Number_____

Nanjing University of Aeronautics and Astronautics

Graduation Thesis

Unmanned Mapless Navigation based on Deep Reinforcement Learning

Student Name	Asmita Tiwari (提娃)
ID Number	191664128
College	College of International Education
Major	Software Engineering and Management
Class	1916641
Advisor	Associate Prof. Li Bo Han

June 2020

Nanjing University of Aeronautics and Astronautics

Graduation Thesis Letter of Commitment

Private solemn statement: the whole content of this Graduation Thesis (Title：Unmanned Mapless Navigation based on Deep Reinforcement Learning) is made and the whole results are experimented personally under the guidance of my faculty advisor. Except the deliberately added remarkable contents, this graduation thesis is completed with my own knowledge and doesn't involve other thesis written by any other person or group.

Personal's signature：　　Date:

(Student ID Number): 191664128

Unmanned Mapless Navigation based on Deep Reinforcement Learning

Abstract

Dynamic path planning for unknown environments has always been a challenge for mobile robots. Map- based navigation technology requires complex programming, a large number of data sets, and cannot adapt to changes in the environment. Thus, introducing mapless navigation can achieve the goal of robots or mobile agents with the best collision-free results. This navigation model takes cameras and sensors as input and commands the mobile robot thus simplifying the process of perceiving the environment and taking action decisions using Deep Reinforcement Learning (DRL) techniques. The main objective of this thesis is to compare and analyze different reinforcement learning techniques and deep neural networks to understand how to use it for vision-based autonomous navigation. This is done by examining similar projects that solves the problems associated with the use of DRL. The main technique considered in DRL is Q-learning. This is achieved by implementing a prototype with an agent which automatically navigates in a real environment. In addition, this research will focus on current research in several areas related to DRL algorithms such as Markov decision process (MDP), Advantage Actor-Critic (A2C), Deep Q Networks (DQNs), Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO) and Convolutional Neural network (CNN). To analyze the functionally and robustness of these algorithm a simulated experiment is conducted in Gazebo using PyTorch. Furthermore, the parameters such as Environment Variation, Sensors, Trajectory Planning, Reward Function, equally proves the effectiveness of simulation and experiment. Hence, training an agent with consideration of all these aspects can safely reach navigation targets in a dynamic environment along with less interference of human and homogenous programming.

Keywords: Deep Reinforcement Learning, Mapless Navigation, Reward Function, Sensor, Trajectory Planning

基于深度强化学习的无人无地图导航

摘要

对于未知环境，动态路径规划一直是移动机器人面临的挑战。基于地图的导航技术需要复杂的编程，大量的数据集，并且无法适应环境的变化。无地图导航可以实现机器人或移动代理的最佳无冲突结果。该导航模型将摄像机和传感器作为输入并命令移动机器人，从而使用“深度强化学习（DRL）”技术简化了感知环境并做出动作决策的过程。本文的主要目的是比较和分析不同的强化学习技术和深度神经网络，以了解如何将其用于基于视觉的自主导航。这是通过研究解决与 DRL 使用相关问题的类似项目来完成的。 DRL 中考虑的主要技术是 Q 学习。这是通过使用可在真实环境中自动导航的代理实现原型实现的。此外，本研究将集中于与 DRL 算法相关的多个领域的当前研究，例如 Markov 决策过程（MDP），优势参与者关键（A2C），深层 Q 网络（DQN），深层确定性策略梯度（DDPG），近端策略优化（PPO）和卷积神经网络（CNN）。为了分析这些算法的功能和鲁棒性，使用 PyTorch 在凉亭中进行了模拟实验。此外，环境变化，传感器，弹道规划，奖励功能等参数同样证明了仿真和实验的有效性。因此，在考虑所有这些方面的情况下训练代理可以在动态环境中安全地到达导航目标，同时减少对人类和同质程序的干扰。

关键词： 深度强化学习，无地图导航，奖励功能，传感器，轨迹规划

Table of Contents

List of Figures and Tables

Chapter 1 Introduction

1.1 Research Background

Navigation can be defined as the process of determining an appropriate and suitable safe path between the two points i.e. the initial starting point and the final destination point. Mobile robots that operate in real-world environments need to be able to safely navigate their surroundings. However, on inner surroundings or outside urban areas, due to privacy and large amount of data, it is very complex and difficult to build, store, and transmit maps. Nonetheless, maintaining a reliable map is expensive and may not be practical due to the rapid changes in the environment. There is another restriction for such delivery bots that need to be operated in a dynamic environment full of moving objects.

With advances in technology, people began to prefer machines to human labor in order to increase productivity. In the beginning, machines were only used to automate work that did not require intelligence. However, the invention of computers led people to consider machine learning (ML). Just as humans can learn how to navigate a city without relying on maps, GPS location, or other aids, it is our aim to show that a neural network agent can learn to traverse entire cities by using only visual observations. In recent years, most of the robots have already had prior knowledge of the structure of the path, road or the environment that it runs through. They were well programmed with the enormous data sets of images, sounds, 2D and 3D structure.

Today, Artificial Intelligence (AI) is still being a subject of research and continuous experimentation in order to provide machines with learning capabilities. It is vital to understand the nature of learning in order to achieve the target of smart intelligent machines. Although there are large number of algorithms that were developed as supervised and unsupervised learning methods in the ML field, the core idea behind Reinforcement Learning (RL) usage is that of learning from interaction. To maintain the ability to interact, various sensors are mounted on machines, including infrared, sonar, inductive, and diffuse sensors. The best known of all RL algorithms is the Q-learning, proposed in 1989, which is based upon the learning from delayed rewards and punishments [1]. Since then, many researchers have used Q-Learning in a very efficient way for navigation of mobile robots and obstacle avoidance. Q-Learning has also been tested to solve trajectory planning problems for mobile robots in 3D environments. Many authors have also proposed hybrid approaches by combining the un-supervised RL with Artificial Neural Network (ANN).

Recently, Deep Learning (DL) based mapless navigation approaches have gained attention, which directly links mapping sensor perception to steering commands. In contrast to rule-based approaches, learning-based navigation methods are optimized on large amount of training datasets and therefore no longer require the manual adjustment of hyper-parameters and rules. Robots learn the best policies by imitating demonstrations gathered from human experts. Reinforcement learning has been widely used in various research areas. Since it is not possible to sample all problem possibilities in steady state and due to the absence of an explicit teacher, RL algorithms are preferable for supervised learning in the machine learning area, as the optimal control problem has become a popular subject of research. In reinforcement learning, the optimal policies are derived from the training data generated during the interaction between robots and the environment.

In this thesis, we focus on the reinforcement learning, to address the issue of safety and efficient navigation in crowds. In our approach, is modeled by a Deep Neural Network (DNN) that transmits raw data from sensors to a collision-free steering control. It has been shown that in some areas it is possible to learn directly from visual input using Deep Reinforcement learning (DRL) approaches that can learn from task rewards - for example, navigation to a destination. RL techniques have begun to be preferred as learning algorithms because they are more practical and applicable than other techniques that require prior knowledge. The RL approach has been used for many purposes, for instance, selecting algorithm characteristics classification, the optimal path finding, routing for networks, and coordinating communications in multi-agent systems.

1.2 Main Task

The bench marking Q-Learning algorithm has been tested for many scenarios and has proven to be effective because it can handle unknown non-deterministic Markovian systems. The usage of Neural Networks in DL has resulted in an end-to-end framework that allows great advances in human-level agents and autonomous systems such as mapless navigation.

In this thesis, we will address the theme of Autonomous Navigation and its application in real-world. A timeline for the development of RL and DL methods follows, describing the main improvements made in these two fields. Then, we will dive into DRL and have an overview of the state-of the-art of this new and promising field, by reviewing through a set of algorithms (Value optimization, Policy optimization and Actor-Critic).

The main part of the thesis will be the evaluation and analysis of different methodologies of DRL with real-world applications in autonomous navigation. In addition, a simulation experiment will provide us with the state-of-the-art optimized functions in a dynamic environment. In the end, we will discuss some potential possible research directions in DRL for navigation in mapless environment. For which we have great expectations that will lead to a real human level of intelligence.

1.3 Related Works

In the realm of collision prevention, integrated innovation in 3D sensor fusion and the use of dynamic safety zones seems to be a promising method. The potential to predict human actions also has the capability to prevent collision. On the human motion level, recent work has shown that human actions can be predicted from early stages of motion. On the task level, prior work has indicated that observing changes to the entropy rate of a Markov Chain, produced from a task description encoded as a Markov Decision Process, could be utilized to encode the uncertainty of the robot about what action the human will perform next. In other work, the encoding of discrete units of human and robot actions allowed for the incorporation of particular task specific rules and preferences, which could then be utilized to predict likely sequences of human actions. Mnih have presented incredible results to their deep neural network implementation of the Q learning algorithm, which have achieved human level performance in many Atari games [2]. Alongside, algorithm based on the Monte Carlo tree search and the deep integration of RL, beating the world champion in the game of Go. Wang applied the optimization algorithm DQN, but designed a new algorithm, network architecture, separate state value, and action advantage called Dueling DQN [3]. Later this method came up as Double DQN, which uses a second target Q network to avoid maximum deviation in Q learning and obtain better results in many areas. Tai proposed a flawless motion planner formed by deep reinforcement learning methods [4]. Reinforcement learning algorithm based on an uncertainty model to estimate the probability of collision in an unknown environment changed the angle of mapless navigation. This area went for deep research to find the estimated opponent's future behavior using historical information. With this information the method of Asynchronous Advantage Actor-Critic (A3C) was developed for neural networks in order to implement strategic functions superior to standard DQN. Chen used deep reinforcement learning to develop a multi-robot collision avoidance strategy, which also requires the deployment of several sensors to estimate the state of close people and moving obstacles [5]. Nevertheless, the complex structure of these navigation

methods not only require costly online calculations, but also reduce the robustness of the entire system to perceived uncertainty.

1.4 Motivation of Research

In today's world of robotics, there is a general tendency to implement behavioral mechanisms based on human psychology, using natural thought processing as an example. This allows a better understanding of the environment and the objects it contains. This trend is also valued in the field of mobile robot navigation. Navigators have been increasing their level of abstraction over time. However, the level of abstraction has further improved, introducing advance concepts that classify the spaces based on more complex and abstract data such as the utility and the objects they contain.

Another important aspect is the collection of environmental information, which must be available for navigation systems, among others, which are carried out by mobile robots. This information is provided by a perception system, which poses significant problems in terms of object and location recognition. One of the most difficult problems in scene recognition is the appearance of a place. Sometimes the same place may look different or different places may look similar. The position of the robot and the variation of its viewing angle (point of view) can also influence the identification of the place when it is revisited. Environmental models and the way navigation tasks are defined can be redesigned to take into account new technologies and trends.

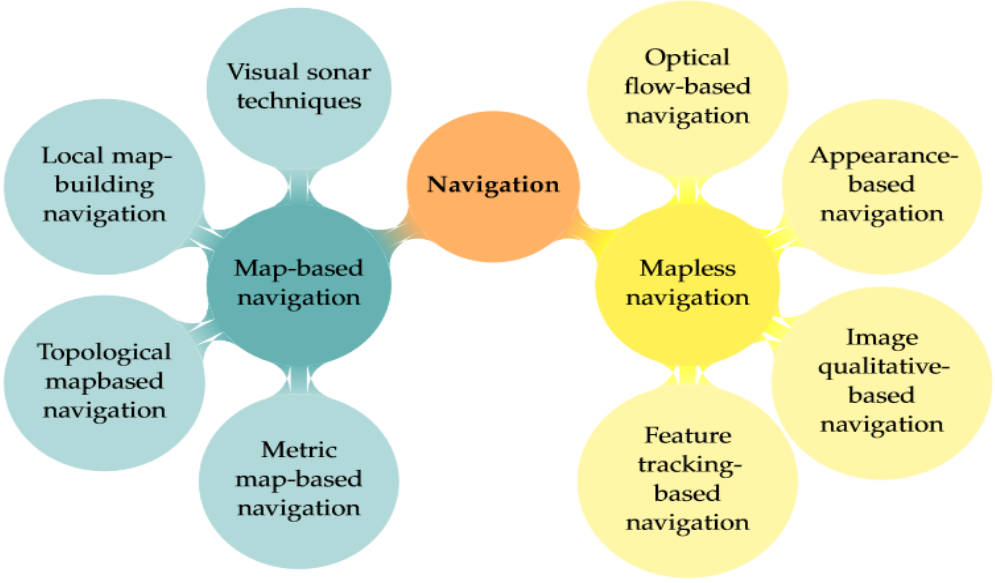

Figure 1.1 Navigation Techniques

This gives mobile robots greater autonomy and facilitate interaction with humans in their familiar environment. In this trend, vision is the main sensor for navigation, localization and scene recognition. Location recognition is a well-known and difficult problem, which relates not only to how a robot can give the same meaning to the same image as a human give to the same image, but also to the variability in the appearance of these images in the real world. Location recognition is closely related to many important areas of research in robotics, including simultaneous localization and mapping.

Regardless of the current task of specific, these machines should pay attention to safety and possible damage to the environment itself and to all persons or agents present in the environment. If the machine is mobile, the minimum operational requirement is the ability to plan its movement and avoid obstacles. Many intelligent machines are expected to perform complex tasks in unstructured environments. Regardless of the specific assignment at hand, such machines should pay attention to safety and possible damage to the environment itself and to all persons or agents present in the vicinity. In mobile robot, the minimum operational requirement is the ability to plan its movement and avoiding obstacles. If the environment is static, the usual approach is to create a map of the environment and use a path planning algorithm in conjunction with a location algorithm to safely navigate the workspace. Provide a model of the environment that can include varying degrees of detail as shown in Figure 1.1, such as a complete computational model or a simple graph of interconnections diagram.

1.4.1 Geometric Navigation

Geometric navigation consists in moving the robot from one point in the environment to another, these points being indicated by their coordinates on a map. The environment map is typically represented by a grid of points, and the path between two points is a sequence of points that the robot must reach in the given sequence. The robot controller has to reach the next point on the path by closing a loop about its position (distance and angle). One of the main goals of geometric navigation research is to plan the path from a starting point to a destination point, creating an algorithm capable of finding a path that guarantees completeness.

Although the trajectory planning task of mobile robots is largely treated, the computing capacity has increased exponentially and more complicated algorithms can be developed. The algorithms try to find the shortest or fastest path, while respecting safety parameters. Another challenge is to try to find solutions that offer smoother paths, trying to mimic human paths. Sample planning is based on an

incremental representation of space and uses collision-free algorithms to find the path. Some of the most commonly used algorithms for the path finding problem are Deterministic and Probabilistic.

