

T.R.
GEBZE TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF DEFENSE TECHNOLOGIES

**PATH PLANNING USING REINFORCEMENT LEARNING
METHODS**

EMRE TAŞÇIOĞLU
A THESIS SUBMITTED FOR THE DEGREE OF
MASTER OF SCIENCE DEPARTMENT OF DEFENCE
ELECTRONICS ENGINEERING

GEBZE
2023

T.R.
GEBZE TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF DEFENSE
TECHNOLOGIES

PATH PLANNING USING
REINFORCEMENT LEARNING METHODS

EMRE TASCIOGLU
A THESIS SUBMITTED FOR THE DEGREE OF
MASTER OF SCIENCE DEPARTMENT OF DEFENCE
ELECTRONICS ENGINEERING

THESIS SUPERVISOR
DR. AHMET GÜNES

GEBZE
2023



YÜKSEK LİSANS JÜRİ ONAY FORMU

GTÜ Enstitüsü Yönetim Kurulu'nun/...../..... tarih ve/..... sayılı kararıyla oluşturulan jüri tarafından/...../..... tarihinde tez savunma sınavı yapılan'ın tez çalışması Anabilim Dalında YÜKSEK LİSANS tezi olarak kabul edilmiştir.

JÜRİ

ÜYE
(TEZ DANIŞMANI) :

ÜYE :

ÜYE :

ONAY

Gebze Teknik Üniversitesi Savunma Teknolojileri Enstitüsü Yönetim Kurulu'nun

...../..... tarih ve/..... sayılı kararı.

İMZA/MÜHÜR

SUMMARY

This study focuses on the development of path planning applications for autonomous vehicles utilizing various reinforcement learning methods. Reinforcement learning methods have been among the most successful techniques in autonomous control systems as they provide optimal actions for the agent. Firstly, the Q-learning method was developed for path planning in static environments, with it operating in discrete state and action spaces. In the second part of the study, deep Q-learning models were developed for dynamic environments, with the extended Kalman filter being used in environment observation. In the last part, all methods applied to the problem were compared with each other.

Key Words: Reinforcement Learning, Path Planning, Dueling Double Q-Learning, Extended Kalman Filter

ÖZET

Bu çalışmada birden fazla pekiştirmeli öğrenme yöntemi kullanılarak otonom araçlar için yol planlama uygulamaları geliştirilmiştir. Pekiştirmeli öğrenme teknikleri başarılı otonom kontrol sistemlerinden biridir. Pekiştirmeli öğrenme ajan için ideal hareketin bulunmasını sağlamaktadır. Uygulanan ilk metot Q-öğrenmesi olmuştur. Q öğrenmesi ayrık durum ve politika uzayında statik çevre için çalışmaktadır. Çalışmanın ikinci bölümünde dinamik çevre için derin Q-öğrenmesi modelleri geliştirilmiştir. Çevre gözlemlenmede Kalman filtresi kullanılmıştır. Son bölümde tüm yapılan çalışma ve metotlar karşılaştırılmıştır.

Anahtar Kelimeler: Pekiştirmeli Öğrenme, Derin Q-Öğrenmesi, Genişletilmiş Kalman Filtresi

ACKNOWLEDGEMENTS

I would like to express my deep thanks to my supervisor, Dr. Ahmet GÜNEŞ. His guidance provided me experiences and information.

I am grateful to my parents for their love and support.

Finally, I want to thank to my wife Elif TAŞÇIOĞLU for her limitless support and understanding.



TABLE of CONTENTS

	<u>Page</u>
SUMMARY	iv
ÖZET	v
ACKNOWLEDMENTS	vi
TABLE of CONTENTS	vii
LIST of ABBREVIATIONS and ACRONYMS	ix
LIST of TABLES	x
LIST of FIGURES	xi
1. INTRODUCTION	1
1.1 Motivation	2
1.2 Thesis Objectives and Methodologies.....	3
1.3 Main Results	3
1.4 Thesis Structure	4
2. PATH PLANNING WITH VALUE BASED METHODS	5
2.1. Path Planning with Q-Learning.....	6
2.1.1. Reinforcement Learning Framework	6
2.1.2. Q-Learning.....	7
2.1.3. Simulation Environment.....	9
2.1.4. Simulation Results	10
2.1.5. Path Planning for Passive Sonar Environment	11
2.2. Path Planning with Double Dueling Deep Q-Learning.....	14
2.2.1. Deep Q-Learning.....	14
2.2.2. Simulation Environment	20
2.2.3. CNN Approach and Path Planning	21
2.2.4. Statistical Approach and Path Planning	25

2.2.5. Extended Kalman Filter	30
2.2.6. CNN Approach Path Planning with EKF.....	32
2.2.7. Statistical Approach Path Planning with EKF.....	34
2.2.8. Statistical Approach Path Planning with EKF Using Uncertainty Parameter.....	38
 3. RESULTS	 40
3.1. Results	41
3.2. Conclusion	43
 REFERENCES	 44

LIST OF ABBREVIATIONS AND ACRONYMS

<u>Abbreviations</u>	<u>Explanations</u>
<u>and Acronyms</u>	
ϵ	: Epsilon
α	: Learning Rate
γ	: Gamma
a	: Action
r	: Reward
S	: State
$Q(s, a)$	: Q-Value
SL	: Source Level
TL	: Transmission Loss
NL	: Noise Level
AG	: Antenna Gain
P_D	: Probability of Detection
P_{FA}	: Probability of False Alarm
τ	: Hyper parameters
$A(s, a)$	: Advantage Value
$V(s)$	: Value
$L(w)$	: Loss
CNN	: Convolutional Neural Network
QL	: Q-Learning
$DDQN$	: Double Deep Q-Learning
DQN	: Deep Q-Learning
EKF	: Extended Kalman Filter
K	: Kalman Gain
P	: Covariance Matrix
$h(x)$	: Measurement Model Function
R	: Measurement Noise Covariance Matrix
Q	: Process Noise Covariance Matrix
PRM	: Probabilistic Road Map

RRT : Rapidly Exploring Random Trees
RL : Reinforcement Learning
DDDQN : Dueling Double Deep Q-Learning



LIST OF TABLES

<u>Table No:</u>	<u>Page</u>
2.1: Hyper Parameters.	5
3.1: Training Statistics.	41



LIST OF ALGORITHMS

<u>Table No:</u>	<u>Page</u>
2.1: Q-Learning control algorithm.	8
2.2: Dueling DDQN network input math.	22
2.3: Pseudocode of reward function.	25
2.4: Pseudocode path planning using EKF.	36



LIST OF FIGURES

<u>Figure No:</u>	<u>Page</u>
2.1: Q-learning principle.	7
2.2: Q-learning action directions.	9
2.3: Q-learning map and optimum path.	10
2.4: Q-learning score curve.	11
2.5: Q-learning with sensors optimum path.	14
2.6: Double deep q-learning diagram.	16
2.7: Dueling DDQN network.	19
2.8: Random taken two test scenarios.	20
2.9: CNN input frames.	22
2.10: Final path CNN represented scenarios.	24
2.11: Dueling DDQN statistical input math.	25
2.12: Random taken two test scenarios.	29
2.13: Extended Kalman filter sequence.	31
2.14: CNN represented EKF input frames.	32
2.15: CNN represented path planning with Kalman filter result.	33
2.16: Graph of P_{kk} diagonals.	36
2.17: Statistical represented path planning with Kalman filter result.	42
3.1: Score curve of five models.	43

Chapter 1

INTRODUCTION

1.1 Motivation

1.2 Thesis Objective and Methodology

1.3 Main Results

1.4 Thesis Structure

1.1 Motivation

Research on autonomous systems is rapidly increasing. In this study, we focused specifically on autonomous driving, which is a subcategory of autonomous decision-making. The main goal of autonomous driving is for the vehicle to reach its destination by observing the environment and controlling its systems. With the ever-increasing capabilities of processors, artificial intelligence studies are becoming more prevalent, and research into autonomous driving has become more feasible.

There are many traditional methods for path planning. However, in all traditional methods the environment is evaluated as static. *PRM*, A-star, *RRT* are some of these methods. In real world applications, the environment is dynamic. For complex environments, these methods are not efficient. In traditional methods, path planning is re-executed for each environment, this increases the computation time [1].

With the increasing processing power, *RL* methods can be an alternative to traditional methods. Path planning studies using reinforcement learning methods have been frequently published in the literature in recent years. There are path planning studies with Q-learning in discrete state space for static environments [2]. The need for retraining the Q-learning method for each environment can be time-consuming and inefficient. Therefore, new methods have been developed to overcome this limitation and improve the efficiency of training in dynamic environments. These new methods often use continuous state spaces and neural networks to learn and adapt to changing environments. The Q-learning method has been expanded to continuous state space using artificial neural networks [3].

Q-learning works in discrete space and the training times are limited; studies have been carried out with advanced methods of RL. There are studies using policy based deep RL methods. The policy gradient-based methods are suitable for parallel training increasing the training speed [4].

The aim of this thesis is to contribute to the development of autonomous technologies by using value-based RL techniques. Models were created for path planning using different RL methods. These methods are applied to both static and dynamic environments. Different feature extraction methods applied to the observed environments.

The actual driving scenarios are much more complex than ideal simulators. Solutions that are valid in the simulation world may not be successful in real life. In this study, Kalman filter is used to mitigate the uncertainty of the sensors.

1.2 Thesis Objective and Methodology

The basis of this study is to build an agent that could autonomously control itself, decide on no-actions or choose a motion direction. With the development of technology, studies for vehicles that will operate autonomously independently of people are increasing. Safe and fast movement are among the most important expectations for these vehicles.

In this study, a general solution is sought for autonomous systems to take a safe route. It is intended that the results can be applied to different environments, regardless of the platform.

The input information of the intelligent agent is observed environment and agent's status parameters. In this study, two different observation approaches are followed. The first one takes sequential images as input. It uses convolutional layers to extract information from these images. The second approach is based on the statistical representation of the environment. The EKF outputs are used as inputs to the models.

In this thesis the value-based *RL* methods were used. These methods are Q-learning, deep Q-learning, double deep Q-learning and DDDQN.

1.3 Main Results

Path planning problem is very complex. The complexity of the problem is related to the environmental characteristics. The dimension of environment, step size, velocity of vehicle, velocity of obstacles, obstacle count, and obstacle size define complexity. It is easier to find the optimum path in static environments. Satisfactory results can be obtained with traditional methods such as A-star and Dijkstra algorithms. However, the complexity increases in dynamic environments, and traditional methods may not be sufficient for such complex environments. Models were created with RL methods for complex environments.

In the statistical input approach, all parameters such as target position, ego position, obstacle positions, velocities, etc. are augmented to form a state vector and given to the system as input. In CNN, only last four frames of map are given to the model as input. Although statistical methods provide a faster solution, the input counts are limited. With the increasing number of obstacles, the network must be retrained. However, in the CNN input approach, since all features are already present in the frames, there is no need for retraining.

Scores, success rates, and training times are compared each other's. As a result, DDDQN method is found to learn faster in path planning problem.

1.4 Thesis Structure

In the first section, the path planning problem is defined. The discrete environment definitions are made. A definition for Q-learning algorithm is given. State space, action space and reward functions are explained. Path planning is implemented in a static environment with Q-learning, which is a tabular method. The results are evaluated.

For dynamic environments, deep Q-learning, double deep Q-learning and DDDQN methods were applied. Two different space representation approaches are given to the model. These two approaches are the statistical approach and the *CNN* approach. In this study, these methods were used to construct the state space of the model. In dynamic environments, a single moving obstacle is present with constant speed and course.

Perfect observations can be made in the simulation environment. To obtain more realistic results that resemble real environments, in some scenarios the environment is estimated using the Kalman filter. In last episode of the study, the obtained results are compared.

