TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Yuzhou Liu 221144 IAFM

Using Reinforcement Learning for Multiple Way-points Path Planning of Single Mobile Robot in the Dynamic Obstacle Environment

Master's thesis

Supervisor: Andreas Bresser
Dipl.-Inf

Laura Piho
Ph. D

Pascual Campoy
Prof.

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Yuzhou Liu 221144 IAFM

Stiimulõppe Rakendamine Mobiilse Roboti Mitmepunktilise Teekonna Planeerimiseks Dünaamilises Takistuskeskkonnas

Magistritöö

Juhendaja: Andreas Bresser
Dipl.-Inf

Laura Piho
Ph. D

Pascual Campoy
Prof.

Tallinn 2022

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Yuzhou Liu

08.08.2022

Abstract

Nowadays, mobile robots are applied to increasingly complex scenarios. Whether it is a closed scenario, such as the transportation of goods or medical supplies in large supermarkets, factories and hospitals, or an open scenario, such as the delivery of unmanned vehicle delivery service in urban roads, all of them put forward higher requirements for the multi-destination(waypoint) path planning performance of the mobile robot. The purpose of this thesis is to design and prove a multi-waypoint path planning algorithm for the mobile robot, which combines reinforcement learning and heuristic search algorithm. so that the mobile robot can carry out continuous multi-waypoint path planning in a dynamic environment with multiple moving obstacles.

The algorithm first selects the next waypoint through the Q-learning algorithm, and then performs path planning between the current position of the robot and the next waypoint through the Anytime Repairing A* (ARA*) algorithm as the global planner. At the same time, the Two-Step Vector Field Histogram with Look-Ahead Verification with Recovery method (2S-VFH*-R) algorithm is used as the local planner to plan the avoidance or escape path when the mobile robot encounters a moving obstacle.

This thesis builds a simple Q-learning problem model to solve the Traveling Salesman Problem (TSP). The performance of Q-learning algorithm is evaluated, which proves the feasibility of using Q-learning combined with heuristic search algorithm to solve the multi waypoint path planning problem in static simulation environment. Finally, the thesis analyses the performance of the multi-waypoint path planning algorithm and draws a conclusion that in the unknown dynamic environment, the algorithm has better performance than the traditional greedy method (local optimal solution) with the minimum overall time taken as the optimization objective.

This thesis is written in English and is 48 pages long, including 7 chapters, 21 figures and 3 tables.

Annotatsioon

Stiimulõppe Rakendamine Mobiilse Roboti Mitmepunktilise Teekonna Planeerimiseks Dünaamilises Takistuskeskkonnas

Tänapäeval rakendatakse mobiilseid roboteid üha keerukamate stsenaariumide jaoks. Olgu tegemist suletud stsenaariumiga, nagu kaupade või meditsiinitarvete transport suurtes supermarketites, tehastes ja haiglates, või avatud stsenaariumiga, nagu mehitamata sõidukite kohaletoimetamise teenuse osutamine linnateedel, kõik need esitavad kõrgemad nõuded mobiilse roboti mitme sihtkoha (teekonnapunkti) tee planeerimise jõudlus. Käesoleva lõputöö eesmärk on kavandada ja tõestada mobiilse roboti jaoks mitme teekonnapunktiga teeplaneerimise algoritm, mis ühendab endas armeerimisõppe ja heuristilise otsingu algoritmi. et mobiilne robot saaks dünaamilises keskkonnas, kus on palju liikuvaid takistusi, teostada pidevat mitme teekonnapunkti tee planeerimist.

Algoritm valib esmalt järgmise teekonnapunkti läbi Q-õppealgoritmi ja seejärel planeerib roboti praeguse asukoha ja järgmise teekonnapunkti vahel teed globaalse planeerijana läbi Anytime Repairing A* (ARA*) algoritmi. Samal ajal kasutatakse kaheastmelist vektorvälja histogrammi koos taastamismeetodiga ettevaatava kontrolliga (2S-VFH*-R) algoritmi kohaliku planeerijana vältimis- või põgenemistee kavandamisel, kui mobiilne robot satub liikuva takistusega. .

See lõputöö koostab lihtsa Q-õppe probleemimudeli, mis lahendab reisiva müügimehe probleemi (TSP). Hinnatakse Q-õppe algoritmi jõudlust, mis tõestab Q-õppe kasutamise teostatavust koos heuristilise otsingu algoritmiga mitme teekonnapunkti tee planeerimise probleemi lahendamiseks staatilises simulatsioonikeskkonnas. Lõpuks analüüsib lõputöö mitme teekonnapunktiga teeplaneerimise algoritmi jõudlust ja teeb järelduse, et tundmatus dünaamilises keskkonnas on algoritm parema jõudlusega kui traditsiooniline ahne meetod (lokaalne optimaalne lahendus), mille optimeerimiseks kulub minimaalne koguaeg. objektiivne.

Lõputöö on kirjutatud Inglise keeles ning sisaldab teksti 48 leheküljel, 7 peatükki, 21 joonist, 3 tabelit.

List of abbreviations and terms

RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
PPO	Proximal Policy Optimization
DQN	Deep Q-Learning
MPNet	Motion Planning Network
TSP	Traveling Salesman Problem
NPC	Non-deterministic Polynomial Complete problem
VRP	Vehicle Routing Problem
ARA*	Anytime Repairing A*
AD*	Anytime Dynamic A*
IMUs	Inertial Measurement Units
AMCL	Adaptive Monte Carlo Localization
DWA	Dynamic Window Approach
DWB	Dynamic Window B
VFH*	Vector Field Histogram with Look-Ahead Verification
2S-VFH*-R	Two-Step VFH* with Recovery method
MDP	Markov Decision Process
TD	Temporal Difference
ROS	Robot Operating System
XML	eXtensible Markup Language
URDF	Unified Robot Description Format
SDF	Simulation Description Format

Table of contents

List of figures

List of tables

1 Introduction

This chapter provides a comprehensive introduction to the topic of the dissertation. In section 1.1, the motivation for using a combination of reinforcement learning and heuristic search algorithms in the multi-waypoint path planning process of mobile robots is presented. In Section 1.2, the state-of-the-art research results and challenges brought by the application of reinforcement learning to solve the multi-waypoint path planning of mobile robots are introduced, and the research gaps in this field are discussed in order to indicate the value and potential contribution of the selected topic of the thesis. Section 1.3 introduces the sensor parameter configuration, localization algorithm, and navigation algorithm of the MiR100 robot.

1.1 Motivation

In recent years, driven by multiple factors such as increasingly personalized consumer demand, shortage of work force, and the continuous maturity of the new generation of information technology, intelligent logistics and intelligent manufacturing have become important means for enterprises to reduce costs, increase efficiency and improve quality. Mobile robots have begun to appear, and the market demand continues to rise. In the existing industrial applications, mobile robots have been running in a very satisfactory way in static unmanned environments, such as warehouses and factories.

Mobile robots are applied to increasingly complex scenarios. Whether it is a closed scenario, such as the transportation of goods or medical supplies in large supermarkets, factories and hospitals, or an open scenario, such as the delivery of unmanned vehicle delivery service in urban roads, all of them put forward higher requirements for the mobile robot to complete the task of transporting goods to multiple destinations under the constraints of multiple factors, such as unknown obstacles, traffic congestion, time, distance, cost and energy consumption. This is also known as the problem of multi-waypoint path planning in the dynamic environment.

The multi-waypoint path planning problem is widespread in many areas, such as in a factory production line where spare parts need to be transported to multiple areas in an unpredictable situation full of equipment and moving workers, or in a large hospital where robots need to traverse crowded corridors to deliver medicines to multiple wards.

This thesis attempts to explore a potential solution that combines classic heuristics search algorithms with Reinforcement Learning (RL) algorithms to solve the multiple waypoints path planning problem in the dynamic environment. More precisely, the multi-waypoint path planning problem is defined as a Travelling Salesman Problem (TSP), and then the TSP problem is transformed to a RL problem. As a combination problem, TSP was proved to have Non-deterministic Polynomial Complete problem (NPC). A brief description of the problem is: Suppose a travelling salesman wants to visit multiple cities, he need find the shortest path that can pass through each city once.

In this thesis, a mobile robot within a supermarket environment is considered as a use case. The mobile robot needs to deliver goods to several waypoints with the shortest time in a space full of unknown dynamic/static obstacles (such as people, carts, and goods accidentally dropped on the aisle).

Existing state-of-the-art research focuses on end-to-end application of reinforcement learning to solve the problem of multi-waypoint navigation, which means that the two steps in the traditional TSP solution (1- Calculate the shortest path between every two waypoints; 2 - Find the order of tour visits) are replaced by a semi-blind manual process that lacks the interpret-ability of the decision-making process. It would be computationally expensive for reinforcement learning to simultaneously solve multiple tasks of navigation, obstacle avoidance, and multi-waypoint sequencing. Most of research is based on the 2D simulation environment, which cannot completely draw on their research results for the 3D environment with complex or even unexpected physical influence factors.

Most of the research focuses on finding the shortest path through all waypoints in static environment rather than the minimum time. Some researchers try to improve path planning or navigation problems in dynamic environments by setting multi-objective (shortest time and shortest path) reinforcement learning methods. However, designing a

"super-powerful" reward function that balances multiple objectives would be challenging enough and lack interpretability.

The goal here is not to beat the current state-of-the-art solvers of TSP problem, in fact, won't even come close to their solutions. Instead, the training environment with some extreme cases will be considered to test the performance of the RL algorithm: a Q-learning method combined with the ARA* algorithm will be built, and the robot will find the optimal path without any prior knowledge of the TSP structure. The model is guided by only one "signal", which is the observed cumulative "reward" -- in this case, the overall time it takes to reach all the waypoints.

1.2 Related Work

With the increasing popularity of service robots, sweeping robots and automated guided vehicles, path planning, as the core of mobile robot technology, has become a research hotspot. In the face of complex environment, mobile robots need autonomous learning to complete the task of path planning. With the popularity of deep reinforcement learning (DRL) research and its excellent performance in some fields in recent years, more and more scholars and research teams apply deep reinforcement learning to the field of mobile robot path planning.

In the existing research results, many mature algorithms have been accumulated in the field of single-waypoint path planning between waypoints for mobile robots, including particle swarm optimization algorithm [1], genetic algorithm [2], fuzzy logic algorithm [3], artificial potential field method [4] and so on.

The traditional multi-destination (waypoint) path planning is an integration and improvement of single-destination (waypoint) path planning. In the existing research results, the idea of transforming the choice of target location into the Traveling Salesman Problem (TSP) appears. The particle swarm optimization path planning proposed by this method [5] combines the fast convergence characteristics of the ant colony algorithm, and optimizes it through the ant colony algorithm. As one of the most widely used algorithms, the multi-destination path planning algorithm [6] could obtain optimal path planning because it combines global statics and local dynamics. It enables the mobile robot to avoid static obstacles in time and move to the destination accurately.

However, the biggest disadvantage of traditional multi-destination path planning algorithms is that it is easier to fall into local optimum. Moreover, in the global path planning, the existing algorithms have the problems of low utilization rate of environmental information, complex calculation structure and low sorting efficiency.

The publication of [7] shows a unique approach of implementing multiple destination global path planning strategy based on improved Q-learning algorithm without using more complex DRL algorithm. However, they only considered a 2D lattice map containing known static obstacles as a simulation environment for the training of algorithm. This makes the model lack of realistic environmental information that can be used as important feedback for algorithm optimization.

Pengyuan Wei [8] proposed a robot motion planning algorithm based on deep learning, which enables the robot to perform continuous multi-target motion planning in 2D and 3D static environments. The algorithm first uses the Deep Q-Learning algorithm to help the robot select the next target point, and then uses the Motion Planning Network (MPNet) to allow the robot to navigate from the current position to the next target waypoint. Here the author focuses on the shortest total distance after the robot reaches all the waypoints and returns to the starting point. The task requirement does not force the RL-agent to reach all the waypoints but uses DQN to make the robot realize how many waypoints it needs to reach in the task. In addition, the authors also did not test the performance of the algorithm in a dynamic obstacle environment, which is what this thesis will take into account.

The decision of this thesis to use a traditional heuristic search algorithm to deal with the path planning task between waypoints is inspired by Güldenring's thesis [9]. She mentioned the idea of combining traditional planner with RL-agent for navigation in the chapter of future work. In her thesis, the reinforcement learning-based navigation algorithm is trained and tested in both static and dynamic environments, in which the local planning is realized with the state-of-the-art Deep Reinforcement Learning approach -- Proximal Policy Optimization (PPO). Although the author only uses DRL to solve single-waypoint path planning, his setup details and empirical skills for dynamic environments provide a supportive reference for this thesis.

In addition to referencing research results in academia, this thesis also get a lot of inspiration from open-source projects in industry. First, Costa shared his ideas and experience in solving the traveling salesman problem with reinforcement learning in his blog [10]. This provides confidence in the decision of this thesis to transform the multi-waypoint path planning into the TSP problem, as well as solve the TSP problem with Q-learning. In addition, the thesis also references his method of setting a "traffic zone" area in a simulated environment to increase the time the robot spends passing through. His open-source code also provides an important reference for the algorithm implementation of this thesis.

Herzen [11] proposes a more elegant reinforcement learning solution to the TSP problem. He iteratively builds a neural network capable of generating relevant routing decisions on new random graphs by combining neural networks that learn random graph embeddings with reinforcement learning.

Amazon Sagemaker [12] extended the TSP problem to more complex Vehicle Routing Problem (VRP) problems. The Clipped Proximal Policy Optimization (Clipped PPO) algorithm is used, which relies on specialized clipping of the objective function to remove the incentive for new policies to eliminate old policies. Unfortunately, their model took at least 5,000,000 training steps to converge. It is worth noting that the three reward functions they proposed provide important references for the design of reward functions in this thesis.

From the above research, it can be concluded that most of the existing research is about the application of reinforcement learning to achieve waypoint to waypoint navigation and obstacle avoidance. Even if there is research on multi-waypoint path planning, it only considers the problems in the static environment, and focuses on using multi-objective deep reinforcement learning to provide end-to-end multi-waypoint navigation. This makes the design of reward function full of skills very complex and low interpretability.

This thesis will try to combine the traditional single-waypoint heuristic search algorithm with the reinforcement learning algorithm to solve the TSP problem, that is, let the RL algorithm with strong exploration ability be responsible for solving the high-dimensional multi-waypoint path planning problem, and the more reliable traditional heuristic search algorithm be responsible for the low-dimensional single-waypoint navigation and

obstacle avoidance task. This will be a proof process of applying heuristic algorithm and reinforcement learning to solve the robot multi-waypoint path planning problem in the dynamic environment, so the basic Q-learning algorithm is chosen to start with.

