e1a9ee221e1-Paper.pdf

[46] C. Colas, P. Fournier, O. Sigaud, M. Chetouani, and P.-Y. Oudeyer, “CURIOUS: Intrinsically Motivated Modular Multi-Goal Reinforcement Learning,” *arXiv.org*, 2019. https://arxiv.org/abs/1810.06284 (accessed Apr. 29, 2024).

[47] S. Sukhbaatar, Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, and R. Fergus, "Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play," *arXiv.org*, 2017. https://arxiv.org/abs/1703.05407 (accessed Apr. 29, 2024).

[48] C. Florensa, D. Held, X. Geng, and P. Abbeel, "Automatic Goal Generation for Reinforcement Learning Agents," *arXiv.org*, 2017. https://arxiv.org/abs/1705.06366 (accessed Apr. 29, 2024).

[49] S. Racaniere, A. K. Lampinen, A. Santoro, D. P. Reichert, V. Firoiu, and T. P. Lillicrap,"Automated curricula through setter-solver interactions," *arXiv.org*, 2019. https://arxiv.org/abs/1909.12892 (accessed Apr. 29, 2024).

[50] Geoffrey Cideron, "Self-educated language agent with hindsight experience replay for instruction following," 2019, (accessed: Apr. 29, 2024), [Online]. Available: https://openreview.net/pdf?id=S1g_t1StDB

[51] M. Kumar, B. Packer, and D. Koller, "Self-paced learning for latent variable models," Advances in neural information processing systems, vol. 23, 2010. Accessed: May 10, 2024. [Online]. Available: https://proceedings.neurips.cc/paperfiles/paper/2010/file/e57c6b956a6521b28495f2886ca0977aPaper.pdf

[52] M. Gong, H. Li, D. Meng, Q. Miao, and J. Liu, "DecompositionBased Evolutionary Multiobjective Optimization to SelfPaced Learning," IEEE transactions on evolutionary computation, vol.23, no.2, pp.288–302, Apr.2019 https : //doi.org/10.1109/tevc.2018.2850769.

[53] Z. Liu, B. Liu, Z. Zhao, Q. Chu, and N. Yu, "Dual-Uncertainty Guided Curriculum Learning and Part-Aware Feature Refinement for Domain Adaptive Person Re-Identification," Jun. 2023, doi: https://doi.org/10.1109/icassp49357.2023.10097020.

[54]A. Murali, L. Pinto, D. Gandhi, and A. Gupta. Cassl: Curriculum accelerated selfsupervised learning. In 2018 IEEE International Conference on Robotic https://doi.org/10.1109/icra.2018.8463147.

[55] T. Langlois and P. Campos, “Abalearn: Efficient self-play learning of the game abalone,” ICGA Journal, vol. 26, no. 4, pp. 219–228, Dec. 2003, doi: https://doi.org/10.3233/icg-2003-26402.

[56] OpenAI et al., “Dota 2 with Large Scale Deep Reinforcement Learning,” arXiv.org, 2019. https://arxiv.org/abs/1912.06680 (accessed May 10, 2024).

[57] T. A. Berrueta, A. Pinosky, and T. D. Murphey, “Maximum diffusion reinforcement learning,” Nature Machine Intelligence, vol. 6, no. 5, pp. 504–514, May 2024, doi: https://doi.org/10.1038/s42256-024-00829-3.

[58] K. Lee et al., “Generalized Tsallis Entropy Reinforcement Learning and Its Application to Soft Mobile Robots.” Accessed: Jun. 01, 2024. [Online]. Available: https://www.roboticsproceedings.org/rss16/p036.pdf

[59] D. Jarrett, C. Tallec, F. Altché, T. Mesnard, R. Munos, and M. Valko, “Curiosity in Hindsight: Intrinsic Exploration in Stochastic Environments.” Accessed: Jun. 01, 2024. [Online]. Available: https://arxiv.org/pdf/2211.10515

[60] M. Yuan, B. Li, X. Jin, and W. Zeng, “Automatic Intrinsic Reward Shaping for Exploration in Deep Reinforcement Learning.” Accessed: Jun. 01, 2024. [Online]. Available: https://arxiv.org/pdf/2301.10886