1.4.2 Topological Navigation

Topological navigation refers to the navigational processes that take place using a topological representation of the environment. A topological representation is characterized by the reference elements of the environment and are defined according to the different relationships between them. The objective of topological navigation is to develop a navigational behavior closer to that of humans in order to improve the understanding of human-robot collaboration. The main effects of topological navigation:

- Topological navigation allows efficient planning. However, because it is based on relations, it does not minimize the distance traveled or the execution time.
- Topological navigation does not require accurate localization.
- Due to its natural conception, it is easy to be understood by humans.
- A complex recognition model is required.
- In large environments, it involves huge diagrams and diagrams are at a better scale as compared to geometrical representations.
- Considers different kinds of recognition.
- Object recognition, in order to perform various tasks in common indoor environments, mobile robots need to quickly and precisely verify and identify objects, obstacles for feature extraction and prediction.
- Place recognition for a semantic navigation that can improve human-robot and robot-environment interaction.

Topological representation of the environment, which is based on graphs, allows large navigation tasks and uses models similar to those used by humans. Unlikely, they are not able to overcome the unknown obstacles. This is the major disadvantage of all of these mentioned navigation techniques. To solve these problems, a new method has been invented, which is being researched as the context of mapless navigation.

1.4.3 Mapless Navigation

It is not necessary to describe the environment in advance when navigating and to determine the movement of the robot by observing / extracting relevant information on the constantly changing environment. The mobile robot is supposed to plan a trajectory from its current position to the final

location while taking into account indoor environment. Then, the motion controller will guide the mobile robot and follow the trajectories precisely to the target position. Effective motion planning strategies and stable motion controllers are mainly based on the mathematical calculations and geometric relationships.

For mapless navigation tasks, the decision system can be simplified into a movement planning part and a navigation control part. Although useful in many situations, the applicability of traditional decision-making systems is limited by their flexibility and versatility. The complexity of the environment and the dynamic obstacles will increase the calculation efficiency and reduce the navigation performance. In addition, the errors of individual steps will be accumulated to the end. Another approach is to perceive the local environment for autonomous navigation, but this method assumes the environment to be very geometrically predictable. Today's robots are no longer bound or limited to structured environments and are moving towards Unmanned Mapless Navigation (UMN). Navigation without a map is a reactive visual navigation technique where navigation is carried out without prior description of the environment. Motion planning plans at navigating robots to the desired target from the current position without colliding with obstacles. For mobile nonholonomic ground robots traditional methods, such as Simultaneous Localization and Mapping (SLAM), address this problem using the previous obstacle map of the navigation environment based on a dense laser zone findings[6].

Simultaneous Localization and Mapping SLAM

Although map and map-building based approaches have been successfully implemented, their applications are still limited to relatively simple scenarios for relatively straight forward tasks. They are based upon SLAM with a collision-free path within the map. They might not be suitable for an environment full of dynamic obstacles. it is expensive to maintain a reliable SLAM to operate delivery robots. They are mainly used in mobile robotics. It simultaneously estimates a robot's pose and the map of the environment. The movement plan aims to direct the robot from the current position to the desired target without colliding with obstacles. SLAM is a robot map drawing technology. Robots or vehicles draw routes in an area, but at the same time, they also need to know where they are. The SLAM process uses a complex set of calculations, algorithms and sensory inputs to navigate in a previously unknown environment or to revise a map of a previously known environment. SLAM allows for the remote creation of Geographical Information System Mapping (GIS) data in situations where the environment is too dangerous or too small for humans to map. SLAM is a key element in self-driving vehicles and other autonomous robots enabling awareness of where they are and the best routes to their destination.

By creating its own maps, SLAM allows a faster, more autonomous and more adaptive response than pre-programmed routes.

The collision and obstacle avoidance or pre-collision strategy requires knowledge of the geometry of the current environment and computationally intensive motion planning techniques using path and trajectory planning, and shows that the robot, when properly planned using the collision avoidance and path planning algorithm, can adapt to real-time trajectory adjustments. This mapping technology would certainly work within the constraints that has already been given as input to the machine. But in a dynamic and crowded pedestrian environment, SLAM often fails. To overcome such a challenge, robots are now trained to navigate the environment themselves using the visual data they collect from sensors.

1.5 Core of Mapless Navigation

Manually designed features are extracted to localize the robot and create the obstacle map. There are several addressed issues for this task:

- Creating and updating the obstacle map, which is time consuming,
- The heavy reliance on the accurate and dense laser sensor for mapping and forecasting local cost mapping,
- It is always difficult to quickly generate appropriate navigation behaviors for mobile robots with unobstructed maps of sparse information,
- Today, inexpensive methods such as Wi-Fi positioning and visible light communication provide a lightweight solution for positioning mobile robots.

Thus, mobile robots are able to obtain the real-time target position with respect to the robot coordinate frame. For motion planners, it is difficult to directly use local observations and target position information to generate global navigation behavior without a global obstacle map. Therefore, learning-free motion planner based on learning is introduced. In virtual environments, a nonholonomic differential drive robots are trained to learn how to reach the target avoiding obstacles through asynchronous depth reinforcement teaching, the most advanced type of teaching on the machine learning.

1.5.1 Reinforcement Learning (RL)

RL method aims to use observations from the interaction with the environment to take actions that would maximize reward or minimize risk. Reinforcement learning algorithm (called the agent) continuously land iteratively learns from the environment [7]. Thus, the agent learns from its experiences of the environment until it explores the full range of possible states. Reinforced learning is a form of

machine learning and thus also a branch of artificial intelligence. It allows machines and software (Agents) to automatically determine the ideal behavior (State-Action) in a given situation (Environment) in order to maximize performance (Reward) as in Figure 1.2. In order for the agent to learn its behavior, a simple reward in return is needed called a reinforcement signal.

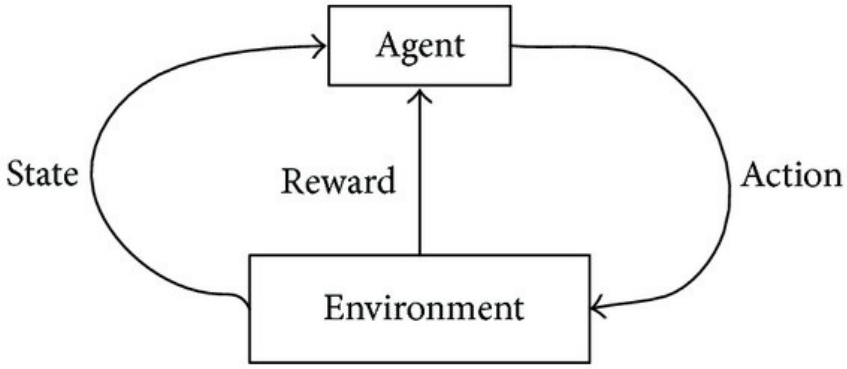

Figure 1.2 Framework of RL

Reinforcement learning is defined by a certain type of problem, and its solutions are classed as Reinforcement Learning algorithms. In the problem, an agent is supposed decide what is the best action to take based on its current state. The iteration of this process is known as a Markov Decision Process.

In order to produce intelligent programs also called agents, reinforcement learning involves the following steps:

- Input state is observed by the agent.
- Decision making function is used to make the agent execute an action.
- After executing the action, the agent receives a reward from the environment.
- The information on the state-action pair about the reward is stored

In recent years the field of Reinforcement Learning has seen a number of breakthroughs. By combining with developments from working with complex neural networks, commonly referred to as Deep Learning, practitioners have begun to offer various DRL solutions, which are capable of learning complex tasks while retaining a limited representation of the information learned. Nowadays, researchers have started applying such methods to different kinds of applications, and are attempting to exploit the ability of this learning approach to improve with experience. The purpose of RL is to teach the agent how to behave in an unknown environment by obtaining the optimal Q-value function that provides the best results for all conditions. The agent uses rewards received from the environment after selecting

actions for each state to update Q-values for convergence of optimality. Similarly, DL is a sub-domain of machine learning based on the concept of artificial neural networks that mimic the human brain when they process data and create models for use in decision making. DL enables automatic features engineering and end to-end learning through gradient descent and backpropagation. There are many types of DL nets, and their use depends on their application and the nature of the problem being addressed.

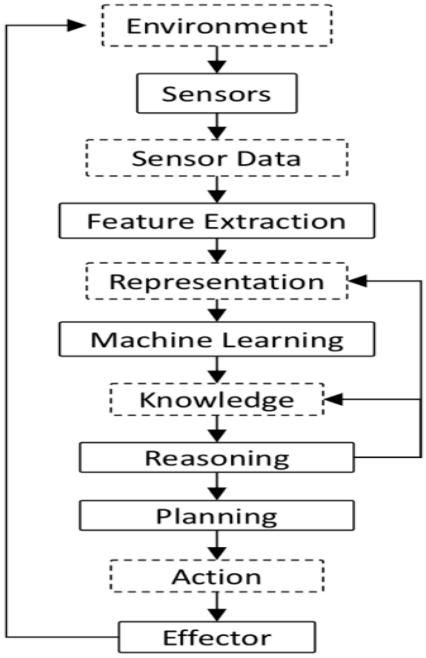

Figure 1.3 Framework of DL with RL

For time-based sequences like speech recognition, natural speech processing recurrent neural network is used. For the extraction of visual features such as image classification and object recognition, a convolution neural network is used. For data pattern recognition, such as classification and segmentation, feed-forward networks are used.

AI researchers have been trying to create an agent capable of thinking and acting independently in the real world thus the relationship between RL and DL technology is established as in Figure 1.3. In fact, in 2015, DeepMind successfully combined the RL decision framework and the DL representation learning framework [8]. This learning framework enabled the extraction of the visual characteristics, thus creating the first end-to-end artificial agent. This new concept named deep reinforcement learning is used now, not only to play ATARI games, but also for the development of the next generation of intelligent self-propelled cars such as Google with Waymo, Uber, Ford and Tesla. However, this mapless navigation

method has not yet been able to replace humans totally. Extensive experiments are carried out daily in this area of robotic technology. With the new development of Deep Reinforcement Learning, which combines deep learning methods with reinforcement learning algorithms, the RL represents a breakthrough in technical applications compared to previously unsolvable problems. Deep Reinforcement Learning allows RL to adapt to decision problems, for example situations with multidimensional state and action spaces.

In summary, this thesis will give an outline of different methodologies of Mapless Navigation based on DRL. The real-world applications of navigation based on DRL will be introduced in Chapter2. Chapter 3, will focus on different constraints and fundamentals of mapless navigation with the detailed description of DRL. In chapter 4, the different approaches (The State-of-the-Art) and significant impact on implementing DRL for a human-level autonomous agent will be discussed with analysis of some experiments and results from research. An overview of the state of the art of DRL algorithms, their functionality and optimality in context of navigation will be presented further.

Chapter 2 Applications of Unmanned Mapless Navigation

The machine in RL is driven by a feedback mechanism based on rewards / penalties, and its objective is to continuously improve the behavior of the machine or robot. Deep Reinforcement Learning is currently used to train systems in games like Atari, Alpha Go, Dota, Assassin's Creed, Chess, etc. [9] The self-navigation techniques have also been implemented in games training the AI with DRL model that can beat the human player. These mapless navigating robotic techniques based on DRL have been further applied to the following fields:

2.1 Medical Field

Figure 2.1 Intelligent Robots used in Hospitals

Intelligent robots are widely used in Medical field shown in Figure 2.1. They are the most intelligent robots of all other types with sensors and microprocessors for data storage and processing. The performance of these robots is based on condition analysis (i.e. self- navigating new obstacles) and task execution capabilities, so it is very efficient. Intelligent robots can sense sensations such as pain, smell and taste, but also have the ability to see and hear, and can perform corresponding actions and expressions, such as emotions, thinking and learning which are very useful in nursing field.

2.2 Industrial Use

Mobile robots are entering new areas usually reserved for humans and manned vehicles. In warehouses (see Figure 2.2) and distribution centers, in factories. These robots have embedded intelligence and adaptive functions in real time (i.e. auto-navigation techniques), expanding their capability of working like humans in industries. These auto navigating robots uses deep reinforcement learning algorithms that helps build complex models to reduce inventory and transport time for receiving

products in the warehouse, thereby optimizing the use of space and warehouse operations. Industries today are using several mobile robots for manufacturing and inventory and supply chain management in order to lessen the human error.

Figure 2.2 Autonomous Mobile Robot for Industrial Logistics

2.3 Automobile Industry

The automotive industry has huge and diverse data sets that supports deep learning by reinforcement. It has been used in autonomous cars and it helps transform factories, vehicle maintenance and automation in the industry. The industry has been driven by safety, quality and cost of DRL. Autonomous cars are the major breakthrough of DRL shaping the concept of mobile robots into real environment and real obstacles. Self-driving cars can sense the surrounding environment and navigate without human intervention. To accomplish this task, each vehicle is generally equipped with a GPS unit, an inertial navigation system and a series of sensors, including a laser rangefinder, radar and video. The vehicle uses information from GPS and inertial navigation systems to locate itself and data from sensors (such as Lidar, Radar, Cameras) to refine its position estimate and build a three-dime4nsional image of its environment. The data from each sensor is filtered to eliminate noise and is often merged with other data sources to enhance the original image. The vehicle then uses this data to make navigation decisions depends on its control system.

The goal of self-driving cars is set to interpret the world in a logical way without remembering a specific environment. This can be achieved by examining the information accepted by the engine and generating countless information generated manually as training data, and using a specific deep learning architecture to minimize over-fitting. If the algorithm specifically defined for training data cannot adapt the new data to a classification, which results in an error, an over-adjustment occurs. With the help of a complete set of tools and a perfect neural network, vehicles can meet the challenges of autonomous driving in various environments. Most autonomous vehicle control systems use a negotiated architecture (see Figure 2.3), which means that they can make intelligent decisions in the following ways:

- Maintain own internal map,
- Use the map to find the best path to the destination, which can avoid obstacles (such as road structures, pedestrians and other vehicles) from a set of possible paths.

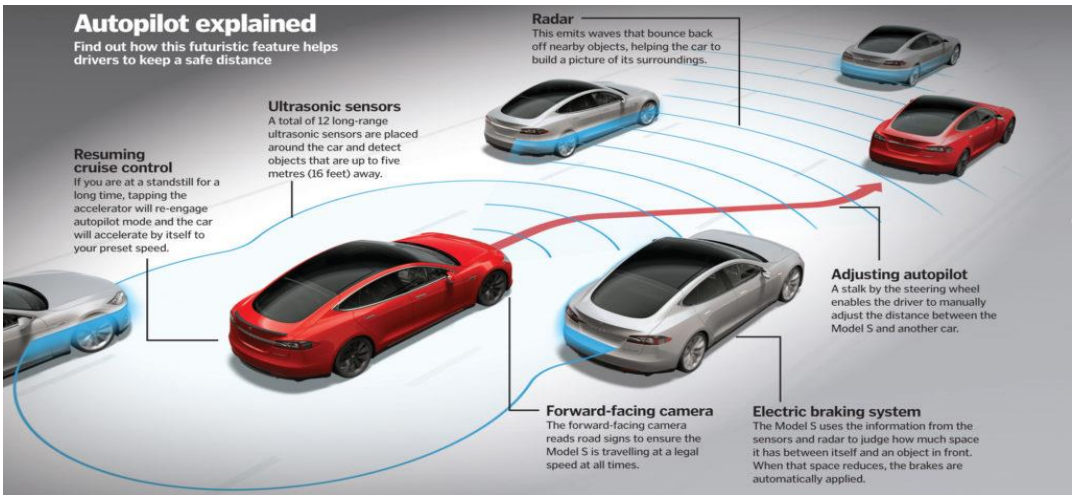

Figure 2.3 Autopilot components

Once the vehicle has determined the best path, the decision is broken down into commands, which are transmitted to the vehicle's actuators. These actuators control the steering, braking and throttle of the vehicle. In short, Self-driving cars are based on the following parameters:

- Mapping and Localization
- Obstacle Avoidance
- Path Planning
- Hardware Components (Sensors and Computational power)

In short, autonomous cars rely on sensors, actuators, complex algorithms, machine learning systems and powerful processors to run software.

