M.S. THESIS

Local Navigation Approach by Learning Collision

충돌 학습을 통한 지역 경로 계획 방법

BY

Howoong Jun

February 2019

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Local Navigation Approach by Learning Collision

충돌 학습을 통한 지역 경로 계획 방법

지도교수 이 범 희
이 논문을 공학석사 학위논문으로 제출함

2018년 11월

서울대학교 대학원

전기 · 정보 공학부

전 호 웅

전호웅의 공학석사 학위 논문을 인준함

2018년 12월

위 원 장:	조 동 일	[illegible]
부위원장:	이 범 희	[illegible]
위 원:	심 형 보	[illegible]

Abstract

This thesis proposes a reinforcement learning based collision avoidance method. The problem can be defined as an ability of a robot to reach its goal point without colliding with other robots and obstacles. There are two kinds of collision avoidance problem, single robot and multi-robot collision avoidance. Single robot collision avoidance problem contains multiple dynamic obstacles and one agent robot. The objective of the agent robot is to reach its goal point and avoid obstacles with random dynamics. Multi-robot collision avoidance problem contains multiple agent robots. It is also possible to include unknown dynamic obstacles to the problem. The agents should reach their own goal points without colliding with each other. If the environment contains unknown obstacles, the agents should avoid them also.

To solve the problems, Collision Avoidance by Learning Collision (CALC) is proposed. CALC adopts the concept of reinforcement learning. The method is divided into two environments, training and planning. The training environment consists of one agent, one obstacle, and a training range. In the training environment, the agent learns how to collide with the obstacle and generates a colliding policy. In other words, when the agent collides with the obstacle, it receives positive reward. On the other hand, when the agent escapes the training range without collision, it receives negative reward. The planning environment contains multiple obstacles or robots and a single goal point. With the trained policy, the agent can solve the collision avoidance problem in the planning environment regardless of its dimension. Since the method learned collision, the generated policy should be inverted in the planning environment to avoid obstacles or robots. However, the policy should be applied directly for the goal point so that the agent can ‘collide’ with the goal. With the combination of both policies, the agent can avoid the obstacles or robots and reach to the goal point simultaneously. In the training algorithm, the robot is assumed to be a holonomic robot. Even though the trained policy is generated from the holonomic robot, the method can be applied to both holonomic and non-holonomic robots by holonomic to non-holonomic converting method.

CALC is applied to three problems, single holonomic robot, single non-holonomic robot, and multiple non-holonomic robot collision avoidance. The proposed method is validated both in the robot simulation and real-world experiment. For simulation, Robot Operating System (ROS) based simulator called Gazebo and simple game library PyGame are used. The method is tested with both holonomic and non-holonomic robots in the simulation experiment. For real-world planning experiment, non-holonomic mobile robot named e-puck is used. The learned policy from the simulation can be directly applied to the real-world robot without any calibration or retraining. The result shows that the proposed method outperforms the existing methods such as Reciprocal Velocity Obstacle (RVO), PrEference Appraisal Reinforcement Learning (PEARL), and Optimal Reciprocal Collision Avoidance (ORCA). In addition, it is shown that the proposed method is more efficient in terms of learning than existing learning-based method.

keywords: collision avoidance, path planning, mobile robot, multi-robot system, reinforcement learning

student number: 2017-28406

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivations

Collision avoidance problem has been key issue in the field of robotics such as humanoid [1], robot arm [2], and mobile robots [3] for many years. In the problem, robots have to achieve their objectives without colliding with other objects such as obstacles and other robots. The concept of the problem is described in Fig. 1.1. Especially, with the rise of autonomous self-driving car, safe navigation for wheel-based mobile robots became critical [4]. The autonomous self-driving robot has two key objectives: reaching a goal location and avoiding collision. It is possible that the goal location and a global path can be given by a central server. However, avoiding collision with unexpected obstacles or other robots cannot be done by a central server since the most worlds are unstructured environments. Therefore, goal-driven decentralized local navigation approach is mandatory for autonomous self-driving robots.

Classical goal-driven decentralized local navigation approaches use hand-crafted rules to derive the solution. They re-plan the optimal path on every iteration based

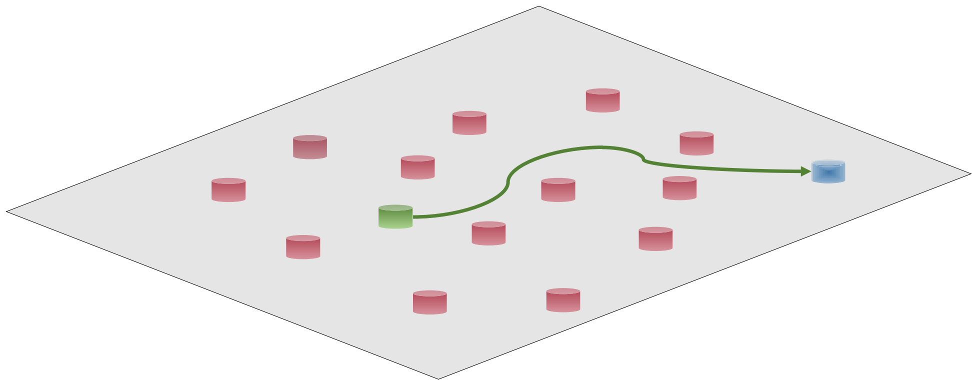

Figure 1.1: An overview of collision avoidance problem. An agent (green) has to avoid obstacles (red) and reach a goal point (blue). The obstacles can be both movable or static.

on the rule to cope with unknown circumstances. However, adjusting paths on every iteration imposes heavy computational burden since the methods should consider every dynamics of obstacles. Some learning-based navigation methods adopt supervised learning to overcome the limitation. With an image of a camera, the agent can learn the circumstances of collision and how to avoid it. However, supervised learning has the issue of collecting large data-set. In other words, to achieve the objective, large amount of image should be collected. Other learning-based navigation methods use reinforcement learning to solve the problem. By training the agent to avoid collision and reach to a goal point simultaneously, it can reach its objective. However, the methods are still inefficient in terms of learning and lacks reproducibility [5] since the agent has to learn two tasks.

To overcome those limitations, reinforcement learning-based local navigation approach named *Collision Avoidance by Learning Collision (CALC)* is proposed in this thesis. The overall concept of CALC is described in Fig. 1.2. CALC learns collision instead of avoiding an obstacle. An agent robot learns how to collide with a single obstacle in a small and restricted *training environment*. In the training environment, the agent can derive a colliding policy based on the reinforcement learning. With the colliding policy, CALC transfers it into high dimensional problems called *planning environment* which comprise multiple robots and obstacles. Since the method learns collision rather than avoidance, the actions generated from the policy should be inverted to avoid obstacles. Likewise, the agent should reach the goal point so the actions should be directly applied to the goal point. Therefore, the proposed method can solve two problems simultaneously, avoiding obstacles and reaching a goal point, by learning only one objective: collision. Also, it can achieve the improved ability in terms of learning efficiency and reproducibility. In addition, the method only requires local information around the agent robot. Since the method is a local path planner, it only needs relative position vectors of the obstacles inside the fixed range around the agent.

The planning environment can be categorized into two, single robot and multi-robot collision avoidance. Single robot collision avoidance problem includes one agent and multiple dynamic obstacles. The agent robot should reach its goal point without colliding with obstacles of random dynamics. The agent and obstacle robots can be both holonomic and non-holonomic robot. Multi-robot collision avoidance problem includes multiple agents. The agent robots should reach its own goal points without colliding with each other. Moreover, dynamic obstacle can be included in the environment. If there are obstacle robots in the environment, the agents should also avoid those robots. Additionally, the agent robot is assumed to be holonomic robot in the training environment. Holonomic robot has advantages compared with non-holonomic robot due to its simple locomotion control. For example, it is easy for the holonomic robot to go east without considering current orientation of itself. However, non-holonomic robot has to consider its current orientation and its dynamics to move east. It is more complicated to move the non-holonomic robot than the holonomic robot. Therefore, holonomic robot is considered in the training algorithm for efficient training. Also, to apply the learned policy from the holonomic robot into non-holonomic robot, a new method for holonomic to non-holonomic conversion is also suggested in this thesis.