Chapter 2

PATH PLANNING WITH VALUE BASED METHODS

2.1 Path Planning with Q-Learning

2.2 Path Planning with Dueling Double Deep Q-Learning



2.1 Path Planning with Q-Learning

2.1.1 Reinforcement Learning Framework

RL is one of the fundamental branches of machine learning, along with supervised and unsupervised learnings. The most important feature that distinguishes it from supervised learning is that it does not require labeled data. RL focuses on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge) [5]. It is a structure in which an agent receives a scalar reward for positive results and a penalty for negative results. The agent's goal is to maximize the reward by fully observing the environment.

The use of *RL* methods is spreading in the field of robotics, as these studies enable automatic decision systems to provide results close to human decisions. The application of *RL* methods to industrial products is increasing day by day. For example, there are studies using *RL* methods for path planning, which is one of the most important modules of robot vacuums [6]. In this study, our aim is to develop an algorithm that can find a safe and short path for an autonomous vehicle by using *RL* methods.

State-Action-Reward-State-Action (SARSA) is an on-policy reinforcement-learning algorithm used to train a new policy for a Markov decision process. A state represents the status of the environment, while an action is the output of an RL method and is determined by the policy. The reward is the value obtained because of each action, and it is used to evaluate the effectiveness of the chosen action in the current state.

The Bellman equation is a fundamental component in solving RL problems and is widely used in the field. It helps to solve Markov decision processes (MDP) by computing the value of a given state. The value of a state is equal to the maximum action that leads to the highest reward in that state, plus a discount factor multiplied by the value of the next state, as derived from the Bellman equation. This equation is essential for finding the optimal policy and value functions in RL problems. Equation 2.1 shows the Bellman equation.

$$V(s_a) = \max_a R(s, a) + \gamma V(s') \quad (2.1)$$

where $V(s_a)$ is value of state for action. $R(s, a)$ is the reward for (s, a) . γ is gamma. It defines the effect of next state's value. $V(s')$ is next state's value.

2.1.2 Q-Learning

One of the most important breakthroughs in RL was the development of an off-policy temporal-difference control algorithm known as Q-learning [7]. Q-learning is a popular RL algorithm that is used to find an optimal action-selection policy for a given Markov decision process (MDP). It is based on the concept of learning the action-value function, which gives the expected return for taking a particular action in each state. Q-learning is a tabular method. q-function values are kept in a lookup table. The flow of Q-learning algorithm is given in figure 2.1.

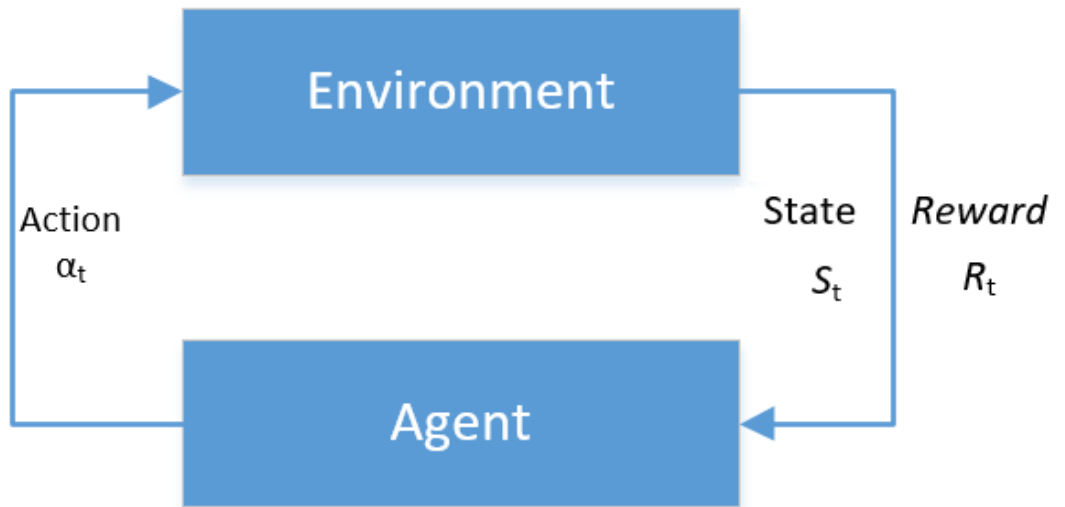


Figure 2.1: The relationships between the parameters in Q-learning.

The Q-learning algorithm uses the BELLMAN equation to update the action-value function iteratively, based on the rewards and transitions observed from the environment. The goal of the algorithm is to find the action-value function that maximizes the expected return for the agent. The main purpose of the algorithm is to examine the next moves and see the score they will earn according to the moves they will make and aim to maximize this score and act accordingly. The Q-learning state

uses the Q-value table, which stores the action and reward relation. This Q-table has a row for each state and a column for each action. Since Q-learning is an iterative algorithm, it implicitly assumes an initial condition before the first update occurs. High initial values, also known as "optimistic initial conditions" [7]. These values are updated iteratively using the reward or penalty obtained on each run. Q-table update function is given in equation 2.2.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_i Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (2.2)$$

where $Q(s_t, a_t)$ is the action-value function, α is learning rate which influences the learning speed and convergence towards the optimal policy. r_t is the reward received after acting. The value table was updated at each episode according to rewards.

In RL, the reward function plays a crucial role in guiding the learning process of the agent. The reward function maps the current state and action of the agent to a scalar value, which serves as feedback on the quality of the agent's behavior. The γ , determines the importance of future rewards in the agent's decision-making process. A higher gamma value places more weight on future rewards, while a lower gamma value prioritizes immediate rewards. The choice of gamma depends on the specific problem and the trade-off between short-term and long-term rewards.

```

Initialize  $Q(s, a)$  arbitrarily
Repeat (for each Episode)
    Initialize  $s$ 
    Repeat (for each Iteration)
        Choose  $a$  from  $s$  using  $\epsilon - Greedy$ 
        Take action  $a$ , observe  $r, s'$ 
         $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_i Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$ 
         $s \leftarrow s'$ 
    Until  $s$  is terminal

```

Algorithm 2.1: Q-learning control logic.

Q-learning control logic is given in figure 2.1. Exploration/Exploitation methodology used for main exploration logic. ϵ and ϵ decay parameters provide to finding optimal solution of problem. Name of this method is ϵ greedy. In this method initialization ϵ is defined as one. ϵ is reduced by ϵ greedy parameter per iteration. It explores when it is close to one and exploits when it is close to zero. Each movement is called iteration and each final is called episode. ϵ is updated at the end of each episode.

2.1.3 Simulation Environment

In the first scenario, discrete action and state spaces were defined. The state size is 100x60, action size is eight. The agent can move up, down, right, left, up-left, down-left, up-right- down-left. The Action space is given in Figure 2.2.

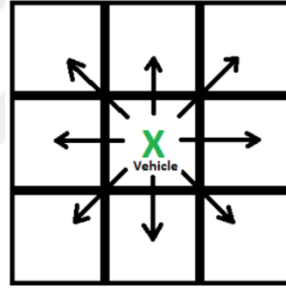


Figure 2.2: Q-learning action space.

In simulation environment, the obstacles are randomly placed. These obstacles represent blocked areas of the environment. The expected behavior of the agent is to avoid obstacles and reach at the target position. The agent gets -0.1 penalty for each movement and when reached to the target gets 100 rewards.

The map is discretized using 60x100 grids. Each grid indicates a state in the value table. The initial value of each grid is zero, within the start of the training each value table row and column fills according to rewards of episode. Greedy mechanism is used for exploration. The parameters of simulation, ϵ starts from 1.0, and decreases in each episode with a decay constant. ϵ decay is 0.99995. γ is 0.99 and α also is 0.99.

In the simulation, a random number is taken for each iteration. If it is bigger than ϵ , chosen optimum policy its mean is selected the action which is belong maximum

When the results are analyzed, static environments can find the optimum path with a Q-learning method for discrete state space. However, when the target or obstacle locations are changed, the existing Q-table values become invalid. In these cases, the Q-tables should be updated, meaning that they need to be retrained. However, continuous training in dynamic environments is not efficient in terms of real autonomous systems. Therefore, for dynamic environments, a continuous state space is required.

2.1.5 Path Planning for Passive Sonar Environment

Training can be provided by using different objective functions in path planning studies [8]. A solution space can be created for different problems by customizing the reward function according to the risk function.

In military applications, autonomous vehicles must penetrate enemy defenses without being detected. In this study passive sonar equations and signal-to-noise ratio are used for risk function. Passive sonar is a technique used to detect and locate underwater objects by listening for sounds produced by the objects. The equations used in passive sonar are based on the principles of acoustics and the properties of sound propagation under the water. In military operations, it is often important to transport personnel or equipment without being detected by enemy forces. For detection of subsurface vehicles there are several detection technologies. One of them is sonobuoys. These can be considered as passive sonars. In this study, passive sonars are simulated. The SNR values of the reflected signals directly affect the probability of detection. In this study, it was aimed to determine the SNR value as a reward function and to perform path planning without being detected. Sonar equations are used to calculate the probability of detection of a vehicle by sonar systems. As a result of these equations, the current SNR value is calculated. The probability of detecting the vehicle is calculated using the SNR value. The variable affecting the SNR value is the distance of the vehicle to the radar. The reward from the current action is obtained by combining the distance to sonar and the energy consumption. w_1 and w_2 represents weight of cost reward parameter. The objective function can be updated by changing

the weights. The distance weight (w_1) directs the agent to take the shortest path, while the sonar reward weight (w_2) causes the agent to move away from the sonar.

$$r_t = w_1 r_e + w_2 r_d \quad (2.3)$$

r_e shows the energy consumption in driving between states. It takes value between 1-200. The distance calculation between the states is given in equation 2.4.

$$r_e = -||s_2 - s_1||^2 \quad (2.4)$$

r_d represents the probability of detection penalty. It takes value between 0-100, calculated using passive sonar equations [9]. The SNR calculation is given in equation 2.5.

$$SNR = SL - TL - NL + AG \quad (2.5)$$

SL represents the source level of sound and is often referred to as the amplitude of the sound. The source level is related to the power of the sound source, and it provides a measure of the potential loudness of the sound at a given distance. TL represents transmission level refers to the power or intensity of the sound waves that are being emitted by an underwater object or activity. The transmission level of a sound is typically measured in decibels (dB) and is an important factor in determining the range and detectability of the sound by a passive sonar system. NL represents noise level; it refers to the intensity of unwanted or background sounds that can interfere with the detection and analysis of signals produced by underwater objects or activities. Noise can come from a variety of sources, including natural sources such as ocean currents and biological activity, as well as man-made sources such as shipping traffic and naval operations. AG represents array gain; it refers to the increase in the strength or intensity of a received signal that is achieved by using an antenna with a particular pattern or shape. Antenna gain is a measure of the antenna's ability to focus or amplify the energy of a received signal and is measured in decibels (dB). In this study, SL, NL, and AG are assumed constant. TL varies depending on distance [10]. TL calculation is given in equation 2.6.

$$TL = 10 \log \left(\frac{1}{2\mu R} \right) - 10 \log (R) \quad (2.6)$$

R is the distance between the vehicle and the sonar. To calculate p_D at each sensor, SNR must be calculated for each sensor. Based on the SNR, a detection threshold is set. When a signal is detected that is above this threshold, the algorithm considers it to be a real target and generates an alarm. If the signal is below the threshold, it is considered noise or clutter and is ignored. A commonly used value for p_{FA} is 10^{-4} . The relation between the p_{FA} and SNR [11] is given in equation 2.7.

$$p_D = p_{FA}^{1/(1+SNR)} \quad (2.7)$$

If $SNR < 0$, p_D is assumed zero [11]. Once $p_D^{(s)}$ which is p_D at the sensor calculated, the probability of being not detected by at least one of the sensors at time t and the corresponding reward is,

$$r_d = 1 - \prod_{s=1}^S (1 - p_D(s)) \quad (2.8)$$

S is the number of sensors. Simulation was conducted on pc which has 16gb ram, AMD Razer 5 1600 CPU. Training simulation took about 124 seconds. Four thousand episodes were run. The environment map and optimum route recommended to the vehicle is given in Figure 2.5. Red heat map circles represent sonars and their coverage. This means the detection zones of the UAV correspond to these areas. This the UAV gets bigger penalties in these areas.