2.3.1 Tesla Self-Driving Cars

From home to destination, Tesla's improved autopilot adds a new experience to autonomous cars. With 40 times processing power and advanced sensor technology, Tesla has paved the way for future driving methods.

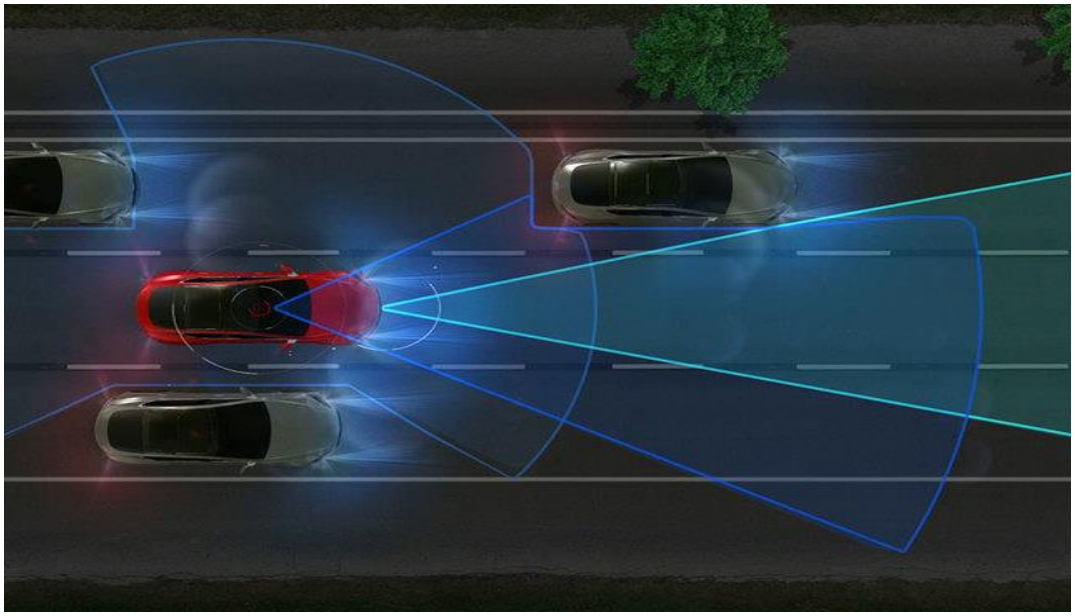

Figure 2.4 Tesla Autopilot Design

Tesla features allow autonomous driving:

- Front radar
- The camera offers visibility at 250 meters, 8 external cameras
- High precision CNC electric assist braking system
- 12 remote ultrasonic sensors in the car
- If covered with debris, these sensors can be affected
- Perceive all objects within 16 feet of the vehicle

The radar and front camera track the position of the car ahead and adjust Tesla's speed accordingly. This function maintains a safe distance between itself and the car in front as in Figure 2.4. The distance between the cars depends on the speed of the two cars. If the car merges into your lane, Tesla will monitor its position and reduce its speed. If the car merges and accelerates in your lane, then Tesla will not panic.

However, when the road markings are not clear and / or the car is traveling below 20 mph, the autopilot function does not work properly. It is not recommended to use this feature in residential areas with street lights and stop signs. The driver must still be careful and keep his hand on the steering wheel.

If the steering wheel does not detect the driver's hand, it will alert the driver visually and audibly to be checked. If the driver still does not get their hands on the steering wheel, the car will safely start to slow down until the car comes to a complete stop or the driver takes control.

2.3.2 Ford Self-Driving Cars

Ford is developing a hands-free driving system for the Mustang Mach-E, which will allow fully electric vehicles to travel on certain roads without the driver having to touch the steering wheel. First, you need to map the path with great accuracy, then use the car and more precise GPS sensors to ensure its correct position in the lane.

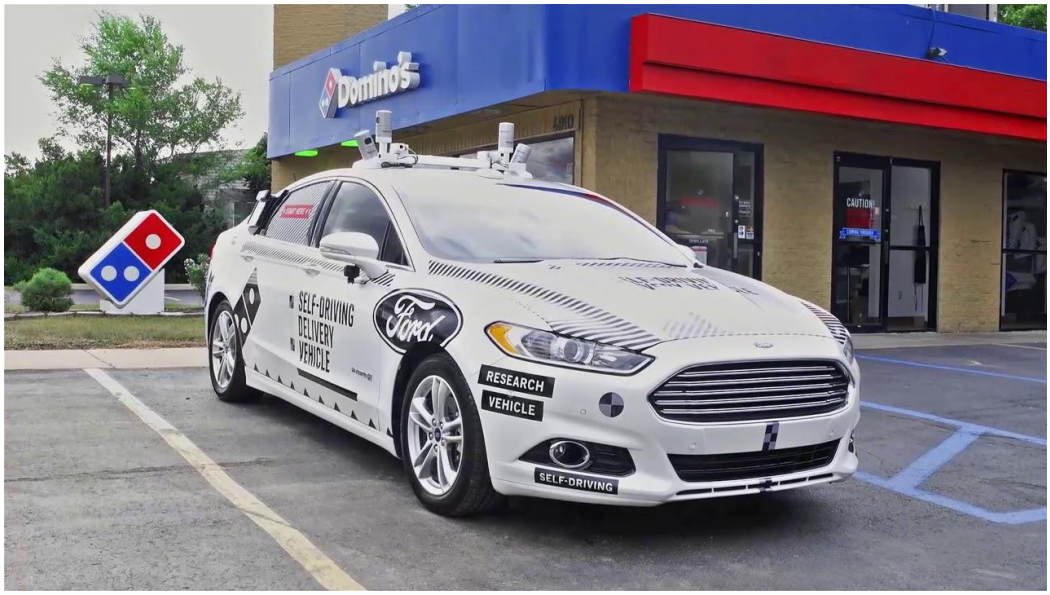

Figure 2.5 Ford Self-Driving Car

The number of sensors integrated in Ford Mustang Mach-E is increasing considerably. There are cameras, LIDARs, ultrasonic and radars currently used. The result is a 360-degree perception of the vehicle. Ford is underway of developing self-driving cars Figure 2.5.

Advantages of Autonomous Cars

- There are limitless options for convenience and quality of life. The elderly and the physically disabled will be independent.
- If anyone forgets anything, one can just send the car to send or pick up stuffs.
- Reduce transport costs by 40% (in terms of vehicles, fuel and infrastructure)
- Free parking for other uses (schools, parks, community centers)
- 80% global reduction in urban carbon dioxide emissions

The challenges of Autonomous Cars

However, if no one takes over the steering, the autonomous car is not yet able to reliably achieve fully automatic driving. In addition, autonomous cars have few challenges on the road and in crowded places.

- Lidar is expensive and always tries to find the right balance between range and resolution. If multiple autonomous vehicles are driving on the same road, their lidar signals interfere another.
- Weather conditions like rain, snow on the road may cause the dividing line to disappear. Then the camera and sensor will not be able to follow the lane mark.
- Human drivers rely on subtle signals and non-verbal communication, such as making eye contact with pedestrians or reading other drivers' facial expressions and body language to make instant judgments and predict behavior. Can autonomous cars make this connection? Will they have vital instincts like human drivers? This is the major challenge for self-driving cars.

Chapter 3 Fundamentals of Unmanned Mapless Navigation

3.1 Markov Decision Process (MDP)

The Markov decision process provides us with a formal sequential decision method. This formalization is the basis for the proper structure of problems solved by reinforcement learning. In MDP, there is a decision maker called an agent which interacts with the environment. These interactions occur gradually over time. At each time step, the agent obtains a sort of representation of the state from the environment as explained in Table 3.1. Given this representation, the agent will choose the action to take. The environment will then change to the new state and the agent will be rewarded for the previous action.

Table 3.1 Components of MDP

Components	Features
Agent	Learner and decision-maker
Environment	where the agent learns and decides what to do
Action (A)	A set of actions that can be carried out in the environment
State(S)	Set of agent's states in the environment
Rewards(S)	Each action performed by the agent will provide positive or negative rewards.

These five components are all interdependent as one's output becomes one's input as shown in Figure 3.1. The reinforcement learning problem aims to directly solve the problem of interaction learning to reach the objective [13]. The agent chooses actions and the environment reacts to these actions and results to a new situation to the agent. The environment grants rewards as numerical values that the agent tries to maximize over time. The full environmental specification defines a task, which is an example of a reinforcement learning problem. The process of selecting an action in a given state, transitioning to a new state and obtaining a reward is constantly repeated. This creates what is called a trajectory that shows the order of state, action and reward. In this process, the agent's goal is to maximize the total rewards for acting in a given state. This means that agents want to not only maximize instant rewards, but also maximize the cumulative rewards they receive over time. The agent's goal is to maximize the cumulative reward.

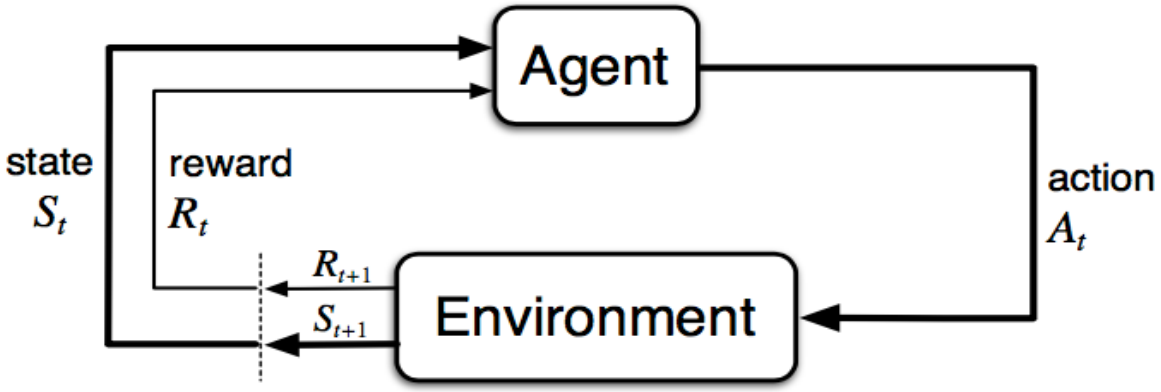

Figure 3.1 MDP Framework

In MDP, we have a set of states S, a set of actions A, and a set of rewards R. Assuming that each of these groups has a limited number of elements. A_t each instant t = 0,1,2, ..., the agent will receive a representation of the environmental state $S_t \in S$. On the basis of this state, the agent will select an action $A_t \in A$. This provides us an action-state pair (S_t, A_t). Then, the time increases at the time step following t + 1, and the environment passes to the new state $S_{t+1} \in S$. Then, the agent receives the reward $R_{t+1} \in R$ for the action at taken from the state S_t. Let, the process of accepting rewards as any 'F' function that maps state-action pairs to rewards. A_t each time t,

$F(S_t, A_t) = R_{t+1}$.

The trajectory of the sequential process of selecting an action in a state, transitioning to a new state and obtaining a reward can be expressed as S0, A0, R1, S1, A1, R2, S2, A2, R3,…. at time t, and the environment is in state S_t. The agent observes the current state and select the action A_t. The environment changes to state S_{t+1}, and the agent receives the reward R_{t+1}. Then, the process restarts at the next time step t + 1.

Transition Probability

Since the sets S and R are finite, variables Rt and St have a well-defined probability distribution. All of the possible values that can be assigned to Rt and St have an associated probability. These distributions depend on the coming states and actions which occurred at the previous time step (t-1). For instance, suppose $s' \in S$ and $r \in R$. Then there is the possibility of $S_t = s'$ and $R_t = r$. This probability is determined by the specific values of the previous state $s \in S$ and of the action $a \in A(s)$. Note that A (s) is the set of actions that can be performed from state s.

For all s ∈ S, r ∈ R and a ∈ A (s), we will define the probability of the transformation to state s′ with reward r when action a is performed in state s.

$P(s', r \mid s, a) = P_r\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}$.

3.2 Policy

A policy is a function that maps a given state to the probability of selecting each possible action in that state [14]. The symbol (π) to represent the policy.

In short, the agent "follows the policy". For example, if an agent follows the policy π at time t, then $\pi(a \mid s)$ is the probability of $A_t = a$ when $S_t = s$, i.e. under the policy π, the probability of taking an action a in the state s is $\pi(a \mid s)$.

3.2.1 On Policy

On-Policy learns the value of the policy done by agents, including the exploration steps. The algorithms related to policies that generate past state action decisions are called on-policy algorithms. The change on-policy Q learning policy is called SARSA. It can be used especially when the agent is likely to choose a random sub optimal operation in the next step. In SARSA, the agent learns the best policy and uses the same policy as the ϵ-greedy strategy to execute. Since the update policy is the same as the behavior policy, SARSA is on-policy.

The experience format in SARSA is ⟨s, a, r, s ', a'⟩, which means that the agent was in state s, performed actions a, received the reward r and finally entered state s ', from which it decided to take action a '. This results a new Q (s, a) update experience.

$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma Q(s', a') - Q(s, a)]$

3.2.2 Off- Policy

Off-policy learning refers to learning a behavioral method, that is to say a target strategy, from data generated by another way of choosing actions as behavior policy. Off-policy learners learn the value of an optimal policy regardless of the agent's behavior. Q-learning is a best suited off-policy. In fact, Q learning is sometimes called SARSA-max. It estimates future rewards and adds value to the new state without following any greedy policy. Three of its main features are:

- More freedom to explore,
- Learn(imitate) from human data,
- Reuse old data

In Q-Learning, the agent uses the absolute greedy policy to learn the best policy and uses other policies such as the ϵgreedy policy to work. Since the update policy is different from the behavior policy, Q learning is off-policy.

$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

When using a deep neural network, the closure strategy method is implemented by the Deep Q network (DQN).

3.3 Value Function

Almost all reinforcement learning algorithms involve an estimated value function, i.e. function of state-action pair, that estimates the performance of an agent in a given state or how feasible is to perform action in a given state [15]. The concept of "good" is defined here in terms of expected future returns, or rather expected returns. The reward the agent hopes to obtain in the future depends on the action it will take. Therefore, the value function is defined with respect to particular polices. The policy π maps the probability π (a | s) of taking the action a of each state s ∈ S and the action a ∈ A (s) when in the state s.