In addition, the thesis will consider that it is necessary for the robot to pass all waypoints to complete the task by default, so reinforcement learning algorithm is not responsible for helping the robot understand how many waypoints there are in the task. Furthermore, the endurance mileage of the robot will not be considered as a limiting factor, because limited by the loading capacity of the cargo frame, the number of shelves that need to be replenished in a single trip of the robot in the actual supermarket will not exceed 8.

1.3 The MiR100 Robot

As shown in Figure 1, the MiR100 robot [13] is a mobile robot designed and manufactured by Mobile Industrial Robots Aps MiR, located in Oden se, Denmark. Mainly used in internal transportation and logistics applications, the MiR100 robot is designed to optimize operational processes, reduce employee workload, and allow users to reduce costs while improving efficiency. The internal positioning system of the

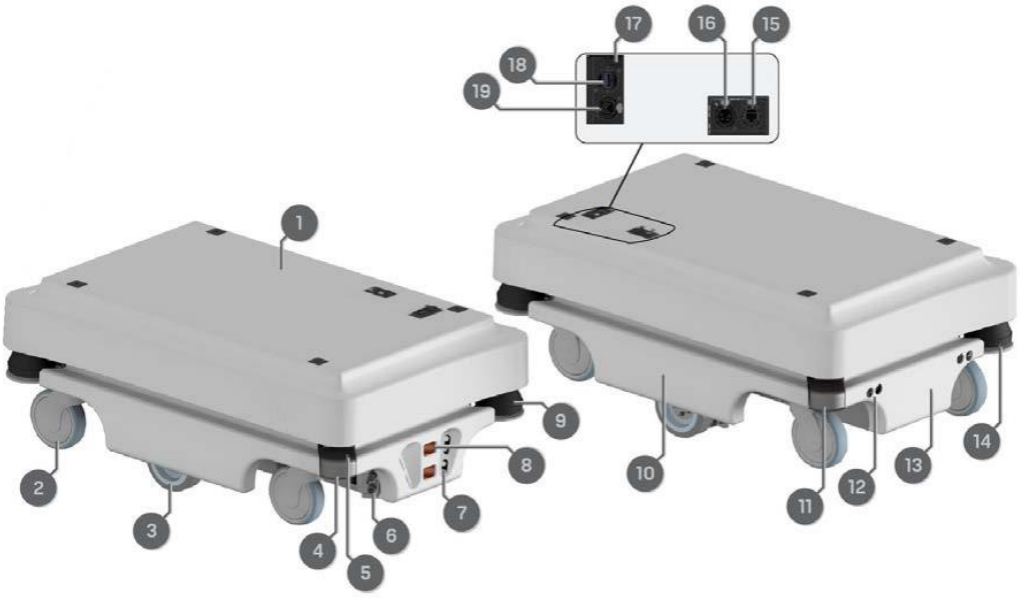

Figure 1. *MiR100 robot of the company Mobile Industrial Robots [14]*

MiR100 robot allows it to recognize the driving area and the surrounding environment, and it can also input a 3D map of the building layout. In addition, the MiR100 robot has

built-in sensors and cameras for cooperative operation, maintaining a safe environment with humans and automatically stopping when it encounters obstacles in the route. MiR100 is 35.2cm in height, 58cm in width and 89cm in length. The device weighs 70kg. 1.5 m/s (maximum 2.0 m/s) is the average speed of the robot.

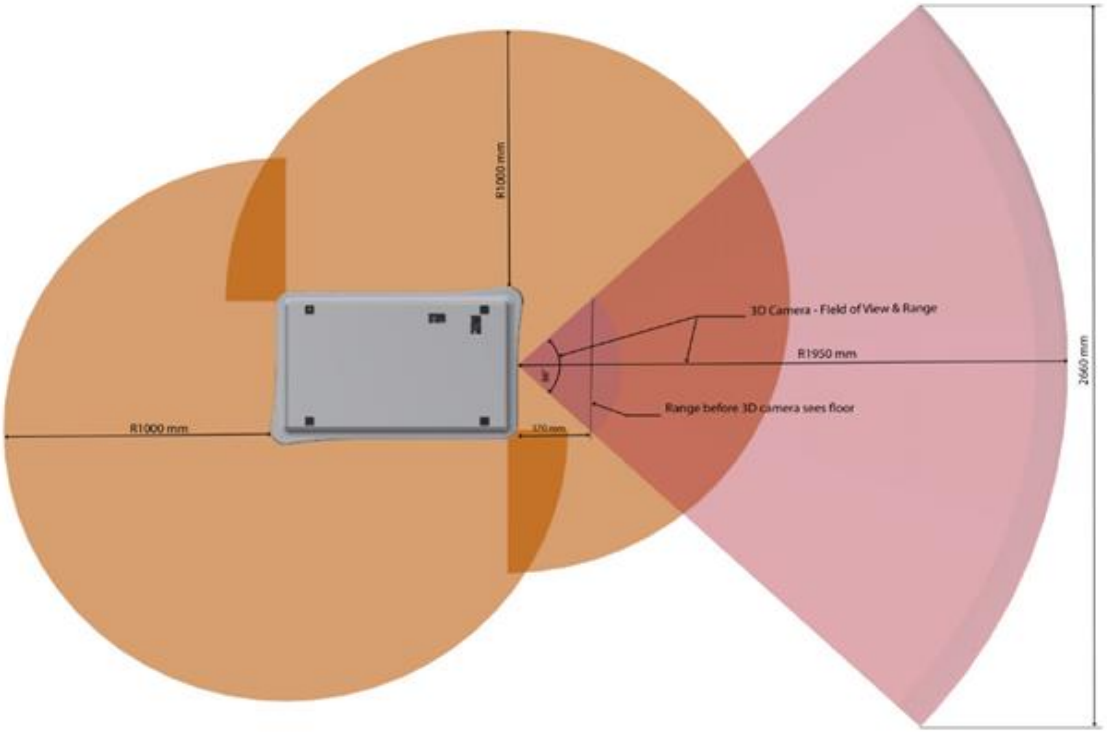

Figure 2. *Sensor setup of the mobile MiR100 robot [15]*

Figure 2 shows the configuration details of the sensors of the MiR100 robot. In general, the robot has a total of 6 wheels, two differentially controlled driving wheels (see No. 3 in Figure 1) are in the middle of the robot body, and the other four corners are Swivel wheels (see No. 2 in Figure 1).

There are three sensor types responsible for detecting obstacles:

1. The S300 safety laser scanner installed on the left side of the front (see No. 9 in Figure 1) and another one in its diagonal direction (see No. 14 in Figure 1). Each safety laser scanner has a detection field of view of up to 270°, which overlaps and thus provides a complete 360° vision protection around the robot. But they can only detect objects that intersect a plane 200mm high from the ground, and do not detect transparent obstacles well.

2. Two 3D depth cameras (see No. 7 in Figure 1) installed in front of the robot's driving direction. The two 3D depth cameras can see objects up to 1800mm vertically at 1950mm in front, as well as horizontally at angles of 118° and 180mm from the first view of the ground. Those 3D depth cameras are for navigation only. They are not part of the robot's safety system.

3. Two ultrasound sensors for detecting transparent objects are installed on the side (see No. 6 in Figure 1) and rear (see No. 12 in Figure 1) of the robot. Those ultrasound sensors are used to detect objects that cannot be seen by cameras or laser scanners.

In addition, it has internal sensors such as gyroscope, accelerometer and motor controller, as well as wheel encoders.

1.3.1 Localization

The robot's current position is resolved with Adaptive Monte Carlo Localization (AMCL), which rely on the particle filter using input data from motor encoders, inertial measurement units (IMUs), and safety laser scanners to determine the robot's most likely position on the map. In more detail:

1. The initial position of the robot serves as a reference point for the method of determining the position of the robot.

2. The data collected by the IMUs, and the motor encoders are used to determine and how fast it has moved over time from its initial position. Considering both sets of data inputs at the same time makes the deduced robot's current position more accurate.

3. Laser scanners data is used as input for particle filter algorithm, which will decide the possible location of the robot by comparing the data to nearby walls or any objects on the map.

However, the limitation of the above method is that the localization is very dependent on the correct initial position of the robot (see Figure 3 for an example). The robot cannot determine where the red point cloud (laser scanner data) matches the black point cloud on the map. When the robot can successfully locate itself, it determines a set of possible

positions, represented by the blue dots in the image above. Because the motor encoders and IMUs data will only be compared from the area where the robot computer expects the robot to be. If there are too many dynamic obstacles around the robot, it will make it unable to detect any static landmarks and thus unable to approximate the current position.

AMCL is the only specified localization algorithm in the ROS [18] Navigation Stack, a probabilistic localization system for robots moving in a two-dimensional environment.

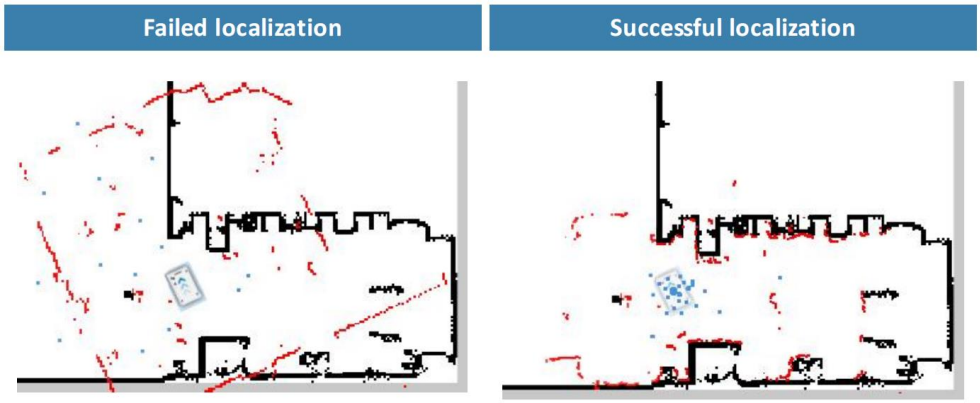

Figure 3. *The failed and successful localization [16]*

To briefly summarize its principle, by scattering particles in the global map, the particles can be understood as the possible poses of the robot. According to the evaluation criteria, such as how well the lidar data matches the map, the particles are scored. The higher the score, the more likely the robot is in this position. After passing through the particle filter, the particles with high scores are left behind. After scattering particles for many times, the particles will be concentrated to the places where the robot's position is likely to be high, which is called particle convergence. In fact, the simple understanding of self-adaptation is to increase or decrease the number of particles according to the average score of particles or whether the particles converge. It can effectively solve the problem of robot abduction and fixed number of particles.

1.3.2 Navigation

The diagram shown in Figure 4 describes the processes in navigation and system of control. The implementation of the navigation function is based on the Navigation Stack in ROS. By inputting odometer, sensor information and target pose, it outputs safe speed

commands that control the robot to reach the target state. The move_base package [19] is the top layer of the entire ROS Navigation Stack. It combines various functional modules, receives target waypoints through SimpleActionServer and completes navigation tasks.

As can be seen in Figure 4, move_base provides the configuration, operation, and interaction interface of ROS navigation. It mainly includes two parts:

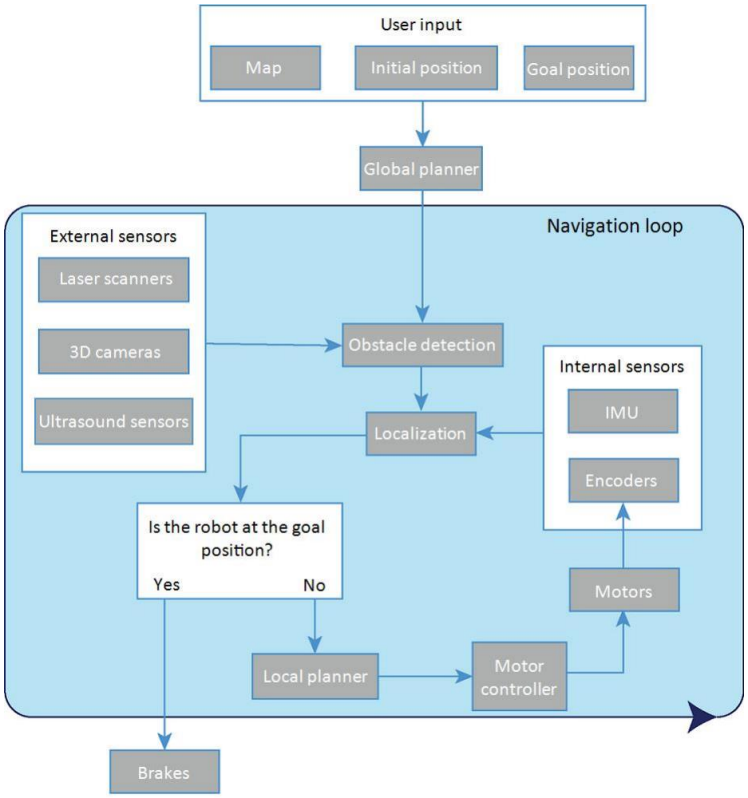

Figure 4. *Flow chart of the navigation and robot system [17]*

1. Global path planning (global_planner): The global planner is responsible for long-distance path planning. According to the global costmap, it plans an optimal path from the starting position to the target position. The global costmap is everything the robot knows from previous visits and stored knowledge, for example, the provided map of the environment and obstacles detected by sensors that are not marked on the map before. However, it needs to know the accurate information of the environment in advance. When the environment changes, such as unknown obstacles, this method is powerless. It is a kind of pre-planning, so it does not require high real-time computing power of the robot system. Although the planning result is global and relatively better, it has poor robustness to the errors and noise of the environment model.

2. Local real-time planning (local_planner): The local planner is responsible for short-range path planning and obstacle avoidance. Based on the local costmap, a new local path is planned to avoid nearby obstacles, and the replanning is continued during the global planned navigation path. The local costmap is everything that can be known from the current position. For instance, walking people and other moving objects, as well as every wall etc. that can be detected by sensors. This kind of planning needs to collect environmental data, and the dynamic update of the environmental model can be corrected at any time. The local planning method integrates the modelling and search of the environment and requires the robot system to have high-speed information processing and computing capabilities. Environmental errors and noise have high robustness and can feedback and correct the planning results in real time. However, due to the lack of global environmental information, the planning results may not be optimal, and the correct path or complete path may not even be found.