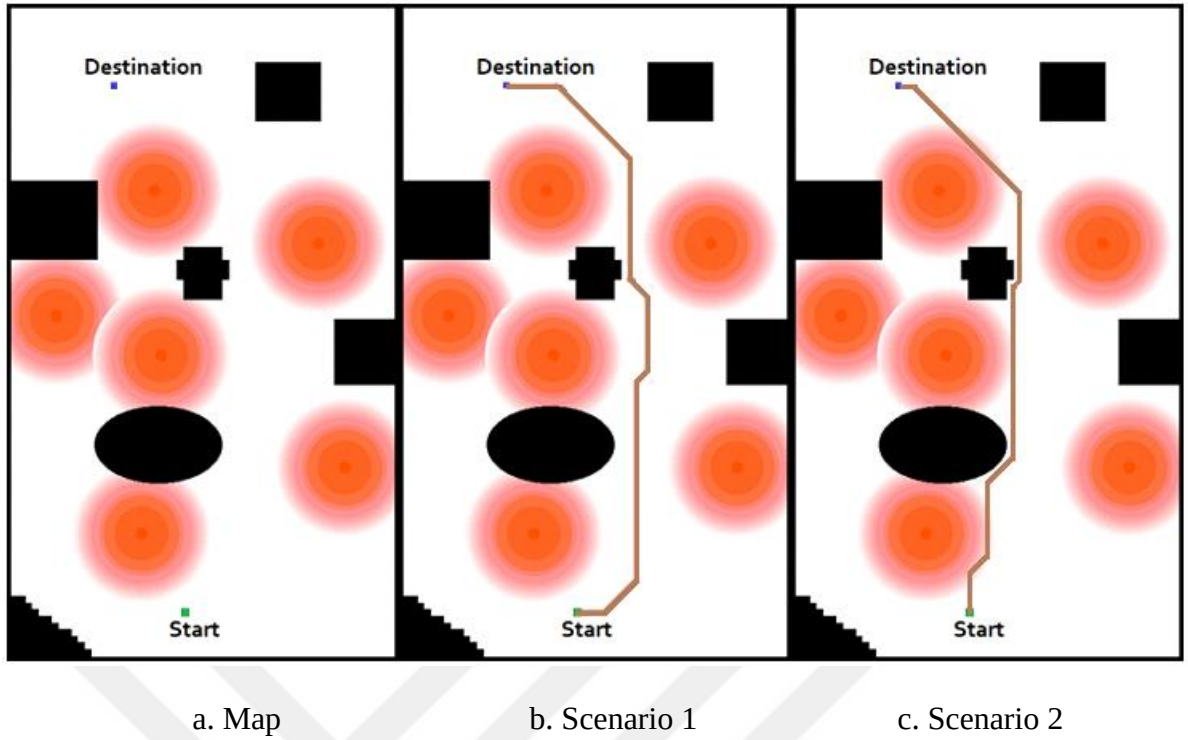


Figure 2.5: Q-learning with sensors optimum path.

In the first scenario, the weight of sonar penalty w_1 is 1, the distance penalty w_2 is 0.2. The p_D for this optimum path is 1.286% and the distance is 10.28 kilometers the result map is given in in Figure 2.5 (b). In the second scenario, the weight of sonar penalty w_1 is 0.2, the distance penalty w_2 is 1. The p_D for this optimum path is 84.852% and the distance is 9.14 kilometers the result map is given in in Figure 2.5 (c). In the results obtained, it is seen that path planning can be done by using different risk functions with Q-learning. But since it finds solution for static environment, it requires retraining for every change in map.

2.2 Path Planning with Dueling Double Deep Q-Learning

2.2.1 Deep Q-Learning Methods

Deep learning provides an objective function by using neural networks to solve a problem. Neural networks are used in deep learning. Deep RL is the use of RL methods by applying deep learning methods. Deep Q-learning combines the Q-learning algorithm with deep neural networks to approximate the Q-value function. Research

in this area aims to establish an autonomous decision mechanism for highly complex environments. [12]. Deep Q-learning is a type of RL algorithm that uses deep neural networks to approximate the Q-value function. In RL, an agent learns to interact with its environment by taking actions and receiving rewards based on those actions. The Q-value function is used to determine the best action to take in each state. In deep Q-learning, the Q-value function is approximated using a deep neural network, which allows the algorithm to handle complex, high-dimensional state spaces. This makes Deep Q-learning a powerful tool for solving problems in RL, such as playing video games or controlling robots. A deep q network (DQN) is a multi-layered neural network that for a given state s outputs a vector of action values $Q(s, \theta)$, where θ represents the parameters of the network [12]. In the deep Q-network method, the current state vector is obtained from the neural network. After the $Q(s_t, a_t)$ for the current s is determined from the network, the action is performed according to a greedy method like Q-learning.

When neural networks are used directly to approximate states, Q-learning becomes unstable. The DQN algorithm has several techniques to stabilize learning, including experience replay, where the agent stores its experiences in a replay buffer and samples them randomly for learning. The experience replay technique was used in the DQN model trained in this study. The goal of the DQN algorithm is to minimize this loss function, which is achieved by adjusting the parameters of the neural network during training. The use of the target network helps stabilize learning by providing a more stable target for the Q-value estimation. Loss calculation is given in equation 2.8. The model trains according to the calculated loss.

$$LOSS = r_{t+1} + \gamma \max_i Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \quad (2.9)$$

Double Deep Q-network (Double DQN) is an improvement to the deep q-network (DQN) algorithm. The basic idea behind Double DQN is to address the overestimation problem that occurs in DQN. In DQN, the target Q-value is calculated using the same network that is used to estimate the current q-value. Double DQN works by using a separate network, called the target network, to select the action. The target network is a copy of the main network that is frozen for a certain number of iterations before

being updated with the main network's parameters. This ensures that the target network does not overestimate the Q-values, and the updates are more stable.

In double DQN, two networks are used: a target network and an evaluation network. The evaluation network selects the best action, and the target network evaluates the expected reward for the selected action. The target network is updated periodically with the weights from the evaluation network, so that it remains relatively stable over time, avoiding the oscillations that can occur in DQN [13]. Loss calculation is given in equation 2.6. The model trains according to the calculated loss.

$$LOSS = r_{t+1} + \gamma \max_i Q_{Target}(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \quad (2.10)$$

Double Deep Q-Learning Diagram is given in Figure 2.6.

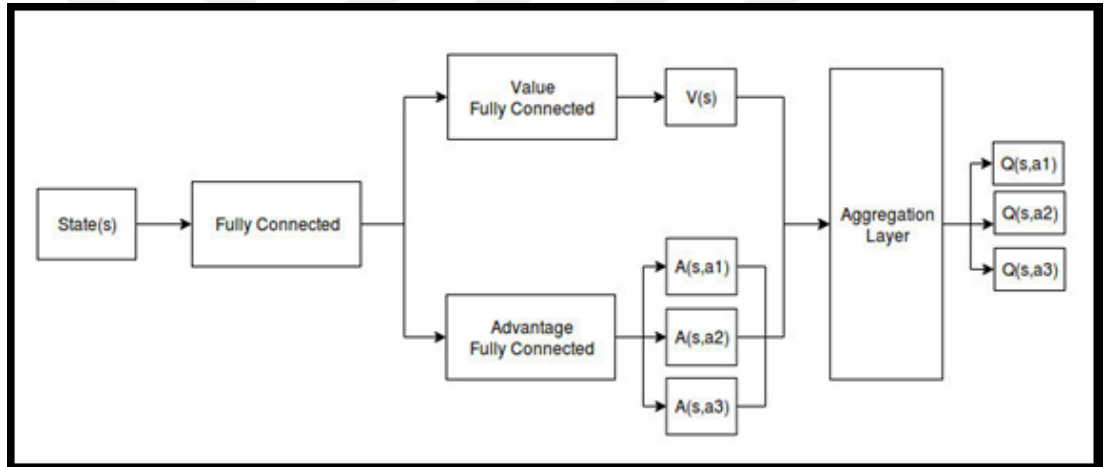


Figure 2.6: Double deep Q-learning diagram.

DDDQN is a variant of the deep Q-learning (DQN) algorithm, which is a type of RL algorithm. It is used to learn a policy for an agent to follow in an environment to maximize a reward.

In traditional DQN, a single neural network is used to estimate the expected return (i.e., the value) of taking a particular action in each state. The DDDQN algorithm, on the other hand, uses two separate neural networks to estimate the value and advantage of each action more accurately. By separating these two streams, the algorithm can learn more efficiently and accurately, leading to better performance. The final action value is calculated by combining the estimates of the value and advantage functions.

Specifically, the Q-value is calculated as the sum of the state value and the advantage, minus the average advantage across all actions.

In DDDQN model one fully connected dense layer which has 256 nodes and 2 different fully connected layers used for advantage and value layers [14]. Network schema of method is given in Figure 2.7.

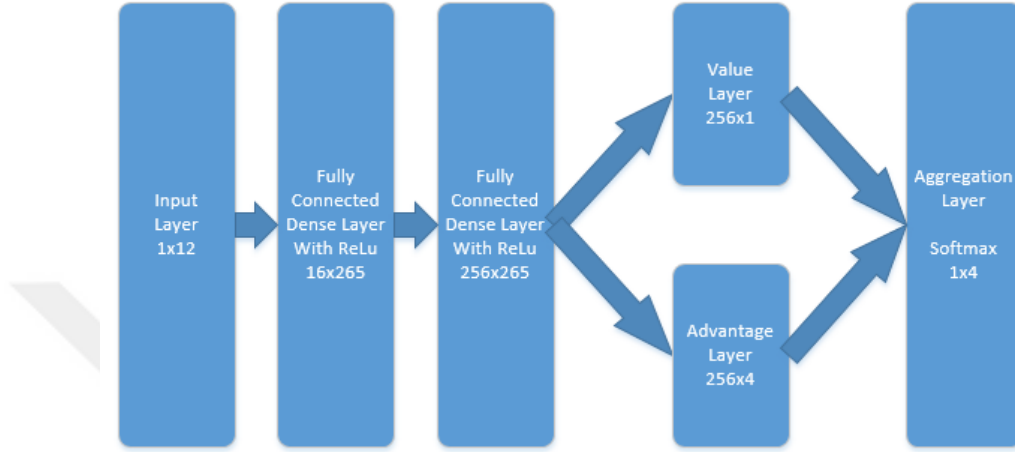


Figure 2.7: DDDQN network.

All input vectors are normalized and given to the network. ReLU was used as the activation function. In the context of neural networks, the ReLU is a type of activation function that has been widely used in deep learning models. It is defined as:

$$f(x) = \max(0, x) \quad (2.13)$$

In other words, the ReLU function takes a real-valued input x and returns 0 if x is less than 0, and returns x if x is greater than or equal to 0. The ReLU function has several advantages compared to other activation functions. It is very simple to compute, as it only requires a comparison and takes the maximum value. It also has a quite simple derivative, which is either 0 or 1 depending on the value of x . This makes it easy to compute the gradient during backpropagation, which is an important step in training a neural network.

The aggregation layer is the transaction layer of dueling networks. In this layer, $Q(s, a)$ is improved by processing $A(s, a)$ pairs from the advantage network with value output.

Overall, DDDQN is a more advanced and refined version of DQN that can improve the performance of RL algorithms in certain environments.

In this study, two different methods were used as the state function. These densifications were examined under two different titles as statistical and CNN, and the results were evaluated. The dueling double deep Q-learning algorithm involves several steps, including:

Initialization: Initialize the neural networks for estimating the value function and advantage function, as well as other parameters such as the γ and α .

Experience replay: Store the agent's experiences (i.e., state-action pairs) in a buffer, and sample a state parameter of past experiences from the buffer at regular intervals. It is a technique used for improving the constancy and performance of algorithms. It involves storing many past experiences or transitions, which consists of the state, action, reward, and next state at each time step, in a memory buffer. The learning algorithm can then sample from this buffer at random to learn from these past experiences.

