Mobile Robot Obstacle Avoidance based on Deep Reinforcement Learning

by Shumin Feng

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
In
Mechanical Engineering

Pinhas Ben-Tzvi, Chair
Erik Komendera
Tomonari Furukawa

December 12, 2018
Blacksburg, Virginia

Keywords: Robotic Systems, Neural Networks, Obstacle Avoidance, Deep Reinforcement Learning

Mobile Robot Obstacle Avoidance based on Deep Reinforcement Learning

by Shumin Feng

ABSTRACT

Obstacle avoidance is one of the core problems in the field of autonomous navigation. An obstacle avoidance approach is developed for the navigation task of a reconfigurable multi-robot system named STORM, which stands for Self-configurable and Transformable Omni-Directional Robotic Modules. Various mathematical models have been developed in previous work in this field to avoid collision for such robots. In this work, the proposed collision avoidance algorithm is trained via Deep Reinforcement Learning, which enables the robot to learn by itself from its experiences, and then fit a mathematical model by updating the parameters of a neural network. The trained neural network architecture is capable of choosing an action directly based on the input sensor data using the trained neural network architecture. A virtual STORM locomotion module was trained to explore a Gazebo simulation environment without collision, using the proposed collision avoidance strategies based on DRL. The mathematical model of the avoidance algorithm was derived from the simulation and then applied to the prototype of the locomotion module and validated via experiments. Universal software architecture was also designed for the STORM modules. The software architecture has extensible and reusable features that improve the design efficiency and enable parallel development.

Mobile Robot Obstacle Avoidance based on Deep Reinforcement Learning

by Shumin Feng

GENERAL AUDIENCE ABSTRACT

In this thesis, an obstacle avoidance approach is described to enable autonomous navigation of a reconfigurable multi-robot system, STORM. The Self-configurable and Transformable Omni-Directional Robotic Modules (STORM) is a novel approach towards heterogeneous swarm robotics. The system has two types of robotic modules, namely the locomotion module and the manipulation module. Each module is able to navigate and perform tasks independently. In addition, the systems are designed to autonomously dock together to perform tasks that the modules individually are unable to accomplish.

The proposed obstacle avoidance approach is designed for the modules of STORM, but can be applied to mobile robots in general. In contrast to the existing collision avoidance approaches, the proposed algorithm was trained via deep reinforcement learning (DRL). This enables the robot to learn by itself from its experiences, and then fit a mathematical model by updating the parameters of a neural network. In order to avoid damage to the real robot during the learning phase, a virtual robot was trained inside a Gazebo simulation environment with obstacles. The mathematical model for the collision avoidance strategy obtained through DRL was then validated on a locomotion module prototype of STORM. This thesis also introduces the overall STORM architecture and provides a brief overview of the generalized software architecture designed for the STORM modules. The software architecture has expandable and reusable features that

apply well to the swarm architecture while allowing for design efficiency and parallel development.

ACKNOWLEDGMENTS

Firstly, I would like to thank my parents for their support and love throughout my years of study. They encourage and support me whenever I face any challenges and difficulties in my life and study. I would like to express profound gratitude to my advisor, Dr. Pinhas Ben-Tzvi for helping and supporting my research and giving me the chance to be a part of his Robotics and Mechatronics Lab at Virginia Tech. I am grateful for my colleagues in our lab for giving me a hand whenever I met difficulties in research. I would like to thank my committee members, Dr. Erik Komendera and Dr. Tomonari Furukawa for serving on my Master's Advisory Committee and helping me to complete my Master’s degree.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

Autonomous navigation is one of the major research topics in the field of mobile robotics. Many robotic applications, such as rescue, surveillance, and mining require mobile robots to navigate an unknown environment without collision. For fully controlled robots, the navigation task [1] can be achieved by a human operator who controls the robot by sending commands to the mobile robot via cable or wireless communication. However, this mode of operation is of limited use in hazardous environments where cable and wireless communication are unable to be set up. Thus, it is necessary for a mobile robot to navigate autonomously in some situations. Additionally, improving the level of autonomy is the current research trend in response to the increasing demand for more advanced capabilities of mobile robots. Advancing autonomous navigation has become one of the most significant missions for improving the human living standard, minimizing human resource requirements and raising work efficiency.

Generally, a global collision-free path from the current location of the robot to the goal position can be planned if an accurate map of the environment is provided. Many approaches were developed to solve global path planning problems, such as A* searching algorithm [2], or Probabilistic Roadmap Method (PRM) [3][4], which rely on prior information of the environment. However, in the real world, an environment is not always static, and in a dynamic environment, robots need to take into account the unforeseen obstacles and moving objects. Local path planning or obstacle avoidance algorithms, such as the potential field method [5] and dynamic window approach [6] can be utilized to avoid collisions in a dynamic environment.

Under some circumstances, no map is available in advance for a robot to implement the navigation task. When exploring an environment without any prior knowledge, obstacle avoidance becomes crucial for autonomous mobile robots to navigate without collision. Although there are many solutions to obstacle avoidance problems, decide the correct direction to avoid collision in a challenge environment without map is still difficult. The proposed obstacle avoidance approach is an attempt to improve the performance of autonomous robot when traversing in such environment. The rest of this chapter presents our contributions and an outline for the following chapters.

1.2 Contributions

In this thesis, an obstacle avoidance approach based on Deep Reinforcement Learning (DRL) was developed for the map-less navigation of autonomous mobile robots. Although the proposed approach is developed for the individual navigation task of a reconfigurable robotic system named STORM, which stands for the Self-configurable and Transformable Omni-Directional Robotic Modules (STORM), it is a generalized method and can be applied to any mobile robot if necessary sensory data is available. The method was applied to the prototype of a locomotion module belonging to STORM and validated via experiments. We also present an overview of the STORM robotic system, along with the electronic system, the software architecture, and the system model of the locomotion module. Our main contributions can be listed as follows:

1. DRL was utilized to solve the obstacle avoidance problem, which enables the robot to learn by itself from its experiences, and then find the optimal behavior when interacting with the environment.

2. A multi-stage training method was proposed to train the neural network for goal-oriented navigation with obstacle avoidance.

3. A simulator was created as the 3D training environment for the obstacle avoidance approach. It is necessary to collect a large-scale data to train the neural network. In the real world, it is impractical to acquire such a large amount of transition data without damaging the robot. The simulator can also be used as the testbed for the robot.

4. An electronic system was designed for the locomotion module.

5. An extensible and reusable software system was developed for the robot. The software architecture has three layers, which divides the robotic software system into small modules. This feature together with the publish-subscribe pattern enables modular design and parallel development without complete knowledge of the whole system to improve the work efficiency.

1.3 Thesis Structure

The structure of this thesis is listed as follows:

Chapter 1: Presents the background and main contributions of this thesis.

Chapter 2: Reviews current literature in the field and discusses the state of the art obstacle avoidance algorithms.

Chapter 3: States the problems exist in the previous work in this area. In addition, different DRL applications together with some basic methods to solve DRL problems are introduced in this chapter to show the capability and potential of DRL as the solution to avoid collision in challenging scenarios.

Chapter 4: Presents the proposed obstacle avoidance approach in detail.

Chapter 5: Presents the overview of the STORM system, along with the mechanical design, electronic system, and software architecture of the STROM locomotion module.

Chapter 6: Describes the system model of the STORM locomotion module.

Chapter 7: Presents the simulation and experimental results of the proposed obstacle avoidance approach.

Chapter 8: Shows the conclusion and future work.

1.4 Publication

Disclosure: Content from the publication was used throughout this thesis.

Journal Papers

S. Feng, B. Sebastian, S. S. Sohal and P. Ben-Tzvi, "Deep reinforcement learning based obstacle avoidance approach for a modular robotic system", Journal of Intelligent and Robotic Systems, Submitted, December 2018.

CHAPTER 2

LITERATURE REVIEW

2.1 Autonomous Navigation

In general, the autonomous navigation task for a mobile robot is composed of several subtasks, such as localization, path planning, and obstacle avoidance. A global path planner is commonly utilized to plan an optimal collision-free path from start to end point for a robot, using a priori knowledge of the environment. However, the trajectories planned by these type of planners are unable to adapt to the changes of environment. In this regard, local path planners, or obstacle avoidance controllers are introduced for the autonomous mobile robot to interact with the dynamic environment according to sensory information.

2.2 Global Path Planning

Global Path Planning is defined as planning a collision-free path intelligently from a starting position to a goal position for mobile robots or automobiles. The trajectory is generated by a global path planner offline from a well-defined map of the environment.

A* searching algorithm is one of the popular global path planning approaches to find out the shortest path based on graph search methods by searching the nodes in a two-dimensional grid from the starting point and identifying the appropriate successor node at each step. A heuristic algorithm is utilized to guide the search while computing a path with minimum cost. The following equation is used to calculate the cost,

$$f(n) = h(n) + g(n) \tag{2.1}$$

where n is the next node on the path, $h(n)$ is the heuristic cost that estimates the cost of the cheapest path from n to the goal, and $g(n)$ is the path cost that recorded the cost from the start node to the current node n. Initially, two empty lists are created to store the nodes. One is the open list which is used to store the currently discovered nodes that are not evaluated yet. Another is the closed list that stores all the nodes that have already been evaluated. The following procedures should be done to find the optimal path.

1. Put the start point in the closed list and put the walkable (not an obstacle node) neighbors in the open list and set the start point as their parent;

2. Repeat the following steps if the open list is not empty;

3. The nodes in the open list with the lowest $f(n)$ is considered as the current node and will be moved to the closed list;

4. For the walkable nodes that are not in the closed list and adjacent to the current node, calculate the $g(n)$, $h(n)$, and $f(n)$ of the nodes. Check if the nodes have already been added to the open list. If not, add it to the open list and set the current node as its parent. If it is already on the open list, compare the $g(n)$ with the $g(n-1)$. If $g(n)$ is less than $g(n-1)$, set the current node as it's parent and recalculate the $g(n)$, $h(n)$, and $f(n)$ of that node;

5. Once the target node can be added to the closed list, work backward from the target node and go from each node to its parent square until reaching the starting node. The resulting path is the optimal solution.

Similar methods such as Dijkstra's Algorithm and Greedy Best First Search are able to find out the optimal path as well. The main differences are that the cost function of Dijkstra's Algorithm and Greedy Best First Search are $f(n) = g(n)$ and $f(n) = h(n)$, respectively. The Dijkstra's Algorithm works well to find the shortest path, but it will take more time since it explores all the possible path even if they are already expensive. Greedy Best First Search can be fast to find out the optimal path, but sometimes it can get stuck in loops. The A* algorithm uses both the actual distance from the start and the estimated distance to the goal, which means that it combined the advantages of the two algorithms. It avoids expanding paths that are already expensive and expands most promising paths first. Although some global path planning algorithms are able to find the optimal path for the mobile robot, global path planning methods have some obvious drawbacks. Primarily, prior knowledge of the environment is always required when planning the path. As a result, the derived path will be not reliable if the map is not accurate enough. Also, in the real world the environment varies with time, and so with global planning approaches the mobile robot is not able to adapt to the changes in the environment.

Thus, a global path planner is not capable of planning a collision-free path without a high precised map. Unforeseen obstacles will cause troubles if the mobile robot takes motions according to the fixed path that is planed at the beginning. Current environment information is necessary for a mobile robot if no prior knowledge is provided or unexpected obstacles exist in the environment. Some obstacle avoidance methods, also known as local path planner, are developed for mobile robots to take actions according to the online sensory information.

Sample-Based Planning (SBP) is another popular category of global path planning methods [7]. In general, SBP operates in the C-space that is the configuration space of all possible transformations that could be applied to a robot. The C-space, C can be divided to free space, C_{free} and obstacle space, C_{obs}. The inputs to the planner are the start configuration q_{start} and goal configuration q_{goal}. A collision-free path P_{free} that connects q_{start} to q_{goal} lies in the C_{free} space can be found out after performing the SBP. The Probabilistic Roadmap Method (PRM) [3][4] and Rapidly-exploring Random Trees (RRT) [8] are the typical SBP methods.

The primary procedures of PRM can be listed as follows:

1. Select a random node q_{rand} from the C-space;

2. Add the q_{rand} to the roadmap if it is in C_{free};

3. Find all nodes within a specific range to q_{rand};

4. Connect all neighboring nodes to q_{rand};

5. Check for collision and disconnect colliding paths;

6. Repeated the above procedures until a certain number of nodes have been sampled to build a roadmap;

7. Performing graph search algorithm to find the shortest path through the roadmap between q_{start} and q_{goal} configurations.

The following shows the procedures of RRT:

1. Select a random node q_{rand} from the C-space in the C_{free} region;

2. Find the nearest node q_{near} to q_{rand} from the current roadmap;

3. Connect q_{near} and q_{rand}. If the length of the connection exceeds the maximum value, a q_{new} at the maximum distance from the q_{near} along the line to the q_{rand} is used instead of q_{rand} itself.

4. Perform collision checking to ensure the path between q_{new} and q_{near} is collision free. If the path is collision free, add q_{new} to the roadmap.