[61] T. Lin and A. Jabri, “MIMEx: Intrinsic Rewards from Masked Input Modeling.” Accessed: Jun. 01, 2024. [Online]. Available: https://arxiv.org/pdf/2305.08932

[62] S. Lee, D. Cho, J. Park, and H. Kim, “CQM: Curriculum Reinforcement Learning with a Quantized World Model.” Accessed: Jun. 01, 2024. [Online]. Available: https://arxiv.org/pdf/2310.17330

[63] D. Rengarajan, G. Vaidya, A. Sarvesh, D. Kalathil, and S. Shakkottai, “Published as a conference paper at ICLR 2022 REINFORCEMENT LEARNING WITH SPARSE REWARDS USING GUIDANCE FROM OFFLINE DEMONSTRATION.” Accessed: Jun. 01, 2024. [Online]. Available: https://arxiv.org/pdf/2202.04628

[64] P. Mazzaglia, O. Catal, T. Verbelen, and B. Dhoedt, “Curiosity-Driven Exploration via Latent Bayesian Surprise,” Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, no. 7, pp. 7752–7760, Jun. 2022, doi: https://doi.org/10.1609/aaai.v36i7.20743.

[65] B. Eysenbach, A. Gupta, J. Google, and B. Levine, “DIVERSITY IS ALL YOU NEED: LEARNING SKILLS WITHOUT A REWARD FUNCTION.” Accessed: Jun. 01, 2024. [Online]. Available: https://arxiv.org/pdf/1802.06070

[66] Agoston Restas, “Drone Applications for Supporting Disaster Management,” *World journal of engineering and technology*, vol. 03, no. 03, pp. 316–321, Jan. 2015, doi: https://doi.org/10.4236/wjet.2015.33c047.

[67] Leandro, R. Albergaria, and V. Barcellos, “The use of UAV and geographic information systems for facility location in a post-disaster scenario,”*Transportation research procedia*, vol. 27, pp. 1137–1145, Jan. 2017, doi: https://doi.org/10.1016/j.trpro.2017.12.031.

[68] Sarra Berrahal, J.-H. Kim, Slim Rekhis, Noureddine Boudriga, D. Wilkins, and J. Acevedo, “Border surveillance monitoring using Quadcopter UAV-Aided Wireless Sensor Networks,”*Journal of communications software and systems*, vol. 12, no. 1, pp. 67–67, Mar. 2016, doi: https://doi.org/10.24138/jcomss.v12i1.92.

[69] F. Marube, O. Odongo, and L. Muchemi, “An Autonomous Unmanned Aerial Security Surveillance System to Enhance Security in Remote Territories,”*International Journal of Computer Applications*, vol. 179, no. 6, pp. 975–8887, 2017, Accessed: Apr. 29, 2024. [Online]. Available: https://www.ijcaonline.org/archives/volume179/number6/marube-2017-ijca-915964.pdf

[70] Yunus Karaca *et al.*, “The potential use of unmanned aircraft systems (drones) in mountain search and rescue operations,” *The American journal of emergency medicine*, vol. 36, no. 4, pp. 583–588, Apr. 2018, doi: https://doi.org/10.1016/j.ajem.2017.09.025.

[71] Marzena Półka, S. Ptak, and Łukasz Kuziora, “The Use of UAV’s for Search and Rescue Operations,” *Procedia engineering*, vol. 192, pp. 748–752, Jan. 2017, doi: https://doi.org/10.1016/j.proeng.2017.06.129.

[72] L. Manuelli and P. Florence, “Reinforcement Learning for Autonomous Driving Obstacle Avoidance using LIDAR. 2017,” Accessed: Apr. 29, 2024. [Online]. Available: https://www.peteflorence.com/ReinforcementLearningAutonomousDriving.pdf

[73] Mihai Duguleana and G. Mogan, “Neural networks based reinforcement learning for mobile robots obstacle avoidance,”*Expert systems with applications*, vol. 62, pp. 104–115, Nov. 2016, doi: https://doi.org/10.1016/j.eswa.2016.06.021.

[74] Zhao Yijing, Z. Zheng, Zhang Xiaoyi, and L. Yang, “Q learning algorithm based UAV path learning and obstacle avoidence approach,” Jul. 2017, doi: https://doi.org/10.23919/chicc.2017.8027884.