Target network updates: periodically update the target network, which is used to estimate future reward.

Value and advantage estimation: Use the neural networks to estimate the value and advantage of all actions for current state.

Action selection: Select the action which has the highest estimated value.

BELLMAN equation update: Use the BELLMAN equation to update the main network's weights based on the estimated value and advantage of the selected action. The specific equation for updating the main network's weights using the BELLMAN equation.

Model is aimed to minimize the error between the estimated Q-values and the second Q-values, with the bellman equation. This is done using backpropagation and gradient descent [16].

The target network is updated periodically to ensure that it stays in sync with the primary network. The update equation for the target network is as follows:

$$w' \leftarrow \tau w + (1 - \tau)w' \quad (2.11)$$

w' is the current weight of the target network, w is the current weight of the primary network, τ is a hyperparameter that controls the rate of update for the target network. τ value of 0 means that the target network is not updated, while value of 1 means that the target network is updated with the primary network's weights at every iteration.

This equation effectively combines the primary network's weights with the target network's weights, with the relative influence of each depending on the value of τ . This helps to ensure that the target network stays in sync with the primary network, while also allowing it to retain some of its own "memory" of past experiences.

In practice, the target network is typically updated at regular intervals, rather than at every iteration, to help stabilize the learning process and improve the overall performance of the algorithm.

Dueling double deep Q-learning and double deep Q-learning algorithms Pseudocode is same. The only difference that separates the two from each other is the calculation of Q-value in the forward transaction. The dueling network has two streams to separately estimate (scalar) state-value and the advantages for each action. Both networks output q-values for each action. The scheme is given in figure 2.7.

In the dueling method outputs shows current state is important or not, this feature prevents retraining for unnecessary states. This is particularly useful in states where its actions do not affect the environment in any relevant way [17].

In the output of dueling double deep Q-learning, two separate values are obtained, the value function and the advantage function. The value function $V(s)$ gives info about future reward for state s . The advantage function gives info about action-reward matches. Combining the value V and the advantage A for each action, the Q-value of state is calculated.

$$Q(s, a) = V(s) + (A(s, a) - \frac{1}{|A|} \sum_{a'} A(s, a')) \quad (2.12)$$

Back-propagation is done according to the reward value obtained from the action performed with the Q-values obtained as a result of this equation. Mean Square error is used as back propagation function.

The score comparison of three deep Q-learning methods are given in figure 2.8.

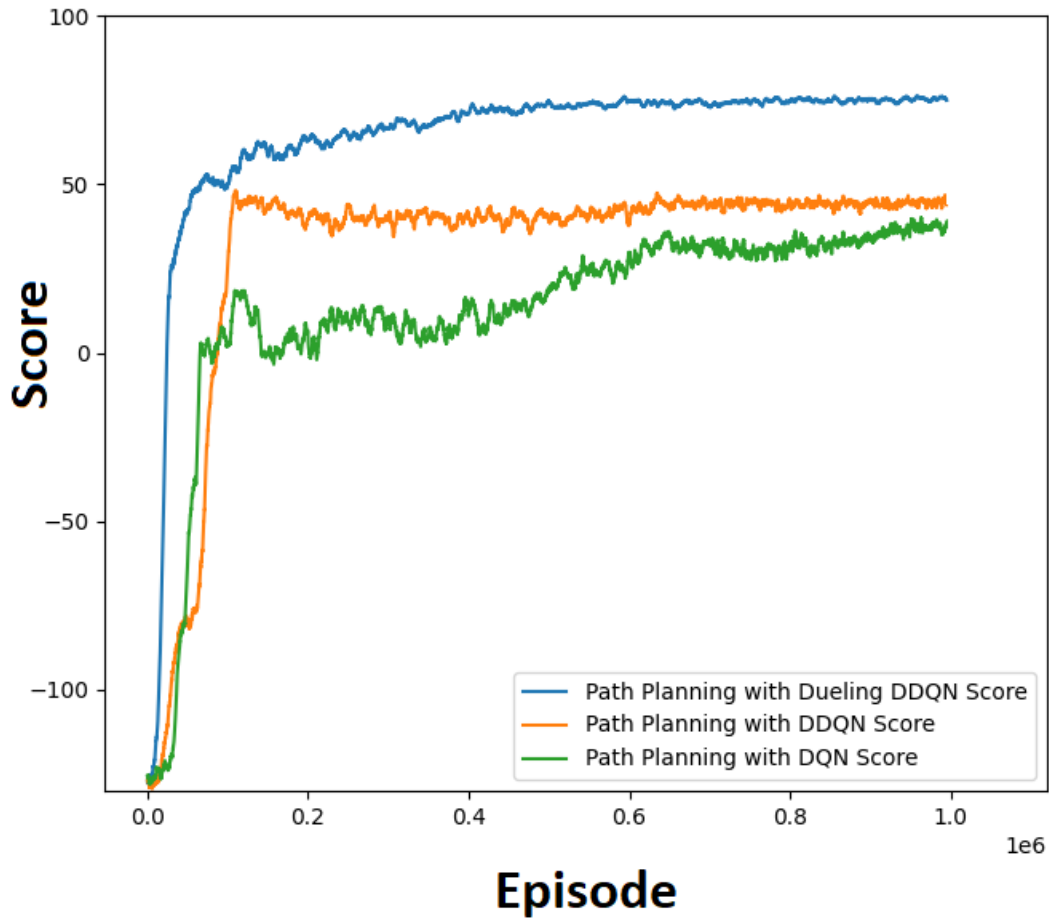


Figure 2.8: DQN methods score curve.

As seen in the score chart, the DDDQN method is more successful than the other two methods. Therefore, DDDQN method was used in this study.

2.2.2 Simulation Environment

Python is a high-level programming language that is widely used for machine learning applications. The models in this study were developed using Python. Python libraries were utilized to develop algorithms and create environments, while the PyTorch library was selected for implementing artificial neural networks.

In this study, a custom environment was created using Pygame environment builder, which is one of the most important modules for simulating the system and finding the closest solution to a real environment. Pygame provides a video game with graphics, sound, and keyboard control that can be used to create custom environments

for Python programming language. Additionally, Pygame offers functionalities such as 2D graphics, text rendering, and audio playback, which can be used to create games or other interactive programs.

Pygame is commonly used for creating simple 2D games, as it provides a simple interface for handling common tasks such as rendering graphics, playing sounds, and detecting user input. It also includes a built-in game loop and support for creating custom game loops, which makes it easy to develop games using Pygame. It provides a wide range of flexibility, easy customization, low CPU power and memory consumption.

The obstacle in simulation is mobile. It has constant velocity vector for each iteration. The input vector is defined as the relative distances to the vehicle and global locations of obstacle, target, and vehicle. The first relative vector is the margin between vehicle and obstacle, the other one is the distance between vehicle and target. In all simulations, the input state vectors are normalized.

The output vector is given as one of the three actions. These are no-action, 30 turn degrees right, turn 30 degrees left, and move straight. The environment is composed of an 8.4x8.4 km area. Each action represents one hundred meters of movement step.

2.2.3 CNN Approach and Path Planning

In RL, CNN can be used to process visual input from sensors, such as cameras, to allow an RL agent to make decisions based on its surroundings. For example, a self-driving car could use a CNN to process video feeds from its cameras to identify other vehicles, pedestrians, traffic lights, and other objects in the environment and make decisions about how to navigate safely.

CNN have also been used in RL applications for playing video games, such as Atari games, where the agent must make decisions based on the game's pixel data [17]. In these cases, CNN layers are used to extract relevant features from the game screen and make predictions about the best actions to take based on the current game state.

Overall, CNN is a powerful tool for processing visual input in RL applications and can make more informed observation according on their surroundings for RL agent. Recent advances in deep learning have made it possible to extract high-level features from raw sensor data, leading to breakthroughs in computer vision [15]. Environment gives output frames as 840x840 pixels in RGB. When examining typical CNN samples, it was seen that 84x84 gray inputs gave the best results for feature extraction. For this reason, 840x840 frames have been reduced to 84x84. Frame translation is given in figure 2.9.

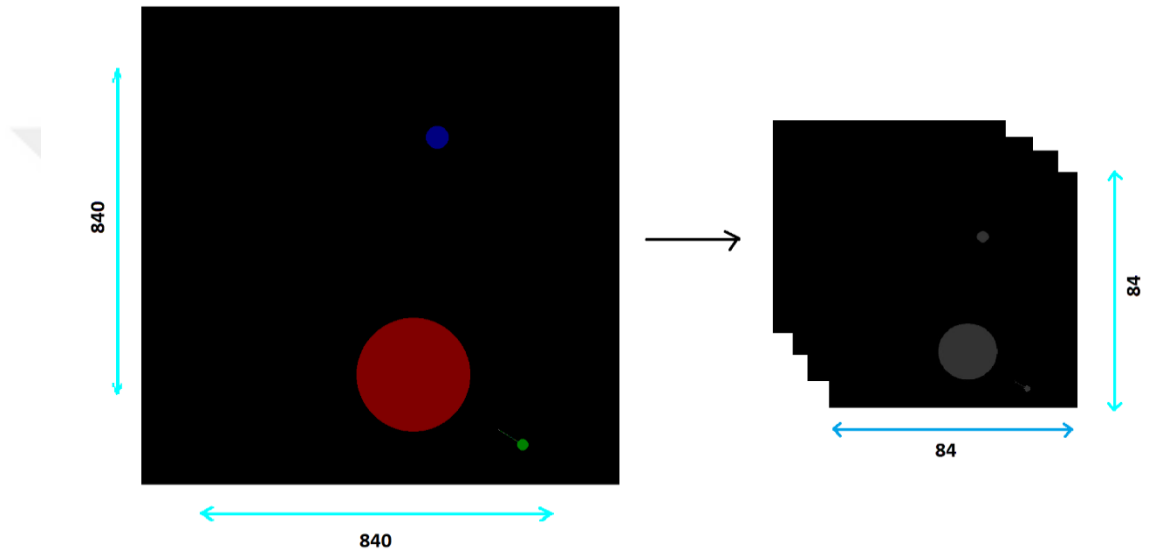


Figure 2.9 CNN input frames.

To capture the motion in addition to current positions, CNN needs more than just the current frame. Therefore, the last 4 frames are included as input to the CNN to capture the direction and speed of movements. .

The main difference of algorithm is the input layer. In the previous section, all distances were calculated and normalized as input layer and given to the neural network. In this method, the frames of the map are used as input. To extract information related to the state of the environment, 4 frames of the movement of the agent are considered after each action. The issue with using these frames as input is that their dimensions are 84 x 84, and they are represented as grayscale pixel values, resulting in a dimension of 1 x 84 x 84. As a result the issue is that the size of four consecutive frames is 28,224 pixels (4 x 1 x 84 x 84). To extract features from these

frames, the CNN architecture developed by DeepMind [17] was used, which consists of four layers.

The CNN model consists of several channels used to extract features. These features include the positions of remaining position and directions of the obstacle, position and direction of the agent and target position. At the end of the CNN layers, a feature vector of size 3136 is obtained. This vector takes statistical input place as DDQN input layer. All subsequent operations are the same as described in the statistical path planning section (Chapter 2.2).

In the algorithm, the initialization frame is created first. The created frame is multiplexed to four and the neural network is fed assuming that objects are stationary at the beginning. In each action taken, the state vector of $4 \times 84 \times 84$ is shifted. The old frame is removed from the array and the new frame is added to the array. Quad state frames are saved in memory. Since each frame contains $4 \times 84 \times 84$ pixel data, memory usage is much higher in the CNN model compared to the statistical model.

For CNN represented path planning two million episodes were trained. In the training, a system with 64gb ram NVidia Quadro 4000 graphics card intel i7, 24 core CPU was used. The total training time is 146 hours and 11 minutes. Successful results can be obtained in very wide scenarios without the need for retraining since training is executed at different locations and speeds. Unlike the statistical approach, the CNN approach is trained more slowly. This is because of Convolutional layers and the rendering time of frames in simulation.