The ultimate purpose of the proposed method is to apply the algorithm into real-world scenarios. In the real world, holonomic robot is less common than non-holonomic robot. Therefore, multiple non-holonomic robot collision avoidance problem is the key target to the proposed method. In order to achieve the objective, single holonomic robot collision avoidance problem should be considered first. CALC for single robot collision avoidance problem is demonstrated in the simulated robot experiment and compared with other methods. Also, experiments for the single robot collision avoidance problem with non-holonomic robot is conducted. Finally, Multi-robot collision avoidance problem is demonstrated in the both simulated and real-robot experiments. It is also compared with other multi-robot collision avoidance methods. In the experiment, non-holonomic robot, which is more closer to the real-world vehicles than holonomic robot, is used. The training process is only done in the simulation. However, the learned policy from the simulation is applied successfully to the real-world experiments without additional re-training.

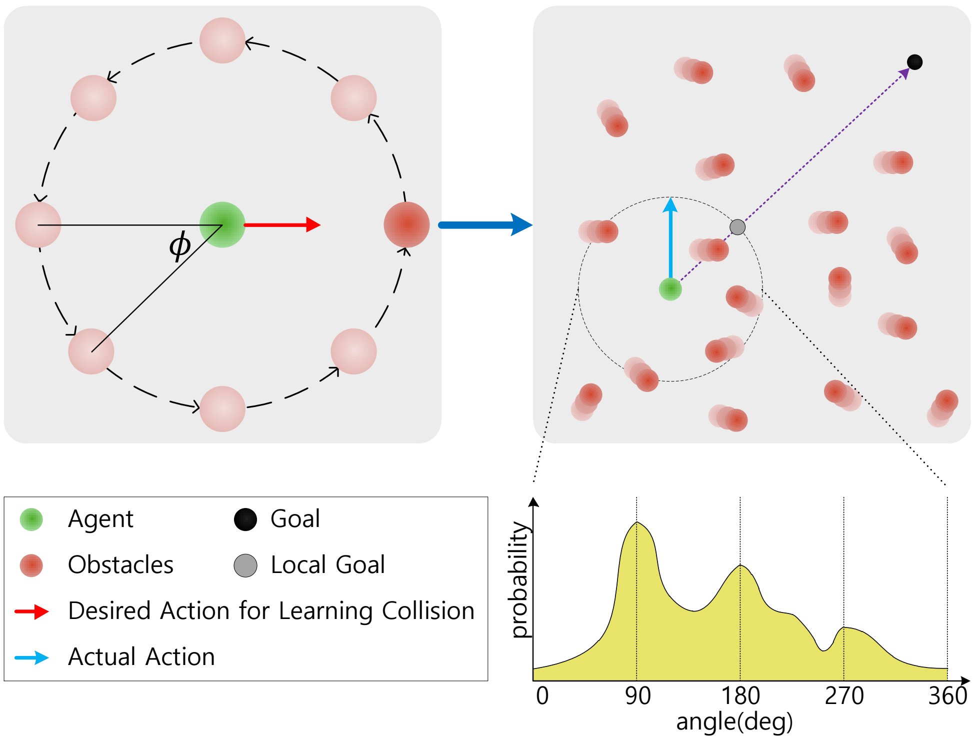

Figure 1.2: The overview of the proposed method. An agent learns how to collide with a static obstacle in a small training environment (left). In the training environment, the obstacle rotates its position based on the angle ϕ. The red arrow indicates the colliding action for the agent. The learned policy from the training environment is utilized in a planning environment (right). The policy can be expressed as a probability distribution of a direction for avoiding obstacles. The agent has to avoid collision with moving obstacles and reach its goal point. Based on the trained policy, the agent can take action that can pursue the goal point without colliding with the obstacles.

1.2 Contributions

The main contributions are listed below.

First, a new small training environment for collision avoidance problem is suggested in Chapter 3. The new environment consists of three elements: one agent robot, one obstacle, and a training range. With those elements, the agent robot can learn how to collide with the obstacle. This enables the proposed algorithm to be converged fast and efficient since the robot has to learn only one task.

Second, the expansion from the small training environment to high-dimensional planning environment is presented in Chapter 4 and Chapter 5. The method utilizes the pre-trained data in the training environment into high-dimensional problems with multiple obstacles. To cope with the obstacles, the method reverses the pre-trained policy. Also, to reach the goal point, the method directly apply the policy. With the combination of the policies, the method can successfully solve the collision avoidance problem.

Thirdly, the applicability of the method is presented. The proposed approach can be applied on both holonomic and non-holonomic robots. Holonomic and non-holonomic robot experiment is presented on Chapter 4.2 and 4.3 respectively. Additionally, it is applied to goal-driven non-holonomic multi-robot collision avoidance problem in Chapter 5.

Lastly, to apply learned policy from the holonomic robot to non-holonomic robot, holonomic to non-holonomic conversion approach is suggested in Chapter 5. This method consists of two parts: angular and linear motion planning. Angular velocity is derived based on the current robot heading and desired action. In the linear motion planning part, three velocity models are suggested: constant, linear, and quadratic.

The suggested models are compared with the experiments. Also, statistical analysis of consuming time for three models are conducted to find the appropriate model for the proposed method.

1.3 Organizations

The rest of this thesis is organized as follows. Chapter 2 reviews key ideas about reinforcement learning. Also, it covers related works about navigation methods including hand-crafted methods and deep learning-based methods. Chapter 3 presents the training process of the proposed method. Chapter 4 addresses the problem of the single robot collision avoidance for both holonomic and non-holonomic robot and shows simulated experiments. Chapter 5 modifies the approach of Chapter 4 into multi-robot collision avoidance problem for non-holonomic robots. In the Chapter 5, both simulated and real-world experiments are presented. Both Chapter 4 and Chapter 5 utilize the result of Chapter 3. Finally, Chapter 6 concludes the thesis with suggestions for future work.

Chapter 2

Related Work

This work is mainly based on the concepts of reinforcement learning and navigation. This chapter introduces background of reinforcement learning and previous works about navigation. Additionally, existing works about navigation using deep learning are introduced in the last section.

2.1 Reinforcement Learning

Reinforcement learning is one category of machine learning field. The concept can be roughly defined as an ability of an agent to learn the objective by trial and error. As the agent experiences diverse circumstances, it can enhance its ability to achieve the goal. Recently, with a combination of deep learning technique, reinforcement learning has improved significantly by defining the field of *Deep Reinforcement Learning (DRL)*. This provoked many researchers to apply deep reinforcement learning into many fields [6] such as video game [7], game of Go [8] [9], navigation [10], and robotic grasping [11].