5. Repeat the above procedures until $q_{new} = q_{goal}$.

2.3 Obstacle Avoidance

2.3.1. Artificial Potential Field

Some local path planning methods are based on the Artificial Potential Field (APF) approach. In these type of algorithms, the target is usually represented as an attractive potential field U_{target}, and the obstacles are defined as repulsive potential fields U_{obs_i}. The resultant potential field is produced by summing the attractive and repulsive potential fields as follows:

$$U = U_{\text{target}} + \sum U_{obs_i} \tag{2.2}$$

To adopt these type of methods, the distances from the obstacles to the goal position are selected as input, and the output is given as the direction of the force generated by the potential field. The mobile robot will take the motions along the direction of that force generated by the potential field. The potential field based methods are straightforward and capable of overcoming unknown scenario, but they have some significant drawbacks [9]:

1. The robot may be trapped in local minima away from the goal.

2. The robot is not able to find a path between closely spaced obstacles.

3. Oscillations occur in the presences of the obstacles and in the narrow passages.

To improve the performance, some algorithms based on the potential field method were developed, as the Virtual Force Field (VFF) method [10], Virtual Field Histogram (VFH) and its' inherent approaches [11][12][13]. The VFF is able to eliminate the local minima problem, but it still cannot pass through narrow paths.

2.3.2. Virtual Field Histogram

The VFH uses a two-dimensional Cartesian histogram grid as a world model that is updated continuously with range data sampled by onboard range sensors [13]. Each cell in the histogram grid holds a certainty value that represents the confidence of the algorithm in the existence of an obstacle at that location.

The VFH utilizes a two-stage data reduction method to process the data. In the first step, the raw inputs for the range sensor will be converted into a 2D Cartesian coordinate system. Each cell in this coordinate contains a certainty value c_{ij}, that is proportional to the magnitude of a virtual repulsive force calculated by the distance measurement obtained from the sensor. Then the 2D coordinate can be reduced to a 1D histogram grid where the certainty value is replaced with an obstacle vector. The following equations are utilized to calculate the direction β_{ij} and the magnitude m_{ij} of the obstacle vector. Where, the direction is taken from the center of the robot to the active cell (x_i, y_j):

$$\beta_{i,j} = \arctan(\frac{y_j - y_0}{x_i - x_0}) \tag{2.3}$$

$$m_{ij} = (c_{ij})^2 (a - bd_{ij}) \tag{2.4}$$

Here, (x_0, y_0) indicates the center of the robot, a and b are the positive constants and d_{ij} is the distance between the active cell and the center of the robot. The 1D polar histogram can be divided into small sectors evenly. For each sector k, the polar obstacle density, h_k can be calculated with the following equation:

$$k = (\text{int})\frac{\beta_{ij}}{\alpha} \tag{2.5}$$

$$h_k = \sum m_{ij} \tag{2.6}$$

The h_k is the sum of all the magnitudes of the obstacle vectors in the sector k. The closest sector k_n to the target direction with a low obstacle density will be selected as the steering angle.

Although the main drawbacks can be overcome with the related methods, deciding the correct direction is still difficult in problematic situations [12]. Combining the global path planner and the local path planner is helpful for dealing with these problematic scenarios. However, it is still unpractical if no prior environment information is provided.

2.3.3. Fuzzy Logic Controllers

Some Fuzzy Logic Controllers (FLC) [14][15] are built and attempt to teach a mobile robot to select the proper action according to the sensory information to improve the performances of the mobile robots in a complex environment. However, the selection of the linguistic variables used for the fuzzy rules and the definition of the membership functions can be more overloaded and tough to handle with the increasing number of the readings from the sensory system.

2.3.4. Dynamic Windows Approach

The dynamic windows approach is another method to solve online path planning problems in dynamic environments. In this method, steering commands are selected by maximizing the objective function, given by:

$$G(v,\omega) = \sigma(\alpha \text{heading}(v,\omega) + \beta \text{dist}(v,\omega) + \gamma \text{vel}(v,\omega)) \tag{2.7}$$

where v and ω are the linear velocity and angular velocity of the robot, respectively; α, β and γ are the weights of the three components: $\text{heading}(v,\omega)$, $\text{dist}(v,\omega)$, and $\text{vel}(v,\omega)$; and the function σ is introduced to smooth the weighted sum.

The maximum of this objective function is computed over a resulting search space V_r that is determined by reducing the search space of the possible velocities. The first step is to find the admissible velocities by which the robot can stop before it reaches the closest obstacle. Then the dynamic window is used to find the reachable velocities.

The advantages of this approach are that it can react quickly and conquer a lot of real-world scenarios successfully. However, this method has a typical problem in real world situations where the robot cannot slow down early enough to enter a doorway.

The problems of the path planning methods will be summarized in detail in the next chapter. In addition, an obstacle avoidance approach based on DRL will be proposed as the solution. In the following chapters, detailed information of developing and implementing the proposed method will be presented.

CHAPTER 3

PROBLEM STATEMENT AND PROPOSED SOLUTION

In the literature review, we presented some current methods for a mobile robots to navigate autonomously without collision. Both the advantages and drawbacks were described for each method. Although some drawbacks have already been solved, problems still exist, and the performance of the robot needs to be improved in more challenging situations.

3.1 Problem Statement

This thesis focuses on navigation problems in an unknown difficult environment such as narrow corridors and underground tunnels. It is assumed that no map is available in these cases and therefore, global path planning methods will not be taken into consideration due to their requirements of prior knowledge of the environment. The local minima problems in APF based approaches have been investigated in numerous studies. It is proven that the local minima problems can be overcome by the VFF, VFH, and any other inherent methods. The performance of the autonomous robot and the obstacle avoidance capability were improved as well. Despite that, these algorithms enable mobile robots to travel without collision in some dynamic environments; a problematic scenario exists where some local path planners may not find the correct direction. It is a common problem in these methods which trigger the avoidance maneuver according to the sensory information within a specific detection range. Once the obstacles are detected in some direction within that range, the local path planners should send commands to the robot

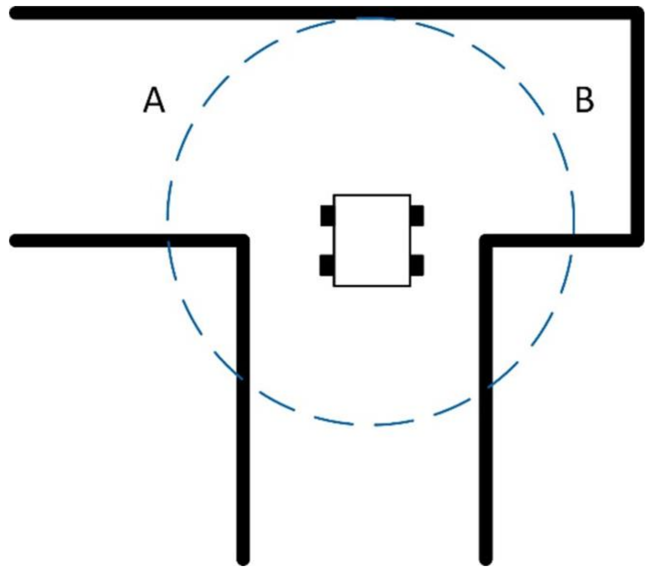

Figure 3.1: Robot in a narrow corridor. The blue circle indicates the trigger distance. Both A and B are the open areas detected by the autonomous robot at the current time step.

and steer it into the direction without obstacles. This may cause problems if the robot is in a problematic situation such as the narrow corridor shown in Figure 3.1. The two openings, A and B, are equally appropriate to local path planner. If the planner chooses to steer in the direction of B, the robot will crash into the wall unless it terminates the current action or moves backward. This problem was first pointed out by Ulrich and Borenstein in [11]. They developed a hybrid path planner by combining the A* search algorithm and VFH to overcome it; however, it still cannot improve the performances of the pure local planner without prior knowledge of the environment.

3.2 Proposed Solution

To overcome this problem without prior knowledge, sensory information is of great importance. The main limitation of the local planners in some previous works is the trigger distance. The key to solve the problem is to develop a new way to trigger the obstacle avoidance maneuver or take all the possible observations from the sensory data into account and specify an action for the robot in each situation. We believed that this can be accomplished with the help of DRL due to its ability to teach an agent to find the optimal behavior from its experiences to respond to thousands of different observations. The following subsections introduce the definition and some applications of DRL. Several elementary RL and DRL methods are overviewed as foundations of the proposed approach.

3.2.1. Applications of DRL

DRL is a combination of reinforcement learning and deep learning techniques. It allows

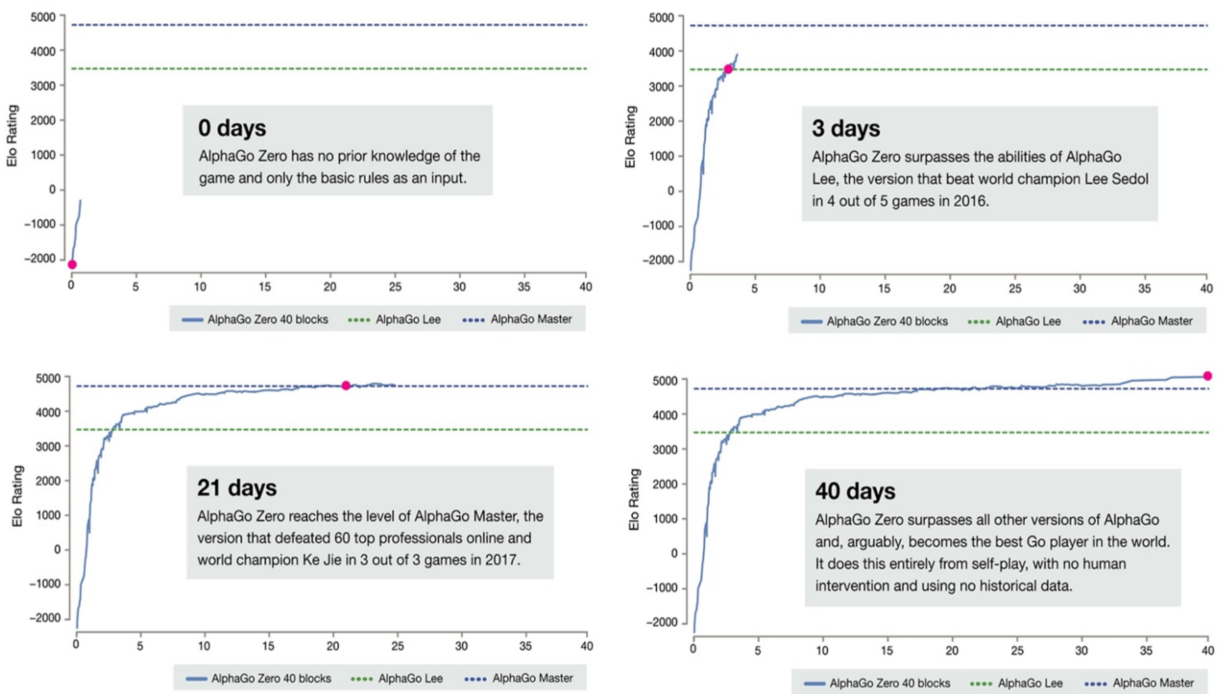

Figure 3.2: The learning process of AlphaGo Zero. The source figures are from [42].

an agent to learn to behave in an environment based on feedback rewards or punishments. A neural network acts as the approximator to estimate the action-value function. A well-known recent application of DRL is AlphaGo Zero [16], a program developed by Google DeepMind. AlphaGo Zero learned to play the game of Go by itself without any human knowledge and successfully won 100-0 against AlphaGo [17], which was the first program to defeat a world champion in the game of Go. Figure 3.2 shows the learning process of AlphaGo Zero.

Researchers are also exploring and applying DRL in robotics [18][19], and believe that DRL has the potential to train a robot to discover an optimal behavior surpassing that taught by human beings in the future. Figure 3.3 shows the application of DRL in robotic manipulation.