For performance evaluation, ten thousand set scenarios were run at random speeds and locations. In 9.826 of these test scenarios, the vehicle was able to reach the target successfully. It has been verified that the method can understand the safe way with a success rate of 98.26%. The average reward obtained in the test scenarios was observed as 74.17. While this value was at the level of -100 when the first training started, it is seen that the amount of reward increased at the end of the training.

In figure 2.18 score episode graph is given. In this graph, the total awards obtained at the end of each episode during the training process are given. In order for the graph to be read more smoothly, the averages of the awards obtained at the end of the section were taken with a buffer of one thousand. As it can be seen from the graph, the system is trained fast during the first 350 thousand episodes and the level of success is increasing rapidly. After the learning process reached a sufficient level, the learning

rate was decreased to avoid overfitting or getting stuck in local optima. Two different scenarios were randomly created from the set of scenarios used in testing, and they are shown in detail in Figure 2.10. Colors from light to dark are used in the images. The most clearly specified images are in the last positions. Darker ones show past positions. In addition, white arrows indicate the direction of movement of the obstacle. The green circle represents the starting point and the blue circle the target point.

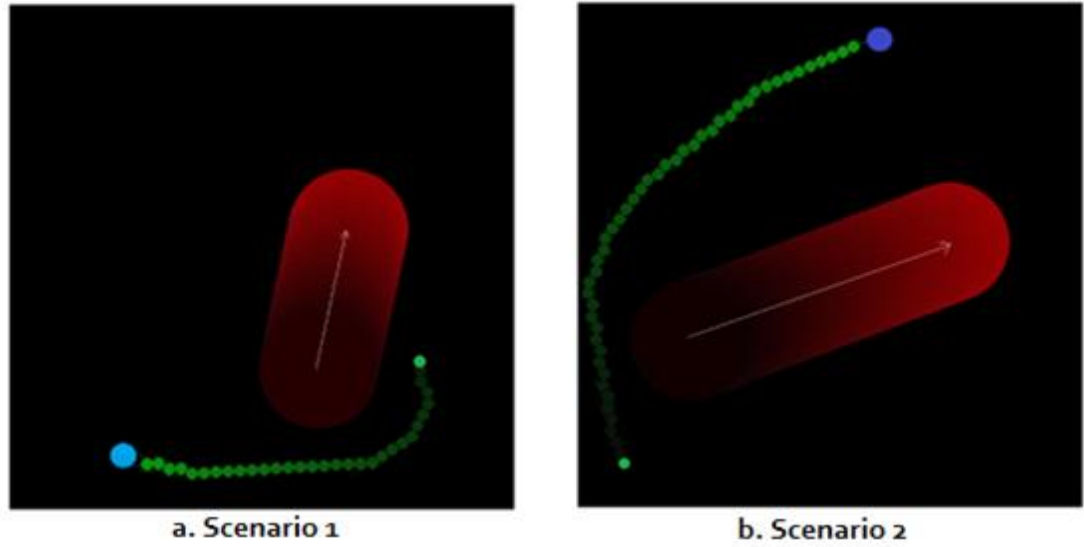


Figure 2.10 Final path CNN represented scenarios.

In scenario 1:

Init positions; vehicle $(7.56, 1.23, 0.87\pi)$, target $(1.25, 0.31)$, obstacle $(5.45, 1.21)$, obstacle velocity $(0.012, 0.062)$. In this scenario, the first heading of the vehicle is towards the target, but since the obstacle is very close to the vehicle, it is seen that the vehicle first maneuvers to avoid the obstacle and reach the target.

In scenario 2:

Init positions; vehicle $(0.38, 0.34, 1.88\pi)$, target $(5.64, 8.11)$, obstacle $(1.12, 2.19)$, obstacle velocity $(0.082, 0.019)$. In this scenario, the obstacle and the starting points are very close to each other in the initial state. Therefore, the vehicle reached the target by passing the obstacle in the opposite direction of the movement direction of the obstacle.

2.2.4 Statistical Approach and Path Planning

The model trained with the input obtained from CNN is explained in the previous section. In the statistical approach, the input layer is changed and given to the model as mathematical data.

The input layer represents the state space. 1x12 feature vector is used for input layer. Detailed input layer is $[x_v, y_v, a_v, a_{vt}, x_{vo}, y_{vo}, r_{vo}, v_{ox}, v_{oy}, x_{vt}, y_{vt}, r_{vt}]$. Current vehicle position $[x_v, y_v, a_v]$, x_v represents vehicle position in x-axis, y_v represents vehicle position in y-axis, a_v represents vehicle heading between 0 and 2π . a_{vt} represents real bearing between vehicle and target point. x_{vo} represents distance between obstacle and vehicle in x axis. y_{vo} represents distance between obstacle and vehicle in y axis, r_{vo} represents Euclid distance between obstacle and vehicle, v_{ox} represents obstacle speed in x axis. v_{oy} represents obstacle speed in y axis, v_{vt} represents distance between target and vehicle in x axis. y_{vt} represents distance between targets and vehicle in y axis and r_{vt} represents Euclid distance between target and vehicle.

Geometric visuals of input parameters are given in figure 2.11.

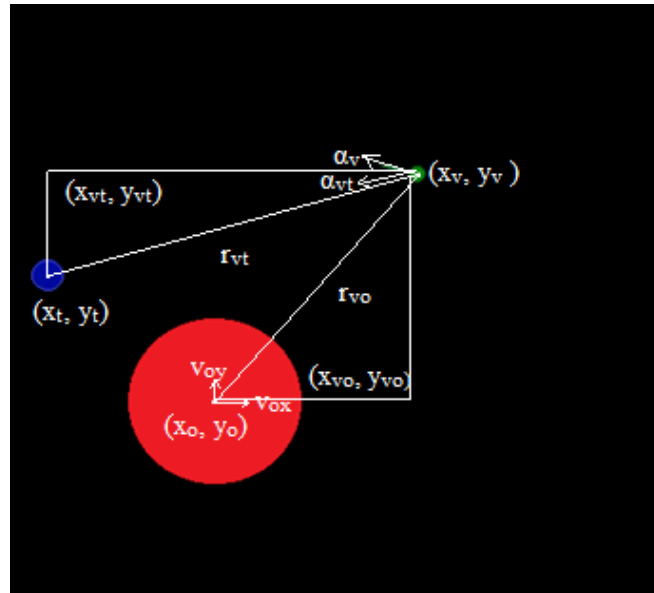


Figure 2.11: Dueling DDQN statistical input math.

Experience replay buffer capacity is defined 20.000. Training of the DQN algorithm using old experiences shortens the training time. The training starts with the agent

performing an action while following an ϵ -greedy policy and transitioning to a new state, s_{t+1} , before the reward, $r_{t+1}(s_t, s_{t+1})$, from the transition is calculated. Then, after the reward is calculated, checks if the new state is a terminal state or not. If s_{t+1} is a terminal state, terminal state flag sets True else false. The initialization value of ϵ is one. It means that the selection of action is made randomly. This behavior is called exploration. ϵ decay algorithm is used for exploitation. As the ϵ value approaches 0, exploitation begins to increase. The ϵ value is updated after every 100 iterations. ϵ is multiplied by ϵ decay. ϵ decay value was determined as 0.9998. The purpose of determining this value is to terminate exploration at the end of one million episodes. Iteration limit is defined one hundred. If the number of steps exceeds one hundred, the epoch is restarted. The update frequency is set to 50 epochs. At the end of every 50 epochs, a loss calculation is made, and backpropagation is performed. Mini batch is made in the training section. The mini-batch size was set to 32. Random samples are taken from the recorded experiences up to the number of mini batches. If the selected state is not terminating state, the expected Q-value is calculated using the next state Q-value.

In this study defined the loss of the model as the mean squared error. Loss is obtained by summing the squares of the difference between the expected Q-value and the current Q-value. Current Q-value is the Q-value obtained from the current neural network. The loss is obtained by the sum of the squares of the difference between the current q and the expected Q-values obtained from the target

$$L(\theta) = \frac{1}{N} \sum_{i \in N} (Q_{\theta}(s_i, a_i) - Q'_{\theta}(s_i, a_i))^2 \quad (2.14)$$

Back propagation operation is performed by using pytorch's backward method for the loss value obtained [18]. The parameter set of algorithms is given in Table 2.1

Learning Rate	0.0004
Mini Batch Size	32
Gamma	0.99
Experience Replay Buffer Size	20000
Initial Epsilon	1.0
Final Epsilon	0.1
Target Network Update Frequency	1000
Optimizer	Adam

Epsilon Decay	0.9998
Training Update Frequency	20

Table 2.1. Hyperparameters

In each iteration, the starting and ending points are determined randomly. The starting heading is determined by adding random noise between $+\pi/2$ and $-\pi/2$ in the direction of the target. Then the position of the obstacle is determined randomly so that it does not intersect with the target and end points. Fixed speed with random heading between 0-100m/s randomly determined for obstacle. If the obstacle intersects with the wall during the simulation, it bounces off the wall and the velocity vector in the direction it hits is inverted. Action is taken from DDQN at the starting position. According to the action taken, the location update is made on the map. The next state, reward, and terminate state flag are returned from the environment and stored in the experience replay buffer. During this process, collision and terminate state checks are made. A terminate state means that the vehicle has successfully reached its target. If a collision occurs, the target is reached, or the iteration limit is reached, the episode is ended.

Before moving on to the new episode, a check is performed to see if it is time to update the neural network. In this study, the update frequency was set to 4, meaning that the loss is calculated and back-propagation is performed at the end of every 4 episodes before starting a new one.

An essential component of this work is the reward system. At each time step, a penalty of -1 is applied. The status of the newly obtained state is checked, and if a collision occurs, the reward is set to -100. If the vehicle is within a distance of 300 meters from the obstacle, the reward is set to -10 to signal that it is in the danger zone. When the vehicle reaches the target, the reward is set to 100. The -10 reward signal helps the vehicle stay at a safe distance from the obstacle. Pseudocode of reward function is given in Algorithm 2.2.

```

Initialize  $Q(s, a)$  arbitrarily
Repeat (for each Episode)
    Initialize  $s$ 
    Repeat (for each Iteration)
        Choose  $a$  from  $S$  using  $\epsilon - Greedy$ 
        Take action  $a$ , observe  $r, s'$ 
         $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_i Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$ 
         $s \leftarrow s'$ 
        If  $s'$  is terminal, update R to 100
        If  $s'$  closer to obstacle than 300 meters, update R to 100
        If  $s'$  intersects walls or obstacle, update R to -100
    Until  $s$  is terminal

```

Algorithm 2.2 Pseudocode of reward function.

The rewards obtained are used in the loss account and network update. It is expected that the reward will increase as the exploitation will increase as the ϵ decreases.

At the end of each episode, the need to update the target network is checked. The update frequency of the target network is determined as one thousand in this study. After every 1.000 episodes, the target network is updated with the current network. This process continues until the ϵ value reaches 0.1. Based on the ϵ calculation made in this study, the training continues for approximately 2.000.000 episodes.

For statistical approach path planning 2.000.000 episodes were trained. In the training, a system with 64gb ram NVidia Quadro 4000 graphics card intel i7, 24 core CPU was used. The total training time is 62 hours and 34 minutes. Successful results can be obtained in very wide scenarios without the need for retraining in the test part, since all training is done at different locations and speeds.

For performance evaluation, ten thousand set scenarios were run at random speeds and locations. In 9972 of these test scenarios, the vehicle was able to reach the target successfully. In the analyzes made, it has been determined that 28 faulty scenarios

occur in cases where the starting points are very close to the obstacle. It has been verified that the method can understand the safe way with a success rate of 99.72%. The average reward obtained in the test scenarios was observed as 76.48. While this value was at the level of -100 when the first training started, it is seen that the amount of reward increased at the end of the training.