To understand reinforcement learning, *Markov Decision Process* (MDP) should be defined first [12]. MDP contains three main concepts, a set of states $\boldsymbol{s}_t \in S$, a set of actions $\boldsymbol{a}_t \in A$, and a reward function $R(s, a)$. In each time-step $t \in [0, T]$, the agent interacts with the environment through the states, actions, and reward function. A state $\boldsymbol{s}_t$ describes the environment. The agent takes an action $\boldsymbol{a}_t$ and transits to the next state $\boldsymbol{s}_{t+1}$. The action can be derived from a function called *policy* $\boldsymbol{a}_t = \pi(\boldsymbol{s}_t)$. In other words, a policy is a mapping from perceived states of the environment to actions. In the stochastic environment, a probability of going from state $\boldsymbol{s}_t$ to $\boldsymbol{s}_{t+1}$ is called *transition probability* $P(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$. Upon taking action $\boldsymbol{a}_t$, the agent receives a reward $R(\boldsymbol{s}_t, \boldsymbol{a}_t)$. The aim of the agent is to maximize the cumulative expected future reward until it reaches the horizon if the problem is a finite horizon MDP as described in equation below.

$$R(\boldsymbol{s}_0, \boldsymbol{a}_0) + \gamma R(\boldsymbol{s}_1, \boldsymbol{a}_1) + \gamma^2 R(\boldsymbol{s}_2, \boldsymbol{a}_2) + \cdots . \tag{2.1}$$

Here, γ is a discount factor which is typically strictly less than one. This discounts the effect of future rewards that are far from the current state.

In the reinforcement learning algorithms, *value function* indicates how good it is for the agent to be in a given state $\boldsymbol{s}_t$ and policy $\pi(\boldsymbol{s}_t)$. In other words, high value function means high expected future rewards. The value function can be defined as follows:

$$V(\boldsymbol{s}|\pi) = \boldsymbol{E}[R(\boldsymbol{s}_0, \boldsymbol{a}_0) + \gamma R(\boldsymbol{s}_1, \boldsymbol{a}_1) + \gamma^2 R(\boldsymbol{s}_2, \boldsymbol{a}_2) + \cdots |\boldsymbol{s}_0 = \boldsymbol{s}]. \tag{2.2}$$

Also, with set of state-action pairs, action-value function $Q(\boldsymbol{s}_t, \boldsymbol{a}_t|\pi)$ can be derived as follows:

$$Q(\boldsymbol{s}, \boldsymbol{a}|\pi) = \boldsymbol{E}[R(\boldsymbol{s}_0, \boldsymbol{a}_0) + \gamma R(\boldsymbol{s}_1, \boldsymbol{a}_1) + \gamma^2 R(\boldsymbol{s}_2, \boldsymbol{a}_2) + \cdots |\boldsymbol{s}_0 = \boldsymbol{s}, \boldsymbol{a}_0 = \boldsymbol{a}]. \tag{2.3}$$

The equation (2.3) is also known as *Q-function*. These functions can be also expressed as Bellman equation form.

$$V(\boldsymbol{s}|\pi) = R(\boldsymbol{s}, \pi(\boldsymbol{s})) + \gamma \sum_{\boldsymbol{s}' \in S} P_{\boldsymbol{s}\pi(\boldsymbol{s})}(\boldsymbol{s}')V(\boldsymbol{s}'|\pi) \tag{2.4}$$

$$Q(\boldsymbol{s}, \boldsymbol{a}|\pi) = R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}' \in S} P_{\boldsymbol{sa}}(\boldsymbol{s}')Q(\boldsymbol{s}', \pi(\boldsymbol{s}')|\pi) \tag{2.5}$$

The optimal value function and Q-function can be defined as optimal expected sum of rewards as follows:

$$V^*(\boldsymbol{s}|\pi) = \max_{\pi} V(\boldsymbol{s}|\pi) \tag{2.6}$$

$$Q^*(\boldsymbol{s}, \boldsymbol{a}|\pi) = \max_{\pi} Q(\boldsymbol{s}, \boldsymbol{a}|\pi) \tag{2.7}$$

Therefore, the optimal policy π^* can be defined as follows:

$$\pi^*(\boldsymbol{s}) = \operatorname*{argmax}_{\boldsymbol{a} \in A} Q^*(\boldsymbol{s}, \boldsymbol{a}|\pi). \tag{2.8}$$

These equations can be solved by dynamic programming-based algorithm. In other words, the solution can be derived by iterative update.

$$V(\boldsymbol{s}) := \max_{\boldsymbol{a}} R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}' \in S} P_{\boldsymbol{sa}}(\boldsymbol{s}')V(\boldsymbol{s}') \tag{2.9}$$

$$\pi(\boldsymbol{s}) := \operatorname*{argmax}_{\boldsymbol{a} \in A} R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}' \in S} P_{\boldsymbol{sa}}(\boldsymbol{s}')V(\boldsymbol{s}') \tag{2.10}$$

The equation (2.9) is called *value iteration* and the equation (2.10) is called *policy iteration* [13]. The principles can be combined with function approximators such as deep neural networks.

In this work, *actor-critic method* [33] is used to train and perform the agent. The approaches include two approximators, actor and critic. The actor updates parameters for policy π and the critic updates value function parameters. Specifically, the actor observes a new state $\boldsymbol{s}_{t+1}$ from the environments and selects an action $\boldsymbol{a}_{t+1}$. The critic also receives the state $\boldsymbol{s}_{t+1}$ and additionally, gets reward $R(\boldsymbol{s}_t, \boldsymbol{a}_t)$ from the previous iteration and derives a value function. With the value function, the critic evaluates the policy and gives the direction to the actor. This method can reduce the variance of policy gradient because of the critic.

2.2 Classical Navigation Methods

Classical navigation methods are categorized into two, centralized and decentralized. The centralized approaches assume the comprehensive knowledge about all robots and their environment [14]. In the approaches, each robot is controlled by a central server. These methods focus on getting an optimal solution for path planning and timing for robots simultaneously. However, since these methods depend heavily on the central server, reliable communication network between all robots and the server is mandatory. Additionally, the system fails to solve the problem if unknown obstacles, which are not in the prior knowledge, appear in the environment. This is a serious problem in the real-world scenarios. Moreover, centralized approaches are inapplicable in the high-dimensional problems, which include large number of robots, because considering every dynamics of robots imposes heavy computational burden.

On the other hand, in the decentralized approaches, each robot independently derives its own solution that can avoid collision. Rapidly-exploring random tree (RRT) based methods generate random tree on each iteration to cope with multiple unknown obstacles [15], [16]. From starting position to goal position, the method formulates a tree-like network randomly and finds the branches that guarantee the collision-free path for the robot. The random tree is updated periodically to handle the changing environment.

Velocity obstacle (VO) [17] based methods use the current velocity information of obstacle robots to avoid collision. In the VO, the obstacles and other robots are represented with the main robot's velocity space. With this information, VO region can be calculated. The main robot can avoid collision with the obstacles and other robots if it chooses the velocity outside of the region. However, the method has a problem of generating oscillatory motions in the chicken scenario.

To overcome the limitation, [18] suggests reciprocal velocity obstacle (RVO). When the two robots are facing each other, each robot takes the half of the responsibility for collision avoidance. With the method, two robots can avoid collision without explicit communication. However, the approach fails in some cases when the two robots fail to agree on which side to pass each other which is known as a reciprocal dance.

To remedy the problem, [19] presented the concept of hybrid reciprocal velocity avoidance (HRVO). In the method, a robot that tries to graze on the other robot has the full responsibility for collision avoidance. However, the HRVO does not guarantee the optimal solution in the multiple robot scenarios. This is because two robots can be influenced by other robots while selecting the velocity. [20] tries to solve the drawback of HRVO by providing additional condition for handling multi-robot collision avoidance. The method is called optimal reciprocal collision avoidance (ORCA) and it is the state-of-the-art method for collision avoidance problem. However, these VO-based methods should also dynamically re-plan the path to handle the moving obstacles.

Artificial potential field (APF) based methods require a set of information to generate a potential field: a starting position, a goal position, attractive potential, and repulsive potential [21]. With the generated field, an agent robot can find an optimal path for avoiding collision. In addition, if the unknown obstacles appeared in the environment, the methods re-generate the potential field and re-plan the path. However, the methods have the assumption that the agent robot has the prior knowledge about the entire environment such as positions and velocities of all obstacles. This assumption runs contrary with the real-world scenarios.