3.2.2. Reinforcement Learning (RL)

RL aims to enable an agent to learn the optimal behavior when interacting with the

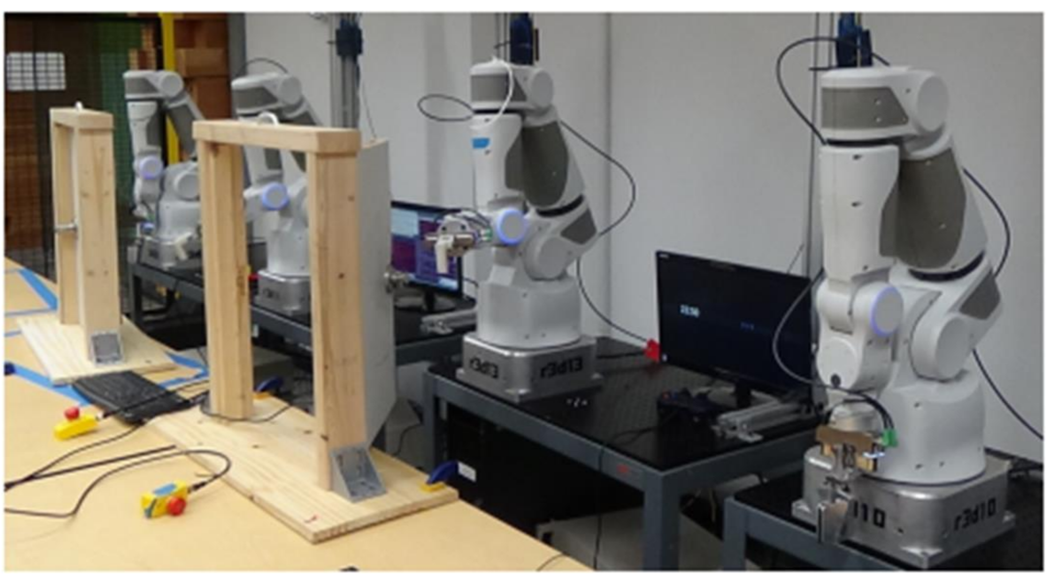

Figure 3.3: Robotic arms learning a door opening task.

provided environment using trial-and-error search and delayed reward [20]. The main elements of an RL problem include an agent, environment states, a policy, a reward function, a value function, and an optional model of the environment. The interaction between the agent and the environment can be simply described as: at each time step t, the agent determines the current environment state s from all possible states S ($s \in S$), then it chooses an action a out of A ($a \in A$) according to the current policy $\pi(s,a)$. The policy can be considered as a map which shows the probabilities of taking each action a when in each state s. After taking the chosen action a, the agent is in a new state s' and receives a reward signal r. The whole process is diagramed in Figure 3.4.

RL is highly influenced by the theory of Markov Decision Processes (MDPs). An RL task can be described as an MDP if it has a fully observable environment whose response depends only on the state and action at the last time step. A tuple $\langle S, A, P, R, \gamma \rangle$ can be utilized to represent an MDP, where S is a state space, A is an action space, P is a state transition model, R is a reward function and γ is a discount factor. Some extensions and generalizations of MDPs are capable of describing partially observable RL problems [21]

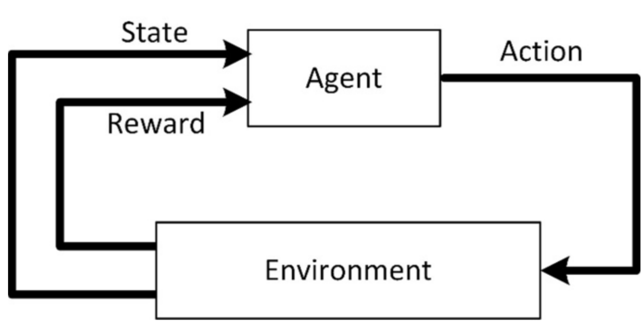

Figure 3.4: The interaction between agent and environment.

or dealing with continuous-time problems [9, 10]. RL algorithms need to be capable of updating the policy due to the training experience and finally determine the policy that fixes one action for each state with the maximum reward.

3.2.3. Q-learning

Q-learning [24] is a fundamental and popular RL algorithm developed by Watkins in 1989. It is capable of finding an optimal policy for an MDP by updating the Q-values table $Q(s,a)$ based on the Bellman Equation:

$$Q(s,a) = r + \gamma(\max_{a'}(Q(s',a')) \tag{3.1}$$

The equation (3.1) can be decomposed into two parts: the immediate reward r and the discounted maximum Q-value of the successor state $\gamma(\max_{a'}(Q(s',a'))$, where γ is a discount factor, $\gamma \in [0,1]$, s' is the successor state and a' is the action to be taken to get the maximum Q-value in the state s'.

3.2.4. Deep Q-networks

Solving an RL problem using linear methods, such as the Bellman equation, is simple and efficient sometimes, but not all situations are suitable for updating the action-value iteratively. Google DeepMind proposed a deep learning model known as DQN[25], which is a convolutional neural network, trained with a variant of Q-learning algorithm with experience replay memory [26], represented as a data set D. The agent's experience at each time step can be stored in D and a minibatch of experiences are sampled randomly when performing updates.

The DQN was tested on several Atari games. In these cases, the game agent needs to interact with the Atari emulator whose states are represented as $s_t = x_1, a_1, x_2, a_2, ..., a_{t-1}, x_t$, where x_t is the image describing the current screen at each time step t and a_t is the chosen action. The states are preprocessed and converted to an $84 \times 84 \times 4$ image as the input of the neural network, defined as the Q-network, which has three hidden layers and a fully-connected linear output layer. The outputs correspond to the predicted Q-values for each valid action. A sequence of loss functions $L_i(\theta_i)$ are adopted to train the Q-network,

$$L_i(\theta_i) = (r \ + \ \gamma(\max_{a'}\left(Q\left(s', a'; \theta_i^{-}\right)\right) - Q(s, a; \theta_i))^2 \tag{3.2}$$

where $r \ + \ \gamma(\max_{a'}\left(Q\left(s', a'; \theta_i^{-}\right)\right)$ is the target for iteration i, and θ_i^{-} are the weights from the previous iteration which are fixed when optimizing the loss function by stochastic gradient descent. The gradient can be calculated by differentiating the loss function with respect to the weights as follows:

$$\nabla_{\theta_i} L_i(\theta_i) = (r \ + \ \gamma(\max_{a'}(Q(s', a'; \theta_{i-1})) - Q(s, a; \theta_i))\nabla_{\theta_i} Q(s, a; \theta_i) \tag{3.3}$$

The work described in [27], applies the DQN to obstacle avoidance for a robot to explore an unknown environment. It shows that the Q-network is able to train a robot to avoid obstacles in a simulation environment with a Convolution Neural Network (CNN), used to pre-process the raw image data from a Kinect sensor.

3.2.5. Double Deep Q-networks

One critical shortcoming of DQN is that it sometimes learns unrealistically high action values, known as overestimation. DDQN [28] is introduced to reduce overestimation by separating the action selection and action evaluation in the target. DDQN was also developed by Google DeepMind in 2015. DDQN is the successor to Double Q-learning [29]. The way that DDQN updates the target networks is the same as DQN but the target of DDQN differs, and can be expressed as follows:

$$y_i = r + \gamma Q(s', \max_{a'}(Q(s', a'; \theta_i); \theta_i^-)) \tag{3.4}$$

It has two sets of weights, θ_i and θ_i^-. The action is chosen greedily from the network with weights θ_i at each step, but the Q-values assigned to that action is from the target network with weights θ_i^-. This method provides a more stable and reliable learning process which decreases the overestimation error.

3.2.6. Double Deep Q-networks Prioritized Experience Replay

Prioritized experience replay of Double Deep Q-networks changes the method from randomly sampling experience to selecting experience based on the priority of each experience stored in the replay memory. The priority value of each experience in the replay memory is calculated using temporal-difference (TD) error [30],

$$\delta_i = r + \gamma Q(s', \max_{a'}(Q(s', a'; \theta_i); \theta_i^-)) - Q(s, a) \tag{3.5}$$

where δ_i is the difference between the target value $y_i = r + \gamma Q(s', \max_{a'}(Q(s', a'; \theta_i); \theta_i^-))$ and the Q-value $Q(s, a)$. The transition from the current state to the next state with largest δ_i are replayed from the memory, which is called pure greedy prioritization. In this case, the transitions with initially high error are replayed frequently. In order to avoid a lack of diversity, stochastic prioritization [30], which interpolates between pure greedy prioritization and uniform random sampling, are developed to sample the transitions. The probability of sampling transition i can be defined as,

$$P(i)=\frac{p_i^{\alpha}}{\sum_k p_k^{\alpha}} \tag{3.6}$$

where p_i is the priority of transition i. α is a value within $[0,1]$, which determines the prioritization percentage used in stochastic prioritization. Pure greedy prioritization is not used and all the experiences are random uniform sampled if $\alpha=0$. Several approaches are available for evaluating the priority, for example, $p_i=|\delta_i|+\varepsilon$, where ε is a positive constant to make sure that p_i is not equal to zero in case of $|\delta_i|=0$. This method is known as proportional prioritization, which converts the error to priority.

The stochastic prioritization also leads to the bias of the estimated solution. To correct the bias, prioritized experience replays add importance-sampling (IS) weights at each transition, which is expressed as [30],

$$\omega_i=(\frac{1}{N}\cdot\frac{1}{P(i)})^{\beta} \tag{3.7}$$

where N is the current size of the memory, $P(i)$ is the possibility of sampling transition i in equation (3.7), and β lies within the range of $[0,1]$.

3.2.7. Conclusion

DRL can be the key to improve the decision making process in obstacle avoidance. It is capable of considering various environment states and respond to specific scenarios. The only requirement that the algorithm needs is sufficient experiences for each state. This can be achieved by providing an abundant simulation environment for training the neural network in the obstacle avoidance problem.

CHAPTER 4

OBSTACLE AVOIDANCE APPROACH

This chapter introduces the collision avoidance approach. This control method was developed to work alongside a high-level planner such as the physics-based global planner introduced in [31]. We defined the collision avoidance problem as a Markov Decision Process which can be solved using DRL. The Gazebo [32] simulation environment was used to train the DRL with a virtual STORM module in a simulated environment. A neural network was used as the approximator to estimate the action-value function.

With regards to obstacle avoidance applications, as compared to existing methods such as the potential field method [5] and dynamic window approach [6], DRL does not require a complex mathematical model to function. Instead, an efficient model is derived automatically by updating the parameters of the neural network based on the observations obtained from the sensors as inputs. Also, this method allows the robot to operate efficiently without a precise map or a high-resolution sensor.

DRL is the key to improve the performances of obstacle avoidance. For example, in our approach, the 50 readings from the laser sensor are converted into values with precision of up to two decimal digits in the range of 0.00m to 5.00m to represent the current state, which means that it has 50*6*10*10 states in total. The robot needs to learn to respond to each specific state. The only requirement for the algorithm is sufficient experiences for each state. This can be achieved by providing an abundant simulation environment for training the neural network. The following table shows the comparison

TABLE 4.1
COMPARISON AMONG PATH PLANNING APPROACH

Methods	Strategy	Prior knowledge	Real-time capability	Comments
A*	Search-based Approach	Yes	Offline planning	Avoids expanding paths that are already expensive
Dijkstra's Algorithm				Explores all the possible path and take more time
Greedy Best First Search				Finds out the optimal path fast but not always work
PRM	Sample-based Planning	Yes	Offline planning	Selects a random node from the C-space and performs multi-query
RRT				Selects a random node from the C-space and incrementally adds new node
PF	Potential Field based Approach	No	Online planning	Has several Drawbacks: local minima, no passage of narrow path, Oscillations in narrow passages
VFF				Overcomes the local minima problem but the other two shortcomings still exist
VFH, VFH^+				Overcome the drawbacks but problematic scenarios exists
VFH^*		Yes		Overcomes the problematic scenarios but rely on information from a map
FLC	Fuzzy logic based method	No	Online planning	Generates various decisions according to different sensory information
Proposed Approach	**DRL based Approach**	**No**	**Online planning**	**Generates optimal decisions from a well-trained neural network. Complex scenarios can be overcome as long as the robot has sufficient experiences during training**

among our approach and the traditional approaches mentioned in chapter 2.

The implementation details are presented in the following sections.

4.1 Problem Formulation

In order to plan a collision-free path, the robot needs to extract the environment information from the sensor data and generate the operation commands. The relationship between the observations from the sensors and actions the robot needs to take is referred to as a mathematical model. In traditional methods, such as potential fields [5], the mathematical models are well-defined for the robot to plan the motion. In our approach, the mobile robot can learn from scratch and fit a mathematical model by updating the parameters of the neural network from a learning experiment.

In general, the problem can be formulated as,

$$(v_t, \ \omega_t) = f(\mathbf{O}_t) \tag{4.1}$$

where v_t, ω_t and $\mathbf{O}_t$ are the linear velocity, angular velocity and the observation from the sensor at each time step.

4.2 Collision avoidance with DRL

The main elements of a RL problem include an agent, environment states, a policy, a reward function, and a value function [20]. Neural Network is utilized as a non-linear function approximator to estimate the action-value function, which enhances the reinforcement learning process to a DRL problem.

Our collision avoidance method is based on DRL. The collision avoidance problem is defined as an MDP. A tuple $\langle S, A, P, R, \gamma \rangle$ can be used to represent an MDP, where all the possible state s formed the state space S ($s \in S$), A is an action space and contains the possible action a ($a \in A$), P is a state transition model, R is a reward function and γ is a discount factor. A neural network with two hidden layers is used as the non-linear function approximator to estimate the action-value function $Q(s, a; \theta)$ with weights θ.