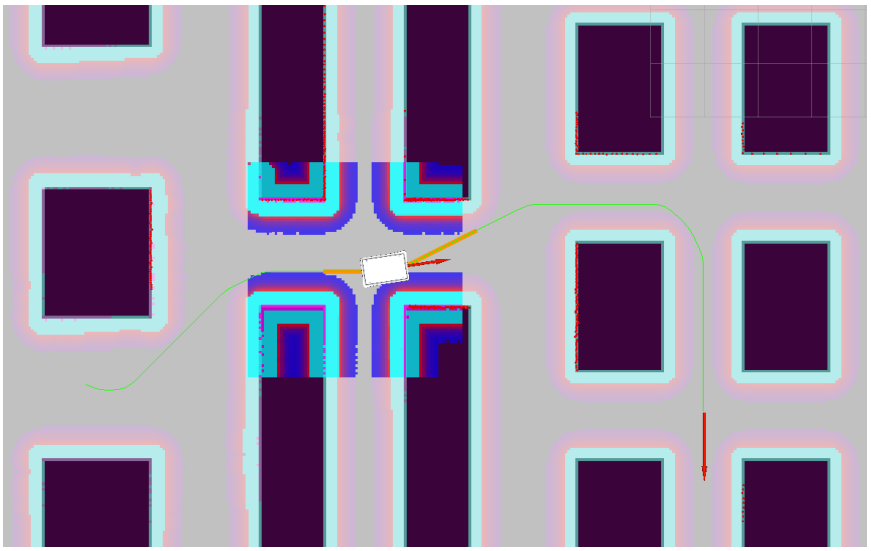

Figure 5. *Local costmap and global cost map*

The costmap of ROS is a grid map where each cell is assigned a specific value or cost in the range of 0 to 255, with a higher cost indicating a smaller distance between the robot and the obstacle. In the Figure 5, the coloured gradient around the robot is the local costmap, while the lightly coloured portion of the map around the edges of the black block fill is the global costmap. The cell costs are divided into three states: occupied (with obstacles), free space (without obstacles), unknown.

For the global planner, it adopts an improved version of SBPL (Search-Based Planner) lattice planner. The sbpl_ lattice_ Planner [20] is the ROS wrapper of SBPL [21] lattice environment. It applies the graph search method to generate the global path from the current position of the robot to the desired target position or waypoint. The search-based planner can generate a path from the start to the target configuration by combining a series of "motion primitives". Motion primitives [22] are pre-calculated motions that robots can take. By transforming the state space into a discrete graph. Each possible state (x, y, theta/yaw) of the robot will be represented by different nodes, which helps to produce a smooth path that takes the orientation of the robot into consideration in the planning. This is particularly important if the robot is not assumed to be circular or has incomplete constraints. In addition, if the probability value corresponding to the costmap in this state is very low, the node will be given a valid mark, otherwise it will be marked as invalid.

There are two planning algorithms to choose from: ARA* (Anytime Repairing A*) planner or AD* (Anytime Dynamic A*) Planner. By default, ARA* algorithm [24] is applied here for global path planning. This algorithm is suitable for situations where the best possible path planning is made in a limited time. The method is to quickly obtain a sub-optimal path, and then continuously optimize this path within the range allowed by time.

The inflated heuristic A* algorithm (A* search with inflated heuristics, hereinafter referred to as: “weighted A*”) is an algorithm that is proved to be faster in most cases but cannot guarantee to find the optimal solution. “Weighted” refers to multiplying the heuristic function of the traditional A* algorithm by an inflation factor ε greater than 1. The parameter ε defines the degree of suboptimal: the length of the suboptimal path is not greater than ε times the length of the optimal path. In this way, the algorithm can use ε to characterize the quality of the corresponding suboptimal solution (which may be the optimal solution, and the suboptimal solutions below have this meaning and will not be repeated here). Therefore, in order to construct an arbitrary-time planning algorithm that can evaluate the pros and cons of the suboptimal solution, the weighted A* can be run iteratively while continuously decreasing ε so that the suboptimal solution keeps getting closer to the optimal solution. This simple idea can lead to a series of suboptimal solutions with a suboptimal factor (i.e., the inflation factor ε, which is used to measure the quality of the suboptimal solution mentioned above). But only this still doesn't solve the problem,

because simply running weighted A* repeatedly will have many nodes that have already been calculated repeatedly, which will waste a lot of computing time.

The ARA* algorithm uses the idea of running weighted A* iteratively but reuses previously computed results while preserving suboptimal factors. In the range allowed by time, the inflation factor ε will become smaller and smaller until it becomes 1; when time is not enough, the ε will be as close to 1 as possible, because when the ε is 1, the ARA* algorithm is close to the A* algorithm, The A* algorithm is theoretically optimal. The advantage of not having to repeat the calculation for nodes that have been calculated before and the results are correct is that the efficiency of the algorithm is greatly improved, which is what this thesis expect.

For the local planner, the dwb_local_planner [25] package realizes the DWB (Dynamic Window B) algorithm which is a modular implementation of the DWA (Dynamic Window Approach) algorithm for local robot navigation on the 2D map. Based on the map data, this package searches multiple routes to the target by algorithm, uses some evaluation criteria (whether it will hit the obstacle, the time required, etc.) to select the optimal path, and calculates the required real-time speed and angle. For an omnidirectional robot like the MiR100, there is an x-direction velocity, a y-direction velocity, and an angular velocity. DWB samples from the set of achievable velocities for only one simulation step given the acceleration limits of the robot. This means, DWB is a more efficient algorithm because it can sample a smaller space.

Among them, the main ideas of the Dynamic Window B algorithms are as follows:

1. Sampling the current control space of the robot (dx, dy, dtheta) discretely.
2. For each sampled speed, calculate the state of the robot after driving at this speed for a period of time, and obtain a driving route.
3. Score multiple routes using some evaluation criteria, such as: proximity to obstacles, proximity to targets, proximity to the global path, and speed.
4. According to the score, choose the optimal path.
5. Repeat the above process.

Here, a motion planning method combining Pure Pursuit Path Tracking algorithm [26] and Vector Field Histogram with Look-Ahead Verification (VFH*) [27] is adopted to implement the local planner. The Pure Pursuit Path Tracking algorithm is responsible for keeping track of the route made by the global planner without losing it when using VFH* to avoid local unmarked obstacles. VFH* is triggered when the distance between the robot and the obstacle falls below a certain threshold. It will build a grid map (Active Window) with the current position of the robot as the centre, establish the reliability probability of obstacles based on the detection information of real-time sensors, and evaluate the density of obstacles in each direction of the robot for the units in the Active Window. Then it will form a histogram based on this and select a route less than the threshold for planning.

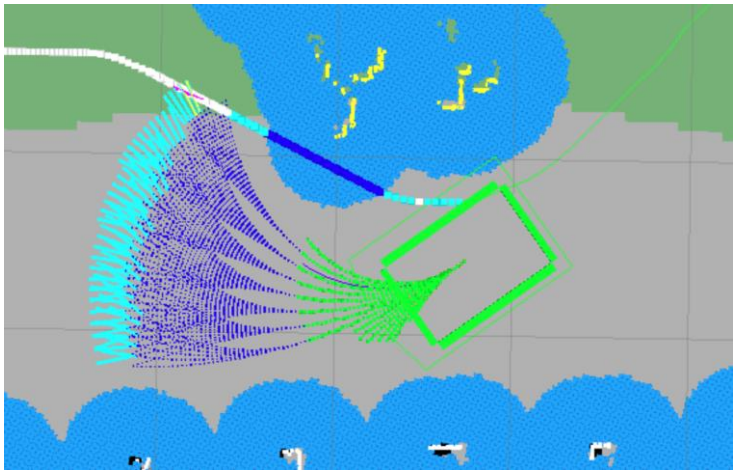

Figure 6. *Two-step VFH*: The thick curve (white) is the global path the robot should have followed, and the thick line (dark blue) and thin line (green) connected to it are the global path the robot has travelled.[9]*

In this thesis, a variant of VFH* algorithm is applied, further referred to as 2S-VFH*-R (two-step VFH* with a recovery method), in the MiR100 robot. As shown in Figure 6, in the first step, a discrete number of possible arcs (green) generated by the robot are spread out. Arcs that do not collide with inflated light blue obstacles above and below are considered valid arcs. These effective arcs are expanded into discrete number of possible arcs through a second finer VFH* expansion (purple). Finally, the extended arc closest to the global path is chosen as the best path around the obstacle. If the robot is trapped by an obstacle and the local planner cannot get it out of the way, the recovery method is triggered, which re-plans the path using the global planner and new local objects (The original map and all new obstacles encountered by the robot) are considered during the re-planning.

2 Fundamentals

This chapter summarizes the relevant basic knowledge to further deepen the reader's understanding of the thesis objective. In Section 2.1, the basic concepts of the Traveling Salesman Problem (TSP) are introduced. In Section 2.2, the use of the classical greedy algorithm to solve the TSP problem is introduced. Section 2.3 covers the basics of traditional Reinforcement Learning (RL) and presents ideas for the problem of translating TSP into RL.

2.1 The Traveling Salesman Problem (TSP)

The Traveling Salesman Problem [28] is a classic combinatorial optimization problem. The classic TSP can be described as: a salesman is going to sell products in several cities. The salesman starts from one city and needs to go through all the cities before returning to the place of departure. The salesman needs to figure out how to choose the route of travel so that the overall path is the shortest. From the perspective of graph theory, the essence of the problem is to find a Hamilton circuit with the smallest weight in a weighted completely undirected graph.

This is an NP-hardness problem. In the worst case, the time complexity of the TSP problem will become super polynomial as the number of cities increases, and the amount of computation will also increase exponentially. This means that the optimal combination of 6 waypoints (6! = 720) can be found by brute force method, but there are already more than 470 million permutations to calculate for 12 waypoints. If it increases to 600 waypoints, it becomes almost impossible to solve by brute force.

2.2 Greedy algorithm

As a simple and fast algorithm commonly used to solve optimization problems, the greedy algorithm makes an optimal choice based on a certain optimization measure based on the current situation, without considering all possible overall situations, which saves a lot of time that must be spent to exhaust all possibilities to find the optimal solution. It adopts

the top-down method to make successive greedy choices in an iterative method. Each time a greedy choice is made, the problem is reduced to a smaller sub-problem. Although it is necessary to ensure that the local optimal solution can be obtained at each step, the resulting global solution is sometimes not necessarily optimal, so the greedy algorithm should not backtrack.

Greedy algorithm solution has the following properties:

1. Greedy choice property: The overall optimal solution of a problem can be achieved by a series of local optimal solutions, and each choice can depend on the previous choices, but not on the choices to be made later. This is the nature of greedy choice. For a specific problem, to determine whether it has the greedy choice property, it must be proved that the greedy choice made at each step ultimately leads to the overall optimal solution of the problem.

2. Optimal substructure property: When the optimal solution of a problem contains the optimal solution of its subproblems, the problem is said to have optimal substructure property. The optimal substructure property of the problem is the key to solving the problem by the greedy method.

The greedy algorithm does not have a fixed algorithm framework. The key to the algorithm design is the choice of the greedy strategy. The premise of the greedy strategy is that the local optimal strategy can lead to the generation of the global optimal solution.

The greedy algorithm also has the following problems:

1. There is no guarantee that the final solution obtained is the best.

2. It cannot be used to find the maximum and minimum solutions.

3. It can only be used under certain conditions, such as the greedy strategy must have no effect, etc.

2.2.1 Greedy algorithm to solve TSP

The basic idea of greedy strategy [29]: Take the case of this thesis as an example, the robot traverses all the next waypoints that can be reached from the starting point, and it will select the nearest waypoint as the next waypoint to go. Then mark the current

waypoint as "reached", the next waypoint as the current waypoint, repeat the greedy strategy, and so on until all the waypoints are marked as "reached", then the loop ends. The above strategy provides an idea for finding local optimal solutions for the problem of multi-waypoint navigation of robots in the environment full of unknown dynamic obstacles. There is an assumption here, that is, the shortest path corresponds to the shortest time. However, in the actual unknown environment, the shortest path and the shortest time are not linearly related.

2.3 Reinforcement Learning (RL)

In machine learning, people are more familiar with supervised learning and unsupervised learning, and another major category is reinforcement learning. Table 1 compares the main differences between these three machine learning categories so that readers can gain a clearer understanding of RL:

Table 1. *Comparison of machine learning methods [30]*

Basis for comparison	**Supervised**	**Unsupervised**	**RL**
Training data	Need domain expert to label data	Unlabeled data	Learn through interaction with environment
Preference	Routine tasks (input output mapping)	Clustering, discovering data correlation and new patterns	Artificial intelligence (Behavioral learning)
Area	Machine learning	Machine learning	Machine learning
Optimal strategy	Depend on the data and learning algorithm	Depend on the data and its classification	Learn optimal strategy from experience
Exploration	No exploration	No exploration	Adaptable to changes through exploration

- Supervised learning is like having a mentor beside the students while they study. Mentors know what's right and what's wrong, but in many real-world problems like chess, Go, where there are thousands of combinations, it's impossible for one mentor to know all possible outcomes. At this time, reinforcement learning will get a result by first trying to make some behaviours without any labels and adjust the previous behaviour through feedback on whether the result is right or wrong. With this continuous adjustment, the algorithm can learn to choose what action to choose under what circumstances to get the best results.

- Both learning methods (supervised learning and reinforcement learning) will learn a mapping from input to output. Supervised learning is the relationship between

them, which can tell the algorithm what kind of input corresponds to what kind of output. What reinforcement learning produces is the feedback reward function to the machine, which is used to judge whether the behaviour is good or bad. In addition, there is a delay in the feedback of reinforcement learning results. Sometimes it may take many steps to know whether the choice of a previous step is good or bad. In supervised learning, if a bad choice is made, it will be immediately fed back to the algorithm.

- Unsupervised is not learning a mapping from input to output, but a pattern. For example, in the task of recommending news articles to users, the unsupervised method will find similar articles that the user has read before and recommend one to them, while reinforcement learning will recommend a small amount of news to the user first, and continuously obtain feedback from the user. Feedback, and finally build a "knowledge graph" of articles that users might like.

Figure 7. *Reinforcement learning in dog training [31]*

The essence of RL is to solve the problem of decision making, where decisions are made automatically and can respond to a dynamic environment by taking a series of actions to maximize the cumulative reward of the agent.

Figure 7 shows a close-to-life example: as the owner, you have an untrained puppy, and you want to train it to follow the command to “raise the right hand for a handshake”. The goal of reinforcement learning is to train a dog (agent) to perform tasks in an environment (the environment around the puppy and the owner). First, the owner will give commands and the puppy will observe (observation). The puppy then responds by taking an “action”. If the puppy moves close to the expected behaviour, the owner will offer a “reward”, such as food or toys; otherwise, the owner will offer no reward, or even a penalty, such as reducing the number of tasty foods. At the beginning of the training, when the command

given was "raise the right hand to shake hands", the dog may take more random actions, such as sitting down or raise the left hand. Puppies will try to associate specific observation states and actions with rewards. This association or mapping between observations and actions is called a strategy.

2.3.1 Markov Decision Process (MDP)

The mathematical basis and modelling tool of reinforcement learning is Markov Decision Process. Almost all RL problems can be formulated by MDP. MDP follows *Markov Assumption*: the next state s_{t+1} only depends on the current state s_t and the performed action a_t. In theory, any reinforcement learning problem or task that satisfies the *Markov assumption* can be represented as five-tuple *Markov Decision Process* $(S, A, P^a_{s,s\prime}, R^a_s, \gamma)$.