2.3 Learning-Based Navigation Methods

Deep learning-based collision avoidance methods have been frequently studied recently.

Several approaches adopt supervised learning to solve the problem. [22] collects image data of crashing trajectories from a drone and trains how to avoid the circumstances. The collected data are distributed manually into two, positive and negative, based on how far the data were collected from the time of collision. However, the method has the issue of collecting large trajectories since supervised learning depends heavily on the large data set. Also, it is dependent on the hand-crafted label. [23] also uses supervised learning with low resolution images. With the images collected by a human driver, the vehicle robot can avoid obstacles. However, this method also has the problem of collecting large data set and additionally, it depends on the skilled human driver. Further, the method can only solve static obstacles. [24] uses depth image to control a mobile robot. However, their commands are empirically chosen discrete actions which is not suitable for elaborate safe driving. [25] applied end-to-end learning with 2D laser scanner. The approach trained the robot by an expert demonstration in a supervised manner. Even though it can handle unforeseen obstacles, the method is still dependent on the data collected by an expert. [26] adopt imitation learning techniques to solve the problem. With a single camera, the UAV can avoid immobile obstacles.

However, the approach can only control left and right discrete movements with fixed forward velocity. Therefore, it can be vulnerable if obstacles are movable. To overcome the limitation, [27] adds forward and spin commands from [26] in the learning process. Although the method can control the UAV more flexibly, it only focuses on avoiding collision with walls on a corridor, not with unknown obstacles. [28] presented a deep auto-encoder based method for the problem. The approach utilize deep auto-encoder to learn actions that can avoid collision with walls and obstacles. However, this method does not have the ability to reach target position since it has never learned the objective.

Several other approaches adopt reinforcement learning to solve the problem. [29] trains an agent robot in a small environment and transfer the learned policy into high-dimensional environments. The small training environment include one agent and one obstacle. If the agent escapes the training range without collision, it receives positive reward and if the agent collides with the obstacle, it receives negative reward. The trained policy can be utilized into high-dimensional environments with multiple obstacles. However, even though the method can avoid collision with multiple obstacles, it requires extra policy for reaching a goal point thus, it is inefficient. Also, the solution for avoiding obstacle is not unique. Therefore, the solution can be easily converged to the local minima.

PrEference Appraisal Reinforcement Learning (PEARL) [30] also trains a robot in a small and restricted environment which includes four obstacles and one goal as described in Fig. 2.1. In the environment, the robot learns how to avoid obstacles and reach a goal. With the trained policy, robot can solve the dynamic obstacle avoidance problem in high dimensional problems. This method does not require whole infor-

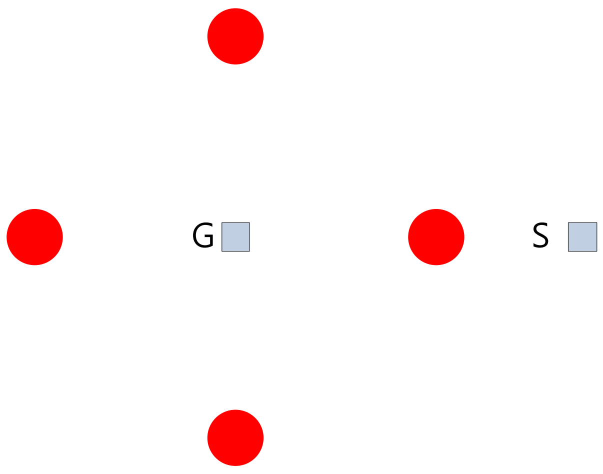

Figure 2.1: Training environment of PEARL. Four red circles are obstacles and S represents the starting position. The agent robot starts from the starting position and should reach the goal point G without colliding with four obstacles (red circles). Only the closest obstacle is considered for the collision avoidance.

mation of the environment since the agent robot generates optimal action only with one closest obstacle robot. Even though the method does not need large data and consumes less computational burden, the inefficiency lies in the process of training the agent since it has to learn two tasks: 1) how to avoid obstacles and 2) how to reach a goal point. Therefore, the challenge posed by reproducibility [5] exists in the method. Moreover, PEARL only considers the closest obstacle from the agent so it can be vulnerable in the crowded environment.

[31] is another example of reinforcement learning-based collision avoidance. The method divides the system into two, target-driven system and collision avoidance system to make learning process efficient. The target-driven system is designed manually. The only thing that the agent learns is avoiding dynamic obstacles with laser scanner

data. Since the agent learns only one task, how to avoid obstacles, the method can reduce the training difficulty of the model. However, the approach does not restrict the number of obstacles. Therefore, there exists myriad of states exponentially proportional to the number of obstacles and this requires large data for learning. Also, the method has the dependency on the manually tuned parameters when combining two sub-systems. Finally, experiments of both [30] and [31] are conducted only on the simulated environments. Therefore, it does not guarantee the applicability of real-world robots.

Chapter 3

Learning Collision

3.1 Introduction

The purpose of the proposed method is to make an agent robot reach a goal point without colliding with multiple obstacles. Since the method adopts the concept of reinforcement learning, the training process for the agent robot is necessary. This chapter introduces how to train the agent robot to achieve its objective by using reinforcement learning. The learned policy from the training process can be applied to both single robot and multi-robot collision avoidance problem which will be covered in Chapter 4 and 5. For training the agent, actor-critic algorithm is used. In the training process, the agent robot is assumed to be a holonomic robot which has more advantages on locomotion control than non-holonomic robots. In addition, the training process is only conducted on a simulation. Even though the trained policy is generated from the holonomic robot, it can be applied to both holonomic and non-holonomic robots. Also, the simulation data can be utilized to both simulation and real world experiment without any re-training.

3.2 Learning Collision

In the training process, an agent robot learns how to collide with a single obstacle robot. The training environment is composed of three objects, an agent, an obstacle, and a training range. When the agent successfully collide with the obstacle, it receives positive reward. However, when the agent escapes the training range without colliding with the obstacle, it receives negative reward. By this process, the agent robot can learn how to collide with the obstacle.

Learning to collide with an obstacle is efficient in terms of learning since the trained policy can be generated only by an attractor. The previous methods with reinforcement learning train the agent to distinguish attractor and repellent. In the methods, the agent has to learn how to avoid repellent and pursue the attractors. However, in the proposed method, only attractor is considered during the training process. Therefore, it is less complicated than the training processes with both attractor and repellent. Also, the trained policy can be utilized for both attractors and repellents. For attractors such as a goal point, the learned policy can be directly applied to the agent so that it can 'collide' with the goal. Simultaneously, for repellents such as obstacles, the policy can be implemented reversely to avoid them. To sum up, with a single set of learned policy, the agent robot can handle both attractors and repellents at once.

3.2.1 Markov Decision Process Setup

Reinforcement learning is based on the Markov Decision Process (MDP). Therefore, MDP should be defined for the algorithm. The MDP state space S can be defined as

$$\boldsymbol{s}_t = \boldsymbol{l}^o - \boldsymbol{l}^a, \forall \boldsymbol{s}_t \in S \tag{3.1}$$

where l^o and l^a are location vectors of an obstacle and an agent robot, respectively. Therefore, state s_t indicated on equation (3.1) is a relative vector between the obstacle and the agent robot. Action space A can be defined as 9 dimensions which include eight cardinal points and one static. The agent receives positive reward $+\delta$ when it collides with the obstacle and negative reward $-\delta$ when it escapes the training range T without the collision.