The total rewards obtained at the end of each episode during the training process are given. In order for the graph to be read more smoothly, the averages of the awards obtained at the end of the section were taken with a buffer of 1000. As it can be seen from the graph, the system is trained fast during the first 250 thousand episodes and the level of success is increasing rapidly.

Four different scenarios randomly from the scenarios used in the test are given in detail in figure 2.12. Colors from light to dark are used in the images. The most clearly specified images are in the last positions. Darker ones show past positions. In addition, white arrows indicate the direction of movement of the obstacle. The green circle represents the starting point and the blue circle the target point.

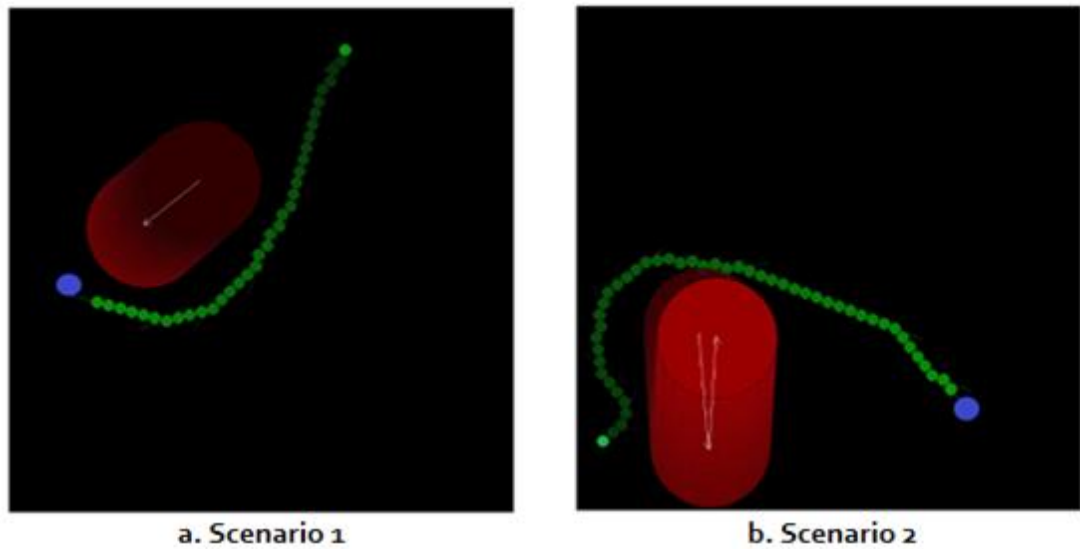


Figure 2.12 Random taken two test scenarios.

In scenario 1:

Init positions; vehicle (7.6, 7.9 , 1.2π), target (0.52 , 4.12), obstacle (2.12 , 6.31), obstacle velocity (-0.022, -0.018). In this scenario, the obstacle is moving towards the target at a constant speed. It is seen in the visual that the vehicle evaluates the obstacle

that enters between the target and the vehicle, and the vehicle reaches the target by not getting closer to the obstacle than the danger area.

In scenario 2:

Init positions; vehicle (0.21, 0.83, 0.22π), target (3.65, 0.92), obstacle (1.07, 1.63), obstacle velocity (0.002, -0.048). In this scenario, the obstacle is pulling from the wall. It is seen in the image that the vehicle reaches the target without colliding with the obstacle.

2.2.5 Extended Kalman Filter

The studies in the previous chapters were made by assuming that the environment and the sensors that detect obstacles are perfect. However, in the real world this is not possible. In this part of the study, to create a simulation environment closer to the real world, the problem is approached by applying the Kalman filter, which can make position vector estimation used in object detections of the sensors. The position of the obstacle is determined using the object matrix obtained from the Kalman filter. It will be applied to both methods, which are statistical, and CNN based input, which have been examined in the previous sections using the DDQN method.

The main purpose of the Kalman filter is to estimate the uncertainty. The Kalman filter has many applications, including object tracking in object detection systems. Hong-Son was proposed the system, which is, depends on pedestrian detection method [19].

In object detection, the Kalman filter can be used to predict the location of an object between frames in a video stream. The filter is typically used in conjunction with a detect-and-track algorithm, which performs object detection in the current frame and then uses the Kalman filter to predict the object's location in the next frame.

The Kalman filter works by using a state-space model to represent the system being tracked. The state-space model consists of a set of variables that represent the system's state (e.g. the position and velocity of an object), and a set of equations that describe how the system's state changes over time. The Kalman filter uses this model to make predictions about the system's state at future times, and then uses measurements of the system to correct these predictions.

The extended Kalman filter (EKF) is an extension of the Kalman filter that is designed to handle systems that are nonlinear. It works by linearizing the system at each time step and using the linearized version to make predictions. The main difference between the Kalman filter and the extended Kalman filter is that the Kalman filter can only be used for systems that are linear, while the extended Kalman filter can be used for systems that are nonlinear.

In object detection, the Kalman filter can be used to smooth out noisy or uncertain object detections, and to predict the object's location in the next frame even if it is not detected in the current frame. This can help to improve the overall accuracy and reliability of the object detection system.

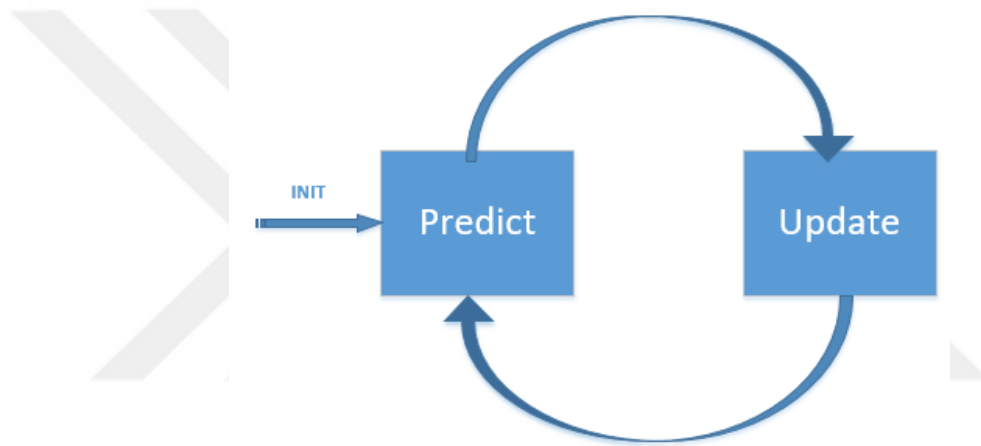


Figure 2.13 Extended Kalman filter sequence.

The extended Kalman filter (EKF) is for estimate the status of a system when the measurements are made in different coordinate systems, or when the system is nonlinear. The EKF works by linearizing the system dynamics and measurement models at each time step, and then using the Kalman filter equations to estimate the state of the system.

The EKF algorithm consists of two steps: prediction and update. In the prediction step, the EKF estimates the state of the system at the current time step based on the state at the previous time step and the system dynamics. In the update step, the EKF corrects the state estimate based on the measurement at the current time step.

The EKF algorithm can be summarized as follows:

- Initialize the state estimate and the covariance matrix.

- For each time step:

Predict the state estimate and the covariance matrix based on the system dynamics.

Linearize the system dynamics and measurement models at the predicted state

Correct the state estimate and the covariance matrix based on the measurement

The EKF states and equations are as follows:

Prediction step:

State estimate: $x' = f(x)$

Covariance matrix: $P' = F * P * F^T + Q$

Update step:

Kalman gain: $K = P' * H^T * (H * P' * H^T + R)^{-1}$

State estimate: $x = x' + K * (z - h(x'))$

Covariance matrix: $P = (I - K * H) * P'$

Where x is the state vector, P is the covariance matrix, $f(x)$ is the system dynamics function, $h(x)$ is the measurement model function, Q covariance matrix, z is the measurement vector, R is the measurement noise covariance matrix, and I is the identity matrix. The matrices F and H are the Jacobians of the system dynamics and measurement models, respectively.

2.2.6 CNN input Path Planning with EKF

In this part of the study, the ground truth with CNN input and the path planning algorithm, which was performed in the previous sections, were re-performed using the Kalman filter outputs. Feature extraction was performed using the same network as the Convolution Neural Network in section 2.2. However, in this method, images with the outputs of the Kalman filter are used instead of the ground truth parameters of the obstacle. There is no ground truth obstacle position in the images. An example image is given in Figure 2.14.

In cases where the Kalman filter gives results outside the map, the map is drawn without obstacles. In this case, the agent acts as if it is acting in without obstacle scenario.

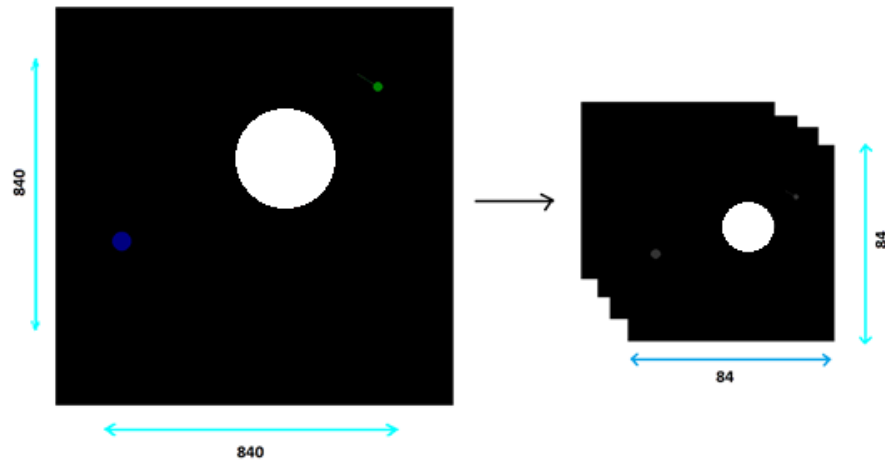


Figure 2.14 CNN represented EKF input frames.

One million episodes were trained. In the training, a system with 64gb ram NVidia Quadro 4000 graphics card intel i7, 24 core CPU was used. The total training time is 158 hours and 05 minutes. Successful results can be obtained in very wide scenarios without the need for retraining in the test part since all training is done at different locations and speeds like previous test scenarios.

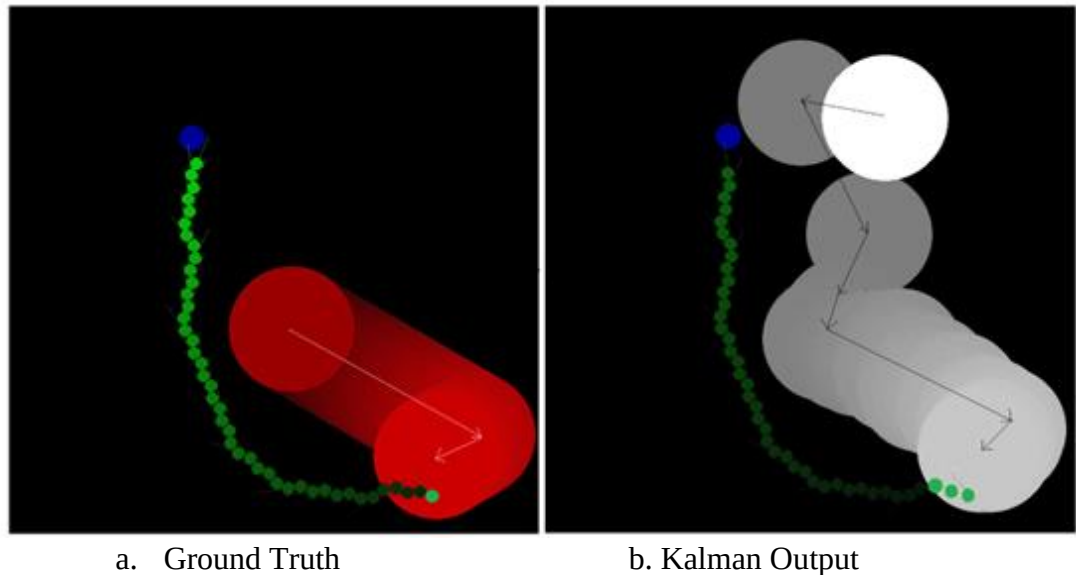


Figure 2.15 CNN represented path planning with Kalman filter result.