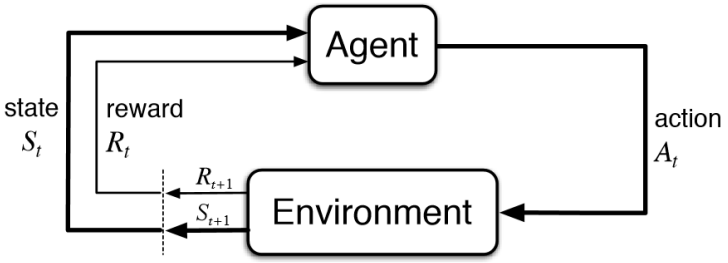

Figure 8. *Formulate the RL problem [32]*

- S represents a finite number of state sets.
- A represents a finite number of action sets in state S.
- $P^a_{s,s\prime}$ are transition probabilities from state s to s' when action a is taken at time step t. $P^a_{s,s\prime} = \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)$.
- R^a_s are reward probabilities. It returns a scalar (real number), which means: Assuming the current state is S_t, and action is a, how much reward R^a_s (Immediate Reward) can be obtained from the next state S_{t+1}. $R^a_s = \mathrm{E}[R_{t+1} \mid S_t = s, A_t = a]$ is the only thing to be considered.
- As the discount factor, γ is used to calculate the *discounted expected* return. $\gamma \in [0,1]$, $\gamma = 0$ means to only look at the present, $\gamma = 1$ means that the long-term is

as important as the present. The degree of emphasis on the long-term is controlled by adjusting the value of γ.

Episodes are often mentioned in reinforcement learning, especially in game scenes. The concept of "episode" comes from the game, which refers to the process from the beginning of the game to the clearance or end of the agent. In reinforcement learning, every continuous series of States, actions and rewards is called an episode. As shown in the following sequence:

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, \cdots, s_{t-1}, a_{t-1}, r_t, s_t \tag{1}$$

One episode of the MDP forms a finite sequence of states, actions and rewards. Take Figure 8 as an example, at time t, state s_t is observed, then the action a_t is taken, and the reward r_{t+1} is obtained later, and the new state is transformed into s_{t+1} until the final state s_{final} is reached (terminal state).

RL requires a high number of samples. Even a simple game requires tens of thousands of rounds of games to learn good strategies. Epoch is a similar but different concept, which is often used in supervised learning. An epoch means that all training data are used for forward calculation and back propagation, and each data is used just once.

2.3.2 Discounted Return for long-term strategies

The return of the current step won't be focused, but the sum of the total long-term benefits, which called the Discounted Return (Expected Cumulative Reward), expressed by G_t: starting from the moment t, the sum of the total discounted returns in the future:

$$U_t = R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k} \tag{2}$$

$\gamma \in [0,1]$ is the discount factor. The further into the future, the bigger the discount on rewards. In finite-duration MDPs, the discount can be absorbed into the reward function, i.e., treat $\gamma^i \cdot R_{t+i}$ as a new reward function R_{t+i}. Note that the new reward function is no longer stationary at this point, even though the original reward function was stationary. In an infinite MDP, the discount factor plays an important role, which together with the boundedness of the reward function guarantees the convergence of the infinite summation series above. Whether it is a finite-term MDP or an infinite-term MDP, this thesis always

considers discounted rewards whenever the reward function is assumed to be stationary. The goal of reinforcement learning is to find a strategy that maximizes the Discounted Return. This policy is called the optimal policy (π^*).

2.3.3 Q-Learning: Learning optimal action-value function

Q-learning is a value-based model-free reinforcement learning algorithm. The value-based algorithm updates the value function according to the Bellman equation, and outputs the value of all actions. Actions are selected according to the highest value. Such methods cannot select continuous actions. Since real-world environments may lack any prior knowledge of the environment's dynamics, model-free RL methods come in handy in such situations. Model-free means the agent doesn't try to understand the environment, but waits step by step for real-world feedback, and then takes the next action based on the feedback.

Q-learning is an off-policy learner. Meaning it learns the value of the optimal policy. The expected value (cumulative discounted future reward) of doing action a in state s is represented by the function $Q_*(s, a)$. Matiisen [33] provides an apt explanation for the Q-function, which is "the best possible score at the end of the game after action a in state s". It's called the Q-function because it represents the "Quality" of taking action a in the state s. The Q-function is also known as the optimal action-value *function*. Equation 3 expresses the relationship between the optimal action-value function and the general action-value function:

$$Q_*(s_t, a_t) = \max_{\pi} Q_\pi(s_t, a_t), \qquad \forall\, s_t \in S, \quad a_t \in A. \tag{3}$$

That is, the optimal action-value function is the maximum of the action-value function under all policies. Through such a definition, the uniqueness of the optimal action value can be achieved, so that the entire MDP can be solved. Here π denotes the policy, which summarizes how the agent choose to act in each state. There are many policy functions π to choose from, and here the best policy function π^* is chosen:

$$\pi^* = \operatorname*{argmax}_{\pi} Q_\pi(s_t, a_t), \qquad \forall\, s_t \in S, \quad a_t \in A. \tag{4}$$

It excludes the influence of policy π, and only evaluates the quality of the current state s_t and action a_t.

In reinforcement learning, the agent interacts with the environment, record observed states, actions, rewards, and use these experiences to learn a policy function. In this process, the strategy that controls the agent's interaction with the environment is called the behaviour policy. The role of behaviour policy is to collect experience, that is, the observed environment, actions, and rewards. The Q-learning algorithm collects four-tuple like (s_t, a_t, r_t, s_{t+1}) with arbitrary behaviour policy, and then record them into an array. This array is called the "experience replay buffer". These experiences are used repeatedly to update (train) the target policy, and this training method is called "experience replay"

Temporal Difference (TD) algorithm is used to estimate the value of $Q_*(s, a)$. TD is that an agent learns from the environment through episodes without prior environmental knowledge [34]. In the simplest case, the Q function is implemented as a table of $Q[S, A]$, which is also called Q-table, with rows and columns as set of states S and set of actions A respectively:

If there is only the current state and action, and the future state and action has not yet occurred, how to estimate the return Q at the end of the episode? The answer is obtained iteratively through the Bellman equation:

$$\underbrace{Q_*(s_t, a_t)}_{\text{expectation of } U_t} = \mathbb{E}_{S_{t+1} \sim p(\cdot|s_t, a_t)} \left[R_t + \gamma \cdot \underbrace{\max_{A \in \mathcal{A}} Q_*(S_{t+1}, A)}_{\text{expectation of } U_{t+1}} \mid S_t = s_t, A_t = a_t \right] \quad (5)$$

The idea of Q Learning is completely based on value iteration. But it needs to be clear that value iteration updates all Q values every time, that is, all states and actions. In fact, it's not possible to traverse all the states, and all the actions, instead, only a limited series of samples can be achieved. Therefore, only limited samples can be used to operate. Q Learning proposes a way to update the Q value: Below is the Q-value update rule that is the core of the Q-learning algorithm:

$$\underbrace{Q(s_t,a_t)}_{\text{new value}} \leftarrow (1-a)\cdot \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\substack{\text{learning}\\ \text{rate}}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\substack{\text{discount}\\ \text{factor}}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\substack{estimate\ of\\ optimal\\ future\ value}} \right)}^{learned\ value} \quad (6)$$

Although the target Q value is calculated according to the value iteration, this Q value (which is the estimated value) is not directly assigned to the new Q, but a "gradual" method similar to random gradient descent is adopted, that is, each time a small step is taken towards the target Q, the "step size" of each small step depends on the learning rate α, which can reduce the impact caused by the estimation error, and finally ensure that the model can converge to the optimal Q value.

After the training, the policy function (Q table) is used to control the agent. In simple words, the agent only needs to choose the action with the largest Q value at each step. Based on the current state s_t, the formula used by the agent to make a decision is:

$$a_t = \underset{a \in A}{\operatorname{argmax}}\, Q_*(s_t,a) \quad (7)$$

It means to find the row corresponding to s_t (one of the 3 rows), find the maximum value of the row, and return the action corresponding to the element. See Table 2 as an example, the current state s_t is the state 2, then the agent will look at the second row and find that the maximum value of the row is 210, corresponding to the fourth action. Then the action a_t that should be performed is the action 4.

As shown in Figure 9, the execution flow of the Q-learning algorithm is as follows:

1. Initialize the Q-values $Q(s,a)$ arbitrarily for all state-action pairs.
2. For life until learning is finished…
    a) Choose an action a in the current world state s based on current Q-value estimates $Q(s,\cdot)$
    b) Take the action a and observe the outcome state s' and reward r

c) Update $Q(s,a) = Q(s,a) + \alpha \cdot \left(r + \gamma \cdot \max_a Q(s',a) - Q(s,a)\right)$. New Q value = Current Q value + learning rate * [Reward + discount rate * (highest Q value between possible actions from the new state s') — Current Q value].

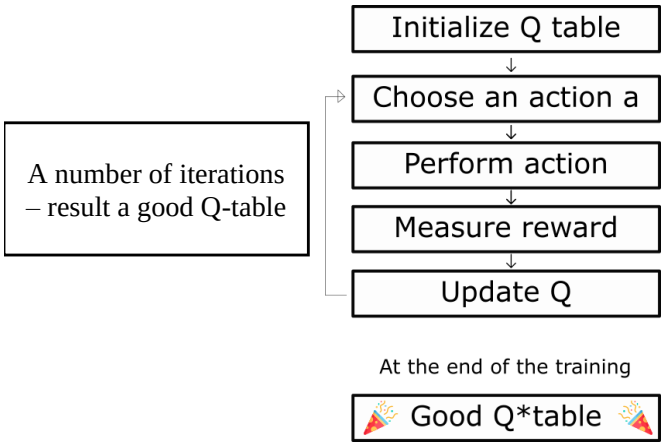

Figure 9. *The Q learning algorithm process [35]*

2.3.4 Explore-exploit dilemma: Epsilon-greedy exploration

In order to build an optimal policy, the agent is faced with the dilemma of exploring new states while maximizing its overall reward. This is called the Explore-exploit dilemma, as known as the exploration-exploit trade-off. Exploration will try a lot of different things to see if they are better than what you have tried before. While exploitation tries the most efficient behaviour from the past experience. To balance those two, the best overall strategy may involve short-term sacrifices. Therefore, the agent should gather enough information to make the best overall decision in the future.

Should the agent leverage existing strategies or explore new ones? In Q-learning, after the Q table is randomly initialized, the Q value (prediction) obtained with the initialized model must also be random. At this time, when the agent chooses the action with the highest Q value, it is equivalent to randomly selecting an action. At this time, the agent is actually exploring. When the algorithm gradually converges and the estimated value of Q becomes more and more stable, for each state, the corresponding action with the highest Q value becomes more and more fixed, which also means that the strategy becomes more and more stable. At this point, the agent is exploiting. Therefore, Q learning itself already has a certain "exploration", but this "exploration" is very short-lived and "greedy". It only

explores at the beginning of training but stops exploring after the first stable policy is explored, which makes it easy to get a local rather than a global optimal policy.

A simple and effective way to solve this problem is the epsilon (ϵ)-greedy exploration strategy. That is, each time an action is selected, an action is randomly selected with a certain probability ϵ, and in other cases, the action with the largest Q value is selected. DeepMind has adopted the method [36] of decreasing ϵ from 1 to 0.01, so that a large number of state spaces can be explored in the early stage of training, and then gradually stabilized to a smaller exploration ratio in the later stage.

$$a_t \begin{cases} \underset{a \in A}{\operatorname{argmax}}\, Q\,(s_t, a), & \text{with probability } (1-\epsilon); \\ uniformly\ select\ an\ action\ from\ \mathcal{A}, & \text{with probability } \epsilon. \end{cases} \tag{8}$$

In detail:

1. An exploration rate "ϵ" is specified, which is initially set to 1. This is the step size will be randomly taken. Each time the agent performs an action, it is regarded as a "step", and the new exploration rate is equal to the current exploration rate multiplied by an exploration decay rate (for example, set to 0.999 in this thesis), and so on. In the beginning, this exploration rate should be at the maximum value (for example, set to 1.0 in this thesis), since the Q-table is empty. This means that a lot of exploration need to be done by randomly choosing actions.

2. Generate a random number. If this number is greater than epsilon, then the agent will "exploit" (meaning the agent uses the information it knows already to choose an action at each step). Otherwise, it will keep exploring.

3. When first training the Q-function, a large epsilon is needed. The epsilon will decrease gradually until the minimum exploration rate (for example, set to 0.01 in this thesis) as the agent becomes more confident about the estimated Q value.

2.3.5 Transforming the TSP to a RL problem

In the classic version of the TSP problem, the salesman would choose to start in one city, travel through the remaining cities and return to the city from which he originally started. An instance is given by the set of cities and their paired distances. [37]

In this thesis, the problem is redefined as a mobile robot that needs to complete the task of restocking multiple shelves in a supermarket in the shortest possible time. The robot carrying the goods will start from the warehouse and transport the goods to each designated shelf position (waypoint). From a practical point of view, the robot needs to respond to the needs of shoppers in the supermarket as quickly as possible, so when the robot reaches the last shelf position without return to the warehouse (the starting point), the task is considered complete, then the timer ends.

Each time the robot reaches a waypoint, it will get a corresponding reward, and the reward is inversely proportional to the time it takes to perform this action (choose the next waypoint). In this way, the robot will be able to figure out an optimal order to visit all the waypoints in order to obtain the maximum cumulative reward, which is the shortest total time to complete the task. In this thesis case, an episode of RL is one instance of the TSP.

3 Simulation Environment

This chapter presents a 3D model of a virtual supermarket built for training and testing the algorithm. The modelling parameters of stationary objects and dynamic obstacles in this 3D environment are described in detail. The training is limited to 3D space, and two safety laser scanner sensors installed along the diagonal of the robot are used to detect the surrounding static and dynamic obstacles. Here, Gazebo [38] is used as a 3D physics simulation platform. The robot must have a map for each area it operates. It is important to create a complete and reliable map so that the robot can perform effectively and safely.

3.1 Gazebo simulator

Gazebo is a powerful 3D physics simulation platform with powerful physics engine, high-quality graphics rendering, convenient programming and graphics interface, and most importantly, its open source and free features. The robot model in gazebo is the same as the model used by RViz (the Robot Visualization tool), but the physical properties of the robot and the surrounding environment, such as mass, friction coefficient, elastic coefficient, etc., need to be added to the model. Although RViz and Gazebo are very

similar in terms of interface, they are actually very different. Gazebo implements simulation and provides a virtual world, while RViz implements visualization and presents received Information. The plug-in on the left is equivalent to a subscriber, RViz receives information and displays it.