3.3.1 State Value Function

The state value function of policy π is represented by V_π, which tells how applicable the given state is for the agent following strategy π. In other words, it provides with the state value under π. Formally, the value of state s under policy π is the expected return from state s at time t with policy π. Mathematically, we define V_π (s) as

$$V_\pi(s) = E[G_t \mid S_t = s]$$
$$= E\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right]$$

The optimal State Value Function is : $V^*(s) = \max_\pi V_\pi(s)$

3.3.2 Action Value Function

The action value function of the policy π (Q_π) tells us what extent it is good for the agent to take given action from a given state by following the policy π. Simply, it gives the value of the action under π.

Formally, the value of action a in state s under policy π is the expected return from initial state s at time t taking action a under policy π. Mathematically, we define Q_π (s, a) as:

$$Q_\pi(s, a) = E[G_t \mid S_t = s, A_t = a]$$
$$= E\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]$$

Formally, the action value function Q_π is called the Q function, and the output of this function is used for any given state-action pair, called the Q value. The letter "Q" is used to indicate the quality of taking a given action in a given state.

The optimal Action Value Function is: $Q^*(s,a) = \max_\pi Q_\pi(s,a)$

The optimal Action-Value is further written in Bellman Equation:

$$Q*(s,a) = E[R_{t+1} + \gamma max_{a'} Q*(s',a')]$$

In short, the goal of Q-learning is to find the optimal policy by learning the optimal Q-values for each action-state pair.

3.4 Monte Carlo Method

The Monte-Carlo method only requires experience, i.e. a sequence of samples of states, actions and rewards in a real or simulated interaction with the environment[16]. The Monte Carlo method is a method for solving reinforcement learning problems based on average sample yields. It has two major features:

- *Policy evaluation:* use the current policy π to predict Q_π or V_π.
- *Policy improvement:* develop better policies by choosing actions in a deterministic way.

$$\pi(s) = argmax_a Q(s,a)$$

MC is used to make predictions by learning a state value function $V_\pi(s)$ which follows a given policy π, and by iterating Generalized Policy Iteration (GPI) and the action value function Q as shown as Pseudocode in Figure 3.2.

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \leftarrow$ arbitrary
  $Returns(s,a) \leftarrow$ empty list
  $\pi(a|s) \leftarrow$ an arbitrary ε-soft policy

Repeat forever:
  (a) Generate an episode using π
  (b) For each pair s,a appearing in the episode:
    $G \leftarrow$ return following the first occurrence of s,a
    Append G to $Returns(s,a)$
    $Q(s,a) \leftarrow$ average$(Returns(s,a))$
  (c) For each s in the episode:
    $a^* \leftarrow \arg\max_a Q(s,a)$
    For all $a \in \mathcal{A}(s)$:
      $$\pi(a|s) \leftarrow \begin{cases} 1-\varepsilon+\varepsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$$

Figure 3.2 Pseudocode for Monte-Carlo

In order to ensure a clearly defined advantage of Monte Carlo method, an episodic task is the best scenario. In other words, assuming that the experience is divided into several episodes, and whatever

action is selected, all the episodes will eventually end. It is only when the episode is completed that value estimate and policy are changed. The Monte Carlo method samples and average return for each state-action pair similar to the average return for each action. The reward after taking an action in one state depends on the action taken in a later state in the same episodes. Since all the action choices are learned, the problem becomes unstable from the point of view of the previous state. The Monte Carlo method can therefore be incremented in episodic sets, but not in step by step task.

3.5 Deep Learning (DL)

The term neural derives from the name of the basic unit of the nervous system and is called a "neuron," so this type of network is called a Neural Network (NN) [17]. It would be a neural network in the human brain, but what if the same power is applied in a group of artificial objects that can mimic the same behavior - this is where the artificial neural network appears. Artificial neural networks can best be described as bio-inspired simulations that are run on a computer to perform certain sets of tasks (such as clustering, classification, pattern recognition, etc.). Generally, artificial neural networks are biologically inspired essentially artificial neural networks configured to perform a specific set of tasks. The neural network is the main method to reinforcement learning in order to improve the generalization capacity. The optimized design of the neural network architecture can reduce over-fitting and improve the accuracy of predictions with high training speed and low computational cost.

In short, deep learning is based on deep hidden layers and extended hidden neurons neural networks, which perform high level abstractions in data based on an architecture composed of several layers of nonlinear neurons. Each neuron of the hidden layer linearly combines its inputs and applies a non-linear function (Relu, Softmax, Sigmoid, tanh, etc.) to the result, allowing the neurons of the next layer to separate the classes (hyper-curves by curves / hyperplane). The structure is shown in Figure 3.3.

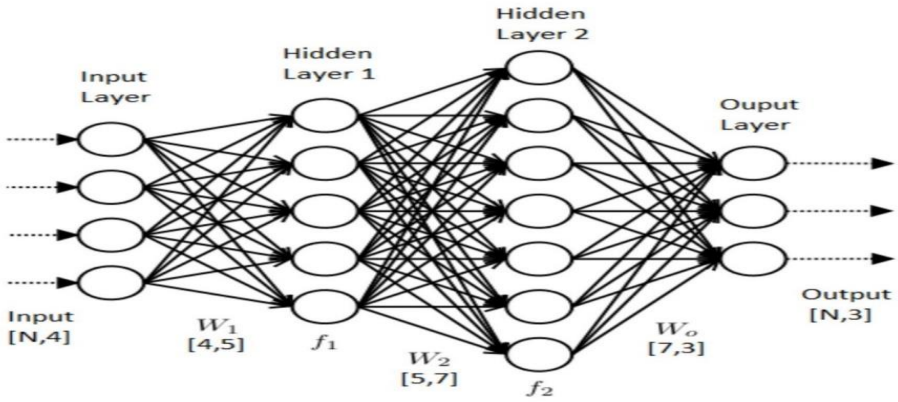

Figure 3.3 Structure of Deep Neural Network

Input layer

The input layer contains artificial neurons called units that receives inputs from the outside world. This is where learning or recognition takes place on the network, which it will be processed.

Output layer

The output layer contains units that respond to information entered into the system and whether it has learned tasks.

Hidden layer

The hidden layer is between the input layer and the output layer. The only job of the hidden layer is to convert the input into something meaningful that the output layer / unit can be used in one way or another. Most artificial neural networks are connected to each other, which means that each hidden layer is connected to the neurons in its input layer and the output layer, and no objects are suspended. This completes a full learning process, and when the weights inside the artificial neural network are updated after each iteration, learning will reach the maximum level.

Mechanism of the neural network

It is best to think of an artificial neural network as a weighted oriented graph, where the nodes are formed by artificial neurons, and the connection between the output of neurons and the input of neurons can be represented by directed edges with weights [18]. The artificial neural network receives input signals from the outside world in the form of patterns and images in the form of vectors.

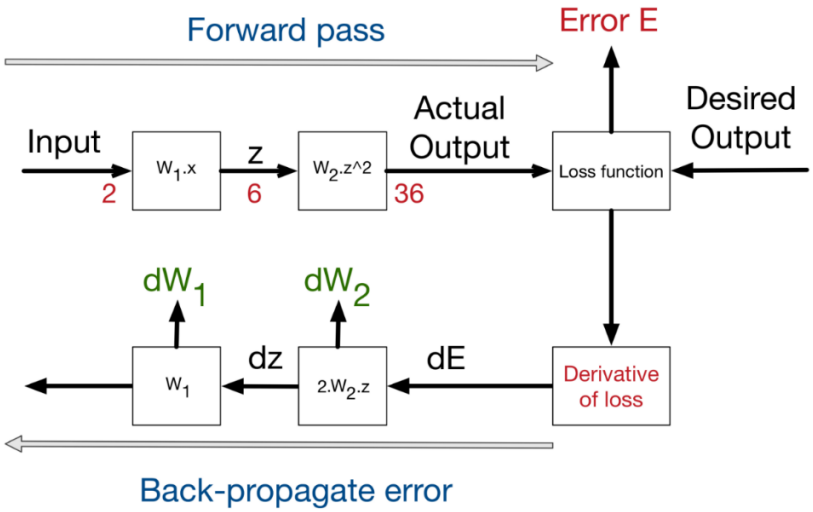

Figure 3.4 Framework of CNN

Weight (W)

As mentioned above, weights are connections between neurons with values. The higher the value, the higher the weight. Similarly, in mathematics and programming, we visualize the weights in matrix format. In general, these weights generally represent the strength of the interconnection between the neurons within the artificial neural network. All weighted inputs are accumulated in the computing unit, but another artificial neuron.

Bias

Bias is also similar to weight. Considering all possible or observable factors but there are always some parameters that is not taken into account. In neural networks, this unpredictable or unobservable factors is resolved as a bias. Each neuron that is not on the input layer is accompanied by a bias, as the weight has its own value. If the weighted sum is zero, a bias is added so that the output is not zero or scaled according to the system response. The weight of the bias is always equal to 1. The sum of the weighted entries ranges from 0 to positive infinity. In order to keep the response within the expected range, a certain threshold is calibrated.

Activation Function

Generally, the activation function is a set of transfer functions used to obtain the desired output. There are many forms of activation functions, linear or non-linear functions. Some of the most commonly used sets of activation functions are binary, Sigmoidal (linear) and Tan Hyperbolic S-shaped (non-linear) functions.

Forward propagation

After randomly initializing the model, the next natural step to take is to check its performance. starting with the existing inputs is passed them to the network layer and directly the actual output of the model is calculated. This step is called forward propagation, because the calculation process goes in natural direction entry from through the neural network to output.

Loss function

At this point, the actual output is already obtained from a neural network initialized at random. On the other hand, there is the desired output of network to learn. The performance measure about NN trying to reach its goal of generating an output as close as possible to the desired value is called the loss function.

Loss = (required output - actual output)

Back propagation

Back propagation is the essence of the formation of neural networks. It is a method for refining the weight of the neural network as a function of the error rate obtained during the previous period (i.e. iteration). Proper adjustment of the weights can reduce the error rate and make the model reliable by increasing generalization. Back propagation is the abbreviation for "backward propagation of error". It is the standard method for the formation of Artificial Neural Networks (ANN). This method calculates the gradient of the loss function relative to the weights in the network.

There are many kinds of Neural Network like, CNN, RNN, etc. Here, CNN will be focused as most of the autonomous navigation uses this technique of NN.

3.5.1 Convolutional Neural Networks (CNN)

It is a deep learning algorithm in which the input is used as an image, and weights and biases are effectively applied to its objects, which ultimately distinguishes the images from each other. In recent years. Convolutional Neural Networks (CNN) have made breakthroughs in the field of image recognition. CNN has a unique network processing mode, which has the features of local connection, pooling and weight which effectively reduces the cost, the complexity of the and the number of training parameters [19]. By using the CNN method, the image model in translation, scaling, and other transformations can be kept unchanged to some extent. Therefore, improving the robustness of the defects of system. Regarding the classifiers that process sensor data, CNN have become a very popular solution.

A convolutional neural network (CNN) consists of one or more convolutional layers (usually with subsampling steps), then one or more fully connected layers, just like in a standard multilayer neural network. CNN architecture is designed as a 2D structure that uses input images or other 2D inputs, such as voice signals. This is achieved by local connection and weight bonding and subsequently some form of polling, which results in invariant translation functionality. Another advantage of CNNs is that they are easier to train and have fewer parameters than fully connected networks with the same number of hidden units. As in Figure 3.5 CNN consists of four basic components:

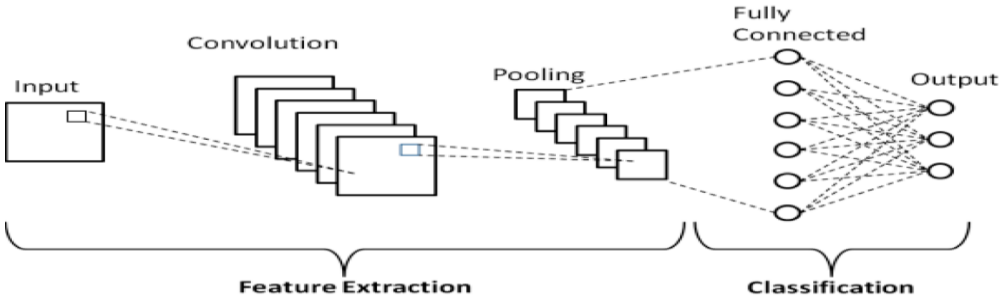

Figure 3.5 Structure of CNN

Convolutional layer

This layer is not only fully connected as a feed-forward network, but a 2D square matrix of neurons called filters (Kernel of as 3 × 3 pixel) is analyzed. The entire image extracts the patterns (local characteristics) by applying effects such as blurring, sharpness, outline, etc. to extract the visual characteristics. Each neuron of Kernel in the hidden layer is be connected to a small area (for example 3 × 3 pixels) of the input image (for example 200 × 200 pixels) given by the previous layer.

Zero padding: It places zeros on the border of the image so that the convolution output size is the same as the input size image.

Strides: It decides when scanning an image, how many pixels the kernel window will slide at each step.

Local connectivity: Each input neuron in the Kernels receives input from only a small part of the local pixels called local receptive fields in the input image by cutting connections with the other pixels. The input neurons represent overlapping receptive fields which form a complete visual space map.

Feature extraction: Each kernel can detect only one localized feature. Therefore, to find 10 different models, there must be 10 Kernels in the convolutional layer and each Kernels searches for a specific pattern on the image.

Learning hierarchical characteristics: Inspired by the visual cortical organization of animals, networks of multiple convolutional layers allow learning of hierarchical visual characteristics. The deeper the layer, the more complex the characteristics detected.

Pooling layer

Polling layer is immediately used after the convolutional layer to use nonlinear subsampling to reduce the output image and increase some translation invariance. There are several ways to implement pooling. But No learning happens in the pooling layer. For backpropagation, only the convolutional layers are involved, and when we want to "learn" certain locations of certain objects like in reinforcement learning pooling is not used.

Fully connected layer

It is a common feed-forward network layer, which establishes a connection between each input dimension pixel position and each output category, mixes the signals received from the learning layer of the characteristics and determines the classification according to the whole image.

Normalization layer

The layer is used to apply batch normalization on the input and hidden layers to resize the input data which avoids the decent problem of the (disappearance / explosion) gradient thus having a deeper network.

3.6 The Environment

Definition of a disordered environment consists of a two-dimensional or three-dimensional workspace, which contains a set of simple, closed and impenetrable obstacles [20]. This means that the robot must not overlap obstacles. The area outside the obstacle is considered to be uniform and it is just as easy to navigate in all its parts. Almost all methods model robots moving in space as points, circles, or polygons approaches virtually.