To train the neural network, a large-scale interaction data, such as the transition $\left(s_t, a_t, s_{t+1}, r_{t+1}\right)$ needs to be collected. A simulator with a virtual STORM prototype and a virtual world is developed to acquire the data and train the neural network. At each time step t, the virtual robot sends the current state s_t to the neural network as the input. The output is the value of each action in the current state. Then the virtual robot should choose an action according to the decaying ε-greedy policy. The value of ε decreases from 1 with a decay rate β,

$$\varepsilon_{k+1} = \beta \varepsilon_k \tag{4.2}$$

where k is the epoch, and ε will stop decaying and stay fixed at 0.05 when it reaches that value. After taking the chosen action a_t, the robot is in a new state s_{t+1} and receives a reward signal r_{t+1}. These transitions are recorded for updating the neural network.

4.3 Implementation Details

In our approach, the state space S is formed by the observation from the laser sensor and $s_t = \mathbf{O}_t$. High resolution is not required, and 50 distance readings are picked evenly from the sensor containing the distance values measured by 512 laser rays in a 180 degrees span. The raw data is preprocessed to decrease the number of all the possible states. The distance measurements are converted to values with the precision up to two decimal digits in the range of 0.00m to 5.00m to represent the current state.

The action space A consists of 11 possible actions $a_i = (v, \ \omega_m)$ with the same linear velocity v but different angular velocities $\omega_m = -0.8 + 0.16 \times m \quad (m = 0, 1, 2, \cdots 10)$.

The state transition model P is not required in our approach. The immediate reward at each time step is assigned based on the following equation:

$$r_{t+1} = \begin{cases} 5 & \text{(without collision)} \\ -1000 & \text{(collision)} \end{cases} \tag{4.3}$$

Once the robot is bumping into the wall, the robot will receive a reward of -1000. A new epoch starts with the robot in a random position in the world.

The neural network is implemented using Keras with Tensorflow as the backend. The inputs of the networks are the preprocessed laser data which represent the current state of the environment. Two hidden layers, each with 300 neurons are added to the network with ReLU as the activation function. The outputs should be the Q-values of the 11 actions. To update the weights of the neural network, we choose to perform mini-batch gradient descent on the loss function,

$$L(\theta) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - Q(s_i, a_i; \theta))^2 \tag{4.4}$$

where y_i is the current target outputs of the action-value function, and n is the size of the mini-batch. To avoid overestimation, we update the target y_i as given by the following,

$$y_i = r_{i+1} + \gamma Q(s_{i+1}, \max_a(Q(s_{i+1}, a; \theta); \theta^-)) \tag{4.5}$$

where r_{i+1} is the immediate reward after it taking the action a_i, γ is the discount factor. Another neural network $Q'(s, a; \theta^-)$ is initialized with random weights θ^-. The action a with the maximum values are chosen from the neural network Q', but the value

Algorithm 1

1. Initialize the Gazebo simulator;
   Initialize the memory D and the Q-network with random weight θ ;
   Initialize the target Q-network with random weights θ^-

2: **for** episode =1, k **do**

3: Put the visual robot at a random position in the 3D world;
   Get the first state s_1

4: **for** t = 1,T **do**

5: With probability ε select a random action a_t

6: Otherwise select $a_t = \max_a \left(Q\left(s, a; \theta_i\right)\right)$

7: Take action a_t; get reward r_{t+1} and state s_{t+1}

8: Store transition $\left(s_t, a_t, s_{t+1}, r_{t+1}\right)$ in D

9: Sample random mini-batch of transitions $\left(s_i, a_i, s_{i+1}, r_{i+1}\right)$

10: **if** r_{i+1} = -1000 **then**

11: $y_i = r_{i+1}$

12: **else**

13: $y_i = r_{i+1} + \gamma Q(s_{i+1}, \max_a (Q(s_{i+1}, a; \theta); \theta^-))$

14: **end if**

15: Calculate θ by perform mini-batch gradient descent on the
    Mini-batch of loss $L(\theta_i) = (y_i - Q(s, a; \theta_i))^2$

16: Replace the target network parameters $\theta^- \leftarrow \theta$ every N step

17: **end for**

18: **end for**

of a is decided in the target network. A similar approach was used by H. van Hasselt, A. Guez, and D. Silver in [33]. The proposed approach is summarized in Algorithm 1.

4.4 Goal Oriented Navigation with Obstacle Avoidance

The proposed obstacle avoidance approach is first trained to avoid the obstacles without considering a goal position for the robot to arrive. We proposed a multi-stage training method for extending our obstacle avoidance approach. In the first stage, the rewards for training the neural network as described in equation (4.3) is used to encourage the robot to avoid the collision. The second stage initiates training with the trained neural network derived in the first stage. All the parameters of the neural network stay the same except the rewards. Equation (4.6) shows the new rewards utilized for training the neural network in the second stage.

$$r_{t+1} = \begin{cases} 20 & \text{(close to the goal)} \\ 10000 & \text{(arrive at the goal)} \\ -1000 & \text{(collision)} \end{cases} \tag{4.6}$$

Benefited from our multi-stage training, the robot can now learn to navigate to its goal position step by step. This method is inspired by how human beings learn to walk and run. Human infants always learn to creep before they can walk, and then they can learn to run. When training the neural network, complicated rewards may increase the difficulty for the neural network to converge. Detailed information of this multi-stage training method and validation will be described in section 7.1.

CHAPTER 5

ROBOTIC SYSTEM DESIGN

The generalized collision avoidance architecture developed for the STORM modules was experimentally validated on a STORM Locomotion module as mentioned before. Development of the electronic and software system, as well as the system modeling of a locomotion module belonging to STORM are part of this Thesis work. An overview of the STORM, followed by the detailed information of the system development are presented in this section.

5.1 Overview of STORM

A vast majority of current research in robotics is concerned with mobile robots. In this domain, modular self-reconfigurable mobile robotic systems are expected to adapt to the changes in the environment and perform difficult missions [34]. Modular robotic systems offer significant functional advantages as compared to using individual robotic systems. For example, when performing search and rescue operations, a group of modules exploring an unstructured environment simultaneously dramatically improves the efficiency. Provided the systems are capable of docking with each other, they can extend the capabilities of individual modules to suit the task at hand. While small modules can navigate through compact spaces, the modules can also dock together to form a larger configuration in order to traverse larger obstacles. In other words, modular robotic systems with docking abilities can combine the advantages of both small and large robots. In contrast, a conventional single robot system cannot have all these properties at the same time. Within the STORM architecture, each module is capable of navigation and interaction with the environment individually. In addition, each module is capable of communicating with every other module and assembling autonomously into other configurations to perform complex tasks cooperatively. Electronic and software systems on robots tend to be complicated due to the increasing degree of autonomy and the wide variety of tasks that modern robotic systems are required to perform. In this regard, the software and electronic architecture of reconfigurable swarm robotic systems are required to be more advanced [35]. In order to efficiently handle the complexity while allowing for further expansion and parallel development, a robust hardware and software architecture was developed for STORM as detailed in this thesis.

The STORM is a swarm robotic architecture designed with modular mobility and manipulation capabilities[36][37]. It is composed of two major kinds of robotic modules, namely locomotion modules and manipulation modules, each being a complete mobile robot on its own. Also, the STORM modules can dock together to form larger hybrid configurations depending upon the terrain layout or the requirements of the desired task. The ability to use multiple locomotion and manipulation modules, each with individual mobility and the capability of docking with one another, STORM can combine the flexibility of small robots and the functional capability of large robotic systems. In addition, the docking capabilities allow STORM to scale up locomotion and manipulation capabilities as per requirements of the task at hand. These capabilities allow the STORM system to better respond to challenges of unknown tasks and hazardous environments, by bridging the gap between both small and large robotic systems. The possible applications for this robotic system include but are not limited to performing search and rescue tasks in an unstructured environment, surveillance and remote monitoring, warehouse automation etc.

The locomotion and manipulation modules of STORM are shown in Figure 5.1. The manipulation module is a tracked mobile robot with an integrated manipulator arm and an end-effector, capable of lifting heavy payloads. In addition the arm can be used as leverage to enhance mobility in rugged terrain and while traversing obstacles. The locomotion module is designed as a multi-directional hybrid mobile robot with both tracked mobility and the wheeled mobility.

As mentioned earlier, the STORM is designed to execute multiple tasks by leveraging the capabilities of individual modules. For example, a complex task such as climbing can be decomposed into multiple steps or subtasks: wireless communication, individual navigation, autonomous reconfiguration, and cooperative actuation. Once a module detects difficult terrain which it is not able to go over, such as stairs, a reconfiguration

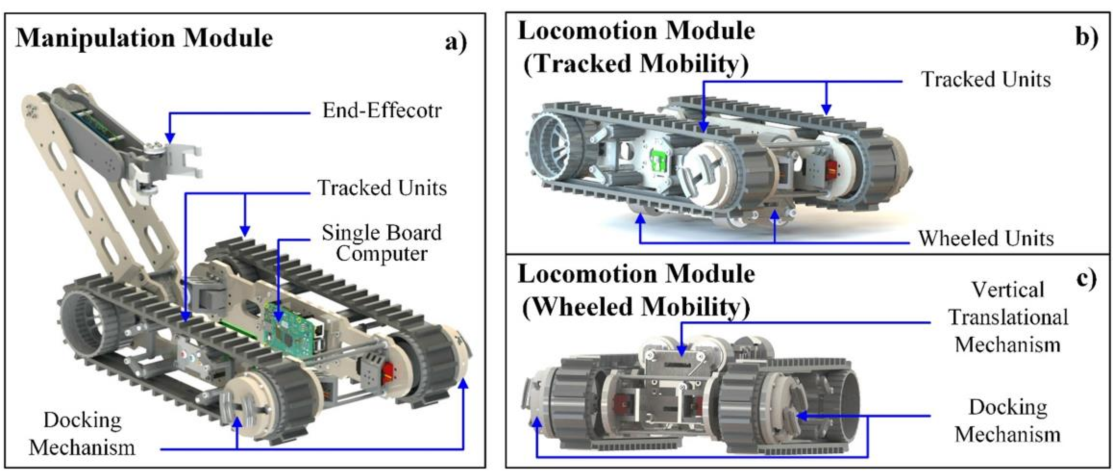

Figure 5.1: 3D models of the individual STORM modules a) the manipulation module, b) the locomotion module in wheeled mobility mode , c) locomotion module in tracked mobility mode

requirement can be issued to the supervisory system. The supervisory system will be in charge of directing the overall SWARM, based on the position and environment information received from each module. Two other modules will receive commands from the supervisor to execute a navigation task, which involves finding a collision-free path from their current position to the first robot and then perform the autonomous coupling.

A key feature of STORM modules is their ability to autonomously dock with one another. Autonomous navigation can bring together the individual modules in close proximity before initiating autonomous docking. On-board depth cameras and IMUs will be used to estimate the robots' positions and orientations in real-time [38]. In order to accomplish autonomous docking, a coupling mechanism [39], [40], is attached to each module. Figure 5.2 shows the main steps of the autonomous docking task. A potential three module configuration composed of one central manipulation and two locomotion modules is also shown in figure to illustrate the concept of reconfiguration. After docking, the modules can form a larger configuration capable of climbing stairs. Similar

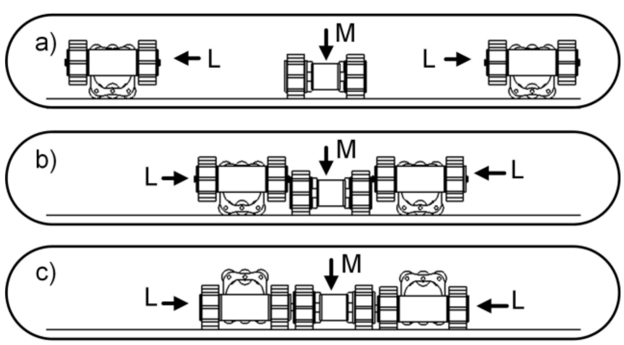

Figure 5.2: The main steps of the autonomous docking. L and M represent the locomotion module and manipulation module, respectively. a) The required modules gather together in close proximity. b) The locomotion modules align with the docking points. c) Modules couple via the powered docking mechanism to become single system.

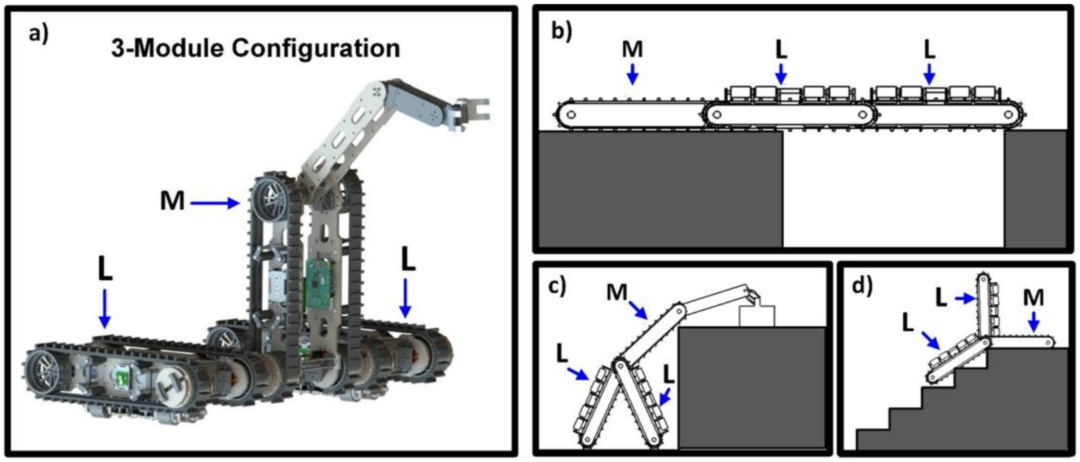

Figure 5.3: Possible multi robot configurations of STORM for performing different tasks. M denotes the Manipulation module and L denotes the locomotion module a) Three-module configuration for carrying objects: (b) crossing wide ditches, (c) retrieving objects from high ledges, (d) stair climbing.

approaches can be used for other high-level tasks such as crossing wide ditches, and retrieving objects from high ledges, as shown in Figure 5.3.