3.2.2 Training Algorithm

In the training algorithm circular shaped training range T with radius $r^T$is assumed. An obstacle is located on the boundary of T and changes its location on every training episode with respect to the following equation:

$$\boldsymbol{l}^o_{i+1} = \begin{pmatrix} cos\phi & -sin\phi \\ sin\phi & cos\phi \end{pmatrix} \left(\boldsymbol{l}^o_i - \boldsymbol{c}^T\right) + \boldsymbol{c}^T \tag{3.2}$$

where c^T indicates the center of T, i represents the current episode ($1 \leq i \leq n^e$), and ϕ implies unit angle for rotation. n^e is the maximum number of episode. The agent robot is assumed to be holonomic robot since it is more efficient in terms of learning than non-holonomic robot which has restrictions on changing the direction. The agent robot is initially located on the center of the training range c^T. In addition, margin m is applied on the size of the obstacle robot to improve the performance. The margin acts as a safety zone. Even though the actual radius of the obstacle, r^o, is same as that of agent robot, r^a, the agent perceive the size of the obstacle as $r^o(1 + m)$ in the learning process. When the agent robot collides with the obstacle robot including margin, it receives positive reward $R(s_t, a_t) = \delta + F(s)$. $F(s)$ is a heuristic calibration parameter to avoid the graze collision. In the proposed method, calibration factor is

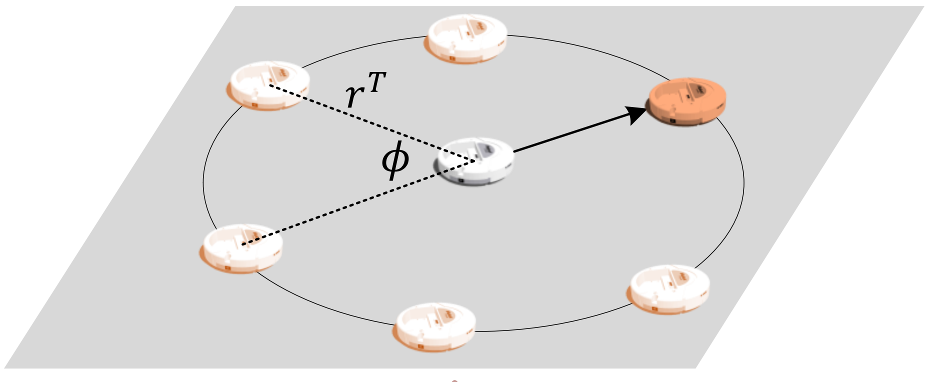

Figure 3.1: Training environment overview. An agent (White robot) can learn how to collide with an obstacle robot (Red robot). The obstacle robot rotates its position based on the unit angle ϕ. The expected position of the obstacle robot with respect to ϕ is described as blurred robot. The radius of the training environment is r^T

determined as follows.

$$F(\boldsymbol{s}) = r^e - \|\boldsymbol{c}^e - \boldsymbol{c}^a\| \tag{3.3}$$

The purpose of equation (3.3) is to lead frontal collision between the agent and the obstacle since learning non-frontal collision can decrease the performance. In other words, the agent can get higher reward for frontal collision than graze collision. On the contrary, when the agent escapes the range T without collision, it receives negative reward $R(s_t, a_t) = -\delta$. Finally, to prevent local minima, the agent receives $R(s_t, a_t) = -0.1$ on every iteration. The overall process is descried on Algorithm 1 and Fig. 3.1.

For training the agent, actor-critic algorithm is used [33]. The two networks, actor and critic, are composed of simple structure, one hidden layer with 128 nodes. For optimizer, Adam [34] is used with 0.00002 actor learning rate and 0.00005 critic learning rate. Even though the network is simple, it can solve the high dimensional problems. Also, it is easy for the training algorithm to be converged fast due to the light network. The training process is only done on simulation. The trained policy from the simulation can be applied to both simulated and real-world experiments. The neural network is implemented and executed by Keras 2.1.2.

Algorithm 1 Training Algorithm

```
Require: MDP $(S, A, D, R)$, training range $\boldsymbol{E}$
1: take action $a_t \in A$
2: if $\left\|\boldsymbol{c}^a - \boldsymbol{c}^T\right\| \geq r^T$ then
3:     $R(\boldsymbol{s}_t, \boldsymbol{a}_t)$ = -$\delta$
4: else if $\|\boldsymbol{c}^a - \boldsymbol{c}^o\| \leq r^a + (1+m)r^o$ then
5:     $R(\boldsymbol{s}_t, \boldsymbol{a}_t)$ = $\delta$ + $F(\boldsymbol{s})$
6: else
7:     $R(\boldsymbol{s}_t, \boldsymbol{a}_t)$ = -0.1
8: end if
9: update parameters
```

3.2.3 Experimental Results

To validate the reproducibility and convergence of CALC, the convergence rate of CALC is compared with that of PEARL [30]. The training environment of PEARL includes one agent, one goal, and four obstacles. The agent has to avoid static obstacles and reach the goal point. The agent receives 2000 rewards when it reaches its goal point and -30 rewards when it collides with one of the obstacles. Also, it receives negative rewards for every iteration based on the following equation:

$$R(\boldsymbol{s}, \boldsymbol{a}) = \frac{\sqrt{(x - G_x)^2 + (y - G_y)^2}}{100000.0} \tag{3.4}$$

where (x, y) is the current location of the agent and (G_x, G_y) indicates the position of the goal.

The overall convergence rate for CALC and PEARL are shown in Table 3.1. The proposed method has shown the perfect convergence rate whereas PEARL performs poorly. Fig. 3.2 shows reward plots for three cases, convergence, divergence, and local minima. As described in Fig. 3.2b, PEARL converges to 0 reward even though the agent has experienced 2000 rewards in the early episodes. In Fig. 3.2d, PEARL oscillates its reward values. This describes the situation where the agent performs collision and reaching the goal in turn. However, CALC converges safely into the optimal solu-

Table 3.1: Convergence rate of CALC and PEARL

Methods	Convergence	Divergence	Local Minima
CALC	100%	0%	0%
PEARL	22.22%	29.63%	48.15%

tion which can make the agent get maximum rewards (Fig. 3.2a) Additionally, CALC converges at the earlier episodes (Fig. 3.2a) compared with PEARL (Fig. 3.2c), which indicates the efficiency of the algorithm. To be specific, PEARL converges at about 1500 episodes whereas CALC converges at 600 episodes.

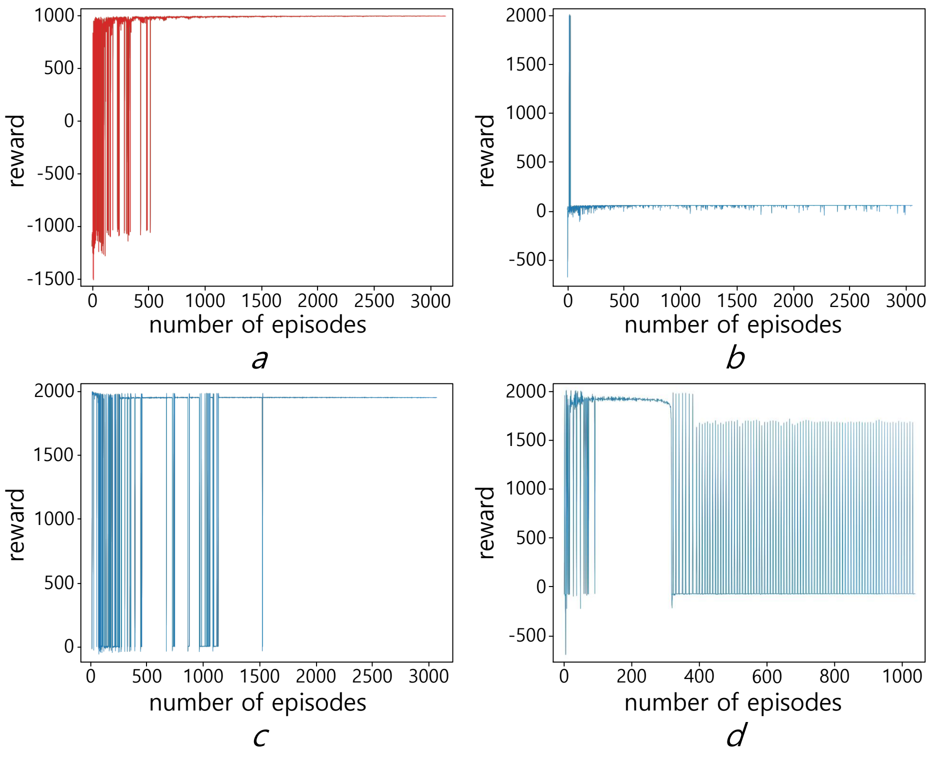

Figure 3.2: Reward plots for CALC and PEARL. x-axis indicates number of episodes and y-axis is received reward of the agent. CALC converges at around 600 episodes and PEARL converges at around 1500 episodes.