[75] J. Han, H. Liu, and B. Wang, “A deep Q network for robotic planning from image,” Aug. 2017, doi: https://doi.org/10.1109/icarm.2017.8273235.

[76] J. Wu, S. Shin, C.-G. Kim, and S.-D. Kim, “Effective lazy training method for deep q-network in obstacle avoidance and path planning,” Oct. 2017, doi: https://doi.org/10.1109/smc.2017.8122877.

[77] L. Tran *et al.*, “Reinforcement Learning with Autonomous Small Unmanned Aerial Vehicles in Cluttered Environments.” Accessed: Apr. 29, 2024. [Online]. Available: https://ntrs.nasa.gov/api/citations/20160006499/downloads/20160006499.pdf

[78] I. Karino, K. Tanaka, R. Niiyama, and Y. Kuniyoshi, “Switching Isotropic and Directional Exploration with Parameter Space Noise in Deep Reinforcement Learning,” *arXiv.org*, 2018. https://arxiv.org/abs/1809.06570 (accessed Apr. 29, 2024).

[79] M. Fortunato *et al.*, “Noisy Networks for Exploration,” *arXiv.org*, 2017. https://arxiv.org/abs/1706.10295 (accessed Apr. 29, 2024).

[80] M. Plappert *et al.*, “Parameter Space Noise for Exploration,” *arXiv.org*, 2017. https://arxiv.org/abs/1706.01905 (accessed Apr. 29, 2024).

[81] J. Zhang, Y. Pan, H. Yang, and Y. Fang, “Scalable Deep Multi-Agent Reinforcement Learning via Observation Embedding and Parameter Noise,” *IEEE access*, vol. 7, pp. 54615–54622, Jan. 2019, doi: https://doi.org/10.1109/access.2019.2913235.

[82] C. Szegedy *et al.*, “Intriguing properties of neural networks,” *arXiv.org*, 2014. https://arxiv.org/abs/1312.6199 (accessed Apr. 29, 2024).

[83] X. Yuan, P. He, Q. Zhu, and X. Li, “Adversarial Examples: Attacks and Defenses for Deep Learning,” *arXiv.org*, 2017. https://arxiv.org/abs/1712.07107 (accessed Apr. 29, 2024).

[84] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” *arXiv.org*, 2017. https://arxiv.org/abs/1607.02533 (accessed Apr. 29, 2024).

[85] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, “Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition,” *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS’16*, 2016, doi: https://doi.org/10.1145/2976749.2978392.

[86] E. Wong and K. J. Zico, “Provable defenses against adversarial examples via the convex outer adversarial polytope,” *arXiv.org*, 2017. https://arxiv.org/abs/1711.00851 (accessed Apr. 29, 2024).

[87] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, “Fast and Effective Robustness Certification.” Accessed: Apr. 29, 2024. [Online]. Available: https://typeset.io/pdf/fast-and-effective-robustness-certification-231phozt8x.pdf

[88] T.-W. Weng *et al.*, “Towards Fast Computation of Certified Robustness for ReLU Networks,” *arXiv.org*, 2018. https://arxiv.org/abs/1804.09699 (accessed Apr. 29, 2024).

[89] I. Ilahi *et al.*, “Challenges and Countermeasures for Adversarial Attacks on Deep Reinforcement Learning,” *arXiv.org*, 2020. https://arxiv.org/abs/2001.09684 (accessed Apr. 29, 2024).

[90] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, “Adversarial Attacks on Neural Network Policies,” *arXiv.org*, 2017. https://arxiv.org/abs/1702.02284 (accessed Apr. 29, 2024).

[91] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and Harnessing Adversarial Examples,” *arXiv.org*, 2014. https://arxiv.org/abs/1412.6572 (accessed Apr. 29, 2024).

[92] Chao-Han Huck Yang *et al.*, “Enhanced Adversarial Strategically-Timed Attacks Against Deep Reinforcement Learning,” *arXiv (Cornell University)*, May 2020, doi: https://doi.org/10.1109/icassp40776.2020.9053342.

[93] Ajay Mandlekar, Y. Zhu, A. Garg, L. Fei-Fei, and S. Savarese, “Adversarially Robust Policy Learning: Active construction of physically-plausible perturbations,” Sep. 2017, doi: https://doi.org/10.1109/iros.2017.8206245.