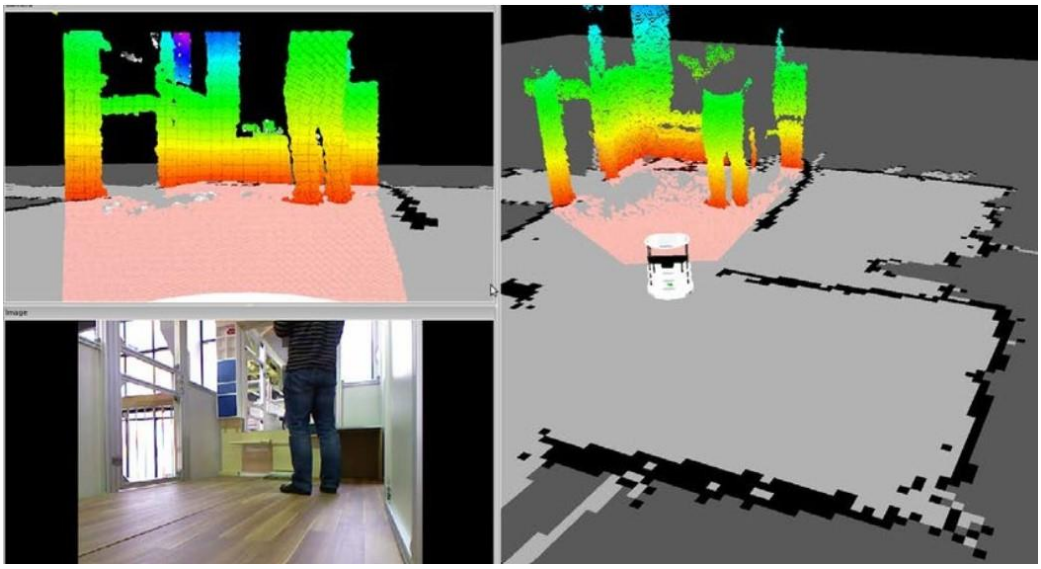

Figure 3.6 Turtlebot on 3D home environment

Kinematics of mobile robots have to operate in a chaotic environment like in Figure 3.6. For example, UAV, drones, surface ships, and submarine unmanned vehicles.

3.6.1 Types Kinematics of Vehicles

Holonomic Kinematics

These models are the most popular for finding mobile robots that have control capabilities in all directions. Complete kinematics is encountered on certain types of wheeled robots such as helicopters and equipped with omni-directional wheels. For the purposes of trajectory planning, these models do not involve the concept of body orientation, but only take into account the position coordinates. However, when applying directional navigation rules to real vehicles, orientation can be taken into account, although this has nothing to do with the map.

Unicycle kinematics

These models describe the mobile robot associated with a specific angular direction, which determines the direction of the speed vector. The change of orientation is limited by the speed constraints. The unicycle model can be used to describe various types of robotic vehicles, such as differential-wheeled mobile robots, drones and underwater vehicles, missiles, fixed-wing aircraft, etc., which in some cases have similar equation control differentials.

Bicycles kinematics

This type of model is very popular in the study of mobile robots such as cars, which have a pair of steered wheels, usually front wheels and a pair of fixed wheels, usually rear wheels. Kinematically, this means that the maximum speed of rotation is directly proportional to the speed of the robot. These determine the absolute limits of curvature of any path the robot can follow, regardless of its speed. This limitation usually requires high-level planning to navigate in a tight environment. In addition to these basic variants of kinematics, the associated linear and angular variables can be controlled in speed or limited in acceleration. Mobile robots with limited acceleration control inputs are often difficult to navigate.

3.7 Sensors

The navigation decisions of most autonomous cars are based on data reported by on-board sensors which provide information on the current environment of the vehicle[21] . There are several kinds of sensors used for navigation these days.

3.7.1 Types of Digitization of Sensor Models

Abstract sensor model

The label can be applied to any method based on a sensor; in these methods it is assumed that the law of navigation can certainly know if a given point is in a given obstacle. In general, any blocked area (without the robot's view) is considered to be part of the obstacle. Although it is not possible to use physical sensors to fully identify obstacle boundaries, LiDAR (light detection and ranging technology), in Figure 3.7 can achieve sufficiently high accuracy that no sampling effect will retain the people's attention. However, with a lower resolution, this model may not be suitable for the design of navigation.

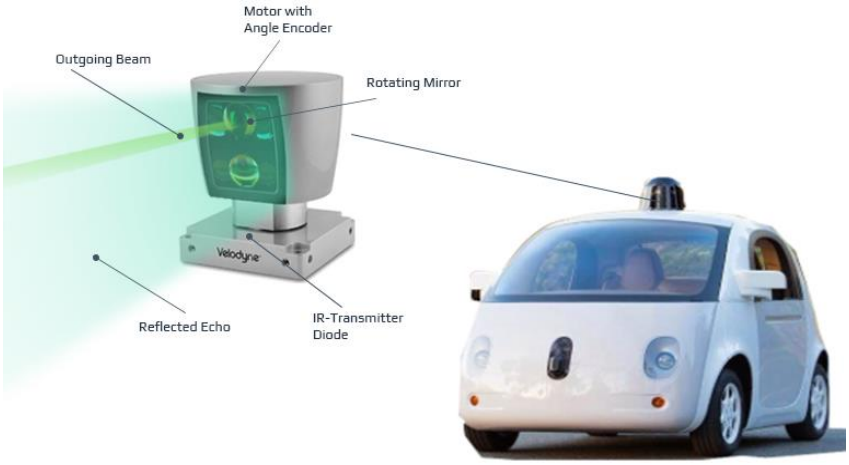

Figure 3.7 Lidar Sensor in Autonomous Vehicles

Ray sensor model

These models transmit the distance to obstacles in a limited direction around the robot to the law of navigation. Compared to abstract sensor models, this is a physically more realistic model of laser-based sensors, and may be suitable for dealing with the effects of low-resolution sensors. A simplified version of this model is the basis of certain limit tracking applications, where a single detection ray in a fixed direction relative to the robot body is used.

Minimum distance measurement

The model shows the distance to the nearest obstacle. This can be achieved with certain types of large aperture sensors, such as acoustic or optical flow sensors. The use of this type of measurement results in a less efficient mode of movement when avoiding obstacles, because it may not be possible to immediately clear which side of the robot the obstacle is located.

Tangent sensor

The sensor model indicates the angle the robot sees relative to the visible edge of the obstacle. As long as a method of detecting the edge of an obstacle from the video stream can be used, this can be done by a camera sensor.

3.8 Trajectories Constraints

There are several ways to prefer one possible path over another. Many path optimization algorithms can be implemented with various such measures or their combination.

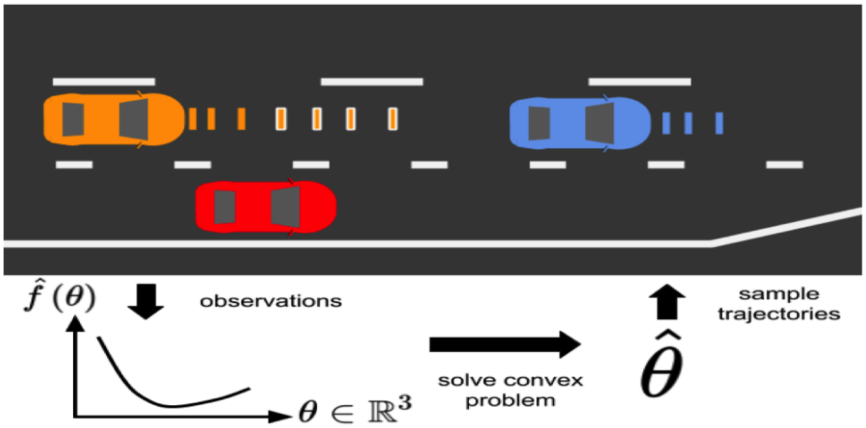

Figure 3.8 Trajectory planning of autonomous vehicles in highway

3.8.1 Measurements of Trajectories

The minimum travel distances

Most path planning diagrams use this metric. Its ease of use is partly because it can be separated from the workable speed curve of the robot. The optimal limited curvature path of the mobile robot between two configurations in the barrier-free workspace: the optimal path consists of a sequence of no more than three motion primitives. One of them goes straight ahead, the other two turn left and right respectively. To mobile wheeled robots with differential drive, minimum wheel rotate is considered and, in most cases, there is only a slight difference from the minimum distance metric above. However, in some cases, particularly when fine movement is required, this standard may require better performance.

The shortest time

The travel time of the journey depends on the speed curve of the mobile robot, and therefore on the kinematic constraints (possibly dynamic). In most cases, this metric is more appropriate for selecting the most efficient operation than the minimum distance. This standard is often used in methods based on MPC.

Navigation algorithms based on DRL include all of these fundamental as foundation. The factors such as environment, sensors, trajectory planning and DRL algorithms are what helps for the best mapless navigation. The DL-based method uses deep neural networks to extract reasonable patterns of navigation behavior from a large amount of labeled expert data [22]. However, to navigate an unknown environment takes a lot of time and energy, which prevents the DL-based method from being widely used to solve the proposed problem. DRL, on the other hand, cannot learn from labeled data, but gains experience in the interaction between the agent and the environment. Using values and policy-based methods, on policy

and off policy-based methods, as well as discrete and continuous domains, neural networks can be forms stability through reinforcement learning.

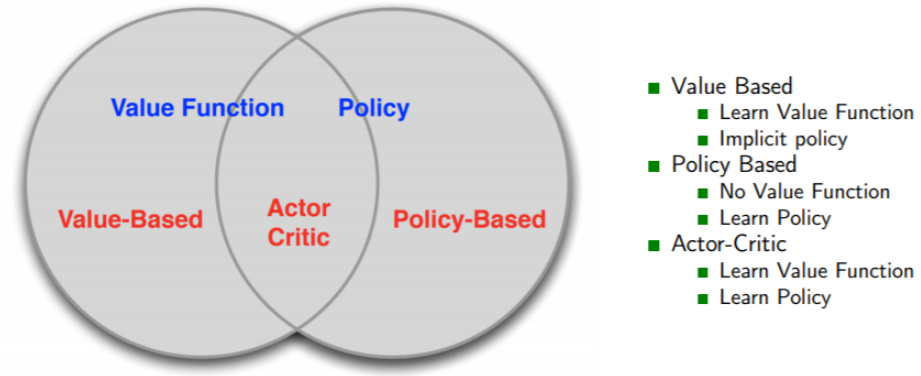

Figure 3.9 Inter-relationship between Value-base, Policy-based and Actor-Critic

Generally, they can be divided into two types, namely DRL based on value (DQN) and based on policy (DDPG, A3C, PPO) as shown in Figure 3.9. These frameworks will be discussed in next chapter.

Chapter 4 Methodology and Analysis

Deep reinforcement learning is applied to autonomous navigation based on capturing visual information, and has achieved remarkable success. With the latest developments in reinforcement learning (RL) and the methodological applicability of RL to agents capable of sequentially detecting the environment and generating actions, navigation policy is to be optimal. Most of these optimal policy takes the scan of the environment, the angle and the distance to the target position as the representative of the observation of the world, and issues the robot's movement commands [23]. The DRL algorithm has been applied to a variety of problems. It is now possible to learn the robot control policy directly from the camera input in the following controllers in the real world. The state of the robot is learned in the dimension function. Deep reinforcement learning achieves excellent performance in complex sequential tasks by using deep neural networks as a function approximator and by learning directly from the original images. A disadvantage of using the original image is that in addition to the learning policy, the deep RL must also learn the representation of the state characteristics from the original image. Therefore, deep RL may requires a large amount of training time and data to achieve reasonable performance.

This chapter will analyze the navigation in dynamic environment using the Deep Q Network (DQN) and Asynchronous Advantage Actor-Critic(A3C) review algorithms to evaluate the method based on experience. Environmental navigation model follows a basic foundation MDP (Agent, Environment, State, Action, Award) and optimize behavioral policy in the context of deep reinforcement learning. The agent is initialized randomly in the environment, then the state S and performs the action A according to the policy π (s): S $\rightarrow$ A. This brings the agent to new state S1 and obtains the reward R1. This process is repeated until the terminal state is reached. The terminal state is most of the cases is either the state where the robot has reached the target state, or the state where the robot has collided with an obstacle.

The DRL algorithm can be divided into value-based RL or value-optimized (Q-Learning), policy-based RL or policy-optimized. There are different state-of -the-art of Deep Reinforcement Learning which are discussed below.

4.1 Deep Q Network (DQN)

In deep Q learning, a neural network is used to approximate the Q value function. This state is given as input and the Q values of all possible actions are generated as output. The following Figure 4.1 is a good illustration of the comparison between Q learning and deep Q learning:

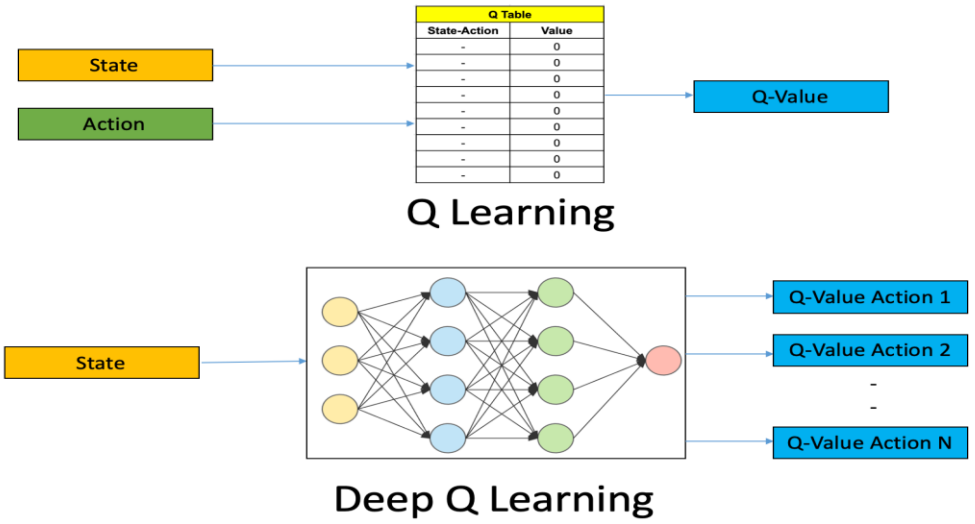

Figure 4.1 Q Learning vs. Deep Q Learning

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)$$

The highlighted part represents the objective. It predicts its own value, but since R is a real impartial reward, the network will use backpropagation to update its gradient to finally converge.

The following steps are involved in the Deep Q Network (DQN):

•The input screen preprocess (state (s)) and sends it to DQN, DQN will return the Q value of all possible actions in this state.

•The epsilon-greedy policy is used to select an action. For the epsilon probability, a random action a; for the probability (1- ε) an action with the largest Q value, for example a = argmax (Q (s, a, w)).

•This action in performed in state s, then goes to new state s′ to get reward r. State s′ is the preprocessed image of the next input screen. This transition is stored under <s, a, r, s ′> in the replay buffer.

• The loss function is calculated, which is the mean square error of the predicted Q value and the target Q * value. This is essentially a regression problem. Return to the Q value update equation derived from the Bellman equation.

• A gradient descent based on real network parameters is performed to minimize this loss

• After each iteration, actual network weights are copied to the target network weights in the neural network.