The sensor information of the robot can also be added to the simulation environment in the form of plug-ins and displayed in a visual way. It can accurately and efficiently simulate the functions of robot work in complex indoor and outdoor environments and is usually used in conjunction with ROS to provide developers with an excellent simulation environment. gazebo supports URDF/SDF format files. The difference between the two will be mentioned later. They are both used to describe the simulation environment. The official also provides some integrated and commonly used model modules, which can be imported and used directly.

The Unified Robot Description Format (URDF) is an eXtensible Markup Language (XML) file format used by ROS to describe all elements of a robot. To use a URDF file with gazebo, some emulation specific tags must be added to work properly with gazebo.

Although URDF is a useful and standardized format in ROS, they lack many features and have not been updated to meet the evolving needs of robotics. URDF can only specify the kinematics and dynamics of a single robot alone, not the pose of the robot itself in the world. It is also not a universal description format, as it cannot specify joint rings (parallel connections) and lacks friction and other properties. Also, it cannot specify non-robots such as lights, heightmaps, etc.

In terms of implementation, the URDF syntax makes heavy use of XML attributes to break proper formatting, which in turn makes the URDF more inflexible.

To address this, a new format called Simulation Description Format (SDF) was created for use by gazebo to address the shortcomings of URDF. The SDF is a complete description of everything from world-class to robotics, capable of describing every aspect of robotics, static and dynamic objects, lighting, terrain, and even physics. SDF can accurately describe various properties of robots. In addition to traditional kinematics, it can also define sensors, surface properties, textures, joint friction, etc. SDF also provides methods to define various environments, including ambient lighting, terrain, etc. SDF is

also described in XML format. In conclusion, SDF is an evolution of URDF that better describes real simulation conditions.

3.1.1 Static Environment

Figure 10 is a floor plan of a 41×35 m supermarket built by the author of the thesis. The location marked with a red flag is the entrance of the supermarket warehouse, which will also serve as the starting point for the robot to start replenishment task. A 2D map of a

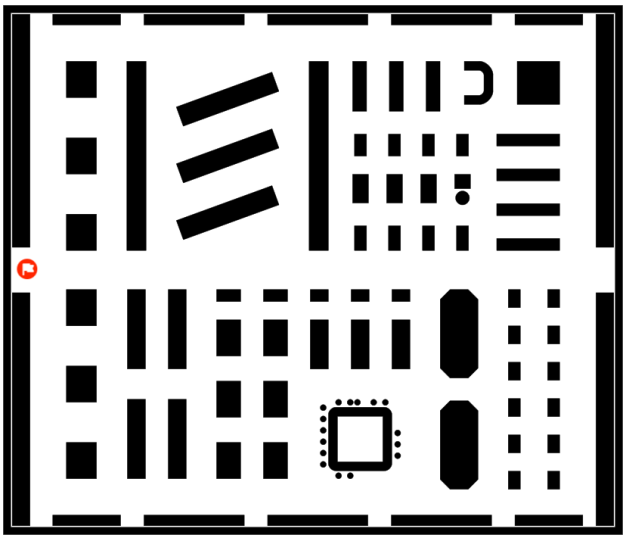

Figure 11. *The 2D map of the simulation environment (Area: 1435 m^2; Length: 41 m, Width: 35 m)*

static environment is created by loading a defined map from a yaml file, which contains all the relevant information for the map: floor plan, resolution and origin.

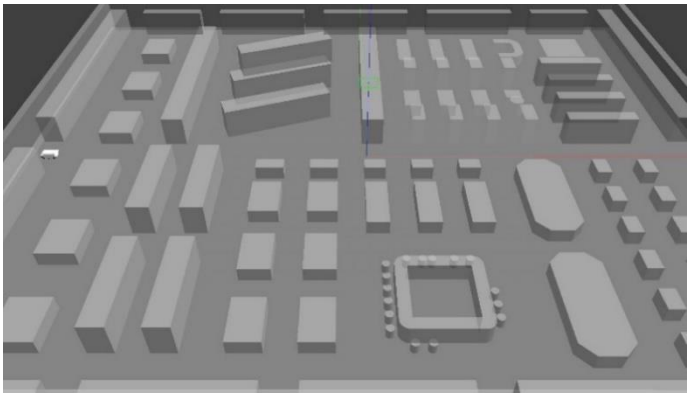

Figure 10. *The 3D map of the simulation environment*

Figure 11 shows the 3D map defined by the SDF file, where the shortest passable distance between static obstacles (shelves) is 1.2 meters.

3.1.2 Moving Obstacles

The moving obstacles in the simulated environment are also defined by the SDF files, and these moving obstacle objects are designed to abstract the behaviour of shoppers and staff in the supermarket. They will move along a predetermined trajectory or stay in one position for a certain period. As shown in Figure 12, the moving obstacles are purple

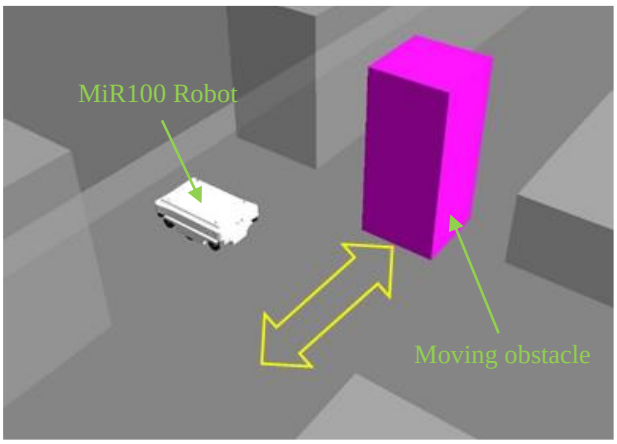

Figure 12. *The 3D model of the moving obstacle (Purple cuboid: 1×1×1.8 m). The yellow arrow indicates the direction in which the obstacle moves.*

cuboids, which are easy to distinguish from the grey static environment. The dimensions of the purple cuboid are: 1 meter long, 1 meter wide, and 1.8 meters high. Their moving speed varies from 0.6 meters per second to 1.5 meters per second.

Each moving obstacle will have a sensing range with a radius of 2 meters. Whenever the robot enters this range, the moving obstacle the robot encounters will stop until the robot leaves this sensing range. This is to prevent the robot from colliding with moving obstacles, which will force the training process to stop.

Additionally, in order to prevent the robot from being completely blocked by moving obstacles and unable to find a way out, a "jumping mechanism" is set for moving obstacles: suppose a moving obstacle moves along a square path. The four vertices of the square are set as the waypoints of the moving obstacle. When the obstacle stops for more than 30 seconds after encountering the robot, the obstacle will be teleported to two waypoints further, that is, it will be spawned in the diagonal direction of the square path, which can help the robot get out of trouble. This is in line with the purpose of this thesis, that is, the case that moving obstacles trap the robot completely should be avoided, instead, the

obstacles should delay the robot's way to the waypoint, so that the robot can recognize the traffic congestion the area in the global map.

4 Methods and Setup

This chapter describes the methods used and the specific training settings. Section 4.2 contains information on setting up static and dynamic environment tasks. Section 4.3 presents the parameter settings of RL-agent inspired by the OpenAI-Gym [39] environment design framework, including different observation spaces, action spaces, reward functions, and hyperparameters. Section 4.4 introduces the parameter settings of the Greedy method as a control group.

4.1 Navigation Stack Setup

RL-agent will integrate global planner and local planner in traditional navigation software introduced in Section 1.3.2. This means that in low-dimensional tasks, the navigation between waypoints will rely on the ARA* algorithm in the SBPL lattice planner to provide global planning, while the local obstacle avoidance part relies on the 2S-VFH*-R algorithm in dwb_local_planner to generate the local obstacle avoidance route. The Q-learning algorithm will be responsible for solving the high-dimensional TSP problem defined in this thesis, that is, the shortest time-consuming path planning task of multiple waypoints. Furthermore, in order to avoid performance impacts due to the global planner, and to isolate other possible sources of errors, the AMCL localization disable and instead provide perfect localization.

4.2 Task Setup

The task of the agent (MiR100 robot) is to plan a route through all 6 waypoints in a given 1435 m^2 simulated environment with the shortest total time. Shelves (global static objects) in the simulation environment are fixed objects listed in the global map, so they are considered by the global planner. Moving obstacles (local dynamic objects) are objects in the global map that are not marked ahead of time. The robot's laser scanning sensors

will detect and record these most challenging moving objects. This is also the most interesting part of this thesis, because the dynamic environment will test the forward-looking behaviour of the agent.

In this thesis, two environmental settings, which are static and dynamic, will be used to test and compare the decision-making performance of reinforcement learning and greedy methods. Both setups will share the same global map with 6 fixed-position waypoints (see Figure 13). A robot is considered to have reached a target waypoint if its centre point is within a 0.6 m radius of its target waypoint.

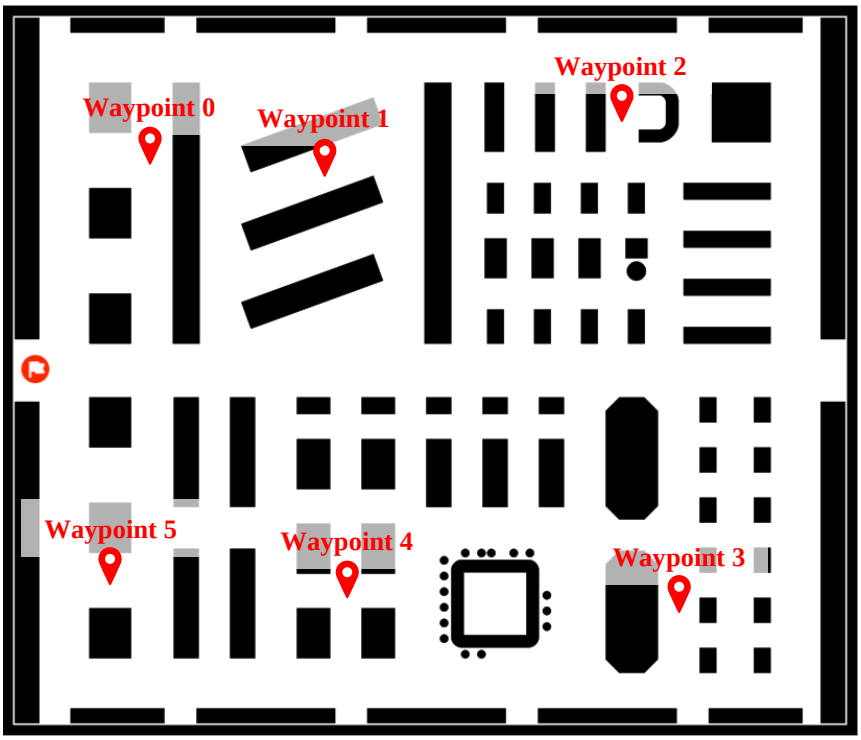

Figure 13. *The location of the 6 waypoints in the global map*

An episode is considered successfully completed if the robot has reached all 6 waypoints. The reason why the robot is not forced to reach the precise position of the target waypoint is because, due to various interference factors in the simulated environment, when the robot has reached the target waypoint, it always takes a lot of time to Iteratively adjust the position and orientation to match the exact position. Therefore, it is unnecessary for

the robot to reach the precise location of the target waypoint, and it will also increase the complexity of the learning problem.

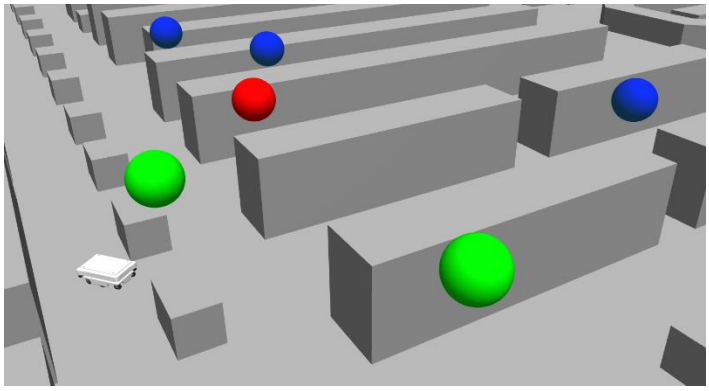

Figure 14. *Legend for waypoints in gazebo（The locations of the waypoints in the diagram are for illustration purposes only, not the actual locations in the simulated environment.）*

In order to achieve a good visual interaction experience, different colours are used to represent the status of waypoints. As shown in Figure 14, the spheres with a diameter of one meter floating in the air represent waypoints for the task:

- Blue sphere: those waypoints that have not been visited yet
- Green sphere: those waypoints that have been visited
- Red sphere: the waypoint that the robot is currently heading to

In addition, a maximum time value is set in consideration of unexpected situations such as the robot getting stuck on obstacles in the simulated environment. If the total time exceeds this maximum, the episode will be stopped and then restarted.

4.2.1 Static Environment Setup

For each episode, the static objects (the shelves) are spawned into the 3D environment. As shown in Figure 15, in the initial state of each new episode, 6 blue spheres representing waypoints with collision-free properties will be spawned at a height of 2 meters above the ground plane. The current episode ends when the robot reaches all the waypoints. The

global cost map will then be reset and the robot will be teleported to the initial position (the entrance of the warehouse) for starting the new episode.

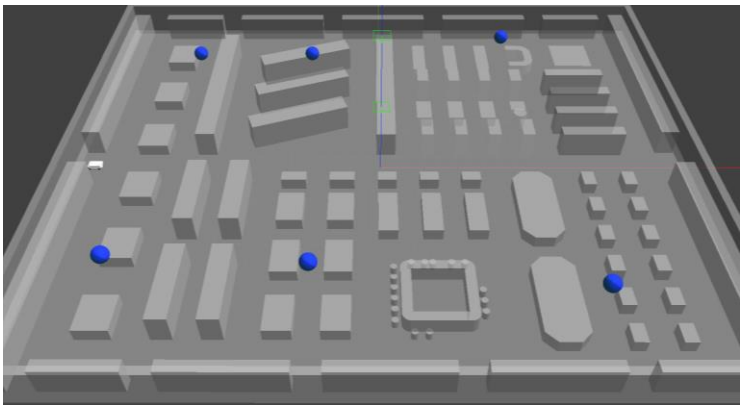

Figure 15. *The static environment (The blue floating spheres represent the waypoints)*

4.2.2 Dynamic Environment Setup

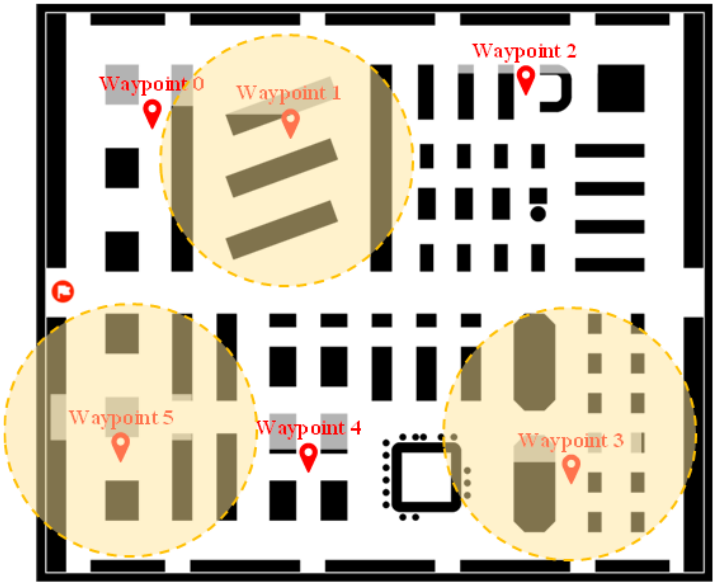

Figure 16. *The dynamic environment with traffic congestion areas (yellow area)*

The same global map and known static environment (shelf) as the static environment are used here. The difference is that here moving obstacles are placed deliberately in the area around 3 non-adjacent waypoints (waypoint 1, 3, 5) among the 6 waypoints (See Figure 16), and these areas are defined as "traffic congestion areas". And every time the robot starts a new episode, the moving obstacles in the environment are not reset to the initial

position synchronously, which means that in each episode, if the robot chooses to follow a different order to visit the waypoints, the moving obstacles will appear at different positions at the same time step. This is in line with the task specification of this thesis, that is, from the perspective of the TSP problem, RL is expected to help the robot "learn" how to identify which are more time-consuming congested areas (requires repeated stops or re-planning of local routes to avoid moving obstacles) , which are non-congested areas (without stopping or avoiding unknown obstacles), o that the robot will eventually intelligently arrange the order of waypoint visits to achieve the goal of least time consuming.