[94] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine, “EPOpt: Learning Robust Neural Network Policies Using Model Ensembles,” *arXiv.org*, 2016. https://arxiv.org/abs/1610.01283 (accessed Apr. 29, 2024).

[95] F. Muratore, F. Treede, M. Gienger, and J. Peters, “Domain Randomization for Simulation-Based Policy Optimization with Transferability Assessment.” Accessed: Apr. 29, 2024. [Online]. Available: https://proceedings.mlr.press/v87/muratore18a/muratore18a.pdf

[96] W. Uther and M. Veloso, “Adversarial Reinforcement Learning,” 2003. Accessed: Apr. 29, 2024. [Online]. Available: http://reports-archive.adm.cs.cmu.edu/anon/anon/usr0/ftp/usr/ftp/2003/CMU-CS-03-107.pdf

[97] A. Gleave, M. Dennis, C. Wild, N. Kant, S. Levine, and S. Russell, “Adversarial Policies: Attacking Deep Reinforcement Learning,” *arXiv.org*, 2020. https://arxiv.org/abs/1905.10615 (accessed Apr. 29, 2024).

[98] J. García and F. Fernández, “A Comprehensive Survey on Safe Reinforcement Learning,” *Journal of Machine Learning Research*, vol. 16, 2015, Available: https://www.jmlr.org/papers/volume16/garcia15a/garcia15a.pdf

[99] Z. Deng *et al.*, “A Probabilistic Model for Driving-Style-Recognition-Enabled Driver Steering Behaviors,” *IEEE transactions on systems, man, and cybernetics. Systems*, vol. 52, no. 3, pp. 1838–1851, Mar. 2022, doi: https://doi.org/10.1109/tsmc.2020.3037229.

[100] Z. Zhang, M. Leibold, and D. Wollherr, “Integral Sliding-Mode Observer-Based Disturbance Estimation for Euler–Lagrangian Systems,” *IEEE transactions on control systems technology*, vol. 28, no. 6, pp. 2377–2389, Nov. 2020, doi: https://doi.org/10.1109/tcst.2019.2945904.

[101] Z. Zhang, D. Wollherr, and Homayoun Najjaran, “Disturbance estimation for robotic systems using continuous integral sliding mode observer,” *International journal of robust and nonlinear control*, vol. 32, no. 14, pp. 7946–7966, Jun. 2022, doi: https://doi.org/10.1002/rnc.6252.

[102] Z. Zhang, K. Qian, B. W. Schuller, and D. Wollherr, “An Online Robot Collision Detection and Identification Scheme by Supervised Learning and Bayesian Decision Theory,” *IEEE transactions on automation science and engineering*, vol. 18, no. 3, pp. 1144–1156, Jul. 2021, doi: https://doi.org/10.1109/tase.2020.2997094.

[103] C. Li, Z. Zhang, A. Nesrin, Q. Liu, F. Liu, and M. Buss, “Instantaneous Local Control Barrier Function: An Online Learning Approach for Collision Avoidance,” *arXiv.org*, 2021. https://arxiv.org/abs/2106.05341 (accessed Apr. 29, 2024).

[104] X. Huang, Ashkan Jasour, M. Deyo, A. Hofmann, and B. C. Williams, “Hybrid Risk-Aware Conditional Planning with Applications in Autonomous Vehicles,” *DSpace@MIT (Massachusetts Institute of Technology)*, Dec. 2018, doi: https://doi.org/10.1109/cdc.2018.8619771.

[105] T. Kim, H. Lee, and W. Lee, “Physics Embedded Neural Network Vehicle Model and Applications in Risk-Aware Autonomous Driving Using Latent Features,” *arXiv.org*, 2022. https://arxiv.org/abs/2207.07920 (accessed Apr. 29, 2024).

[106] T. Nyberg, C. Pek, Laura Dal Col, C. Noren, and J. Tumova, “Risk-aware Motion Planning for Autonomous Vehicles with Safety Specifications,” *KTH*

Publication Database DiVA (KTH Royal Institute of Technology), Jul. 2021, doi: https://doi.org/10.1109/iv48863.2021.9575928.

[107] Majid Khonji, J. Dias, Rashid Alyassi, Fahad Almaskari, and L. Seneviratne, "A Risk-Aware Architecture for Autonomous Vehicle Operation Under Uncertainty," Nov. 2020, doi: https://doi.org/10.1109/ssrr50563.2020.9292629.