4.1.1 Double DQN (D-DQN)

Deep reinforcement learning through double Q learning is a double Q network algorithm that selects actions by maximizing the value of the predicted Q network and updating the predicted Q network with the selected action value in the Q network. The Q network prediction process is directly updated with the maximum value of the target Q network. Double DQN (D-DQN), which uses two sets of weights θ_i and θ_{i-1}, where the online network Q (s, a | θ_i) is used to select actions and target networks Q (s, a | θ_{i-1}) Used to assess the value of the action.

$$Y_t DoubleQ = R_{t+1} + \gamma\, Q(S_{t+1}, argmax_a\, Q(S_{t+1}, a|\theta_i)|\theta_{i'})$$

Implementation of DDQN in mapless navigation

Lei proposed a new trajectory planning algorithm using DDQN by the deep reinforcement learning, with Lidar sensor inputs where DDQN can solve the problem of dimensionality curse in high dimensional state space with the optimal action value function Q* parameterized by an approximation function [24].

$$Q\ (s, a; \theta) \approx Q^*\ (s, a)$$

By adding a target Q network, whose update has been delayed compared to the predicted Q network. Thus, updating the target value of Q by limiting the large deviation of Q caused by the fast-dynamic update of the Q network, the loss function is improved:

$$L_i(\theta_i) = E\,[(r + \gamma\, max_{a'}\, Q(s', a'|\theta_i - 1) - Q(s, a \mid \theta_i)]$$

Where θ_i is the network parameter Q during the iteration of, and θ_{i-1} is the network parameter used to calculate the target in the iterative calculation.

Likewise, in most of the real-world application DDQN is implemented using CNN to train path planning for the local dynamic environment. The architecture of a Q network composed is of three convolutional layers and a fully connected layer is designed. The rectified linear unit (ReLU) can be used as an activation function in CNN and to improve the performance of CNN. In short, local path planning is carried out by a trained DDQN algorithm. The entry is the angle, the distance detected by the lidar and the local planning point. The output is the direction of movement. And, at the same time, the trajectory is smoothed. During the trip, the agent builds a map and navigates in real time. Many experimental results show that by using knowledge of lidar data, the DDQN algorithm can perform local path planning in unknown dynamic environments and has great flexibility.

4.1.2 Duel DQN

For states with multiple action choices, DQN typically aims to estimate the Q value of each pair of state actions. However, sometimes for states where certain actions do not affect the environment in a relevant way, it is not necessary to calculate the value of each action.

$$Q(s, a) = A(s, a) + V(s)$$

A dual network architecture has two network flows which are used to estimate the state value function V (s) and the associated advantage function A (s, a), then is combined to estimate the action value function Q (s, a).

4.2Advantage Actor-Critic (A2C)

The Advantage Actor-Critic (A2C) learning algorithm is used to represent the policy function independently of the value function. A2C is a modeless method that uses an advantage function to optimize policy. The policy functional structure is called an actor and the functional value structure is called a critic. Taking into account the current state of the environment, the actor generates an action, and taking into account the state and the resulting reward, the critic generates an error signal TD (Time Difference) as shown in Fig: 4.2. If the critic estimates the action value function Q (s, a), the actor's exit is also required. The output of critics promotes the learning of actors and critics. In deep reinforcement learning, neural networks can be used to represent the structure of actors and critics. Many different policy optimization methods (Vanilla PG, TRpO, PPO) can be used to implement the Actor part of A2C. The algorithm can be equated as:

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta \, log\pi_\theta \, (a_t \mid s_t) \mid A(st, at)$$

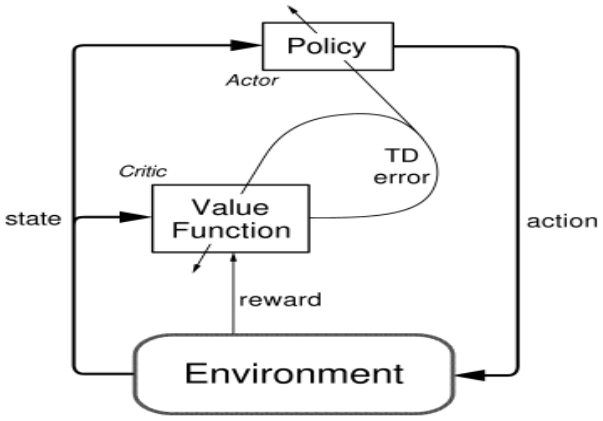

Figure 4.2 Actor-Critic Framework

The advantage function is calculated by the critic, who can be trained to approximate the function. The action network is a behavior policy represented by a neural network which generates the probability that each action is the best action in the current state.

$$\pi (a_k \mid s_k \mid \theta_a) = \Pr (a = a_k \mid s = s_k \mid \theta = \theta_a)$$

The vector θ_a is the weight of the neural network. The objective is to learn the weight θ_a on the basis of performance indicators so that the policy is close to the best policy by maximizing the actualized rewards accumulated. Given the status, the network of evaluators estimates the expected future rewards:

$$V (s_k \mid \theta_c) = E [R_k + \gamma R_{k+1} + \gamma 2R_{k+2} \ldots \mid s_k, \theta_c]$$

Where θ_c is the weight of the network of critics. It should be noted that the network of actors and critics has the same weight on the lower layers.

Implementation of A2C in mapless navigation

Dobrevski designed framework to learn navigation policy based on targets without maps that try to get closer to the target at each time step while avoiding collisions with obstacles using the latest A2C reinforcement learning algorithm [25]. Where an agent is fully trained with A2C in simulation, proving that the policy can be directly transmitted to the real robot without any domain adaptation. The network architecture is built as CNN. The flow of actors and critics on network is rapidly divided, with only the first three layers being shared, and each individual flow having three layers. The network providing a categorical output for the actor and a linear output for the critic, thus showing the success of A2C.

4.2.1 Asynchronous Advantage Actor-Critic (A3C)

The A3C model uses critical stakeholder methods for reinforcement learning by combining policy and state value functions. At each discrete time step t, the robot in the state performs a navigation action a_t, which moves the robot in the state s_t to maximize the expected future returns. These navigation actions include the robot turning forwards, backwards or right or left. Here, the participant retains the policy $\pi (a_t \mid s_t \mid \theta)$, which is determined by the neural network with the parameter. The critical estimation function $V (s_t \mid \theta_v)$ so that actors can adjust their policies accordingly. When the robot obtains new information, actors and critics update their networks. Every 1/3 step or when the terminal status is reached, the policy and value function is updated. The reward is used to calculate and accumulate the gradient for each step.

$$\nabla_\theta = \Delta_\theta, log\pi(a_t \mid s_t \mid \theta')\ \ A(s_t, a_t \mid \theta, \theta v)$$

$A (s_t, a_t; \theta, \theta_v)$ is the estimate of the advantage function.

Since the A3C deploys multiple agents that operate independently in its own environment, each agent provides updates to global policies and value functions asynchronously.

Implementation of A3C in mapless navigation

Zhang introduced a terrain modeling method in which noise sensor data from stereo cameras and LiDAR were used to estimate the height of the ground and vegetation and classify the obstacles mainly in the agricultural environment [26].

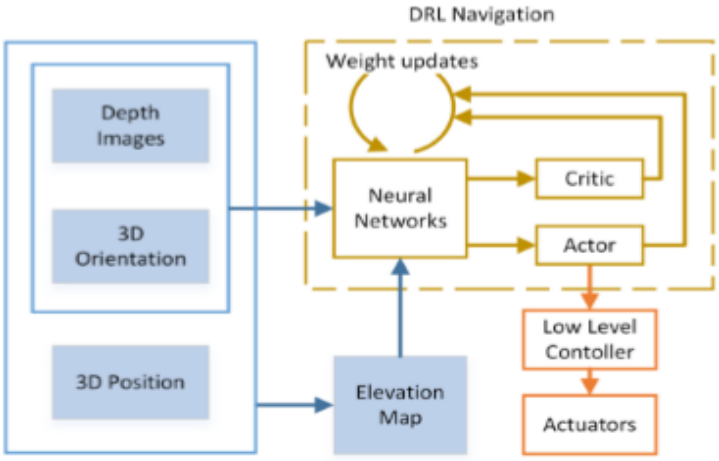

Figure 4.3 Architecture for navigation with A3C

A network based on the A3C architecture which uses depth images, elevation maps and 3D directions (in Figure 4.3) as inputs to determine the optimal navigation action of the robot. Likewise, A3C model learns by running multiple copies of the learning thread in parallel, and updates a shared set of model parameters asynchronously.

The A3C, which solves the problem of navigation in an unknown dynamic environment, still faces the following challenges. In an unknown environment, the robot cannot collect incomplete and noisy environmental data within a certain range thanks to its own sensors, which makes the robot controlled by the A3C fall into the local poles of the valleys, walks and other difficult terrain areas small trap. It is also difficult for other DRL algorithms to avoid local minima. These issues mentioned above can be now solved by using MK-A3C.

4.2.2 Memory and Knowledge-Based (MK-A3C)

Zeng built a MK-A3C that also collects noisy environmental data along with Gated Recurrent Units (GRU-based) neural memory network to improve the robot's temporal reasoning ability [27]. MK-A3C robots with a certain storage capacity can avoid the minimum local traps by estimating the environment

model. Likewise, MK-A3C can combine the reward function based on domain knowledge and the training task architecture based on transfer learning, which can solve the problem of non-convergent policy caused by sparse rewards. These improvements to the MK-A3C can effectively steer the robot through an unknown dynamic environment and respond to dynamic constraints when it comes to moving objects.

4.3 Deep Deterministic Policy Gradient (DDPG)

The Deep Deterministic Policy Gradient (DDPG) is an algorithm that learns Q functions and policy simultaneously. It uses non-strategic data that uses Bellman equations to learn the Q functions, and uses the Q functions to learn the policy. It is a method of actor-critic. In addition, it adopts the concept of DQN to stabilize learning, but at the same time solves the continuous action space, so it is more suitable for control tasks. It takes DQN (discrete motion only) and modify it to use with continuous motion space. DDPG is a combination of DQN and Policy Gradient (PG). Using the semi-gradient descent method, and all DQN techniques are applicable to DDPG. It is only used in an environment with a continuous action space, i.e. it is rarely used in video games, but it is used in actions or robot actions that require control continued. Unlike DQN, the target network is updated every k steps, and in the case of DDPG.

Randomly initialize critic $Q(s,a|\theta^{\mu})$ and actor $\mu(s|\theta^{\mu})$ neural networks with weights θ^{Q} and θ^{μ}.
Initialize target networks Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^{Q}$, $\theta^{\mu'} \leftarrow \theta^{\mu}$.
Initialize replay buffer b.
for episode = 1,...,M **do**
  Receive first observation s_1.
  for t = 1,...,T **do**
    Select a_t based on ϵ-greedy algorithm: select random action a_t with ϵ probability, otherwise $a_t = \mu(s_t|\theta^{\mu})$ according to the current policy.
    Execute action a_t and observe reward r_t and new state s_{t+1}.
    Store transition $[s_t, a_t, r_t, s_{t+1}]$ in b.
    Sample a random batch of N transitions $[s_j, a_j, r_j, s_{j+1}]$.
    Set $y_j = r_j + \gamma Q'(s_{j+1}, \mu'(s_{j+1}|\theta^{\mu'})|\theta^{Q'})$.
    Update critic by minimizing the loss:
      $L = \frac{1}{N}\sum_j (y_j - Q(s_j, a_j|\theta^{Q}))^2$
    Update the actor policy using policy gradient:
    $\nabla_{\theta^{\mu}}\mu|_{s_j} \approx \frac{1}{N}\sum_j \nabla_a Q(s,a|\theta^{Q})|_{s=s_j, a=\mu(s_j)} \nabla_{\theta^{\mu}}\mu(s|\theta^{\mu})|_{s_j}$
    Update the target networks:
      $\theta^{Q'} \leftarrow \nu\theta^{Q} + (1-\nu)\theta^{Q'}$
      $\theta^{\mu'} \leftarrow \nu\theta^{\mu} + (1-\nu)\theta^{\mu'}$
  end for
end for

Figure 4.4 DDPG Pseudocode

DDPG mainly uses two neural networks, one for actors and the other for critics. These networks calculate the current state of the action prediction and generate a temporal difference error signal at each time step. The actor network input is the current state and the output is actual value, which represents the action selected in the continuous action space. The output of the critic is only the estimated Q of the current state and the actions taken by the actor. The deterministic policy gradient theorem provides an update rule for the weights of the participant networks. The neural networks are used for the approximation of the functions. As shown in Figure 4.6 Actor and Critic have two NN structures; Actor Network and Critic Network where, weight of the Actor Neural Network is expressed as θ_μ and the weight of the Critic Neural Network is expressed as θ_Q. Therefore, objective Q_{target} can be rewritten as follows:

$$Q_{target} = r_i + \gamma\, Q'\left(s_{i+1}, \mu'\left(s_{i+1} \mid \theta_{\mu'} \mid \theta_{Q'}\right)\right)$$

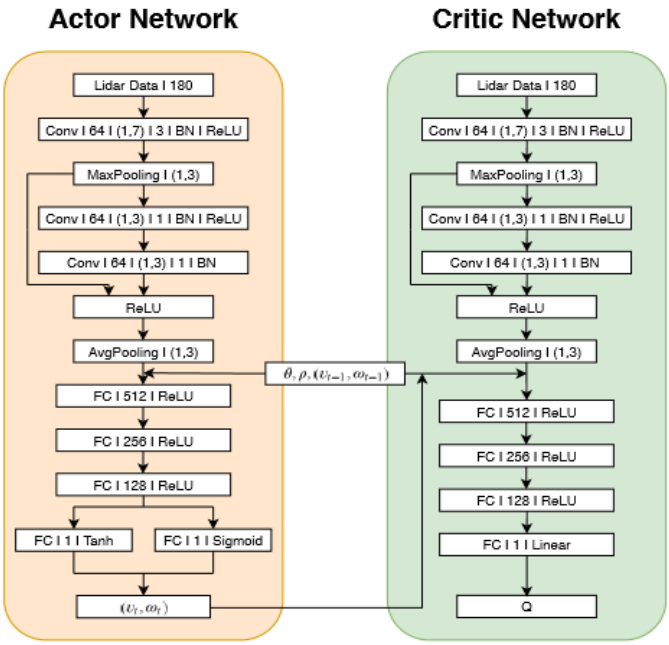

Figure 4.5 Structure of DDPG

4.3.1 Parallel Deep Deterministic Policy Gradient (PDDPG)

PDDDG algorithm can train robots in parallel for mapless navigation. In a multi-robot system, the observation value of each robot is included in other robots, which can be considered as dynamic obstacles. This method uses the same policy module to make decisions, and the robot's trajectory will be recorded in the shared experience replay buffer. The basic network structural unit of the PDDPG algorithm is the same as the enhanced DDPG. But, achieves the goal of the entire multi-robot system by sharing

experience storage data and navigation policy. Only the input size and the state of the robot observed by lidar is modified. The reward function of the single robot navigation task to a multi-robot version is adjusted. The proximity reward of each robot is summarized as a set of proximity rewards. When a robot collides, the environment is reset and the plot ends. Considering training a good reinforcement learning agent, finishing the course at an appropriate stage can effectively speed up agent training. After that, the robot is trained to complete the collaborative navigation task according to the training policy module.