The above approach of dividing a complex task, that requires multiple robotic modules, into subtasks results in swarming behaviors that enable the STORM architecture. One of the main subtasks is autonomous navigation with obstacle avoidance. This work investigates the use of deep reinforcement learning to improve the autonomy level of the STORM system, specifically on obstacle avoidance approach. The proposed obstacle avoidance method is universally applicable to all the submodules in the STORM system, as well as to the general class of mobile robots. The proposed method is validated using a locomotion module prototype.

5.2 Locomotion Module of STORM

A hybrid-locomotion mobile robot, first described in [41], was designed as one module that belongs to a multi-robot system named STORM. This is an on-going research investigation for modular mobility and manipulation [36][37], and is composed of two locomotion modules and a manipulation module. STORM was developed to execute multiple challenging tasks. This increased demand for performance adds difficulties in the design of the mechanical structure and the development of an electronic system for each robot. For example, a complex task such as climbing can be decomposed into multiple steps or subtasks: wireless communication, individual navigation, autonomous reconfiguration, and cooperative actuation. Once a module detects a difficult terrain which it is not able to go over, such as stairs, a reconfiguration requirement can be sent to the supervisor. The other two modules receive the information and commands from the supervisor and then execute the navigation task to find a collision-free path from their current position to the goal position to perform the autonomous coupling task.

To perform the tasks effectively and successfully, a system controller is proposed to take charge of domination, supervision, and communication. This type of centralized control model can be realized by a workstation computer, single board computers embedded in each module, and a wireless router. An efficient sensing system is helpful for obtaining information from the environment for the navigation task. For accomplishing the autonomous coupling task, the modules should have the ability to dock together automatically via a coupling mechanism [39], [40] attached to each robot. Multi-directional mobility must be taken into consideration when designing the mechanical system to improve the spatial alignment capabilities concerning a docking target. The proper motors were selected to form an actuation system to provide the propulsion to achieve the multi-directional mobility.

A reusable and extensible software system is necessary for better exploitation of the robotic platform and for further investigation in on-going research. Although this work is focused on only one locomotion module of the STORM project, the proposed three-layer software architecture allows parallel development. The related projects, such as multi-robot cooperation and global path planning [31], can be developed in parallel without the knowledge of the whole system.

In the following subsections, a short explanation of the mechanical system together with the detailed information of the electronic system and the software architecture are described.

5.2.1. Mechanical System Design

The robotic platform consists of a differential steering tracked unit, differential steering wheeled unit and a vertical translational mechanism (VTM), as shown in Figure 5.4. A Hokuyo laser sensor is mounted on top of the VTU to assist with autonomous navigation. This mobile robot has two locomotion modes. Different modes can be selected according to different task requirements. For instance, wheeled units can be translated down to

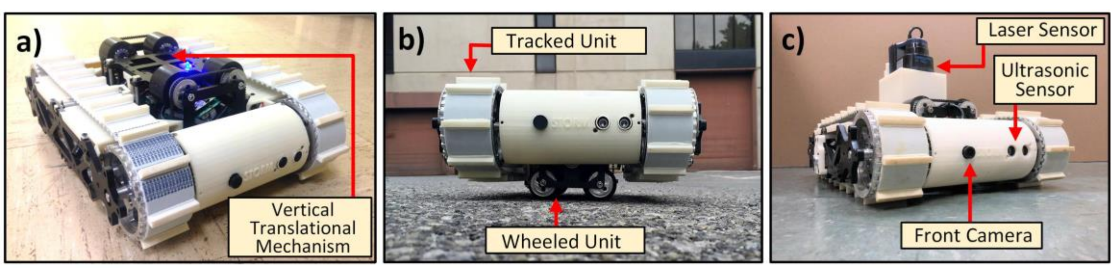

Figure 5.4: The STORM prototype with a laser sensor. Some sensors are pointed out. The tracked units, wheeled units, and the vertical translational mechanism are indicated in the top view.

enable the wheels to touch the ground for higher speed and better performance in position control. The tracked mode can be selected with the tracked units engaged while operating in an unstructured environment. Furthermore, this mechanical structure enables the robot to have vertical mobility, and it can change the moving direction from longitudinal to lateral without turning. It increases spatial alignment capabilities to fulfill the reconfiguration procedure of STORM.

5.2.2. Electrical System Design

The electrical system was designed to meet all the hardware requirements for implementing multiple tasks. The electrical hardware architecture is presented in Figure 5.5. All the components built in or mounted on the robot platform shown inside the

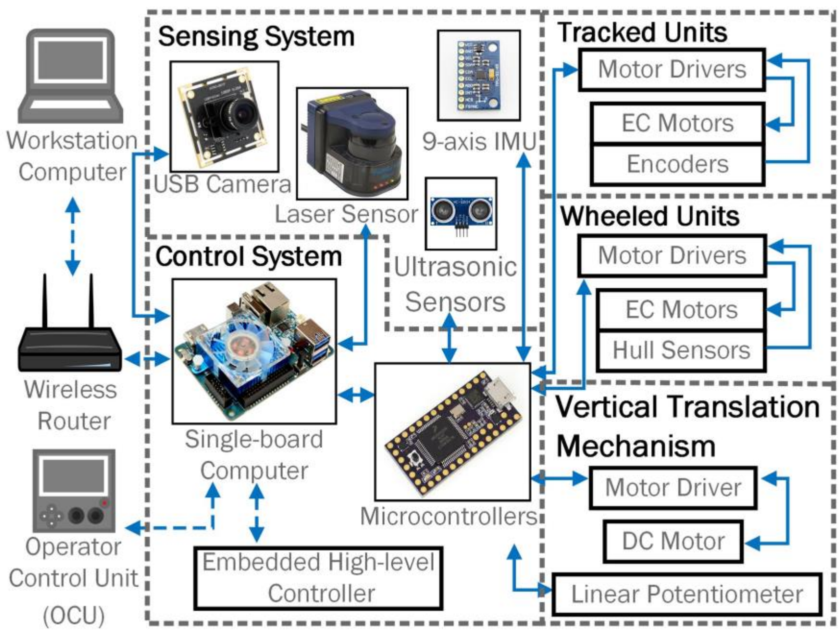

Figure 5.5: Electrical architecture of STORM. All the components built in or mounted on the robot platform are shown inside the dashed line, classified as the internal hardware group. A workstation computer, a wireless router, and an optional OCU form the external hardware group.

dashed line are classified as the internal hardware group. A workstation computer, a wireless router, and an optional operator control unit (OCU) form the external hardware group.

The internal hardware group can be divided into three subsystems: control system, sensing system, and the actuation system. The core of the mobile robot is the control system which contains a single-board computer and two microcontrollers. The control system is in charge of sending commands, acquiring sensor data, and communicating with the external hardware or other robots if required. The main components of the actuation system are five motors. Each tracked unit is actuated by a high-torque EC motor with an encoder. Flat EC motors with hall sensors drive the wheeled units. A DC motor is used to drive the VTM to provide the vertical mobility. A potentiometer is attached to monitor the position of the VTM and to give the feedback to a PID controller for the position control. The sensing system has four ultrasonic sensors, two USB cameras, a laser sensor, and a 9-axis motion tracking device.

5.2.3. Software System

In general, the software architecture was developed in a publish-subscribe pattern based on the Robot Operating System (ROS). As shown in Figure 5.6, the software architecture has three layers: the actuation layer, the platform layer, and the control layer. This architecture divides the robotic software system into small modules. This feature together with the publish-subscribe pattern enables modular design and parallel development without complete knowledge of the whole system.

The software in the actuation layer is embedded in the microcontroller and developed in Arduino. Proper digital and analog signals can be sent to the motor driver and sensors

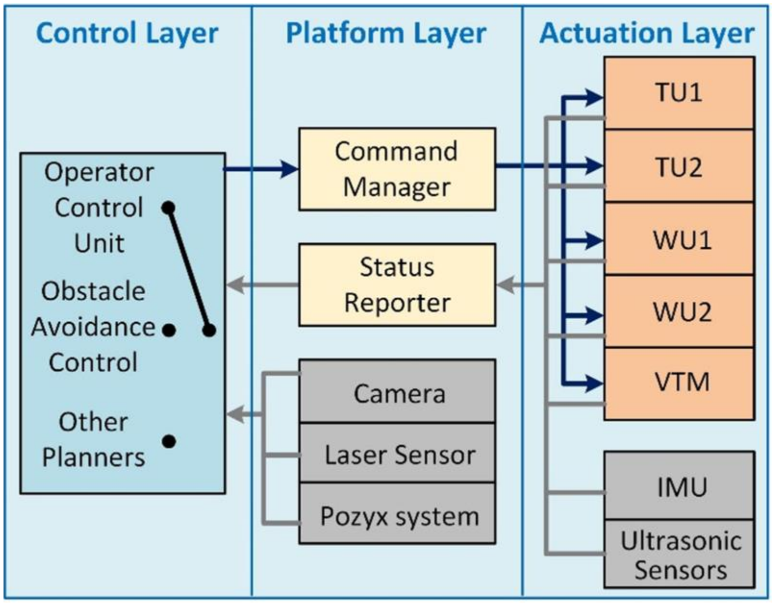

Figure 5.6: Software architecture: The various programs are divided into three different layers. TU1, TU2 are the wheeled units; WU1, WU2 are the tracked units.

according to the commands published by the command manager in the platform layer. Low-level controllers such as the position control of the VTM, PWM speed control of the tracks and wheels are also developed in the actuation layer. The commands from the higher level controllers can be published to the command manager in the platform layer and converted into an 8-bit unsigned integer array which contains the enable signals for the devices and speed information for the motors. All the components in the actuation layer are programmed as objects and waiting for the commands from the command manager. Similar with the function of a state machine, the objects exchange between the standby status to execution status according to the commands. For example, three elements in the command array contain the enable signal and speed information for the motor drive in the wheeled unit. The motor object has an update rate of 10 Hz. The wheeled unit should stay in the standby status after the whole robot platform is activated.

Once the wheeled unit receives the enable signal together with the speed information from the command manager, the wheeled will be activated and operate at the required speed. Low-Level speed control is also implemented with the output speed from the hall sensor. Simultaneously, the output speed information is sent to the status reporter. The speed can be changed every 0.1 seconds if necessary. The wheeled unit will return to the standby status from the excitation status if it receives a deactivated command.

Due to the properties of different sensing products, the software for the sensing system is separated into the actuation layer and the platform layer. The ultrasonic sensors, the potentiometer, and the 9-axis IMU are interfaced with the microcontrollers. The raw data from these sensors are published out to the status reporter together with the feedback information about the motors. The cameras, laser sensor and an optional localization device named Pozyx are directly connected to the single board computer. The processes to interface with these sensors are in the platform layer. Each sensor publishes out a topic for acquiring and sending the raw data. The high-level controllers can subscribe to the status reporter and the sensor topics for any required information.

All the programs in the platform layer and the actuation layer are reusable. The control layer represents the extensible nature. Different control methods can be developed in parallel without the knowledge about each other. They can choose the required information provided by the status reporter and the sensor processes in the single board computer. Some higher level controllers require more computing power than what can be provided by a single board computer. Because of this, the neural network in the proposed obstacle avoidance approach is implemented in the workstation computer. The platform layer is able to subscribe to the topics published by the workstation computer, which is a benefit of using ROS. This property also can enable wireless communication among the multi-robot system.

CHAPTER 6

ROBOTIC SYSTEM MODELLING

In general, the commands for operating the robot in the tracked mobility mode or the wheeled mobility mode from the high-level controller are angular velocities and linear velocities. These inputs should be converted to PWM speed information for each motor. In this chapter, kinematic models are developed for the locomotion module.

6.1 Skid-steering Modelling

Skid-steer locomotion is commonly used on tracked mobile robots, as well as on some four- or six-wheeled mobile robots. Skid-steering mobile robots are not equipped with explicit steering mechanisms that enable the robot to change their heading by changing the steering wheel angle directly. Skid-steering is accomplished by changing the relative velocities of the left and right side drives, which is similar to differential drive. However, the required angular velocities and linear velocities of the mobile robot cannot be derived from a simple differential drive kinematic model with the velocities of left and right drives as inputs, due to the slippage that occurs during turning. In general, pure rolling and no-slip assumptions considered in the differential drive kinematic model are adequate for wheels with a relatively small contact area. For skid-steering mobile robots, slippage cannot be ignored due to the large contact area.