[108] V. Francois-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An Introduction to Deep Reinforcement Learning," *Foundations and trends in machine learning*, vol. 11, no. 3–4, pp. 219–354, Jan. 2018, doi: https://doi.org/10.1561/2200000071.

[109] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation," *arXiv.org*, 2017. https://arxiv.org/abs/1703.00420 (accessed Apr. 29, 2024).

[110] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-Aware Reinforcement Learning for Collision Avoidance," *arXiv.org*, 2017. https://arxiv.org/abs/1702.01182 (accessed Apr. 29, 2024).

[111] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," *arXiv.org*, 2015. https://arxiv.org/abs/1506.02142 (accessed Apr. 29, 2024)

[112] I. Osband, C. Blundell, A. Pritzel, and V. Roy, "Deep Exploration via Bootstrapped DQN," *arXiv.org*, 2016. https://arxiv.org/abs/1602.04621 (accessed Apr. 29, 2024).

[113] B. Lütjens, M. Everett, and J. P. How, "Safe Reinforcement Learning with Model Uncertainty Estimates," *arXiv.org*, 2018. https://arxiv.org/abs/1810.08700 (accessed Apr. 29, 2024).

[114] T. Fan, P. Long, W. Liu, J. Pan, R. Yang, and D. Manocha, "Learning Resilient Behaviors for Navigation Under Uncertainty," *arXiv.org*, 2020. https://arxiv.org/abs/1910.09998 (accessed Apr. 29, 2024).

[115] K. Cho, van Merrienboer, D. Bahdanau, and Y. Bengio, "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches," *arXiv.org*, 2014. https://arxiv.org/abs/1409.1259 (accessed Apr. 29, 2024).

[116] A. Garivier and O. Cappé, "The KL-UCB Algorithm for Bounded Stochastic Bandits and Beyond," vol. 19, pp. 359–376, 2011, Accessed: Apr. 29, 2024. [Online]. Available: https://proceedings.mlr.press/v19/garivier11a/garivier11a.pdf

[117] X. Zhu *et al.*, "Interaction-Aware Cut-In Trajectory Prediction and Risk Assessment in Mixed Traffic," *IEEE/CAA journal of automatica sinica*, vol. 9, no. 10, pp. 1752–1762, Oct. 2022, doi: https://doi.org/10.1109/jas.2022.105866.

[118] G. Li, Y. Yang, S. Li, X. Qu, N. Lyu, and Shengbo Eben Li, "Decision making of autonomous vehicles in lane change scenarios: Deep reinforcement learning approaches with risk awareness," *Transportation research. Part C, Emerging technologies*, vol. 134, pp. 103452–103452, Jan. 2022, doi: https://doi.org/10.1016/j.trc.2021.103452.

[119] D. Kamran, C. F. Lopez, M. Lauer, and C. Stiller, "Risk-Aware High-level Decisions for Automated Driving at Occluded Intersections with Reinforcement Learning," *arXiv.org*, 2020. https://arxiv.org/abs/2004.04450 (accessed Apr. 29, 2024).

[120] Sarvesh Kolekar, Joost de Winter, and D. Abbink, "Human-like driving behaviour emerges from a risk-based driver model," *Nature communications*, vol. 11, no. 1, Sep. 2020, doi: https://doi.org/10.1038/s41467-020-18353-4.

[121] A. Majumdar and M. Pavone, "How Should a Robot Assess Risk? Towards an Axiomatic Theory of Risk in Robotics," *arXiv.org*, 2017. https://arxiv.org/abs/1710.11040 (accessed Apr. 29, 2024).

[122] Y. C. Tang, J. Zhang, and R. Salakhutdinov, "Worst Cases Policy Gradients," *arXiv.org*, 2019. https://arxiv.org/abs/1911.03618 (accessed Apr. 29, 2024).

[123] M. G. Bellemare, W. Dabney, and R. Munos, “A Distributional Perspective on Reinforcement Learning,” *arXiv.org*, 2017. https://arxiv.org/abs/1707.06887 (accessed Apr. 29, 2024).