Implementation of DDPG in Mapless Navigation:

Bouhamed researched to use self-training drones as flying mobile units to reach mobile or static targets spatially distributed in a given three-dimensional urban area [28]. Using DDPG with continuous action space drones are trained through obstacles to achieve their assigned objectives thus solving the problem of trained path planning to improve UAV navigation.

Likewise, Chen in “collaborative navigation without mesh of multi-robot systems” proposed method that improves the classic DDPG to solve the task of navigation of single robot [5]. Using 2D lidar sensors along with CNN, the group of robots can perform training construction tasks and cooperative training navigation tasks. The classic DDPG has been improved to directly solve the problem of mapless navigation to the input of raw lidar data. A PDDPG is implemented to complete the training of multi-robot training and the maintenance of training during collaborative navigation.

4.4 Proximal Policy Optimization (PPO)

The PPO algorithm was launched by the Open-AI team in 2017 and has quickly become one of the most popular RL methods for altering Deep-Q Learning methods. It involves collecting a small amount of experience interacting with the environment and using this experience to update decision policy. Once the policy is updated using the batch, the experience is skipped and the updated batch is collected using the newly updated policy. This is why it is a "policy-based learning" method, where the collected experience samples are only useful for updating the current policy once. The main goal of the PPO is to ensure that new policy updates do not change much from previous policy. In other words, PPO makes a balance between complexity of sample and its implementation by computing and updating at each step by minimizing the cost function ensuring that the deviation from latest policy is relatively less.

The Trust-Region method in PPO (TRPO) is a feature that defines a region during the current iteration, in which the model is trusted as an appropriate representation of the objective function, then

the step is selected as the approximate minimum value of the model in this region. The function is given as:

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)]$$

Where, θ is the policy parameter, $\hat{E}_t$ denotes the expectations over timestep, rt is the ratio of the probability under new and old policies, $\hat{A}_t$ is the estimated advantage at time t and ε is hyperparameter either 0.1 or 0.2.

In short, in the optimization process, after determining the direction of the gradient limits step size to the confidence region so that the local estimate of the gradient remains confidence. In TRPO, the average difference between the old policy and the updated policy is used as a measure of the confidence zone. The objective substitution function is limited and maximized by the size of the policy update.

Implementation of PPO in mapless navigation:

Fan in "Crowd Move: seamless autonomous navigation in crowded scenes" illustrates that using a robust policy gradient algorithm the optimization of 3M training framework (i.e. multi-robot, multi-stage and multi-stage) is possible extending the state-of-the-art reinforcement learning algorithm (PPO) to parallel multi-robot training framework [29]. This approach allows different types of mobile platforms to navigate safely in complex and highly dynamic environments (such as pedestrian overcrowding). This policy has received experience training from all robots collected in parallelly with relatively low cost of change in policies.

4.5 Experiment on Turtlebot

This experiment is a basic approach to survey DRL approaches for a simulated mapless navigating environment. The main gist of this experiment is to evaluate the different types of DRL methods that can be implemented for self-navigation.

Experimental Setup

Turtlebot as an agent: TurtleBot is a classic robot which can move around the circuit, the circuit is a complicated labyrinth, with very contrasting colors between the floor and the obstacle.

Sensor: Lidar is used as an input to train the robots to navigate the environment. The goal is to steer the robot without colliding with the obstacle.

Simulation platform: Environment is simulated and controlled by the gazebo, ROS and Gym. (https://github.com/erlerobot/gymgazebo).

Platform specifications: The experiment equipment is executed in a laptop with 8 GB of RAM, a multi-core processor and an Ubuntu 16.04 system. The version of the experimental software is: Python3.7, Pytorch, Gazebo7.

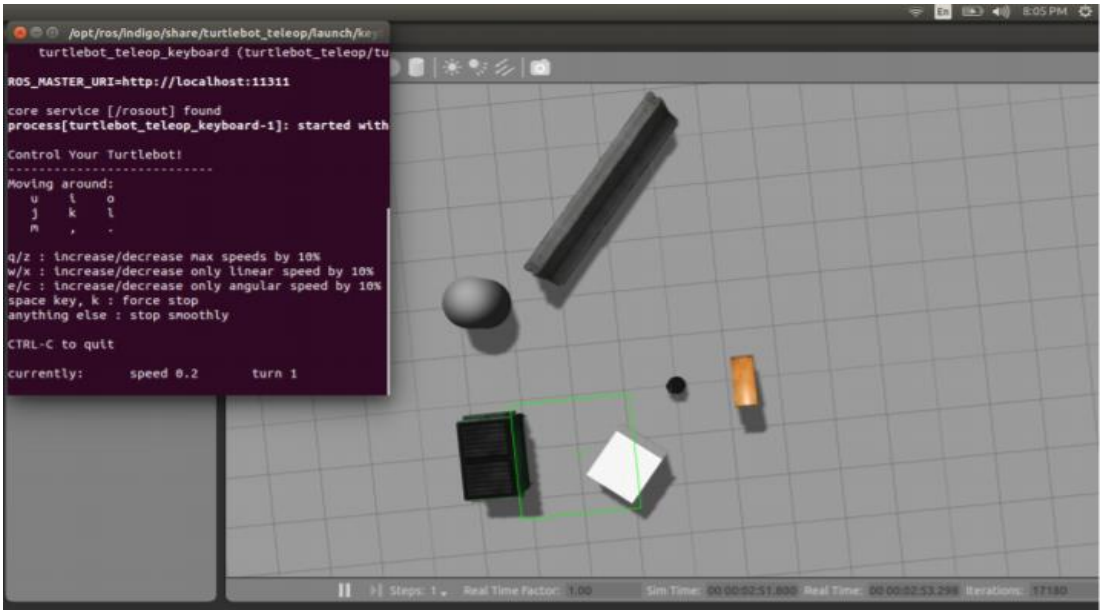

Figure 4.6 Turtlebot simulation in Gazebo

4.5.1 Parameters for the Experiment

Training configuration:

Using different random seeds to run each algorithm five times. Then taking the average of these five iterations. Finally, training TurtleBot in 200,000 steps.

Reward function model:

The infinite horizontal discount model E ($P \infty t = 0\gamma_t r_t$), $0 < \gamma < 1$ is used for the performance analysis. In DQN, the discounted Q value function (action value) is used to calculate the Q target and benefit functions. In the discreet version, it has three discreet actions (front, left and right). And a five-dimensional viewing space, all of which represent the scanning range r in the range area. The reward is defined as: each time the robot moves left or right, the reward is once, the forward movement the reward is five. In the continuous version, it has a one-dimensional action space with a limit of [-0.21, 0.21], which represents the angular acceleration of rotation which controls the angular speed. Reward is defined as a reward function based on the value of the action. It can be understood as a normal distribution, when it is close to the middle of the action value, the reward will be the highest. By randomly initializing the

robot positions at the four corners of the labyrinth in the initial state. In 10 consecutive tries, if the plot reward is greater than or equal to 600, the task is considered solved.

Exploration policy:

For DQN, main exploration method; The Epsilon Greedy policy is used. For DDPG, a random process function is used as mentioned above in the state-of-the-art, which implements A2C method. For PPO, random samples from a defined normal distribution is explored by drawing.

For the functional approximation of evaluation algorithm, a conventional deep neural network architecture is used. There are three fully connected hidden layers, which contain 24 neurons, and each layer is activated by a corrected linear unit. This architecture for DQN, A2C, DDPG and PPO with different inputs and outputs.

Table 4.1 Methods based on parameters constraints

Parameters / Methods	DQN	DDPG	PPO
Q target	Target Model Update Interval	Target Model Update Interval	Continuous
Model configuration	Optimizer, Lost Function	Actor/Critic Network learning rate	Actor/Critic Network learning rate
Exploring strategy	Epsilon Greedy Policy	Gaussian Noise Process	Gaussian Distribution
Reward Function	Infinite discount model, avg reward model	Infinite discount model, avg reward model	Infinite discount model, avg reward model
Training Configuration	Free steps before training, training interval	Free steps before training, training interval	Loop update steps, Batch size
Experience Replay	Max memory size, batch size	Max memory size, batch size	KL penalty method

Table 4.2 Values given to the parameters

Methods	Parameters	Value given to Turtlebot

DQN	Learning Rate	0.001
	Memory size	10000
	Warm Up Steps	2000
	Batch Size	64
	Target Net Update	0.01
	Train Interval	1
	Discount Rate	0.99
	Exploration	Epsilon= 0.2
DDPG	Actor Learning	0.0001
	Critic Learning	0.001
	Memory size	50000
	Warm Up Steps	1000
	Batch Size	32
	Target Net Update	0.001
	Train Interval	1
	Discount Rate	0.99
	Exploration	Theta= 0.15, Sigma= 0.3
PPO	Actor Learning	0.00001
	Critic Learning	0.0001
	Batch Size	32
	Discount Rate	0.99
	Loop Update Steps	(10,1)
	Exploration	Random samples
	Constrain Methods	Clipped surrogate, epsilon=0.2
	Num Workers	2

4.5.2 Result and Analysis

The Turtlebot is trained in the above-mentioned parameters implementing these state-of-the-art (DQN, DDPG and PPO).

As this experiment was done to check which of these methods are more reliable and robust.

Comparing training time

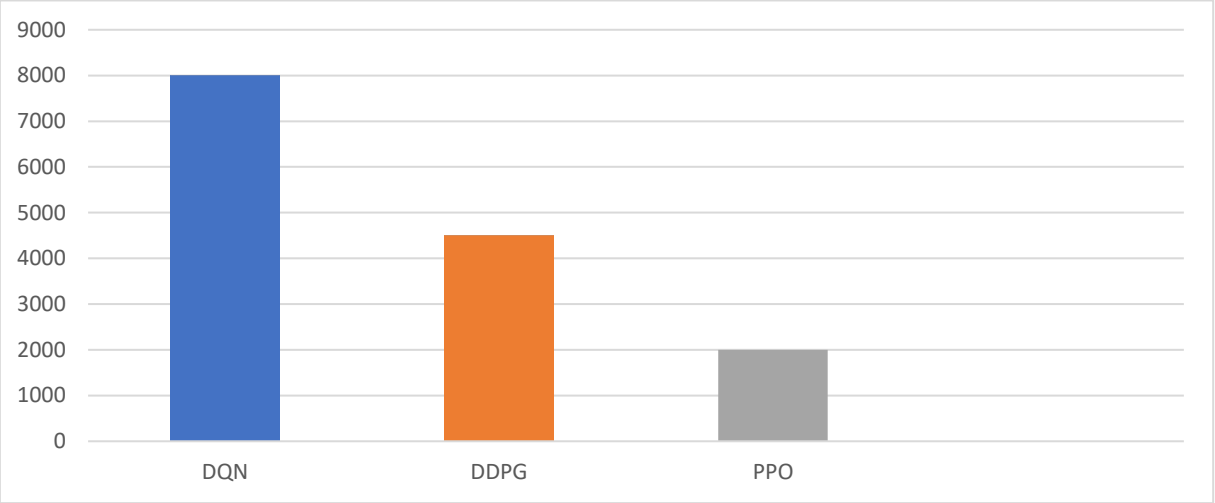

Figure 4.7 Turtlebot Training Time

When the experiment completed its all episodes, the average training time was calculated to compare which of the methods have less execution time. From above chart, PPO took least duration of time to complete the whole training compared to DDPG and DQN. DQN took the maximum time to complete the training. Hence, proving that 'policy-gradient' based algorithms can execute faster than only 'value-based algorithms' with optimal loss function.

Comparing Rewards per episode

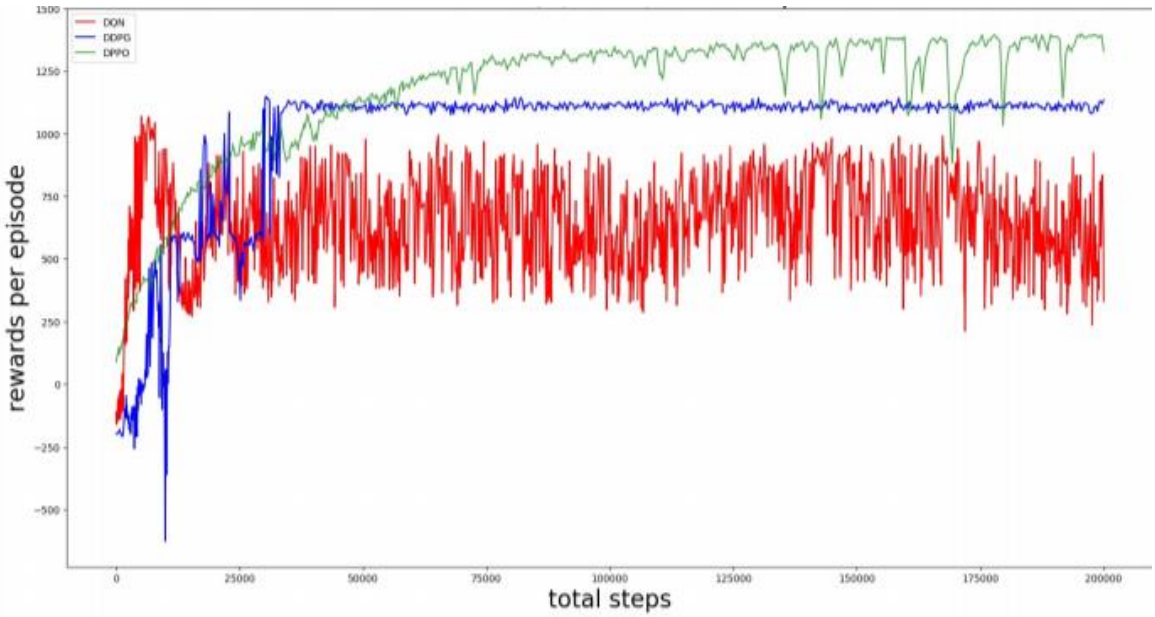

Figure 4.8 Rewards per episode in Turtlebot

For TurtleBot in this experiment, DDPG and PPO have higher performance, and PPO has higher score and shorter training time. DDPG is durable and has higher sampling efficiency. Overall, PPO performed better than DQN and DDPG in terms of response time complexity, with a higher average score.

4.6 Overall Analysis

From above definitions of different state-of-the-art (from 4.1 to 4.4) and implementation of all these DRL methods by researchers one can state that mapless navigation is not a virtual task rather a challenging yet accomplishing one. The application of these algorithms is not limited to Atari or AlphaGo games, but is now implemented in autonomous navigation robots and autonomous vehicles. Each item has the same basic DRL method, using different DRL states or techniques in its specific environment settings, and the results are also different as shown in Table 4.3.

Table 4.3 Functionality of State-of-the-Art DRL

Algorithms	Model	Policy	Action-Space	Operator
Q-learning	Model-Free	Off-Policy	Discrete	Q-Value
SARSA	Model-Free	On-Policy	Discrete	Q-Value

DQN	Model-Free	Off-Policy	Continuous	Q-Value
DDPG	Model-Free	Off-Policy	Continuous	Q-Value
PPO	Model-Free	Off-Policy	Continuous	Advantage

Based on the research works and the experiment in 4.5 the we can point out that each of these methods has unique but interdependent functions. The models of these methods have different impact on the training speed, action speed, memory used, network architecture and overall robustness in navigation period. The differences and similarities of these basis methods are furthermore analyzed in the table below.