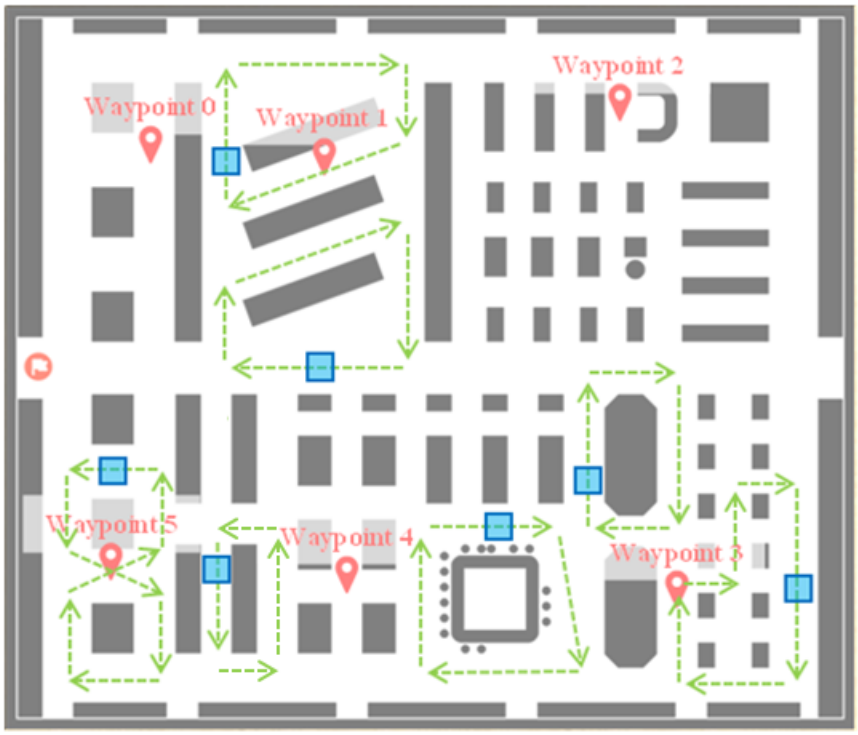

Figure 17. *The trajectories and directions (green dashed arrow) of all 7 moving obstacles (blue square)*

Figure 17 shows the detailed settings of all 7 dynamic obstacles in the simulation space. The blue squares in the figure represent moving obstacles, and the green dashed arrows represent the moving trajectories and directions corresponding to each moving obstacle. It is worth noting that there are some trajectories of moving obstacles that pass the waypoints multiple times, which means that the waypoints are occupied at certain times, and the robot (RL-agent) needs to learn how to adapt to such situations.

In addition, the costmap is cleared once every new episode starts. The purpose of this is to avoid misleading the global planner with the expired position information about the dynamic obstacle, because the actual position of the dynamic obstacle is different at each moment.

4.3 Experimental Group Setup (RL-Agent)

The pseudo-code (see Algorithm 1) shown below demonstrates how the Epsilon-Greedy Q-learning algorithm [40] used in this thesis works.

Algorithm 1: Epsilon-Greedy Q-learning Algorithm

Data: α: learning rate, γ: discount factor, ϵ: exploration rate, ϵ_{min}: minimum exploration rate, λ: exponential decay rate, Q: Q-table generated so far, S: current state

Result: A Q-table containing $Q(S, A)$ pairs defining estimated optimal policy π^*

Function: SELECT-ACTION(Q, S, ϵ) **is**
- $n \leftarrow$ uniform number between 0 and 1;
- **if** $n > \epsilon$ **then**
  - $A \leftarrow \text{argmax}\, Q(S, .)$;
- **else**
  - $A \leftarrow$ select a discrete random action with uniform distribution in the action space;
- **end**
- Return selected action A

end

/* Initialization */
Initialize an empty $Q(S, A)$, except $Q(\text{terminal}, .)$;
$Q(\text{terminal}, .) \leftarrow 0$;
/* For each step in each episode, calculate the Q-value and update the Q-table */
for *each episode* **do**
- /* Initialize state S = the starting point, by resetting the environment */
- Initialize state S;
- **for** *each step in episode* **do**
  - **do**
    - /* Choose action A from S using epsilon-greedy policy derived from Q */
    - $A \leftarrow$ SELECT-ACTION(Q, S, ϵ);
    - Take action A then observe reward R and next state S′;
    - $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$;
    - /* Reduce epsilon until the minimum value, because less and less exploration is needed */
    - **if** $\epsilon > \epsilon_{min}$ **then**
      - $\epsilon \leftarrow \epsilon * \lambda$;
    - **else**
      - $\epsilon \leftarrow \epsilon$;
    - **end**
    - $S \leftarrow S'$;

```
    | | while S is not terminal;
    | end
end
```

An empty Q-table is created at the beginning. The action value of the terminal state is set to zero.

After initialization, in each step, the Epsilon-Greedy action selection is applied, i.e., select an action A from the Q value in the Q-table with a certain probability, or randomly select any action A. When the agent takes action A, its state changes from S to S'. In this way, it gets the reward R. These values are then used to update the Q-table entry $Q(S, A)$. The above steps are repeated in this way until the agent reach a terminal state, that is, all waypoints have shown the status of arrival.

4.3.1 Observation Space

RL-agent needs to obtain all the environmental information that can help it make decisions, because the multi-objective point path planning problem is defined as a TSP problem in this thesis. To demonstrate the feasibility of this definition, a simple definition of the observation space is decided to start with, that is, the point where all robots will stop and visit: the starting point, waypoint 0, waypoint 1, waypoint 2, waypoint 3, waypoint 4, waypoint 5. Here the starting point refers to the warehouse entrance as the initial location of the robot.

4.3.2 Action Space

Here, the action space consists of 6 discrete actions, i.e., 6 waypoints as the robot's moving target: waypoint 0, waypoint 1, waypoint 2, waypoint 3, waypoint 4, waypoint 5.

When the robot is in each state, it will correspond to 6 action choices with different probabilities, and the robot will not choose the waypoint it has reached as the action.

4.3.3 Reward Functions

In order to verify the feasibility of the proposed framework, this thesis decided to try the simplest reward function first, that is, the overall time taken to complete the task, rather than the total path distance. In each episode, the robot will receive -1 reward for every 1

second it consumes, that is, the reward is inversely proportional to time. The longer the robot takes to complete the task, the smaller the reward.

4.3.4 Hyperparameters

In order to verify the feasibility of the proposed framework, this thesis decided to use the same hyperparameters for the static environment and the dynamic environment:

- Learning rate (α): 0.8. Here, the learning rate is also called the step size. It determines the update speed of the Q value, which affects the "speed of learning Q" of the RL-agent. As shown in Equation (6), the new Q value of the state is calculated by multiplying the old Q value by alpha (α) multiplied by the Q value of the selected action. So, higher α value makes the Q value change faster, but this makes the RL-agent overly sensitive to misleading (false) knowledge in noisy environments.

  Alpha is a real number between 0 and 1. If alpha is set to 0, the agent does not learn the "knowledge" that comes with the new action. Conversely, if alpha is set to 1, the agent completely ignores prior knowledge and only focuses on new knowledge.

- Discount factor (γ): 0.95. Gamma (γ) determines the weight to be considered for the best estimate of future rewards. gamma is also a real number between 0 and 1. When gamma is equal to 0, the RL-agent will completely ignore future rewards and short-sightedly consider current rewards. If gamma is set to 1, the RL-agent keeps looking for high reward values in the long term, which will result in blocked transitions [41]: summing up non-discounted rewards results in high Q values.

- Exploration rate (ϵ): 1.0. As shown in Equation (8), the Epsilon (ϵ) parameter determines whether the exploration behaviour of the epsilon-greedy action selection process in the Q-learning algorithm is aggressive or not. It is because of the randomness that epsilon introduces into the algorithm, which forces the RL-agent to try an action that is different from the existing Q-table corresponding to a specific action. This helps the agent not settle for local optima only.

- Minimum exploration rate: 0.01. The exploration rate will be reduced all the way to the minimum value of it, which also contributes to avoid the problem of prematurely trapping the agent in local optima.

- Exponential decay rate: 0.999. This parameter determines how fast the exploration rate exponentially decreases. The higher the exponential decay rate, the slower the exploration rate decreases, and the more sufficient the exploration.

4.4 Control Group Setup (Greedy Method)

The pseudo-code (see Algorithm 2) shown below is the implementation of the Greedy algorithm [42] used in this thesis.

Algorithm 2: Greedy Algorithm

Data: n: number of waypoints, r: the position of the robot, w: the position of one waypoint

Result: Output a series of waypoint coordinates in sequence, which will be used to complete the route planning of multiple waypoints.

```
Function: PATH-DISTANCE-LIST(P) is
    P ← list of unreached waypoints;
    for each unreached waypoint in P do
        /* Use global planner to calculate the distance between the robot and one waypoint */
        dist ← global_planner(r, w);
        D ← list.append(dist);
    end
end

Function: SELECT-NEXT-WAYPOINT(D) is
    shortest_distance ← min(D);
    A ← D.index(shortest_distance);
end

/* For each step in each episode, calculate the distance from the current position of the robot to the
remaining waypoints that have not been reached and select the waypoint with the shortest distance as
the target to go. */
for each episode do
    /* Initialize the position of robot to the starting point, by resetting the environment */
    Teleport robot to the starting point;
    i = 0;
    while i < n do
        Reset the list of distance before move to each waypoint;
        if Number of waypoints that have arrived > 1 then
            /* Calculate the distance from the current position of the robot to the remaining unreached
            waypoints, and save those distance to a list D */
            D ← PATH-DISTANCE-LIST(P);
```

```
            /* Pick the waypoint A with the shortest distance as the next waypoint to go   */
            A ← SELECT-NEXT-WAYPOINT(D);
        else
            If there is only one waypoint left, then directly pick that one as the last waypoint to go;
        end
        i ← i + 1;
    end
end
```

The core principle is the robot departs from the starting point, each time to find the point that is closest to the current point among the points that have not been passed, as the point to be traversed in the next step, until all points are traversed, and return to the starting point.

The global_planner in move_base introduced in the navigation Section 1.3.2 is used to calculate the distance from the robot to the waypoint. At the same time, the local_planner is used to complete the obstacle avoidance task.

Although our goal is to minimize the total time spent, for Greedy algorithm, determining the access order of waypoints by distance can help the robot find the local optimal solution in the environment with obstacles that are not pre-recorded.

5 Evaluation

In this chapter, the RL agent and the Greedy method are tested in both static and dynamic environments to test their decision-making performance and compare the results of the two methods and summarize the possible reasons for their results.

5.1 Static Environment

The RL-agent model is trained based on a static environment without obstacles as the one introduced in Section 4.2.1, and test the performance of the RL-agent and Greedy method. In the training phase, the RL-agent will guide the robot to find the Q-table of the global optimal solution after 1000 episodes of training output. In the testing phase, both RL-agent and Greedy method will go through 10 episodes (rounds) and calculate the average time for completing the task. This average time will be used to compare the decision-making performance of these two algorithms.

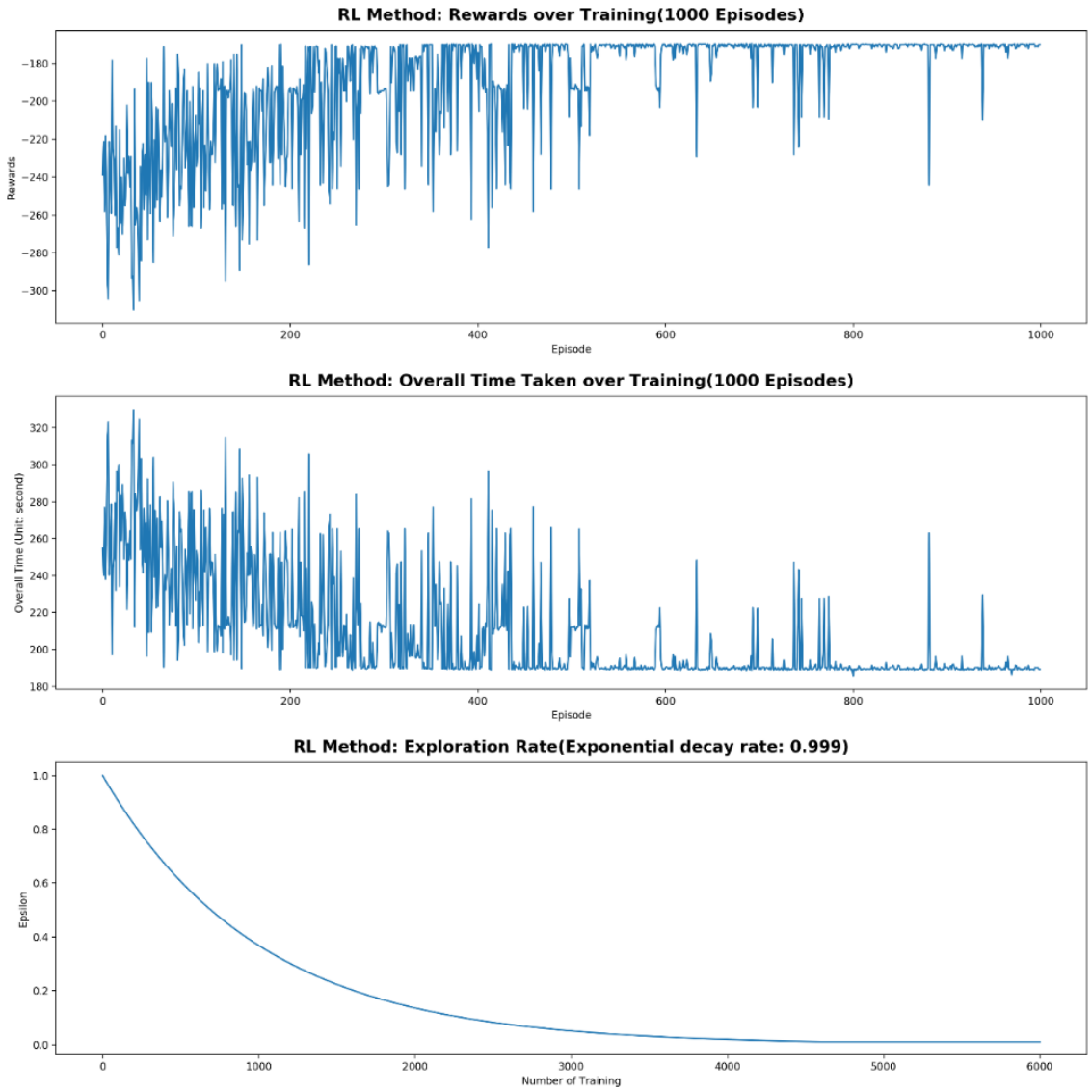

Figure 18. *RL-agent training results in static environment, including rewards, overall time and exploration rate*

5.1.1 RL-Agent

Figure 18 shows three blocks of results, namely rewards over training, overall time taken over training, and exploration rate.