The STORM locomotion modules have both tracked mobility and wheeled mobility. In each locomotion mode, a module can be regarded as a skid-steering mobile robot. The kinematic models of STORM locomotion module are based on the Instantaneous Rotation Center (ICR).

6.2 Kinematic Relations

Kinematic models for the tracked mode and the wheeled mode are developed and embedded in the command manager to get the velocities for each unit. The platform can be regarded as a tracked mobile robot when the VTU is lifting up, and the wheels have no contact with the ground. It can be regarded as a skid-steering mobile robot. When operating in the wheeled mobility mode, it also can be considered as a skid-steering wheeled mobile robot. The kinematics schematics of the locomotion module are shown in Figure 6.1. The geometric center of the module is assumed to coincide with its center of gravity. The body coordinate frame *B(oxy)* is located at the geometric center of the locomotion module.

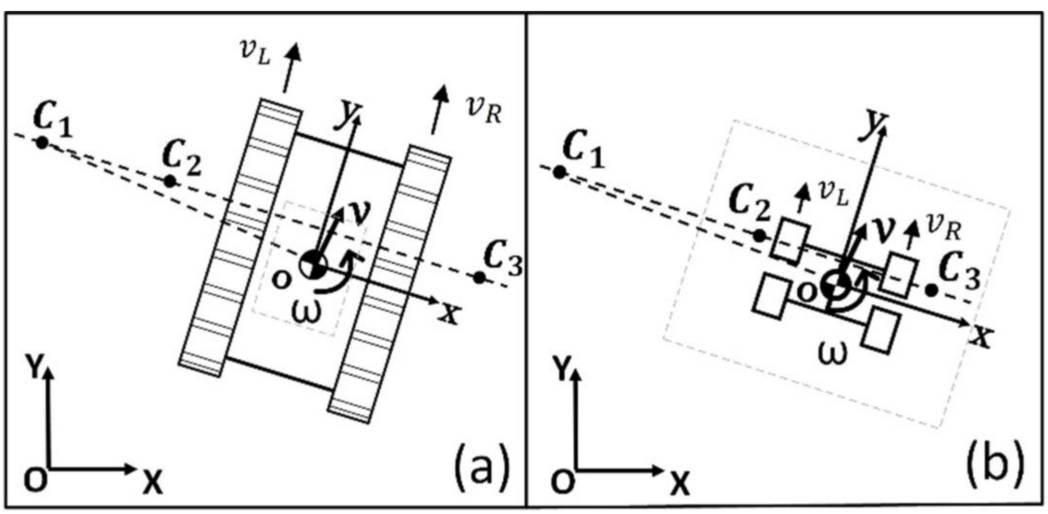

Figure 6.1: Schematic figures of the locomotion module (a)Tracked mobility mode with ICR locations and (b)wheeled mobility with ICR locations are shown.

Any planar movement of a rigid body can be regarded as a rotation about one point. The location of that point can be defined as the ICR. The locomotion module during a steady state turning manoeuver can be considered as a rigid body that is rotated about an ICR. The location of the ICR of the locomotion relative to the ground is represented as $C_1 = (x_{c_1}, y_{c_1})$ in the local coordinate frame, and the x coordinate can be calculated as,

$$x_{c_1} = -\frac{v_y}{\omega} \tag{6.1}$$

where v_y and ω are the longitudinal velocity and the angular velocity of the locomotion, respectively. According to Kennedy's Theorem, the three instantaneous centers shared by three rigid bodies in relative motion to one another all lie on the same straight line. The locomotion module and the two tracks can be considered as rigid bodies, respectively. The $C_2 = (x_{c2}, y_{c2})$ and $C_3 = (x_{c3}, y_{c3})$ are ICRs of the left and the right units, as shown in Figure 6.1. The three ICRs have the same y coordinates:

$$y_{c_1} = y_{c_2} = y_{c_3} = \frac{v_x}{\omega} \tag{6.2}$$

The x coordinates of C_2 and C_3 can be calculated as follows:

$$\begin{aligned} x_{c_2} &= \frac{v_L - v_y}{\omega} \\ x_{c_3} &= \frac{v_R - v_y}{\omega} \end{aligned} \tag{6.3}$$

where v_L and v_R are the linear velocity of the left unit and the right unit with respect to the local frame. The direct kinematics can be derived from equation (6.1) to (6.3):

$$v_x = \frac{v_R - v_L}{x_{c_3} - x_{c_2}} y_{c_1}$$

$$v_y = \frac{v_R + v_L}{2} - \frac{v_R - v_L}{x_{c_3} - x_{c_2}} \frac{x_{c_2} + x_{c_3}}{2} \tag{6.4}$$

$$\omega = \frac{v_R - v_L}{x_{c_3} - x_{c_2}}$$

6.3 Inverse Kinematic Modelling

In general, the inverse kinematics can be stated as follows:

$$(v_L,\ \ v_R) = f(v,\ \ \omega) \tag{6.5}$$

Based on the calculated ICRs, the linear velocities of the robot can be calculated as:

$$\begin{aligned} v_L &= -|x_{c2}|\omega + v_y \\ v_R &= |x_{c3}|\omega + v_y \end{aligned} \tag{6.6}$$

where $v = (v_x, v_y)$ is the linear velocity of the robot. v_x and v_y are the components of v along the X and Y axis respectively. ω is angular velocity of the robot respectively, v_L and v_R are the linear velocity of the left unit and the right unit with respect to the local frame $B(oxy)$. The local x coordinates x_{c2} and x_{c3} were estimated via experiments. Known varying values of v_L and v_R were given to the robot and the resulting ω and v_y were measured using the onboard IMU. Using equation (6.6), the average values of x_{c2} and x_{c3} were calculated from the recorded data. The resulting values were, in the tracked mobility mode, $|x_{c2}| = 0.29$ m and $|x_{c3}| = 0.3$ m, in the wheeled mobility mode, $|x_{c2}| = 0.14$ m, and $|x_{c2}| = 0.15$ m.

The proposed obstacle avoidance approach generates ω and v_y values for the robot. Based on the ω and v_y commands from the high-level controller, v_L and v_R values for the robot were generated based on equation (6.6), using the estimated values of x_{c2} and x_{c3} for the respective mobility mode. It should be noted that the PWM signal applied to the motor controller was assumed to have a linear relationship with the velocity of the left and right track or wheel units. The linear relationship was again estimated via experiments. Taking into account the fact that the motors were only operated within the range of the linear relationship, the above assumptions can be considered valid for the experiments presented in this work.

CHAPTER 7

SIMULATION AND EXPERIMENTAL RESULTS

7.1 Simulation Set-up

The skid-steer STORM locomotion module with a 2D laser scanner along with the Gazebo environment used for training is shown in Figure 7.1. The robot was required to learn from scratch, and no map was provided.

The sensor data and state information were obtained from Gazebo using ROS. This information was then provided to the neural network as inputs. The commands to the virtual robot were then sent via ROS based on the outputs of the neural network. OpenAI Gym library was used to generate the environment to implement the DRL algorithm.

7.1.1. Training Map

The 2D map of the virtual world used for training is shown in Figure 7.2. Instead of the simple corridors used in [5], the robot was trained in a complex circuit with different

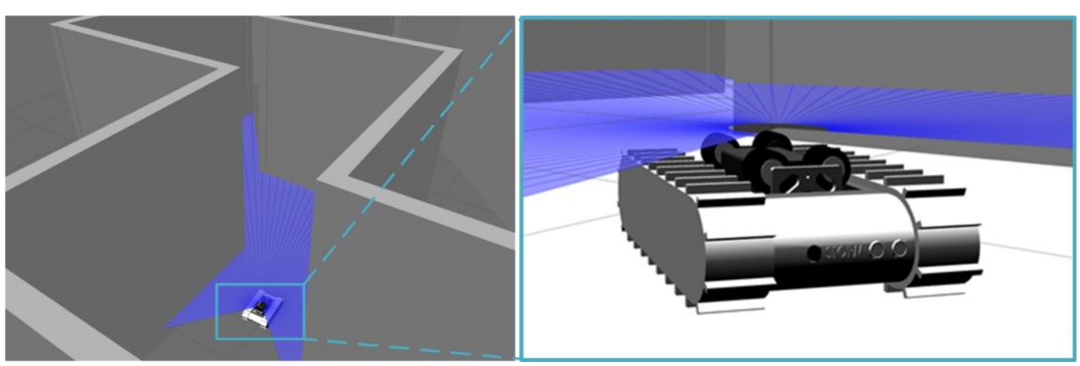

Figure 7.1: Simulation environment. A virtual STORM locomotion module with a 2D range sensor in the simulation environment.

environmental features such as straight tracks, 90-degree turns, acute- and obtuse-angle corners in order to enrich the experience gathered by the robot during learning. This generalizes the approach, such that it applies to more complicated situations after training.

7.1.2. Parameter Selection and Analysis

In our approach, ε-greedy policy is utilized to choose the action. This policy allows the selection of a random action with probability ε. Otherwise, the robot will select the action with the highest value from the neural network. At the beginning of the training, the ε equals 1, and the robot always selects a random action to acquire more different training

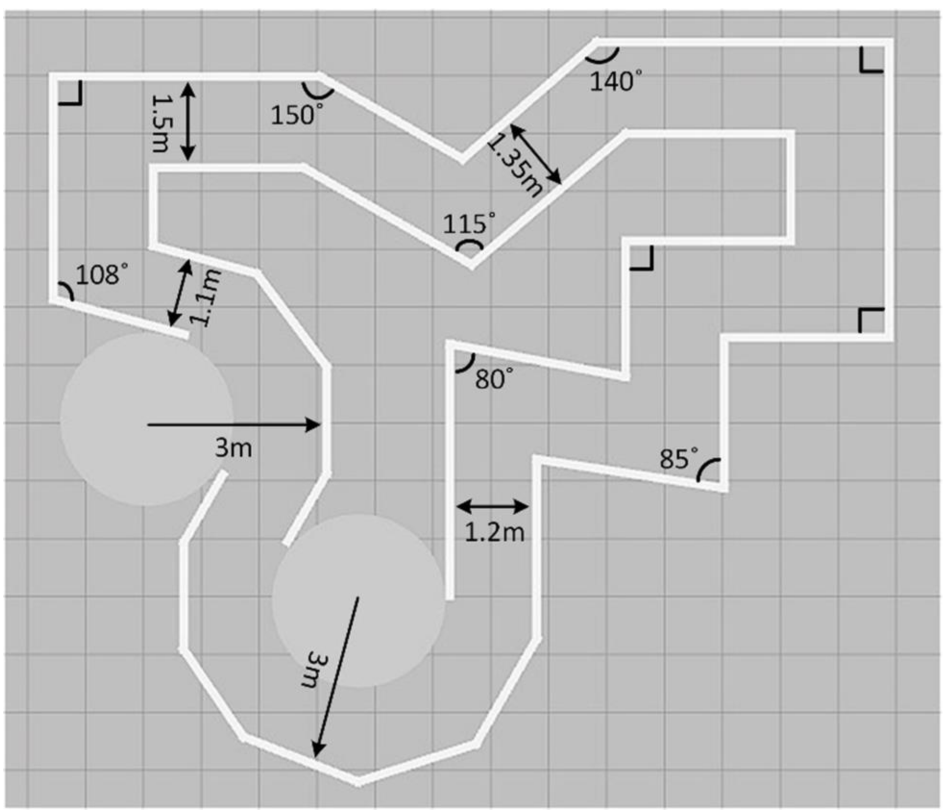

Figure 7.2: The training circuit. The map was created with different features to enrich the experiences of the robot during learning.

data for updating the neural network. This process is defined as exploration. However, the robot cannot always select a random action, as it also needs to exploit the actions. With a probability of $(1 - \varepsilon)$, the robot will follow the output of the current neural network and choose the action with the highest value in the current state. If the robot can receive a similar reward to what it gets in the recorded transitions, the neural network is understood as well-trained. If not, the neural network needs more data to exploit.

The decay rate β of the probability ε is worthwhile to be analyzed for balancing the exploration and the exploitation. To demonstrate the inferences of the parameters β and ε, as well as to evaluate the performances of the virtual STORM, Test 1 was set up as follows:

1. Run the program for three times with the decay rate β at 0.997, 0.998 and 0.999 respectively and get three different trained neural networks.