a. A reward plot for CALC. The algorithm converges to the optimal solution.

b. A reward plot for PEARL - local minima case

c. A reward plot for PEARL - convergence case

d. A reward plot for PEARL - divergence case

Chapter 4

Single Robot Collision Avoidance

4.1 Introduction

In this section, the learned policy from Chapter 3 is applied on the single robot collision avoidance problem. Single robot collision avoidance problem is defined as an ability of one agent robot to reach its goal without colliding with multiple moving obstacles. At first, the problem with holonomic robot is considered in Chapter 4.2 since it has to be proved before applying the method into the problem with non-holonomic constraints. After that, the method is applied on the single non-holonomic robot collision avoidance problem in Chapter 4.3.

In the problem, the agent and the obstacle are assumed as a circular shaped robot. When the agent and obstacle collide with each other, one episode is counted as fail. If the agent reach the goal point without colliding with obstacles, the episode is counted as success. Collision is defined as follows:

$$||\boldsymbol{c}^o - \boldsymbol{c}^a|| \leq r^a + r^o \tag{4.1}$$

where $\boldsymbol{c}^o$ and $\boldsymbol{c}^a$ are the center of the obstacle and the agent respectively. Reaching to the goal is also defined as follows:

$$||\boldsymbol{c}^g - \boldsymbol{c}^a|| \leq r^a \tag{4.2}$$

where $\boldsymbol{c}^g$ indicate the location of the goal point.

The proposed method can be applied to both holonomic and non-holonomic robots. The experiments are conducted only with the simulation. For single holonomic robot collision avoidance problem, game library called PyGame [32] is used. For non-holonomic robot problem, Gazebo simulator with Robot Operating System (ROS) [35] is used. ROS is an open source operating system for robots. It provides functions such as communication and controlling robots. Gazebo is a ROS-based 3D simulator. It provides 3D environments and many models for robots.

4.2 Holonomic Robot Obstacle Avoidance

4.2.1 Approach

In the single holonomic robot collision avoidance problem, the agent and obstacles are assumed as holonomic robots. The only information that the agent can notice is positions of the obstacles inside its sight and ultimate goal point. The algorithm that can solve the problem is defined as “planning algorithm”. The planning algorithm is composed of four steps. First, the agent checks the number of obstacles n^o included in its sight range which has the same size and shape as the training range. By using the same size and shape with the training range, the agent can easily utilize the policies from the training environment into the planning environment. Second, the agent sets a local goal, which is also called a virtual goal, inside the sight range that directs the ultimate goal. Therefore, even though the actual goal is positioned outside of the sight range, the policy learned from the training environment can be applied to the local goal to pursue the ultimate goal. Third, the agent infers colliding action vectors with every n^o obstacles $(\boldsymbol{a}_t^1 ... \boldsymbol{a}_t^{n^o})$ inside the sight range. Also, the agent generates an action vector $\boldsymbol{a}^g$ from the local goal. Finally, to avoid obstacles, the developed action vectors from the obstacles should be reversed. However, since the agent should not avoid the goal, the action vector for the goal $\boldsymbol{a}^g$ does not need to be reversed. Since all collisions are independent, the overall action vector can be generated as follows:

$$\boldsymbol{a}^o = \boldsymbol{a}^g \odot f\left((\mathbf{1} - \boldsymbol{a}_t^1) \odot ... \odot (\mathbf{1} - \boldsymbol{a}_t^{n^o})\right) \tag{4.3}$$

$$\boldsymbol{a}^* = \frac{\boldsymbol{a}^o}{(\boldsymbol{a}^o)^T \mathbf{1}} \tag{4.4}$$

where $\boldsymbol{a}_t^i$ $(1 \leq i \leq n_o)$ indicates the action vector generated from the i^{th} obstacle at time t, $\odot$ implies element-wise multiplication and $f(\boldsymbol{v})$ of equation (4.3) is a quanti-

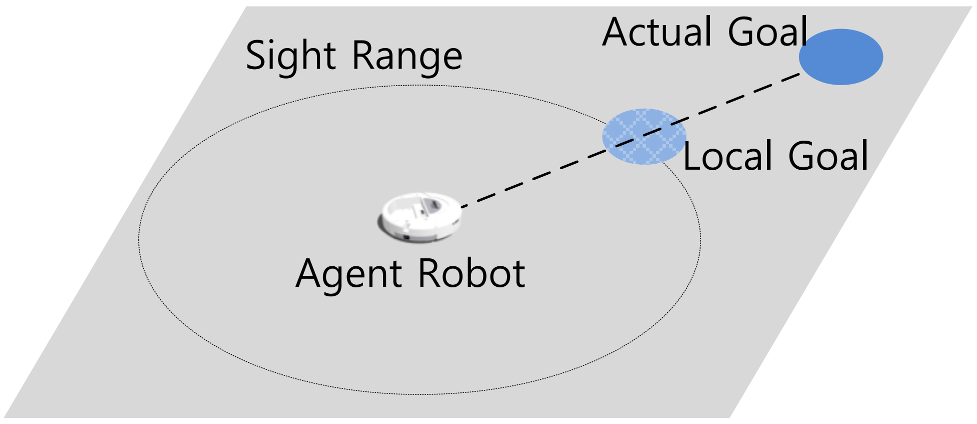

Figure 4.1: Local goal is a virtual goal inside the sight range which directs the global goal. Since the agent robot is trained only inside its sight range, it cannot pursue the global goal outside the range. Therefore, by setting local goal from the direction of the global goal, the agent can easily follow the goal point.

zation function that can be expressed as the following.

$$\{f(\boldsymbol{v})\}_i = \begin{cases} 0, & \text{if } v_i \leq \mu \\ 1, & \text{else} \end{cases} . \tag{4.5}$$

$\{f(\boldsymbol{v})\}_i$ is the i^{th} element of vector $\boldsymbol{v}$ and μ is a constant that can regulate the ratio of 1 and 0 in the vector. Finally, $\boldsymbol{a}^o$ in equation (4.3) can be formulated into multinoulli distribution by equation (4.4). The overall process is described in Algorithm 2.