[124] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos, “Distributional Reinforcement Learning with Quantile Regression,” *arXiv.org*, 2017. https://arxiv.org/abs/1710.10044 (accessed Apr. 29, 2024).

[125] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, “Implicit Quantile Networks for Distributional Reinforcement Learning,” *arXiv.org*, 2018. https://arxiv.org/abs/1806.06923 (accessed Apr. 29, 2024).

[126] Urpí, Núria Armengol, S. Curi, and A. Krause, “Risk-Averse Offline Reinforcement Learning,” *arXiv.org*, 2021. https://arxiv.org/abs/2102.05371 (accessed Apr. 29, 2024).

[127] X. Ma, L. Xia, Z. Zhou, J. Yang, and Q. Zhao, “DSAC: Distributional Soft Actor Critic for Risk-Sensitive Reinforcement Learning,” *arXiv.org*, 2020. https://arxiv.org/abs/2004.14547 (accessed Apr. 29, 2024).

[128] D. Kamran, T. Engelgeh, M. Busch, J. Fischer, and C. Stiller, “Minimizing Safety Interference for Safe and Comfortable Automated Driving with Distributional Reinforcement Learning,” *arXiv.org*, 2021. https://arxiv.org/abs/2107.07316 (accessed Apr. 29, 2024).

[129] J. Choi, C. R. Dance, J. Kim, S. Hwang, and K. Park, “Risk-Conditioned Distributional Soft Actor-Critic for Risk-Sensitive Navigation,” *arXiv.org*, 2021. https://arxiv.org/abs/2104.03111 (accessed Apr. 29, 2024).

[130] Ö. M. Gül and A. M. Erkmen, “Energy-Efficient Cluster-Based Data Collection by a UAV with a Limited-Capacity Battery in Robotic Wireless Sensor Networks,” *Sensors*, vol. 20, no. 20, pp. 5865–5865, Oct. 2020, doi: https://doi.org/10.3390/s20205865.

[131] Y. Huang, L. Chen, and P.H.A.J.M. van Gelder, “Generalized velocity obstacle algorithm for preventing ship collisions at sea,” *Ocean engineering*, vol. 173, pp. 142–156, Feb. 2019, doi: https://doi.org/10.1016/j.oceaneng.2018.12.053.

[132] Ö. M. Gül, A. M. Erkmen, and Burak Kantarcı, “UAV-Driven Sustainable and Quality-Aware Data Collection in Robotic Wireless Sensor Networks,” *IEEE internet of things journal*, vol. 9, no. 24, pp. 25150–25164, Dec. 2022, doi: https://doi.org/10.1109/jiot.2022.3195677.

[133] J. Simanek, M. Reinstein, and V. Kubelka, “Evaluation of the EKF-Based Estimation Architectures for Data Fusion in Mobile Robots,” *IEEE/ASME transactions on mechatronics*, vol. 20, no. 2, pp. 985–990, Apr. 2015, doi: https://doi.org/10.1109/tmech.2014.2311416.

[134] Y. Chen, S. Huang, and R. Fitch, “Active SLAM for Mobile Robots With Area Coverage and Obstacle Avoidance,” *IEEE/ASME transactions on mechatronics*, vol. 25, no. 3, pp. 1182–1192, Jun. 2020, doi: https://doi.org/10.1109/tmech.2019.2963439.

[135] H. Zhang, J. Wen, Y. Liu, and N. Xiong, “Mobile Robot Localization Based on Gradient Propagation Particle Filter Network,” *IEEE access*, vol. 8, pp. 188475–188487, Jan. 2020, doi: https://doi.org/10.1109/access.2020.3031618.

[136] M. Hessel *et al.*, “Rainbow: Combining Improvements in Deep Reinforcement Learning,” *arXiv.org*, 2017. https://arxiv.org/abs/1710.02298 (accessed Apr. 29, 2024).

[137] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, “Evolution Strategies as a Scalable Alternative to Reinforcement Learning,” *arXiv.org*, 2017. https://arxiv.org/abs/1703.03864 (accessed Apr. 29, 2024).

[138] Atikah Surriani, Oyas Wahyunggoro, and Adha Imam Cahyadi, “Noise Parameterization of Continuous Deep Reinforcement Learning for a Class of Non-linear System,” Oct. 2022, doi: https://doi.org/10.1109/icitee56407.2022.9954121.