2. Apply the three neural networks to virtual STORM in the training map successively and let the robot navigate in the training map for five minutes with each of

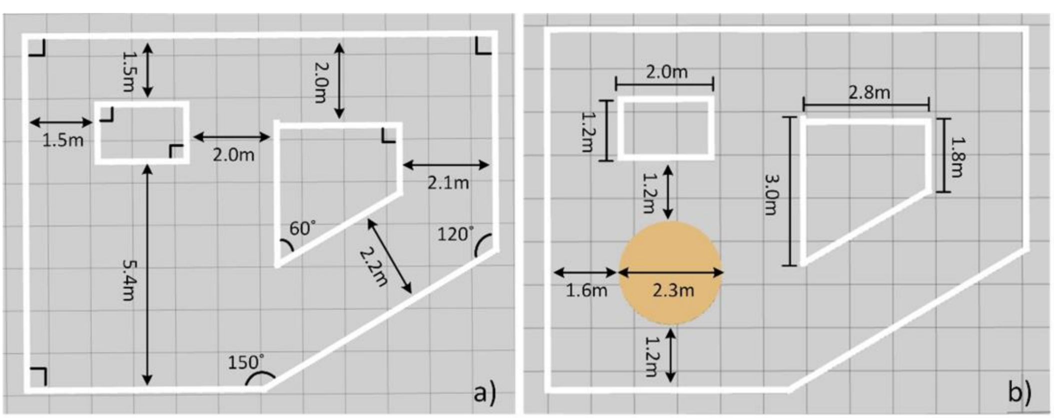

Figure 7.3: The test maps. a) Test map 1 with a large open area b) Test map 2 with a cylindrical obstacle put in the open area.

the networks. The metric for evaluation of the performances of the neural networks is chosen as the number of collisions undergone within the five minutes of simulation.

7.1.3. Validation in Simulator

To prove that the proposed approach can help the robot navigate an unknown environment, we introduce Test 2:

1. Put the virtual STORM with the best trained neural network from Test 1 in two test maps different from the training map. The detailed information about the maps is shown in Figure 7.3.

2. Run the program twice for five minutes with each of the different test maps and record the number of collisions. The virtual robot will be put in a random location with different orientations on the map. After the robot travels for 1000 time steps without collisions, the robot will be spawned to another random location. This is to demonstrate that the proposed obstacle avoidance approach is robust and capable of navigating an unknown environment regardless of the starting position and the heading. Once the robot is too close to the wall, the algorithm will stop the virtual storm and reset at another random location.

7.1.4. Multi-Stage Training Method

In order to realize goal-oriented navigation, we proposed the multi-stage training method to train the neural network with different rewards. Figure 7.4 shows the map for the goal-oriented training. At the first stage, the neural network is trained for 3000 epochs with the rewards in equation (4.3). Five different locations will be randomly selected before the training. The virtual STORM locomotion module will be put at one of these points at

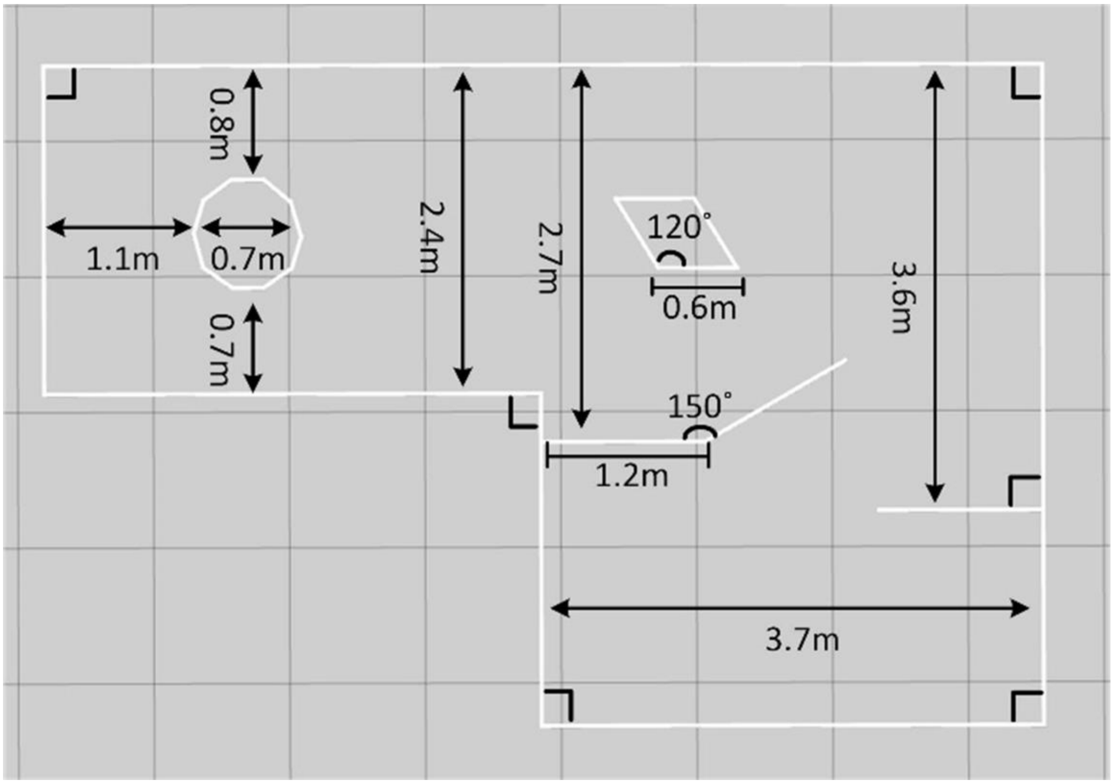

Figure 7.4: Training map for goal-oriented navigation.

each epoch. After the first stage, a trained neural network is applied to the virtual robot, thus marking the start of the second stage. Twenty-five different pairs of starting and goal positions are set up for this stage of training. A random pair of starting point and goal point is assigned at each epoch. The neural network was trained for 3000 epochs at this stage as well.

To demonstrate the advantages of this multi-stage training method, we trained the neural network with the goal-oriented reward without the first stage training for 6000 epochs as a comparison. The result of the trainings and tests will be shown in the next section. The capabilities and limitations of our approaches will be presented as well.

7.2 Simulation Results

7.2.1. Training Results

Figure 7.5 shows the training results from the training map. The three training sessions were performed on the same map, each with a different decay rate of ε. In all three

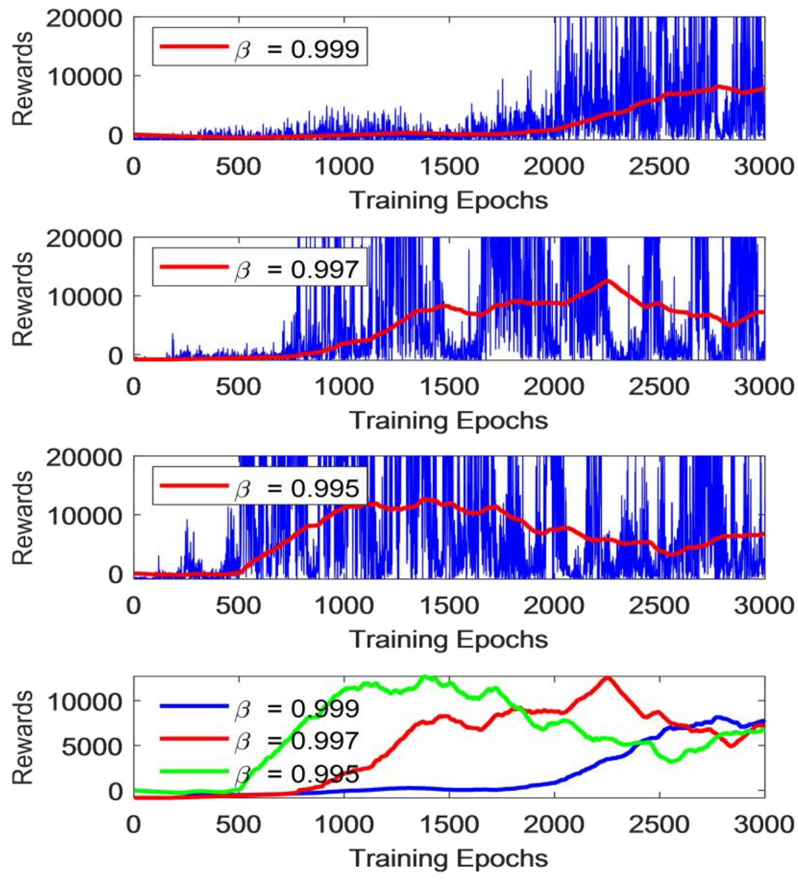

Figure 7.5: Simulation results. In the first three plots, the blue lines indicate the reward values, whereas the red line is the data processed by a moving average filter with the window size of 500. The blue, red, green lines in the last plot stand for the trainings with different decay rates

TABLE 7.1
TEST 1: RESULTS

Decay rate β	Number of collisions in 5 min
0.999	0
0.997	1
0.995	2

setups, the robot has a high exploration rate in the beginning to collect more data for different transition samples.

With a lower β in equation (4.2), the neural network tends to converge faster with fewer scores. A higher β causes the robot to explore more and gain more experience, which is essential for training a neural network in a large state space. The robot was tested in the same simulation environment after training. The exploration rate was set to zero, and the robot choose the action according to the greedy policy.

7.2.2. Test Results

The Table 7.1 shows the performance results from Test 1, where robot ran with different neural networks over a duration of five minutes. It can be recognized from the results that a higher β provides better performance.

The results form Test 2 are shown in the Table 7.2. The robot performance is recorded as a unsatisfactory in Test Map 1 altough the map is significantly simple in

TABLE 7.2
TEST 2: RESULTS

Maps	Number of collisions in 5 min
Test Map 1	8
Test Map 2	0

comparison to Test Map 2. This is because the robot lacks experience in large open areas. This underlines the critcality of the experiences a robot acquires during training and demonstrates its significance for superior performance of the robot in an unknown environments. As with all DRL approaches, this method also faces the limitation of real-world transference. The robot may fail to navigate on a simple map in real world, if the features in that map are entirely different from that used in the training environment.

7.2.3. Multi-Stage Training Results with Comparison

Figure 7.6 shows the results from the multi-stage training. After the first stage training, the robot is able to navigate the training map as shown in Figure 7.4 barely with collision. The second stage of training enables the robot to navigate to its goal completely free of collision. In contrast, training the neural network directly with the goal-oriented rewards is not a good choice. When the robot starts to learn from scratch, complex rewards increase the challenges during learning, which can be proved from the results in the third plot of Figure 7.6. The curve eventually converges to an average reward around 2000 in the third plot, which was much lower than the reward value achieved by the multi-stage training.

The multi-stage training method will be tested on more different environments which will be addressed as part of our future work. In addition, advanced end-to-end navigation methods for single robots and multi-robot groups will be developed for challenging unknown environments.

7.3 Real-world Implementation and Experimental Results

The obstacle avoidance controller for the STORM prototype was embedded in the control layer. The neural network trained with the β = 0.999 was selected as it offered the best performance. As mentioned before, the trained neural network was made to run on a workstation since it was beyond the computational power of the onboard single board

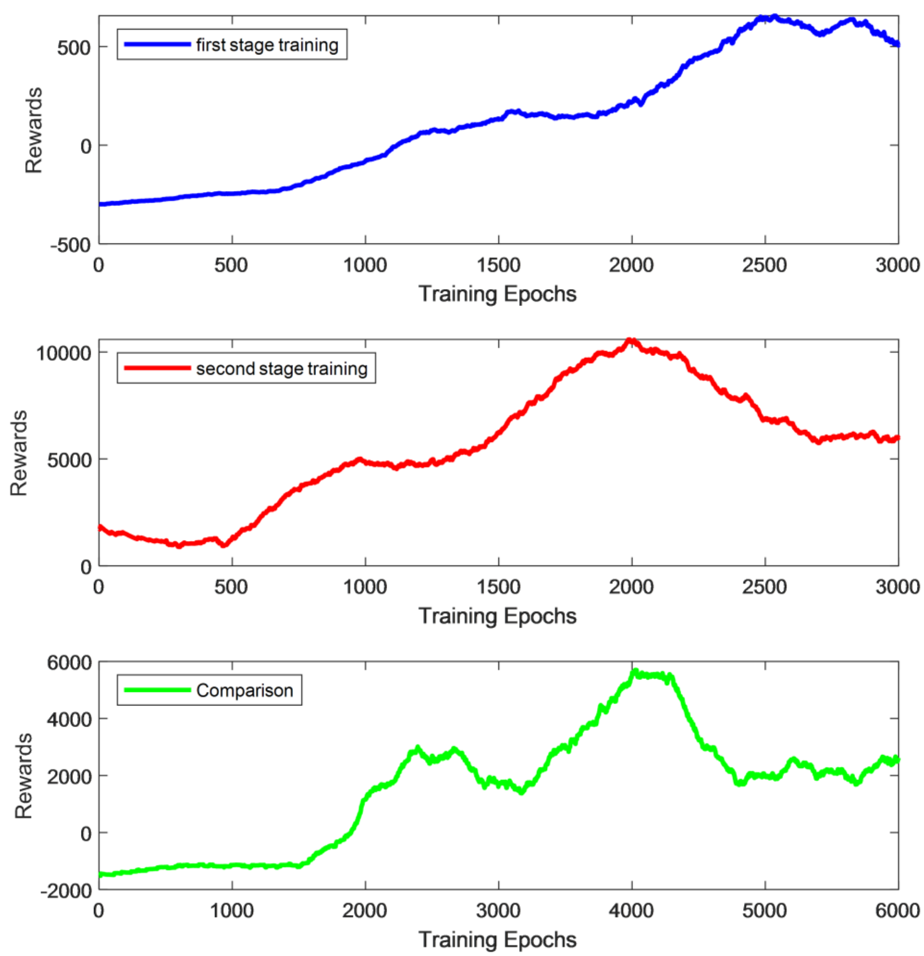

Figure 7.6: Multi-Stage Training Results with Comparison. The first and second plot show the results from the first and second stages of training respectively. The third plot is the result from the training the neural network directly with the goal-oriented rewards

computer. The neural network generated the command $a_i = (v,\ \omega_m)$ with an update rate of 10 Hz. These commands were subscribed by the command manager on board STORM locomotion module and converted into the proper form for the actuation layer.