Algorithm 2 Planning Algorithm

```
Require: MDP (S, A, D, R), $n^o$
Ensure: desired action $\boldsymbol{a}^*$
1: for $i$ = 1, ... ,$n^o$ do
2:     $\boldsymbol{a}^o = \boldsymbol{a}^o \odot (1 - \boldsymbol{a}^i_t)$
3: end for
4: $\boldsymbol{a}^o = f(\boldsymbol{a}^o) \odot \boldsymbol{a}^g$
5: $\boldsymbol{a}^* = \boldsymbol{a}^o \cdot \left((\boldsymbol{a}^o)^T \mathbf{1}\right)^{-1}$
6: return $\boldsymbol{a}^*$
```

4.2.2 Experimental Results

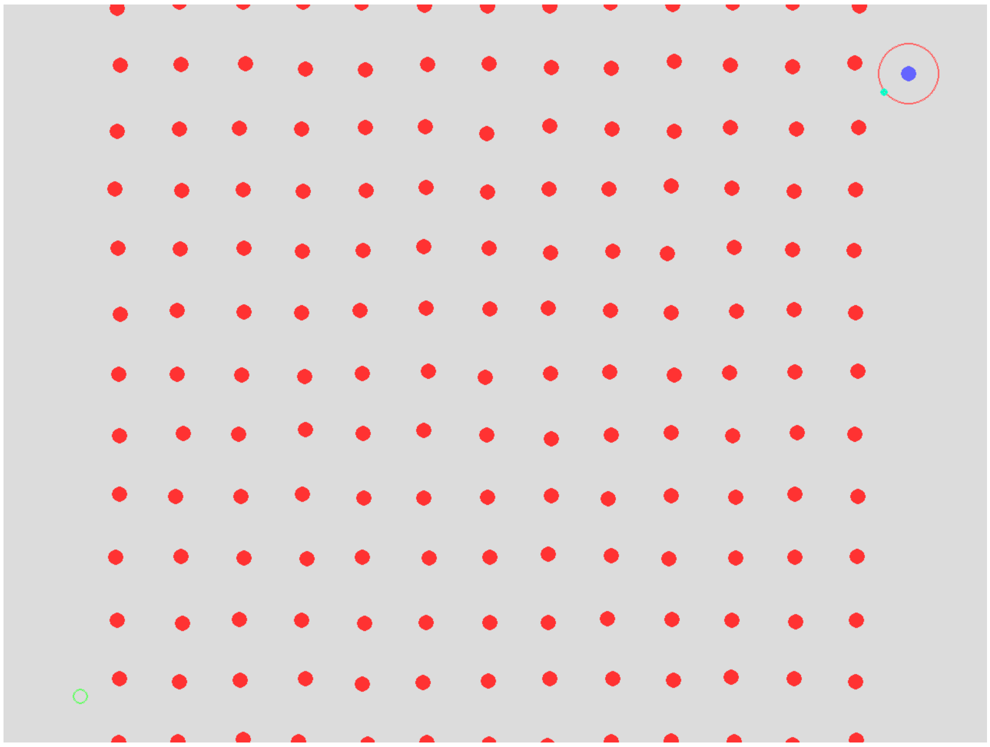

Figure 4.2: An example image of a PyGame simulator. The agent (blue circle) should reach a goal point (green empty circle) without colliding with the obstacles (red circle). The figure is an example of the experiment with 200 obstacles.

The proposed method is implemented with a Python-based game library called PyGame simulator [32] on Ubuntu 16.04 with Intel(R) Core(TM) i7-4790 3.60GHz CPU, 16GB RAM, and NVIDIA GeForce GTX 980. The overall simulation environment is described in Fig. 4.2. In the simulator, the agent and the obstacles are assumed to be circular-shaped holonomic robots with same radius. In the experiment, the agent must travel from a starting position to a goal. Between the starting position and the goal position, moving obstacles are distributed with random dynamics. The agent can only observe the current position of obstacles inside the sight range and the local goal. An episode is terminated when the agent collides with the obstacles or the goal. The experiment is conducted with multiple obstacles such as 50, 100, 150, and 200.

The proposed method is compared with other conventional algorithms. The comparison experiment is conducted with arbitrary reinforcement learning-based method [29], reciprocal velocity obstacle (RVO) [18], and PEARL [30]. For the arbitrary learning-based method which is called forward reinforcement learning (FRL), the agent is trained to avoid collision in the training environment, which is opposite of the proposed approach. FRL is tested on the planning environment to justify the efficiency of the reverse training. Specifically, the agent gets a positive reward when it escapes the training range without collision and a negative reward when it collides with the obstacle in FRL. RVO algorithm is modified to consider 5 closest obstacles from the agent since applying RVO with all obstacles imposes heavy computational burden. Without the modification, the algorithm with 200 obstacles runs in 0.0895 frame per second (fps) whereas fps for 5 closest obstacles is 51.8659. Likewise, the proposed method, CALC, runs in 56.9745 fps with 200 obstacles. PEARL is trained with the environment described in Fig 2.1 and it is trained with 3600 episodes. The overall results are shown in Table 4.1 and they are tested by 250 episodes. The proposed method has the highest success rate among four methods. Also, CALC shows almost consistent performance regardless of obstacles' numbers.

Table 4.1: Success rate comparison with different obstacle number

# of obstacles	RVO [18]	PEARL [30]	FRL	CALC (Proposed)
50	87.6%	95.2%	98.8%	**99.6%**
100	77.2%	92.8%	70.0%	**95.2%**
150	74.8%	70.8%	85.4%	**97.0%**
200	73.6%	66.8%	21.0%	**74.4%**

4.3 Non-Holonomic Robot Obstacle Avoidance

4.3.1 Approach

The main planning algorithm for single non-holonomic robot collision avoidance problem is as same as single holonomic robot collision avoidance in Chapter 4.2.1. Since the agent robot was assumed to be holonomic robot in the training algorithm, conversion from holonomic to non-holonomic is necessary to apply the algorithm into non-holonomic robots. There are three steps for this. First, discrete the angle of robot heading into action space without static (the eight cardinal points) with respect to Table 4.2. Second, find the difference between discrete robot heading and desired action which is derived from trained policy. Finally, apply angular velocity proportional to the difference above. Also, the linear velocity is applied inversely proportional to angular velocity so that it can rotate its heading safely. If the difference of the robot heading and the direction of desired action exceeds 90 degrees, it moves back linearly without angular velocity. The overall arrangement is described in Fig. 4.3.

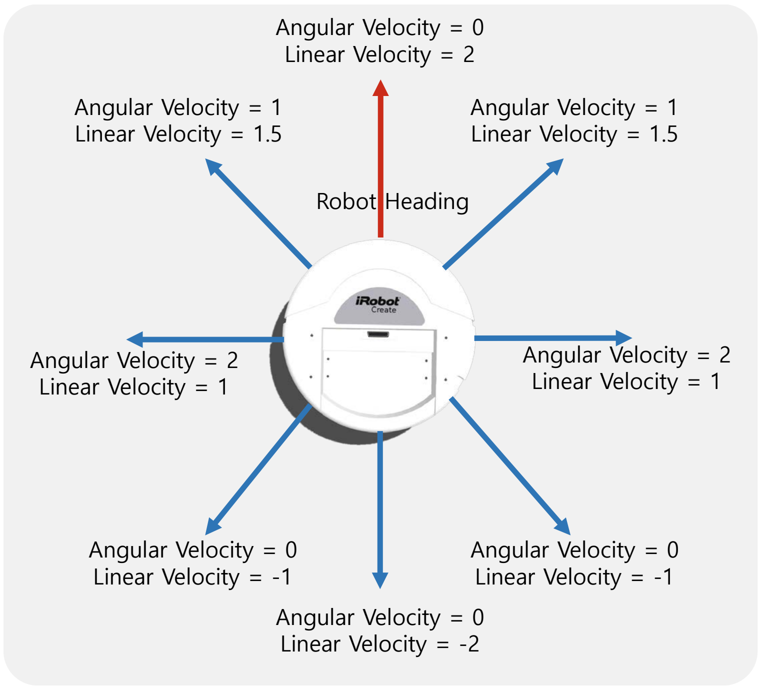

Figure 4.3: An overview of conversion from holonomic to non-holonomic model. Red arrow indicates the heading of a robot. Red arrow and blue arrows represents eight cardinal points which are action vectors.