Table 4.4 Methods Analysis

	DQN	A2C/A3C/PPO	DDPG
Classification	Value based	Policy based	Combination of value and policy based
Training Speed	Slowest	Fastest	Slower than A2C, faster than DQN
Action Speed	Discrete	Both Discrete and Continuous	Only continuous
Memory Consumed	Large replay memory though having one target net	Larger memory required as it has many workers net	Larger memory required as it has two nets (μ and Q)
Parallelization Required	No	Yes, workers network parallelly to update master net	Does not support parallelization
Backpropagation	Happens	Happens	Happens
Sub-Network	Only basic Deep Neural Nets	Actor is π (stochastic) and critic is V (state-value-function). Actors and	Actor μ is deterministic and actor Q is action value

		the workers which run in parallel with one critic i.e. master net	
Weights used	Weights on the main network copied to target nets	Weights of master net are copied to worker nets	Soft nets update of μ and Q separately

An overall outlook of above experiments and research proves that value-based methods such as DQN are slower than the policy-based methods. Likewise, DQN has less functionality (no continuous actions, and does not fully allow the concept of parallelization. As the research in 4.1 DQN and DDQN, the DDQN can improve learning speed and improve performance in many tasks. Therefore, using DQN, the navigation system can effectively steer the robot to the target position in an environment with regional changes. Yet, it seemed to have less memory space since it only has one master net with a simple neural network. Therefore, taking an account for learning algorithms for the continuous action space the Actor-Critic methods are more reliable to use. These Actor-Critic methods like A2C/PPO/DDPG seems to advance in being faster with its policy-based algorithms and also allowing parallelization for training multi-robots. So far from the research, Actor-Critic methods are emerging to be best for training robots. The mapless navigation implementation in 4.2.1 shows that A3C method is robust and consistent in terms of performance because the method is not dependent and the environment map is suitable for situations where many other methods are not applicable. However, from 4.2.2 the MK-A3C algorithm improves the original A3C algorithm by adding a storage mechanism and domain knowledge to the neural nets. By introducing storage mechanisms and domain knowledge into its DRL framework, MK-A3C can solve navigation problems with moving obstacles in unknown dynamic environments.

In short, the policy gradient methods like (A3c, DDPG, PPO) can directly learn the optimization policy without calculating the reward for each action. Therefore, the PG method seems to work well in an environment with unlimited movement. These algorithms have proven themselves in sporting environments. The operating principle of the value iteration method is to estimate the reward of each action and act with the highest reward. Compared to the PG method, the value iteration method has a higher convergence speed, but it is not adapted to the movement environment because it cannot manage infinite movements.

The state-of-the-art of DRL is the heart of unmanned mapless navigation. But, from above research and experiment it is manifested that there are several parameters excluding DRL that affect the results in navigation. Basically, these parameters are to be analyzed equally for a fast, robust, cost effective and optimal navigation. These parameters are analyzed below.

Network Architecture

Today, due to its computing power, deep neural networks can be used to learn effective policy directly from inputs from large sensors. Convolutional neural networks (CNNs) are successfully used in the perception of traffic scenes and can recognize objects, such as pedestrians or vehicles, lane detection and distance estimation. Currently, due to the lack of robustness, the implementation of autonomous driving is not based on computer vision technology. One of the most difficult challenges is to compress the input image into a representative characteristic vector. Currently, there are two ways to solve this problem: the "mediated perception method" and the "behavioral reflection method". The intermediate perception method can analyze the entire traffic scene represented as the input image and analyze the scene involving several sub-scenes components for the recognition of specific objects, such as traffic signs, lanes, pedestrians, vehicles. Using the recognition results, a coherent representation of the world can be created and an engine based on machine learning (ML) will take into account all the information for decision making. The task of understanding the scenes adds more complexity to the neural network (NN) algorithm and causes performance problems. In order to eliminate this gap, instead of detecting the bounding box of the object, a distance prediction can be made on the object.

Deep CNN is used to approximate the action value function with high dimension images as states. Although the benefits of using deep neural networks over traditional neural networks have not been clearly defined, DNN was chosen because DNNs can use graphing units for faster calculations and have certain algorithmic improvements, such as abandonments, regularization and parametric sharing for example in convolutional layers.

Training Program

There are two different environments for training our mobile robots: Simulation and Real-world application. In the context of applications, reinforcement learning provides a framework for the design of complex and difficult-to-conceive behaviors. The challenge is to establish a simple environment in which the machine learning technique can be checked and then applied to real scenarios. DeepMind has opened its own internally developed 3D environment. TensorFlow and Keras5 TensorFlow is an open

source software library for high performance digital computing. Its flexible architecture allows easy deployment of IT between different platforms (CPU, GPU, TPU) and from the office to the server cluster via mobile devices and peripherals. It was originally developed by researchers and engineers from the Google Brain team organized by Google AI, it strongly supports machine learning and deep learning, and the flexible digital computing core has been used in many other scientific fields. There are also many extension packages and software based on Open AI Gym, which can simulate various applications, such as "gym-gazebo". Gym-Gazebo2 is for complex engineering applications, such as robotics, due to the high cost of errors, it is difficult to train reinforcement learning methods using real physical systems. Therefore, simulating complex virtual behaviors and then applying them in real scenarios will become an alternative solution.

Likewise, from simulation to transferring to real-environment has been of the biggest challenges in DRL. Ideally, the simulation will allow the environment and the RL method to understand the behavior and then transfer it to the real application. It depends on the factors such as, computational power, cost of hardware systems, design of complex neural network and the state-of-the-art DRL method used. But experimental results suggest that using methods like Advantage Actor-Critic and DDPG it is more applicable to transfer the simulated environment into real life implementation as these methods uses two forms (Policy based and Value Based) technique along with one output's as input of another.

Reward Function

The reward function is at the heart of reinforcement learning, as it can specify and guide the required behavior. The goal of reinforcement learning algorithms is to maximize the accumulated long-term rewards. Appropriate reward functions can speed up the learning process. It is not easy to define the appropriate reward function for the corresponding application. The first question is how to give a certain amount of reward or punishment. Next, how to define the reward function. The learner must observe the variance of the reward signal in order to be able to improve the policy: if the same reward is always received, it is impossible to determine which policy is better or closer to the optimal value proposed a method to design a reward function which uses the implicit knowledge of the domain. The design of the reward function can also affect the speed of convergence. If the reward is too fast to complete the task, the controller may fall into a local optimal state. The opposite: the controller can slow down the learning process and the worst-case scenario will never reach the end of the task.

Exploration Policy

From the point of view of possible Q-learning exploration / development techniques, we focus on non-directional policy: softmax, ε-greedy. The results show that although it is difficult to adjust its parameters, the performance of softmax or Boltzmann exploration is superior to other policy. The simplest technique. The setting is ε greedy, but ε greed is the worst of all exploration policy. The results also show that learning Q with different exploration policy has serious problems in finding the best policy. An exploration policy for Q learning can be created, which can always find the optimal target state for mobile robots without distracting itself by absorbing the sub-optimal target states. In addition, the effect of the reward function with a higher greed function can be further explored in the dynamic placement of targets and agents.

Therefore, from these parameters explained above, mapless navigation not only rely on the best DRL algorithm but equally seems to be depending on the best sensors used, neural network and simulation simplicity and the computational power of the hardware's used for the best and safe result. Hence, mapless navigation seems to be challenging to choose the best with so many factors it pivots on various aspects of methodology. Hence, one must consider all of these parameters to accomplish navigating in dynamic environment.

Chapter 5 Conclusion

This thesis has provided an overview of reinforcement learning algorithms along with deep neural networks which are used to navigate any agent in a dynamic environment. Different experiments on mapless navigation on real world thus have been proved to be more efficient using DRL algorithmics. Even with training on simple scenario like 'The Turtlebot experiment' different methods can handle complex scenarios robustly, resulting well trained robot at the end that can navigate in obstacles. From all above experimental analysis it is clear that for continuous control tasks, simply discretizing the action spaces and using value-based methods such as DQN is a feasible way. Likewise, forms of Advantage Actor-Critic methods have been the most successful in terms of the action space complexity which takes advantage of prior researching experience and they are capable to select actions in a continuous domain with a temporal-difference learning value function and optimize it by interacting with the environment using the current policy. Moreover, A2C when used with CNN in navigation has demonstrated the maximum optimality of the algorithm it uses. Similarly, by taking the raw 2D lidar sensor data and the relative target positions as inputs, the proposed Parallel Deep Deterministic Policy Gradient (PDDPG) algorithm could directly control the multi- robots to construct the formation and maintain it during navigation. However, the Q Learning algorithm is considered as the foundation of all methods. Stating that the general DRL frameworks (state-of-the-art) can be widely utilized in many applications with different optimization objectives like self-navigating. Hence, the 'trial-and error' technique to train a given agent in any dynamic environment shows that a robot can learn and control policy as similar with human decisions.

5.1 Future work

Agents that can learn the fundamentals of the world through observation and action are growing steadily. However, until robots or autonomous cars can completely cross complex environments and cities, the challenges remain.

Based on the analysis of this thesis, more complex environments, obstacles moving irregularly which can really simulate the real world can be added into future works. In addition, it can further improve the reward function and strategic polices by allowing agents to fully explore. Therefore, providing agents with more knowledge to make definite decisions from its experience and further improve the effectiveness of training. Likewise, input into any neural network can be improved by making precise observations around the training such as distance and angle of obstacles, resolution of the image captured

by the sensor. And constraints such as: a long training time, a slow learning speed, rewarding the algorithm, the variance of biases and weights, are opportunities for improvement.

These challenges open up a wide field of research for new ideas and breakthroughs. Someday, it will result in human robots with superhuman power, but smarter and more efficient than ordinary people.

References

[1] H. Li, T. Wei, A. Ren, Q. Zhu, and Y. Wang, "Deep reinforcement learning: Framework, applications, and embedded implementations," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 847–854.

[2] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[3] Y. Wang, H. He, C. Wen, and X. Tan, "Truly proximal policy optimization," *ArXiv Prepr. ArXiv190307940*, 2019.

[4] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 31–36.

[5] W. Chen, S. Zhou, Z. Pan, H. Zheng, and Y. Liu, "Mapless Collaborative Navigation for a Multi-Robot System Based on the Deep Reinforcement Learning," *Appl. Sci.*, vol. 9, no. 20, p. 4198, 2019.

[6] M. Wu, Y. Gao, A. Jung, Q. Zhang, and S. Du, "The Actor-Dueling-Critic Method for Reinforcement Learning," *Sensors*, vol. 19, no. 7, p. 1547, 2019.

[7] S. S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: an overview," in *Proceedings of SAI Intelligent Systems Conference*, 2016, pp. 426–440.

[8] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electron. Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[9] M. S. Shim and P. Li, "Biologically inspired reinforcement learning for mobile robot collision avoidance," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 3098–3105.

[10] N. Casas, "Deep deterministic policy gradient for urban traffic light control," *ArXiv Prepr. ArXiv170309035*, 2017.

[11] T. Ort, L. Paull, and D. Rus, "Autonomous vehicle navigation in rural environments without detailed prior maps," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 2040–2047.

[12] Q. Tan, T. Fan, J. Pan, and D. Manocha, "DeepMNavigate: Deep Reinforced Multi-Robot Navigation Unifying Local & Global Collision Avoidance," *ArXiv Prepr. ArXiv191009441*, 2019.

[13] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *ArXiv Prepr. ArXiv181112560*, 2018.

[14] S. Levy, W. Xiong, E. Belding, and W. Y. Wang, "SafeRoute: Learning to Navigate Streets Safely in an Urban Environment," *ArXiv Prepr. ArXiv181101147*, 2018.

[15] C. Qiu, Y. Hu, Y. Chen, and B. Zeng, "Deep Deterministic Policy Gradient (DDPG)-Based Energy Harvesting Wireless Communications," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8577–8588, 2019.

[16] P. Gaussier, C. Joulain, S. Zrehen, J.-P. Banquet, and A. Revel, "Visual navigation in an open environment without map," in *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robot and Systems. Innovative Robotics for Real-World Applications. IROS'97*, 1997, vol. 2, pp. 545–550.

[17] U. Farooq, M. Amar, M. U. Asad, A. Hanif, and S. O. Saleh, "Design and implementation of neural network based controller for mobile robot navigation in unknown environments," *Int. J. Comput. Electr. Eng.*, vol. 6, no. 2, p. 83, 2014.

[18] Y. Guo, W. Wang, and X. Chen, "FreeNavi: landmark-based mapless indoor navigation based on WiFi fingerprints," in *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, 2017, pp. 1–5.
[19] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.
[20] Y. G. Rocha and T.-Y. Kuc, "Mental Simulation for Autonomous Learning and Planning Based on Triplet Ontological Semantic Model," 2019.
[21] P. Mirowski *et al.*, "Learning to navigate in cities without a map," in *Advances in Neural Information Processing Systems*, 2018, pp. 2419–2430.
[22] Y. Zhu *et al.*, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*, 2017, pp. 3357–3364.
[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *ArXiv Prepr. ArXiv170706347*, 2017.
[24] X. Lei, Z. Zhang, and P. Dong, "Dynamic path planning of unknown environment based on deep reinforcement learning," *J. Robot.*, vol. 2018, 2018.
[25] M. Dobrevski and D. Skočaj, "Map-less goal-driven navigation based on reinforcement learning," p. 8.
[26] K. Zhang, F. Niroui, M. Ficocelli, and G. Nejat, "Robot navigation of environments with unknown rough terrain using deep reinforcement learning," in *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2018, pp. 1–7.
[27] J. Zeng, R. Ju, L. Qin, Y. Hu, Q. Yin, and C. Hu, "Navigation in Unknown Dynamic Environments Based on Deep Reinforcement Learning," *Sensors*, vol. 19, no. 18, p. 3837, 2019.
[28] O. Bouhamed, H. Ghazzai, H. Besbes, and Y. Massoud, "Autonomous UAV Navigation: A DDPG-based Deep Reinforcement Learning Approach," *ArXiv Prepr. ArXiv200310923*, 2020.
[29] T. Fan, X. Cheng, J. Pan, D. Manocha, and R. Yang, "Crowdmove: Autonomous mapless navigation in crowded scenarios," *ArXiv Prepr. ArXiv180707870*, 2018.

Acknowledgement

This thesis wouldn't have been possible without the help of many people in so many ways. Firstly, I would like to thank my supervisor Li Bo Han for giving me such a golden opportunity to get this wonderful thesis topic. Without his constant guidance, this project wouldn't have been possible. Without the moral support of my family and friends, I wouldn't have completed this project. So, I would also like to thank my friends and family. Finally, I would also like to appreciate College of International Education and College of Software Engineering and Management, in Nanjing University of Aeronautics and Astronautics for providing wonderful research-based topic for thesis.