After about 500 episodes, the model started to converge, at which point the exploration rate dropped to around 0.1. Then the model tends to stabilize at around the 800th episode, but also with very individual exploration behaviour, which is that the exploration rate is already very close to the minimum value (0.01). In the end, the overall time for the RL-agent to complete the task stabilized at around 190 seconds, which saves about 42% of the time compared with the time consumption of nearly 330 seconds at the beginning. It took a total of 207633.25 seconds to train the model for 1000 episodes, which is about 57.68 hours.

Table 2. *Q-table for static environment (1000 episodes)*

	waypoint 0	waypoint 1	waypoint 2	waypoint 3	waypoint 4	waypoint 5
Starting point	-19.210000102398944	-26.057589760170394	-50.17767999665183	-53.004800128	-31.209999999993183	-15.207756804259843
waypoint 0	0.0	-21.19870975672319	-41.176384639867884	-74.20999987302399	-50.170063238302156	-33.27378240157498
waypoint 1	-39.20040000000001	0.0	-28.19999991808	-45.209679999139844	-30.2019999744	-46.207879168000005
waypoint 2	-58.376451182011415	-36.208000005270684	0.0	-41.24198157441198	-50.209999999962776	-73.03590480820856
waypoint 3	-70.20902249025535	-68.84034826980688	-45.85004816814059	0.0	-37.20987250781323	-46.20999795199968
waypoint 4	-50.17160014679955	-47.169280183535996	-57.18439180800106	-35.20999999659235	0.0	-29.009999589567357
waypoint 5	-27.201923195083165	-32.08959859153357	-57.20966719931209	-56.896717796383925	-47.3301635262122	0.0

Table 2 shows the Q table after training, the columns represent the 7 states in the state space, and the rows represent the 6 actions in the action space. The robot finally chose such a "shortest time-consuming path": Starting point→waypoint 5→waypoint 0→waypoint 1→waypoint 2→waypoint 3→waypoint 4.

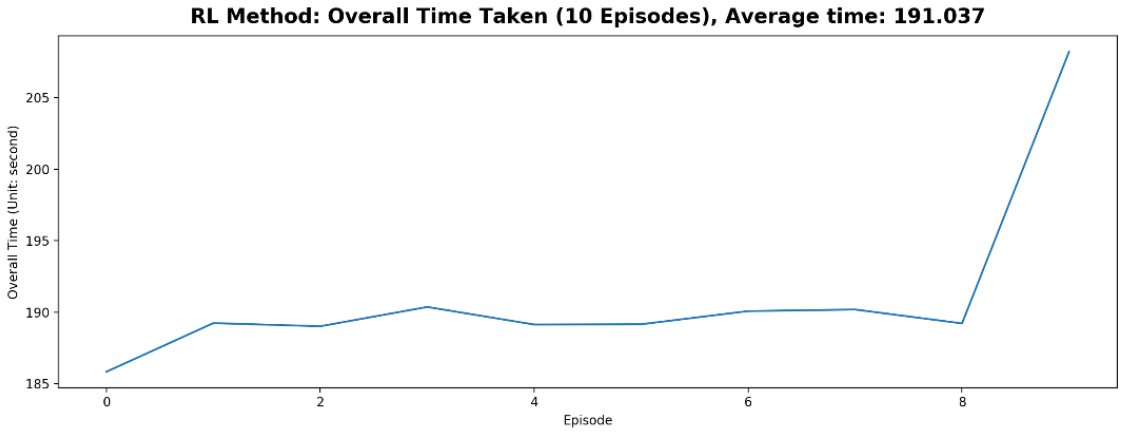

Figure 19. Test results of RL-agent in static environment (10 episodes)

It is worth noting that the RL-agent will not select the waypoint corresponding to the current state as the next action target, so the Q value corresponding to the same state and action in the Q table is 0.0 to avoid selecting repeated actions.

Figure 19 shows the performance after testing 10 episodes on the RL-agent that has been trained on 1000 episodes. Except for the last episode where the time fluctuated greatly, in the remaining episodes, the overall time for the RL-agent to complete the task was controlled within 190 seconds. Finally, the average time taken by RL-agent to complete the multi-waypoint path planning task is 191.037 seconds.

5.1.2 Greedy Method

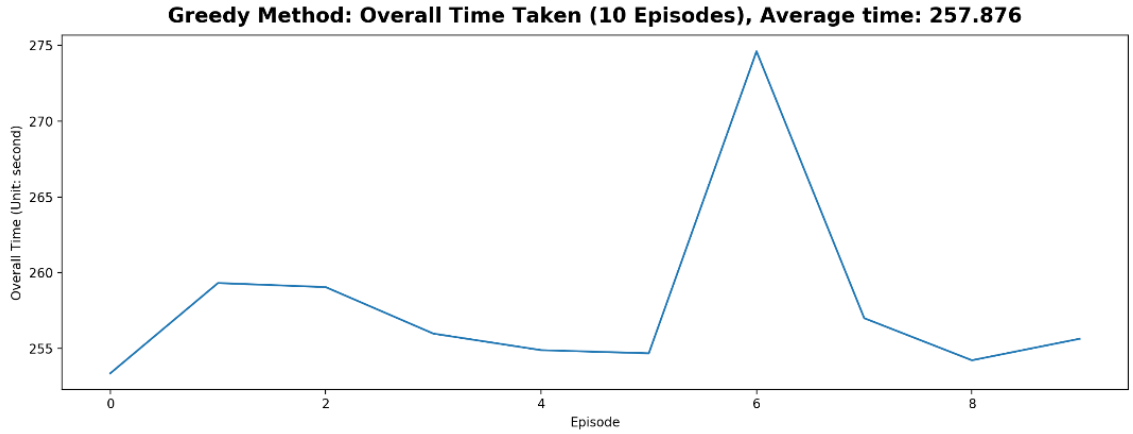

Figure 20. Test results of Greedy method in static environment (10 episodes)

As shown in Figure 20 that after the Greedy method has been tested with the same 10 episodes as the RL-agent, the average time to complete the task is 257.876 seconds, which is 1 minute (66 seconds) more than the average time of the RL-agent. It can be seen that the RL-agent has better decision-making performance than the Greedy method in a static environment without non-recording obstacles.

5.2 Dynamic Environment

The RL-agent model is trained based on a dynamic environment with 7 moving obstacles as the one introduced in section 4.2.2, and test the performance of the RL-agent and Greedy method. In the training phase, the RL-agent will guide the robot to find the Q-table of the global optimal solution after 1000 episodes of training output. In the testing

phase, both RL-agent and Greedy method will go through 10 episodes (rounds) and calculate the average time for completing the task. This average time will be used to compare the decision-making performance of these two algorithms.

5.2.1 RL-Agent

Figure 21 also shows three blocks of results, namely rewards over training, overall time taken over training, and exploration rate.

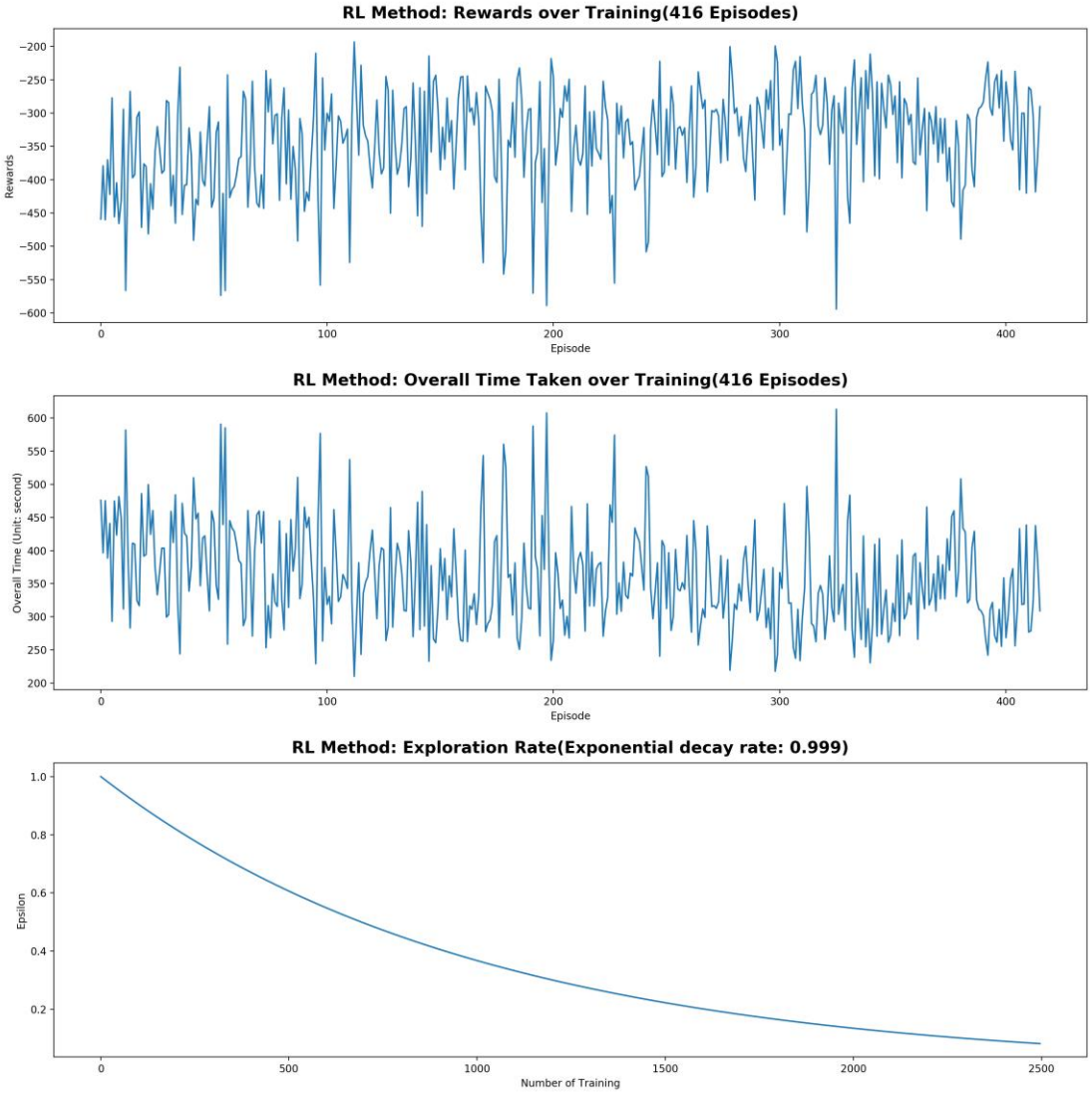

Figure 21. *RL-agent training results in dynamic environment, including rewards, overall time and exploration rate*

This thesis did not complete the RL agent training in the dynamic environment before the deadline for submission, so the latest process results are shown in Figure 22, the model has undergone 416 episodes of training. Due to the addition of moving obstacles in the environment, the time consumed by RL agent to complete the multi waypoint path planning task increases a lot. At the same time, the convergence speed of the model also slows down. The overall time taken is reduced from about 600 seconds to around 250 seconds, but still unstable because RL-agent is still exploring. Moreover, the training of 416 episodes has taken more than 4 days which is unexpected in this thesis.

Table 3. *Q-table for dynamic environment (1000 episodes)*

	waypoint 0	waypoint 1	waypoint 2	waypoint 3	waypoint 4	waypoint 5
Starting point	-19.201615363678208	-55.24110549103463	-76.19612056536009	-112.05982144421033	-31.249861763831163	-20.362000603037103
waypoint 0	0.0	-81.15757370713992	-64.21162551811408	-92.3615112075828	-56.81357591901124	-33.47713145738528
waypoint 1	-52.15687130050675	0.0	-57.934996552390395	-65.5018163719296	-36.18489658559909	-44.89266286634933
waypoint 2	-58.624412867339096	-55.77481685372965	0.0	-66.94506511459133	-73.11369549795354	-98.86488703820464
waypoint 3	-137.05631161662913	-149.72082033836136	-50.982097868946845	0.0	-73.95573807562474	-79.31558275034854
waypoint 4	-52.268951359485925	-78.88139561190125	-87.99701734322022	-80.13653916205966	0.0	-53.63990480065016
waypoint 5	-142.40995136012396	-140.3403964632179	-109.65678352146963	-69.37683748512646	-185.72678117995187	0.0

Table 3 shows the Q table after training, the columns represent the 7 states in the state space, and the rows represent the 6 actions in the action space. Starting point→waypoint 0→waypoint 5→waypoint 3→waypoint 2→waypoint 1→waypoint 4. Compared with the "optimal" path obtained in the static environment, after starting from the starting point, the robot first selects the waypoint 0 which has no moving obstacles, and then goes to the waypoint 5 which has only one moving obstacle. From this behaviour, RL agent shows the ability of "pre perception" for the traffic jam area.

However, the robot then chooses to go to waypoint 3 which has three moving obstacles instead of the nearest waypoint 4 with no moving obstacles. This is most likely because the model has not converged yet. And the reason can be known only after the training of more episodes.