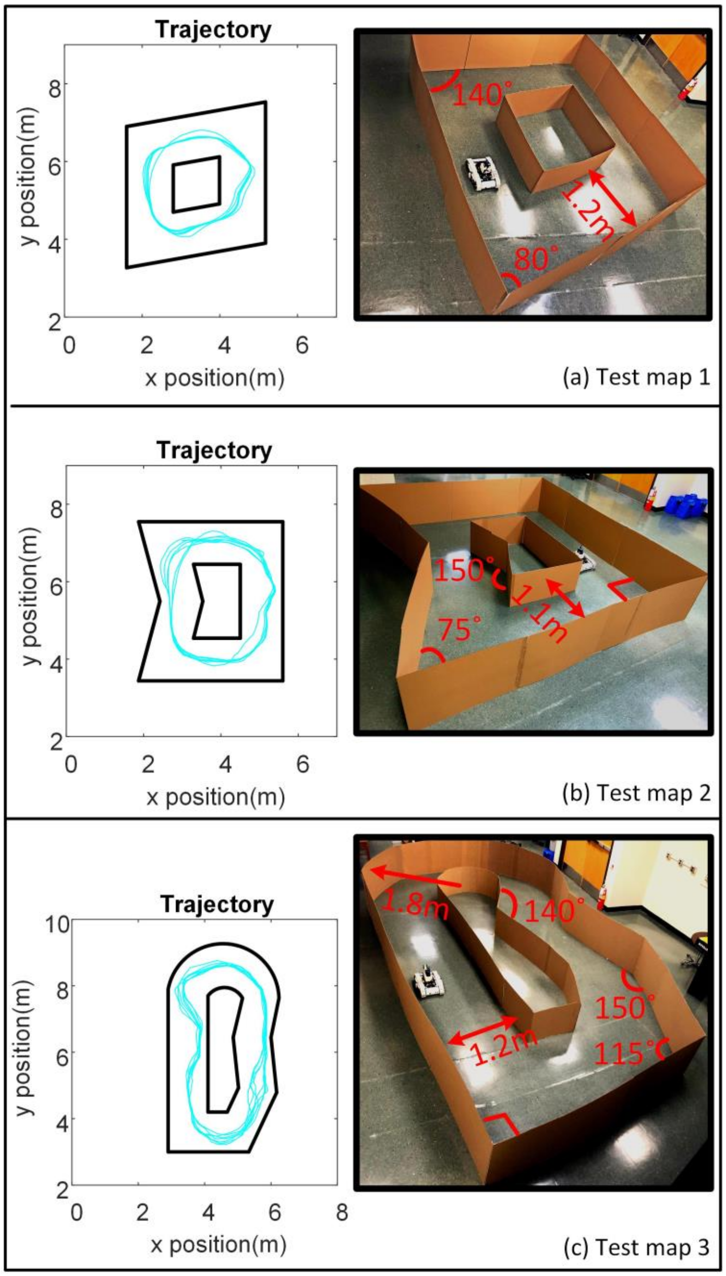

Figure 7.7: Test maps and the trajectories of the robot

As mentioned in Section 5.5, the neural network found optimal actions based on the sensor observations in the simulated world, but the map was the same as the training map. In order to validate that the robot is able to travel without collisions, not only in the same map as the training case but also in environments with similar features, three maps with features different from the simulation world were built to test the robot. One of the maps had one circular feature which had a different diameter with that in the simulation world. The test cases were built to test the robustness of the proposed approach.

Figure 7.7 shows the three test maps and the trajectories followed by the robot when traveling in the maps. The robot traveled in each of the three maps for five minutes without collision. Thus, it was proved through experiments that the proposed approach can handle the situations different from the training scenario provided in the 3D simulation. The proposed collision avoidance method is general in the sense that it can be applied to both the tracked mobility mode and wheeled mobility mode of the locomotion module, as well as to other modules in the STORM architecture for a variety of environmental conditions. Moreover, since the trained architecture works on 50 measurements, a high-resolution sensor is not necessary.

CHAPTER 8

CONCLUSION AND FUTURE WORK

This chapter concludes the thesis with a summary of the current work as well as potential work in the near future.

8.1 Summary

This paper introduced the STORM multi-robot system with a detailed description of the electrical system and software architecture and mechanical structure of the locomotion module. The software and electrical architecture has reusable and extensible features, which allow for further development and expansion. The major contribution of this work is a DRL based obstacle avoidance approach that was tested and validated experimentally. The proposed DRL architecture was trained over a simulated Gazebo environment. This allowed for sufficient data collection over various features without any damage to the real robot. The results from the simulation show that more exploration in training leads to an optimal solution with slow convergence speed.

8.2 Future Work

Future work will explore end-to-end navigation and multi-robot cooperation. As part of the future work, new simulation environments with moving obstacles and uneven terrain will be considered to enrich the experience of the trained neural network. The proposed approach did not take into account any given goal position and was only aimed at making the robot move forward. Future work will incorporate the specified goal position without a prior map, along with the obstacle avoidance capabilities. The more considerable extension of this work is to enable multi-robot exploration and navigation in unknown dynamic environments for the STORM system.

REFERENCES

[1] P. Ben-Tzvi, “Experimental validation and field performance metrics of a hybrid mobile robot mechanism,” *J. F. Robot.*, vol. 27, no. 3, pp. 250–267, 2010.

[2] P. Hart, N. Nilsson, and B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths,” *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968.

[3] L. Kavraki and J.-C. Latombe, “Randomized preprocessing of configuration for fast path planning,” in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, 1994, vol. 3, pp. 2138–2145.

[4] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Feb. 1996.

[5] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, 1985, vol. 2, pp. 500–505.

[6] O. Brock and O. Khatib, “High-speed navigation using the global dynamic window approach,” *Proc. 1999 IEEE Int. Conf. Robot. Autom. (Cat. No.99CH36288C)*, vol. 1, no. May, pp. 341–346, 1999.

[7] M. Elbanhawi and M. Simic, “Sampling-based robot motion planning: A review,” *IEEE Access*, vol. 2, pp. 56–77, 2014.

[8] S. M. Lavalle, “Rapidly-Exploring Random Trees: A New Tool for Path Planning,” Nov. 1999.

[9] Y. Koren and J. Borenstein, “Potential field methods and their inherent limitations for mobile robot navigation,” *Proceedings. 1991 IEEE International Conference on Robotics and Automation*. pp. 1398–1404.

[10] J. Borenstein and Y. Koren, “Real-time obstacle avoidance for fast mobile robots in cluttered environments,” *Proceedings., IEEE Int. Conf. Robot. Autom.*, vol. 19, no. 5, pp. 572–577, 1989.

[11] 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)*, 2000, vol. 3, no. April, pp. 2505–2511.

[12] I. Ulrich and J. Borenstein, “VFH+: Reliable obstacle avoidance for fast mobile robots,” *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2, pp. 1572–1577, 1998.

[13] J. Borenstein and Y. Koren, “The vector field histogram-fast obstacle avoidance for mobile robots,” *IEEE Trans. Robot. Autom.*, vol. 7, no. 3, pp. 278–288, Jun. 1991.

[14] M. Faisal, R. Hedjar, M. Al Sulaiman, and K. Al-Mutib, “Fuzzy logic navigation and obstacle avoidance by a mobile robot in an unknown dynamic environment,” *Int. J.*

Adv. Robot. Syst., vol. 10, 2013.

[15] S. Khatoon, “Autonomous Mobile Robot Navigation by Combining Local and Global Techniques.,” *Int. J. Comput. …*, vol. 37, no. 3, pp. 1–10, 2012.

[16] D. Silver *et al.*, “Mastering the game of Go without human knowledge,” *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[17] D. Silver *et al.*, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[18] S. Gu, E. Holly, T. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3389–3396.

[19] Y. Zhu *et al.*, “Target-driven visual navigation in indoor scenes using deep reinforcement learning,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, vol. 1, no. 1, pp. 3357–3364.

[20] R. S. Sutton and A. G. Barto, “Chapter 1 Introduction,” *Reinf. Learn. An Introd.*, 1988.

[21] T. Jaakkola, S. P. Singh, and M. I. Jordan, “Reinforcement learning algorithm for partially observable Markov decision problems,” *Adv. Neural Inf. Process. Syst.*, vol. 7, p. 345, 1994.

[22] K. Doya, “Reinforcement Learning in Continuous Time and Space,” *Neural Comput.*, vol. 12, no. 1, pp. 219–245, Jan. 2000.

[23] S. J. Bradtke and M. O. Duff, “Reinforcement Learning Methods for Continuous-Time Markov Decision Problems,” *Adv. Neural Inf. Process. Syst.*, pp. 393--400, 1994.

[24] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Mach. Learn.*, vol. 8, no. 3–4, pp. 279–292, 1992.

[25] Y. Zhan, H. B. Ammar, and M. E. Taylor, “Theoretically-grounded policy advice from multiple teachers in reinforcement learning settings with applications to negative transfer,” in *IJCAI International Joint Conference on Artificial Intelligence*, 2016, vol. 2016–Janua, pp. 2315–2321.

[26] L. Lin, “Reinforcement learning for robots using neural networks,” 1993.

[27] T. Lei and L. Ming, “A robot exploration strategy based on Q-learning network,” *2016 IEEE Int. Conf. Real-Time Comput. Robot. RCAR 2016*, pp. 57–62, 2016.

[28] W. Chen, M. Zhang, Y. Zhang, and X. Duan, “Exploiting meta features for dependency parsing and part-of-speech tagging,” *Artif. Intell.*, vol. 230, pp. 173–191, Jan. 2016.

[29] H. Van Hasselt, A. C. Group, and C. Wiskunde, “Double Q-learning,” *Nips*, pp. 1–9, 2010.

[30] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized Experience Replay,” pp. 1–21, 2015.

[31] B. Sebastian and P. Ben-Tzvi, “Physics Based Path Planning for Autonomous

Tracked Vehicle in Challenging Terrain," *J. Intell. Robot. Syst. Theory Appl.*, pp. 1–16, 2018.

[32] "Gazebo [Online]," *http://gazebosim.org/*. .

[33] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," *Assoc. Adv. Artif. Intell.*, Sep. 2016.

[34] P. Moubarak and P. Ben-Tzvi, "Modular and reconfigurable mobile robotics," *Rob. Auton. Syst.*, vol. 60, no. 12, pp. 1648–1663, 2012.

[35] D. Kortenkamp, R. Simmons, and D. Brugali, "Robotic Systems Architectures and Programming," in *Springer Handbook of Robotics*, 2016, pp. 283–306.

[36] P. Ben-Tzvi and P. M. Moubarak, "Mobile robot with hybrid traction and mobility mechanism," US 9,004,200 B2, 2015.

[37] P. M. Moubarak and P. Ben-Tzvi, "A tristate rigid reversible and non-back-drivable active docking mechanism for modular robotics," *IEEE/ASME Trans. Mechatronics*, vol. 19, no. 3, pp. 840–851, 2014.

[38] S. S. Sohal, W. Saab, and P. Ben-Tzvi, "Improved Alignment Estimation for Autonomous Docking of Mobile Robots," in *Volume 5A: 42nd Mechanisms and Robotics Conference*, 2018, p. V05AT07A072.

[39] W. Saab and P. Ben-Tzvi, "A Genderless Coupling Mechanism with 6-DOF Misalignment Capability for Modular Self-Reconfigurable Robots," *J. Mech. Robot.*, vol. 8, no. c, pp. 1–9, 2016.

[40] P. M. Moubarak, P. Ben-Tzvi, Z. Ma, and E. J. Alvarez, "An active coupling mechanism with three modes of operation for modular mobile robotics," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 5489–5494, 2013.

[41] P. Kumar, W. Saab, and P. Ben-Tzvi, "Design of a Multi-Directional Hybrid-Locomotion Modular Robot With Feedforward Stability Control," in *Volume 5B: 41st Mechanisms and Robotics Conference*, 2017, p. V05BT08A010.

[42] "AlphaGo Zero: Learning from scratch." [Online]. Available: https://deepmind.com/blog/alphago-zero-learning-scratch/. [Accessed: 09-Oct-2018].