The most clearly specified images are in the last positions. Darker ones show past positions. In addition, white arrows indicate the direction of movement of the obstacle. For the planned path shown in figure 2.15, the image seen in section a in the left picture

is the ground truth of the obstacle. The image seen in part b in the right picture shows the output of the filter during the remaining motion. As seen in the image of the Kalman output, the first position is irrelevant to the actual position. But after 3-4 steps, the Kalman output converged to the real position very much. This is because the uncertainty field of the Kalman filter is reduced by measurement.

For performance evaluation, ten thousand set scenarios were run at random speeds and locations. In 9728 of these test scenarios, the vehicle was able to reach the target successfully. It has been verified that the method can understand the safe way with a success rate of 97.28%. The average reward obtained in the test scenarios was observed as 71.47. While this value was at the level of -100 when the first training started, it is seen that the amount of reward increased at the end of the training.

In figure 3.1 Score episode graph is given. In this graph, the total awards obtained at the end of each episode during the training process are given. In order for the graph to be read more smoothly, the averages of the awards obtained at the end of the section were taken with a buffer of one thousand. As it can be seen from the graph, the system is trained fast during the first 300 thousands episodes and the level of success is increasing rapidly. After the amount of learning reached an enough level, the learning rate slowed down.

2.2.7 Statistical Approach Path Planning with EKF

In the path planning study conducted with the statistical method in the previous section, the ground truth of the obstacle was used as input parameters. In this part of the study, an input vector is created using the output of the Kalman filter instead of ground truth. Pseudocode of path planning using extended Kalman filter is given in figure 2.3. In the algorithm after EKF is initialized, the input vector is created using predict and update sequence.

The initialization values of some parameters used in EKF equations are given below. In order to indicate that the uncertainty is high, the initialization values in the P_{kk} matrix are set high. d_t represents the time variable. In this study, dt was determined as 1. As X_{kk} initialization, it is set to position (10, 10) and max speed (1, 1).

$$P_{kk(diag)} = \begin{bmatrix} 7056 \\ 7056 \\ 100 \\ 100 \end{bmatrix} \quad X_{kk} = \begin{bmatrix} 10 \\ 10 \\ 1 \\ 1 \end{bmatrix} \quad F = \begin{bmatrix} 1 & 0 & 1(\Delta t) & 0 \\ 0 & 1 & 0 & 1(\Delta t) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```