5.2.2 Greedy Method

In the simulation environment with moving obstacles spawned, MiR100 robot appears to be unable to recover navigation with a certain probability when it is blocked by moving obstacles, and the robot is knocked down by the “remnants” of obstacles due to the defects of Gazebo simulator. At this time, a program for restarting the episode needs to be manually set. The test speed of algorithm is particularly slow due to multiple restarts

(about one episode per hour). Therefore, the test of greedy method in dynamic environment has not been completed as expected.

6 Conclusion

In this thesis, Reinforcement Learning, more precisely Q-learning algorithm, is applied to multi-waypoint path planning in dynamic environment. Its objective is to react to the "traffic congestion area" set with unknown moving obstacles, which are not recorded in the global planner. Different from that most related research, here, the shortest time rather than the shortest distance is regarded as the key indicator to measure the performance of the algorithm. Moreover, the robot is required to reach all the waypoints before the task is completed instead of reaching as many waypoints as possible.

The main idea of this thesis is to retain the traditional global and local planner used for single-waypoint navigation but use RL-agent to be responsible for multi-waypoint path planning as the upper layer. So here, the multi-waypoint path planning problem is transformed into TSP problem instead of navigation problem. For comparison, greedy algorithm was selected to be tested in static and dynamic environments together with reinforcement learning.

In this thesis, a learning architecture has been developed, that integrates the move_base [19] Navigation Stack in the ROS framework and a simulation environment based on gazebo with a self-developed dynamic obstacle simulator.

In the static environment, RL-agent is trained in the low complexity environment without dynamic obstacles. By comparing the overall time taken by greedy method and RL-agent to complete the same multi-waypoint path planning task, it can be found that there is a significant difference between those two: Although RL agent takes about one minute less than the greedy method, RL agent needs nearly two and a half days of training while the greedy method does not.

In the dynamic environment, RL-agent is trained in a more complex environment, in which the moving obstacles are not recorded in the global planner, and they are all in

different positions at different time stamps. Here, the gap between the overall time of RL agent and the greedy method to complete the task is further widened, although the overall time taken of both to complete the task is inevitably increased due to the introduction of moving obstacles.

In addition, the performance bottleneck of both hardware and simulation engine also has a significant impact on the overall time of training. As a physical engine simulator, gazebo can only run programs with a single core in the CPU, and Q learning itself follows an empirical learning mode, which determines that it cannot realize parallel operation. During the training process of single episode, ROS messages are generated to communicate with Gazebo simulator multiple times per second, and Gazebo listens to these messages and translates them back into its internal format which is responsible for controlling the movement of obstacles. In this thesis, gazebo shows that it is not the best option to handle such "heavy" communication tasks.

To sum up, through the experimental demonstration in Chapter 5, the framework of combining reinforcement learning with traditional heuristic search algorithm to solve the shortest time-consuming multi-waypoint path planning proposed in this thesis has been preliminarily proved to be feasible. Although due to the deadline of the thesis, the optimization of hyperparameters to control the convergence speed of the model is not involves, it can be regarded as one of the future works.

The flexibility of RL's framework is reflected in the fact that it can quickly insert different reward functions for different problems. Therefore, the same algorithm can be used on different variants of TSP that deviate significantly from Euclidean. This thesis is a representative example, that is to consider minimizing the travel time rather than the shortest distance.

Another advantage of reinforcement learning is that when some extremely complex problems cannot be directly solved by mathematical modelling in a partially observed environment, which is common in real world, the RL's framework can deal with those “chaos”.

7 Future Work

An important disadvantage of Q-Learning is that for many real-world problems, the Q-table has too many input entries to learn, sometimes even more than the number of all atoms in the universe. In this case, the subject cannot access and learn all the entries in the Q-table. And many states in the Q-table never occur and take a long time to discover through training. For this thesis, in the application scenario of goods replenishment in the supermarket tested, due to the limitation of the MiR100 robot's transportation capacity, which the number of waypoints will not exceed 10 at most, so Q-learning in this application will not face the problem of dimensional disaster.

7.1 Improvement 1

Adding attributes about obstacles, e.g., "Static", "Dynamic", "None") recognized by the 3D cameras in the observation space, and the action space is changed to update the action every second (select the waypoint to go in the next second).

The advantage of this is that the finer the granularity of the environmental information, the more detailed the representation, which makes the robot's decision-making more flexible. In theory, robots can behave more intelligently—when the robot encounters a "congested area", it will temporarily abandon the current waypoint and choose an opportunity to visit the unreached waypoint in the future.

At the same time, if the position and number of moving obstacles have changed while the static map has not changed, the robot can still adapt to these changes and find the shortest route through all waypoints. In other words, the generalization of "model-free" reinforcement learning will be improved.

7.2 Improvement 2

More broader application scenarios should be considered in the future. For example, a large-scale unmanned express delivery robot will carry hundreds of packages to different addresses every day. It will face the congestion caused by tens of thousands of moving obstacles in the city. The congested area changes with time. If the delivery robot wants to avoid the congested area, it involves the processing of time series data. The objectives of

delivery robots may be multiple, such as saving electricity, shortening delivery time, and completing as many delivery goals as possible. So, both the environmental inputs and the multi-objective become incredibly large and complex.

In the face of the exponentially increasing dimensional problem of environmental input information, Deep Q-Learning (DQN) proposed by the DeepMind team in 2013 [36] provides an effective way to solve the problem of dimensional disaster [43]. Ideally, one would also want to have a good guess at the Q-values of never-before-seen states. Due to the shortcomings of Q-Learning, people tend to use Deep Q Network (also known as Deep Q-Learning) to solve this problem. As a combination of deep learning and reinforcement learning, DQN uses a neural network instead of a Q-table, which takes the state of the agent as input and returns the Q-values corresponding to all the actions the agent can take. In this case, when using good learning techniques such as experience replay and target network separation, it can well approximate the untraversable Q-table using a simple neural network model.

However, for multi-objective problems like multi-waypoint path planning, designing a reward function that can efficiently balance overall time taken, obstacle avoidance, and navigation can be a challenging work. As recently debated by academic groups [36], it's worth noting that this would be a semi-blind manual process, and it compromises the interpretability of the decision-making process which would place an undue burden on the engineer to understand the real-world decision problem at hand.

Reference

[1] N. Ahmed, C. Pawase and K. Chang. “Distributed 3-D Path Planning for Multi-UAVs with Full Area Surveillance Based on Particle Swarm Optimization”. In: *Applied Sciences, vol. 11, no. 8, p. 3417*. 2021.

[2] C. Zeng, Q. Zhang and X. Wei, “GA-based Global Path Planning for Mobile Robot Employing A* Algorithm”, In: *Journal of Computers, vol. 7, no. 2*. 2012.

[3] J. Guo, C. Li and S. Guo, “A Novel Step Optimal Path Planning Algorithm for the Spherical Mobile Robot Based on Fuzzy Control”. In: *IEEE Access, vol. 8, pp. 1394-1405*. 2020.

[4] S. Wang, J. Zhang and J. Zhang, “Artificial potential field algorithm for path control of unmanned ground vehicles formation in highway”, In: *Electronics Letters, vol. 54, no. 20, pp. 1166-1168*. 2018.

[5] V. Sangeetha et al. “A Fuzzy Gain-Based Dynamic Ant Colony Optimization for Path Planning in Dynamic Environments”. In: *Symmetry, vol. 13, no. 2, p. 280.* 2021.

[6] M. Li, G. Hua and H. Huang. “A Multi-Modal Route Choice Model with Ridesharing and Public Transit”. In: *Sustainability, vol. 10, no. 11, p. 4275*. 2018.

[7] H. Zhuang, K. Dong, Y. Qi, N. Wang and L. Dong, “Multi-Destination Path Planning Method Research of Mobile Robots Based on Goal of Passing through the Fewest Obstacles”. In: *Applied Sciences, vol. 11, no. 16, p. 7378*. 2021

[8] P. Wei. “Robot Motion Planning Algorithm Based on Deep Reinforcement Learning”. In: *Bachelor thesis, China Agricultural University*. 2021

[9] R. Güldenring. “Applying Deep Reinforcement Learning in the Navigation of Mobile Robots in Static and Dynamic Environments”. In: *Master Thesis, Universität Hamburg, April*. 2019

[10] T. Costa. *Solving the Traveling Salesman Problem with Reinforcement Learning | Eki.Lab*. [Accessed: 31-07-2022] URL: https://ekimetrics.github.io/blog/2021/11/03/tsp/#q-learning-for-a-tsp-with-traffic-zones.

[11] J. Herzen. *Routing Traveling Salesmen on Random Graphs using Reinforcement Learning, in PyTorch*. [Accessed: 01-08-2022]. URL: https://medium.com/unit8-machine-learning-publication/routing-traveling-salesmen-on-random-graphs-using-reinforcement-learning-in-pytorch-7378e4814980.

[12] Amazon Sagemaker. *Traveling Salesman Problem with Reinforcement Learning — Amazon SageMaker Examples 1.0.0 documentation*. [Accessed: 01-08-2022]. URL: https://sagemaker-examples.readthedocs.io/en/latest/reinforcement_learning/rl_traveling_salesman_vehicle_routing_coach/rl_traveling_salesman_vehicle_routing_coach.html.

[13] Mobile Industrial Robots A/S. *MiR100*. [Accessed: 14-07-2022]. URL: https://www.mobile-industrial-robots.com/solutions/robots/mir100/

[14] Mobile Industrial Robots A/S. *MiR100 User Guide.* [Accessed: 14-07-2022]. URL:https://gibas.nl/wp-content/uploads/2021/01/mir100-user-guide_31_en.pdf, p.11.

[15] Mobile Industrial Robots A/S. *MiR100 User Guid*e. [Accessed: 14-07-2022]. URL:https://gibas.nl/wp-content/uploads/2021/01/mir100-user-guide_31_en.pdf, p.60.

[16] Mobile Industrial Robots A/S. *MiR100 User Guide*. [Accessed: 14-07-2022]. URL:https://gibas.nl/wp-content/uploads/2021/01/mir100-user-guide_31_en.pdf, p.65.

[17] Mobile Industrial Robots A/S. *MiR100 User Guide*. [Accessed: 14-07-2022]. URL:https://gibas.nl/wp-content/uploads/2021/01/mir100-user-guide_31_en.pdf, p.54.

[18] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," In: *ICRA Workshop on Open-Source Software*. 2009.

[19] E. M.-Eppstein. *Move_base package*. [Accessed: 16-07-2022]. URL: http://wiki.ros.org/move_base

[20] M. Phillips. *sbpl_lattice_planner package*. [Accessed: 16-07-2022]. URL: http://wiki.ros.org/sbpl_lattice_planner

[21] Search-Based Planning Lab. *sbpl package*. [Accessed: 17-07-2022]. URL: https://wiki.ros.org/sbpl

[22] Search-Based Planning Lab. *Kinematic Constraints and Motion Primitives*. [Accessed: 17-07-2022]. URL: http://sbpl.net/node/48

[23] E. M.-Eppstein. *move_base package*. [Accessed: 17-07-2022]. URL: http://wiki.ros.org/move_base?action=AttachFile&do=view&target=overview_tf.png.

[24] M. Likhachev, G. Gordon and S. Thrun. "ARA*: Anytime A* with Provable Bounds on Sub-Optimality", In: *Advances in Neural Information Processing Systems 16: 6: Proceedings of the 2003 Conference (NIPS-03)*. 2004

[25] D. Lu. *dwb_local_planner package*. [Accessed: 18-07-2022]. URL: https://github.com/locusrobotics/robot_navigation/tree/master/dwb_local_planner

[26] R. C. Coulter. "Implementation of the Pure Pursuit Path Tracking Algorithm". In: *Tech. Report, CMU-RI-TR-92-01, Robotics Institute, Carnegie Mellon University, January*. 1992

[27] I. Ulrich and J. Borenstein. "VFH*: Local Obstacle Avoidance with Look-Ahead Verification". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), vol.* 3. April 2000, pp. 2505–2511 vol.3. 2000

[28] M. M. Flood. "The Traveling-Salesman Problem". In: *Operation Research, Vol.4, No. 1*. 1956

[29] D. Raju. *Travelling Salesman Problem via the Greedy Algorithm*. [Accessed: 18-07-2022]. URL: https://medium.com/ivymobility-developers/algorithm-a168afcd3611.

[30] Z. Ullah, Z. Xu, Z. Lei, L. Zhang, W. Ullah. "RL and ANN Based Modular Path Planning Controller for Resource-Constrained Robots in the Indoor Complex Dynamic Environment". In: *IEEE Access. PP. 1-1. 10.1109*. 2018

[31] E. Tzorakoleftherakis. *Three Things to Know About Reinforcement Learning*. [Accessed: 19-07-2022]. URL: https://www.kdnuggets.com/2019/10/mathworks-reinforcement-learning.html.

[32] S. Bhatt. *Reinforcement Learning 101: Learn the essentials of Reinforcement Learning*. [Accessed: 19-07-2022]. URL: https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292.

[33] T. Matiisen. *Demystifying Deep Reinforcement Learning*. [Accessed: 21-07-2022]. URL: https://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/.

[34] C. Shyalika. *A Beginners Guide to Q-Learning*. [Accessed: 21-07-2022]. URL: https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c.

[35] S. Thomas. *Diving deeper into Reinforcement Learning with Q-Learning (freeCodeCamp.org, 2018)*. [Accessed: 19-07-2022]. URL: https://www.freecodecamp.org/news/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe.

[36] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. "Playing Atari with deep reinforcement learning". In: *ArXiv preprint arXiv:1312.5602*. 2013

[37] S. Hougardy and X. Zhong. "Hard to solve instances of the Euclidean Traveling Salesman Problem". In: *Mathematical Programming Computation, vol. 13, no. 1, pp. 51-74*. 2020.

[38] N. Koenig and A. Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), vol. 3 pp. 2149–2154*. 2004

[39] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba. "OpenAI Gym". arXiv:1606.01540 [cs.LG], 5 Jun. 2016

[40] Parkinson, A. *The Epsilon-Greedy Algorithm for Reinforcement Learning*. [Accessed: 07-08-2022]. URL: https://medium.com/analytics-vidhya/the-epsilon-greedy-algorithm-for-reinforcement-learning-5fe6f96dc870.

[41] baeldung. *Epsilon-Greedy Q-learning*. [Accessed: 28-07-2022]. URL: https://www.baeldung.com/cs/epsilon-greedy-q-learning

[42] D. Xiang, H. Lin, J. Ouyang and D. Huang. "Combined improved A* and greedy algorithm for path planning of multi-objective mobile robot". In: *Scientific Reports, vol. 12, no. 1*. 2022.

[43] A. H. Qureshi, A. Simeonov, M. J. Bency, M. C. Yip. "Motion planning networks". In: *2019 International Conference on Robotics and Automation*. 2019.

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Yuzhou Liu

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Using Reinforcement Learning for Multiple Way-points Path Planning of Single Mobile Robot in the Dynamic Obstacle Environment”, supervised by Andreas Bresser, Laura Piho, and Pascual Campoy
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

08.08.2022

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.