Initialize  $Q(s, a)$  arbitrarily
Repeat (for each Episode)
  Initialize  $s$ 
  Repeat (for each Iteration)
    Predict State
    Get Measurement
    Update  $s$ 
    Update EKF
    Choose  $a$  from  $s$  using  $\epsilon - Greedy$ 
    Take action  $a$ , observe  $r, s'$ 
     $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_i Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$ 
     $s \leftarrow s'$ 
    If  $s'$  is terminal, update R to 100
    If  $s'$  closer to obstacle than 300 meters, update R to 100
    If  $s'$  intersects walls or obstacle, update R to -100
  Until  $S$  is terminal

```

Figure 2.3 Pseudocode path planning using extended Kalman filter.

When observed that the covariance matrix of the stay with the EKF predict update rank became smaller and the P_{kk} matrix became smaller. The diagonal elements of the P_{kk} matrix of EKF provide information about the error of the x, y, Vx and Vy parameters of the obstacle. The fact that these values are low indicates that the accuracy of the relevant variable is high.

$$M = \begin{bmatrix} 7056 \\ 7056 \\ 100 \\ 100 \end{bmatrix} \quad M.T = M^T$$

$$\text{Total estimated Accuracy} = M * M.T$$

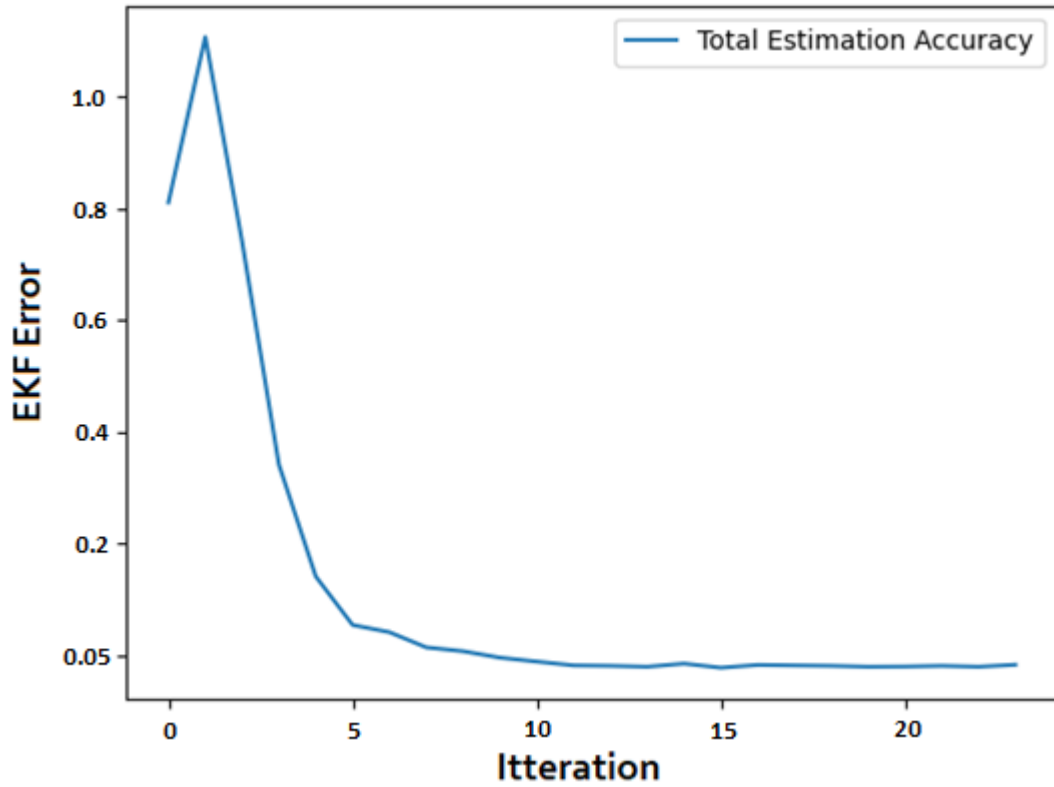


Figure 2.16 Graph of P_{kk} diagonals.

For clarity, the visual figure showing the output of the step-by-step covariance matrix diagonals transpose multiplication of the acquisition is given in 2.16.

The performance of EKF was checked by checking the Total estimated Accuracy. In the last part of this section, increasing the accuracy by adding it to the input of the accuracy model is also tested.

The input layer represents the state space. 1x12 feature vector is used for input layer like statistical DDQN section but only one difference. The obstacle's position and velocity are updated as the output of the Kalman filter. Since the test area is between 8.4km x 8.4km, in case the position and speed of the obstacle obtained at the exit of the EKF are out of range, it has been clipped for the max and min values.

One million episodes were trained. In the training, a system with 64gb ram NVidia Quadro 4000 graphics card intel i7, 24 core CPU was used. The total training time is 73 hours and 11 minutes. Successful results can be obtained in very wide scenarios without the need for retraining in the test part since all training is done at different locations and speeds like previous test scenarios.

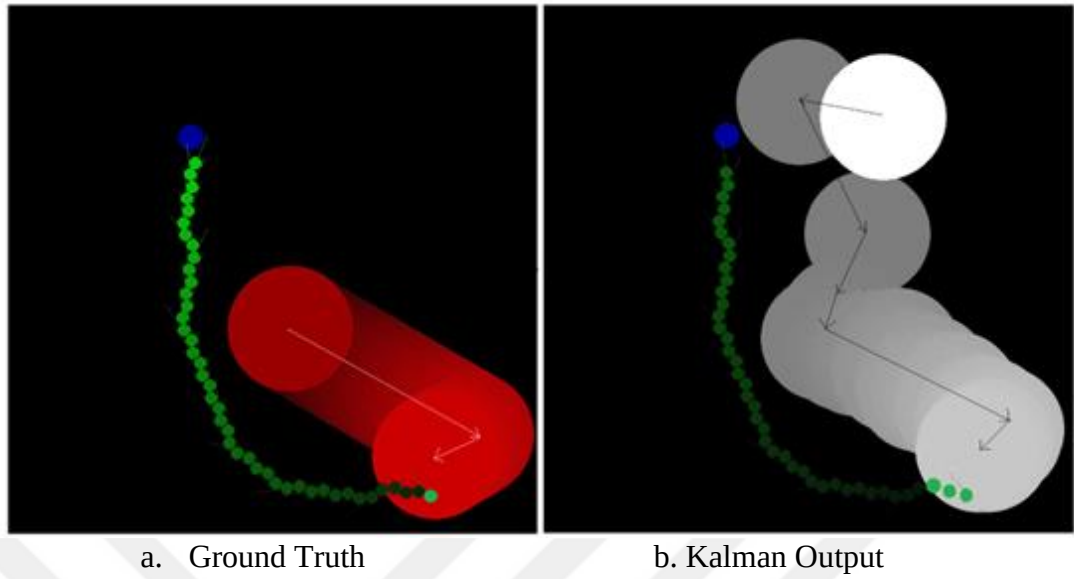


Figure 2.17 Statistical represented path planning with Kalman filter result.

The most clearly defined images show the last positions, with darker ones representing past positions. Additionally, white arrows indicate the direction of the obstacle's movement. Regarding the planned path shown in Figure 2.17, the image seen in section a in the left picture represents the ground truth of the obstacle, while the image seen in part b in the right picture shows the output of the filter during the remaining motion. As seen in the Kalman output image, the first position is irrelevant to the actual position. However, after 3-4 steps, the Kalman output converges to the real position significantly. This is because the uncertainty field of the Kalman filter is reduced by measurement.

For performance evaluation, 10000 set scenarios were run at random speeds and locations. In 9912 of these test scenarios, the vehicle was able to reach the target successfully. It has been verified that the method can understand the safe way with a success rate of 99.12%. The average reward obtained in the test scenarios was observed as 75.27. While this value was at the level of -100 when the first training started, it is seen that the amount of reward increased at the end of the training.

In figure 3.5 Score episode graph is given. In this graph, the total awards obtained at the end of each episode during the training process are given. For the graph to be read more smoothly, the averages of the awards obtained at the end of the section were taken with a buffer of 1000. As it can be seen from the graph, the system is trained fast

during the first 500000 episodes and the level of success is increasing rapidly. After the amount of learning reached an enough level, the learning rate slowed down.

2.2.8 Statistical Approach Path Planning with EKF Using Uncertainty

Parameter

In the previous sections, the statistical method was applied to the path-planning problem by using the ground truth and EKF outputs as inputs. As expected, more successful results were obtained when the inputs prepared using ground truth were compared with those prepared with EKF outputs. Since the uncertainty of EKF outputs is high in the first 4-5 steps, the error rate between ground truth and ground truth is high. It's possible that waiting for the uncertainty to decrease and sampling the environment may be a solution to solve the problem. However, in this solution, it is necessary to add an external control other than the model. In this method, we examined whether it could be a method to overcome this problem by adding uncertainty to the model. In this way, it was desired to eliminate the need for extra control.

Total estimated Accuracy obtained from P_{kk} given in 3.2.1 header has been added to statistical input parameter set. The obtained value was scaled between 0 and 1, normalized and added to the model set. In this case, the parameter set from 1x12 has increased to 1x13. Detailed input layer is $[x_v, y_v, a_v, a_{vt}, x_{vo}, y_{vo}, r_{vo}, v_{ox}, v_{oy}, x_{vt}, y_{vt}, r_{vt}, P_{km}]$. No changes were made to the model hyperparameters. P_{km} represents uncertainty parameter.

One million episodes were trained. In the training, a system with 64gb ram NVidia Quadro 4000 graphics card intel i7, 24 core CPU was used. The total training time is 32 hours and 34 minutes. Successful results can be obtained in very wide scenarios without the need for retraining in the test part, since all training is done at different locations and speeds like previous test scenarios.

For performance evaluation, ten thousand set scenarios were run at random speeds and locations. In 9932 of these test scenarios, the vehicle was able to reach the target successfully. It has been verified that the method can understand the safe way with a success rate of 99.32%. The average reward obtained in the test scenarios was observed as 75.47. While this value was at the level of -100 when the first training started, it is seen that the amount of reward increased at the end of the training.

In figure 3.9 Score episode graph is given. In this graph, the total awards obtained at the end of each episode during the training process are given. In order for the graph to be read more smoothly, the averages of the awards obtained at the end of the section were taken with a buffer of 1000. As it can be seen from the graph, the system is trained fast during the first 150000 episodes and the level of success is increasing rapidly. After the amount of learning reached an enough level, the learning rate slowed down.



Chapter 3

3.1 Results

3.2 Conclusion



3.1 Results

Using the DDDQN method, 5 different models were trained for their dynamic environments using 2 different input feature extraction methods and 2 different obstacle location detection methods. Feature extraction methods created the state parameters of the models.

The first model was trained using the geometric relative positions of the obstacle, vehicle, and target. Ground truth position of obstacle was used for first method. This model is named Path Planning statistical using Ground truth.

In the second model, the state parameters of the environment are defines using the CNN method and the ground truth of the obstacle. This model is named Path Planning CNN using Ground truth.

In the third model, the environment state parameters are used as in the first model. However, the position and velocity vectors of the obstacle were obtained from the EKF outputs. This model is named Statistical using EKF without uncertainty.

In the fourth model, the environment state parameters are used as in the second model. However, the position and velocity vectors of the obstacle were obtained from the EKF outputs. This model is named Path Planning CNN using EKF.

In the fifth model, the environment state parameters are used as in the third model. A new parameter has been added. This parameter is the diagonal error parameter of P_{kk} , which shows the uncertainty status of EKF. This model is named Path Planning Statistical using EKF with uncertainty.

The all-trained models were evaluated with 3 different metrics: score curve, training time and success in the test set. Score curve is given in figure 4.1. When the curve is evaluated, it is seen that the number of episodes required for the training of statistical state models is less than that of CNN state models. As expected, the most successful training was obtained with the training using ground truth. When the results of the statistical input model and the CNN input model are compared, it is seen that the statistical model is more successful than the CNN model. However, when it is desired to change the number of obstacles in the environment, new obstacle vectors should be added to the statistical model as much as the number of obstacles. In the

model with CNN, training can be made for environments that are more complex by keeping the different input state vector constant, only by changing the frames. In order to bring the models closer to the real environment, EKF models were prepared from the target position velocity determination filters used in autonomous vehicles. The performance of these models was lower than the ground truth models. In order to improve the results of the model obtained by using EKF, it is seen that the trained model, which will be given the uncertainty parameter of EKF, provides an improvement when evaluated on the score curve. When the models trained with CNN are compared, the EKF outputs of the model with ground truth complete the learning faster than the model given. The model with the lowest score was the model trained with CNN and EKF outputs.

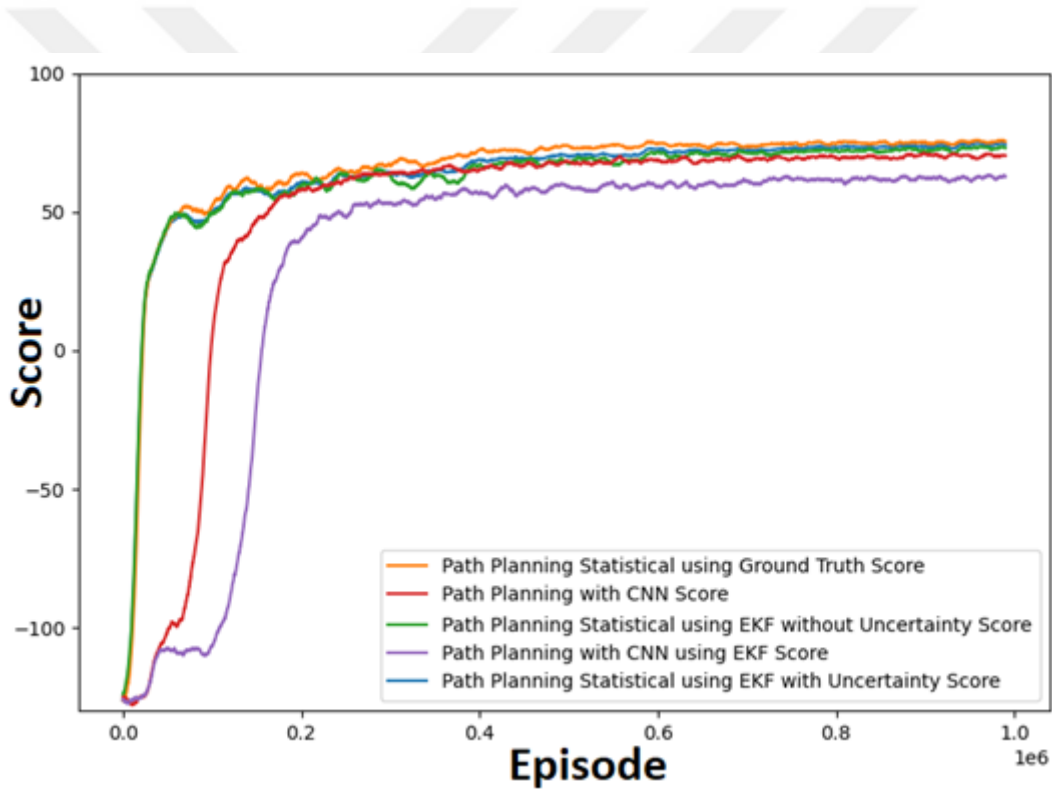


Figure 3.1 Score curves of five models.

The training results of the models are given in Table 4.1. When the mathematical results are compared, it is seen that the training period of the model trained with CNN takes longer. This time difference occurs as the CNN layers enlarges the model. Due to the EKF calculation time cost, the duration of the trainings using EKF is getting longer.

Method	Training Time	Average Score	Success Rates	Train Episodes
Path Planning statistical using Ground truth	62h 34min	76.48	%99.72	1000000
Path Planning CNN using Ground truth	146h 11min	73.12	%98.26	1000000
Statistical using EKF without uncertainty	73h 43min	75.27	%99.12	1000000
Path Planning CNN using EKF	158h 05min	71.47	%97.28	1000000
Path Planning Statistical using EKF with uncertainty	75h 23min	75.47	%99.42	1000000

Table 3.1 Training statistics.

The average test scores are compared, it is seen that the model prepared with the statistical approach achieves more successful results. When all the results are evaluated, it is seen that the solution of the path plan prepared with DDDQN can be used as an alternative method.

3.2 Conclusion

RL has proven to be a powerful tool in solving path-planning problems in various applications, including autonomous vehicles, mobile robots, and UAVs. The ability to learn optimal policies from experience without the need for hand-crafted rules or maps makes it particularly attractive in complex, dynamic, or unknown environments.

In this study, we analyzed traditional RL algorithms and demonstrated that path planning can be achieved using different RL methods. They showed that statistical feature extraction can lead to faster and more successful results compared to using CNN for feature extraction.

The DDDQN algorithm is a solution for path planning, especially in dynamic environments where quickly identifying an optimal direction is crucial. However, a significant drawback of this approach is its discrete action space. To enhance this study, future work can explore path planning models for multiple obstacles, multiple agents, continuous environments, and continuous action spaces using policy-based RL methods such as DDPG.

In conclusion, the future of path planning with RL is promising, with many exciting opportunities for research and development in the years to come.

REFERENCES

- [1] Aisha Muhammad, Mohammed A.H Ali, Ibrahim Haruna Shanono (2022). "Simulation Performance Comparison of A*, GLS, RRT and PRM Path Planning Algorithms". 2th Symposium on Computer Applications & Industrial Electronics (ISCAIE)
- [2] Ahmed M. Walied, Ahmed Onsy, , Shady A. Maged (2021). "Path Planning in a dynamic indoor environment for mobile robots using Q-Learning Technique". 2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)
- [3] Prashant Bhopale 1 , Faruk Kazi1 , Navdeep Singh "Reinforcement Learning Based Obstacle Avoidance for Autonomous Underwater Vehicle". 2019Journal of Marine Science and Application
- [4] Zichen He, Lu Dong "Asynchronous Multithreading Reinforcement-Learning-Based Path Planning and Tracking for Unmanned Underwater Vehicle". Ieee Transaction on systems, man and cybernetics: systems
- [5] Kaelbling, Leslie P.; Littman, Michael L.; Moore, Andrew W. (1996). "Reinforcement Learning: A Survey". Journal of Artificial Intelligence Research. 4: 237–285. arXiv:cs/9605103. doi:10.1613/jair.301. S2CID 1708582.
- [6] Woohyeon Moon, Bumgeun Park, Sarvar Hussain Nengroo 2021. "Path Planning of Cleaning Robot with Reinforcement Learning", 2022 IEEE International Symposium on Robotic and Sensors Environments (ROSE)
- [7] Richard S. Sutton and Andrew G. Barto, (2014), Reinforcement Learning: An Introduction, 2th Edition.
- [8] J. Kim and J. P. Hespanha, "Discrete approximations to continuous shortest-path: Applications to minimum-risk path planning for groups of UAV's" in 42nd IEEE Conference on Decision and Control, Dec. 2003.
- [9] R. J. Urick, Principles of Underwater Sound, 2nd ed. McGraw-Hill,1983
- [10] Emre TAŞCIOĞLU, Ahmet Güneş , "Path-planning with minimum probability of detection for AUVs using reinforcement learning" in in IEEE Innovations in Intelligent Systems and Applications Conferences. 2022.
- [11] L. J. Ziomek, Fundamentals of Acoustic Field Theory and Space-Time Signal Processing. CRC Press, 1995.
- [12] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare and Joelle Pineau (2018), "An Introduction to Deep Reinforcement Learning", Foundations and Trends in Machine Learning: Vol. 11, No. 3-4. DOI: 10.1561/22000000071.

- [13] H. V. Hasselt, A. Guez, D. Deep Reinforcement Learning with Double Q-learning . arXiv:1509.06461v3, 2015.
- [14] Z. Wang, T. Schaul , Dueling Network Architectures for Deep Reinforcement Learning. arXiv:1511.06581v3, 2016.
- [15] Manjunath Jogin, Mohana Mohana , Feature Extraction using Convolution Neural Networks (CNN) and Deep Learning. 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)
- [16] W. Weng, H. Gupta, N. He, L. Ying, R. Srikant, The Mean-Squared Error of Double Q-Learning. arXiv:2007.05034v3, 2020.
- [17] V. Minh, K. Kavukuoglu, D. Silver, A. Graves, I. Antonoglou, Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602v1, 2013.
- [18] V. Minh, K. Kavukuoglu, D. Silver, A. Graves, I. Antonoglou, Human-level control through deep reinforcement learning . doi:10.1038/nature114236, 2015.
- [19] Hong-Son Vu, Jia-Xian Guo, Kuan-Hung Chen, Shu-Jui Hsieh, and De-Sheng Chen.“A Real-Time Moving Objects Detection and Classification Approach for Static Cameras”,978-1-5090-2073- 7/16/\$31.00 ©2016 IEEE
- [20] Dong Yuansheng, Zou Xingjie 2020 “Mobile Robot Path Planning Based on Improved DDPG Reinforcement Learning Algorithm” Software Engineering and Service Science (ICSESS), 2020 IEEE 11th International Conference on. :52-56 Oct 2020