Table 4.2: Discretization

Robot Heading	Action Space
$-\pi/8 \leq \theta \leq \pi/8$	0
$-3\pi/8 \leq \theta \leq -\pi/8$	7
$-5\pi/8 \leq \theta \leq -3\pi/8$	6
$-7\pi/8 \leq \theta \leq -5\pi/8$	5
$\theta \leq -7\pi/8$ or $7\pi/8 \leq \theta$	4
$5\pi/8 \leq \theta \leq 7\pi/8$	3
$3\pi/8 \leq \theta \leq 5\pi/8$	2
$\pi/8 \leq \theta \leq 3\pi/8$	1

4.3.2 Experimental Results

CALC for single non-holonomic collision avoidance is implemented on a Gazebo simulator with Robot Operating System (ROS) [35] and RosPy. The overall simulation environment is described in Fig. 4.4. For the agent and obstacles, iRobot Create mobile robot [38], whose radius equals to 0.18m is used. The only thing that the agent can perceive is the current position of obstacles inside the sight range and the local goal as described in Fig. 4.1.

On the planning algorithm, the agent must travel from starting position [0m 0m] to the goal point [10m 10m]. Between the start position and the goal point, moving obstacles are distributed with random dynamics. The experiments are proceeded with different number of obstacles from 5 to 10. When the agent collides with both obstacles or goal point, the current episode is terminated. The overall experimental result is listed on the Table 4.3. In every experiments, there exists 2 to 3 collision cases that the agent cannot handle. For example, if the agent is surrounded completely by obstacles, the agent cannot do anything but waiting. These cases do not indicate algorithmic flaw so it can be neglected. Thus, the success rate of Table 4.3 can increase if those cases are excluded. The experimental result proved that the performance of CALC is consistent regardless of obstacle numbers. For the experiments, 200 episodes are used to check the success rate.

Table 4.3: Comparison with different obstacle number

Number of obstacles	Success Rate
5	100.0 %
6	95.52 %
7	89.05 %
8	87.56 %
9	87.87 %
10	90.05 %

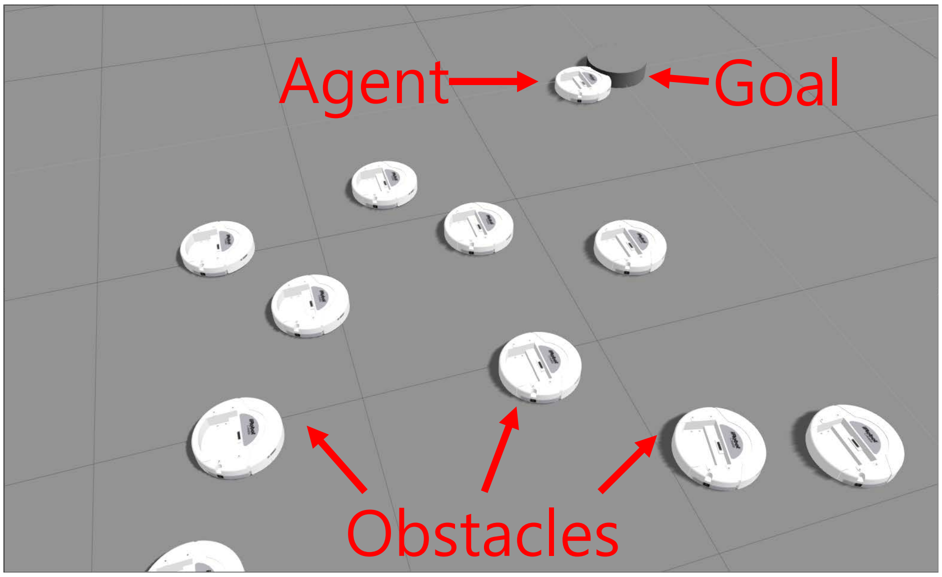

Figure 4.4: Gazebo simulation image of planning environment. Both agent and obstacles are iRobot Create model and goal point is expressed as a cylinder. The image describes the situation when the agent reaches the goal point.

Chapter 5

Multi-Robot Collision Avoidance

5.1 Introduction

In a multi-robot collision avoidance problem, multiple agent robots are included. The purpose of agent robots is to reach their own goal points without colliding with each other. In the point of one agent robot, other robots are regarded as obstacles. In some circumstances, there exists unknown obstacles with random dynamics. If there are obstacles inside the environment, the agent robots should avoid them also. The collision can be defined as illustrated in Chapter 4.1. Each robot derives its own action based on the learned policy of Chapter 3 which is expressed as a probability distribution. Also, to apply the learned policy into non-holonomic robots, holonomic to non-holonomic conversion method is proposed in this chapter. The experiments are conducted with both simulation and real-world experiment. For simulation, Gazebo simulator with Robot Operating System (ROS) is used. For real-world experiment, e-puck mobile robot is used.

5.2 Approach

The main planning algorithm for multi-robot collision avoidance problem is described in Algorithm 3. First, the agent robot detects the local obstacles and a local goal. Goal information is given for the local goal. In the point of one agent robot, other robots are perceived as obstacles. Second, the agent robot derives the colliding policy for each obstacle and reverse the policy. Colliding policy for the local goal is also calculated. After that, the policies are converged with element-wise multiplication and the optimal action vector is derived. The process is proceeded in each agent robot.

Since the agent robot of the training algorithm from Chapter 3 is assumed to be a holonomic robot, conversion from holonomic to non-holonomic is necessary to apply the algorithm into non-holonomic robots. The optimal action derived from the planning algorithm gives information about the direction that the robot should move which is called a desired direction. Therefore, the robot should simultaneously change its

Algorithm 3 Planning Algorithm for Multi-Robot Collision Avoidance

```
Require: MDP (S, A, D, R), $n^o$, $n^a$
Ensure: action vector $\boldsymbol{a}^*$
1: for $j$ = 1, ... ,$n^a$ do
2:     for $i$ = 1, ... ,$n^o$ do
3:         $\boldsymbol{a}^o = \boldsymbol{a}^o \odot (1 - \boldsymbol{a}_t^i)$
4:     end for
5:     $\boldsymbol{a}^o = f(\boldsymbol{a}^o) \odot \boldsymbol{a}^g$
6:     $(\boldsymbol{a}^*)_j = \boldsymbol{a}^o \cdot \left((\boldsymbol{a}^o)^T \mathbf{1}\right)^{-1}$
7: end for
8: return $\boldsymbol{a}^*$
```

heading by angular velocity and move toward the direction by linear velocity. First, the angular velocity is set as proportional to the difference between the heading of agent robot, θ_h, and the desired direction, θ_*, derived from the policy. The angular velocity can be expressed as follows:

$$v_{ang} = k(\theta_h - \theta_*) \tag{5.1}$$

where v_{ang} stands for angular velocity and k indicates the empirical proportional factor. The linear velocity can be calculated based on the following equation:

$$v_{lin} = -\frac{2v_{max}}{\pi^2}(\theta_h - \theta_*)^2 + v_{max} \tag{5.2}$$

where v_{max} and v_{lin} stands for maximum and linear velocity respectively. With the equation (5.2), the robot can move forward when the difference between θ_h and θ_* is from $-\pi/\sqrt{2}$ to $\pi/\sqrt{2}$. This can make the robot pursue forward movement rather than backward which is more reasonable in the real-world scenario.

The overall concept is described in Fig. 5.1. Each robot has its own goal point and they are described as same color. For example, red robot which is in the lower left part of Fig. 5.1 should reach to the dotted red circle which is located in the upper right part. Each robot derives its own probability distribution and chooses the action that has the highest probability of avoiding other robots and obstacles.

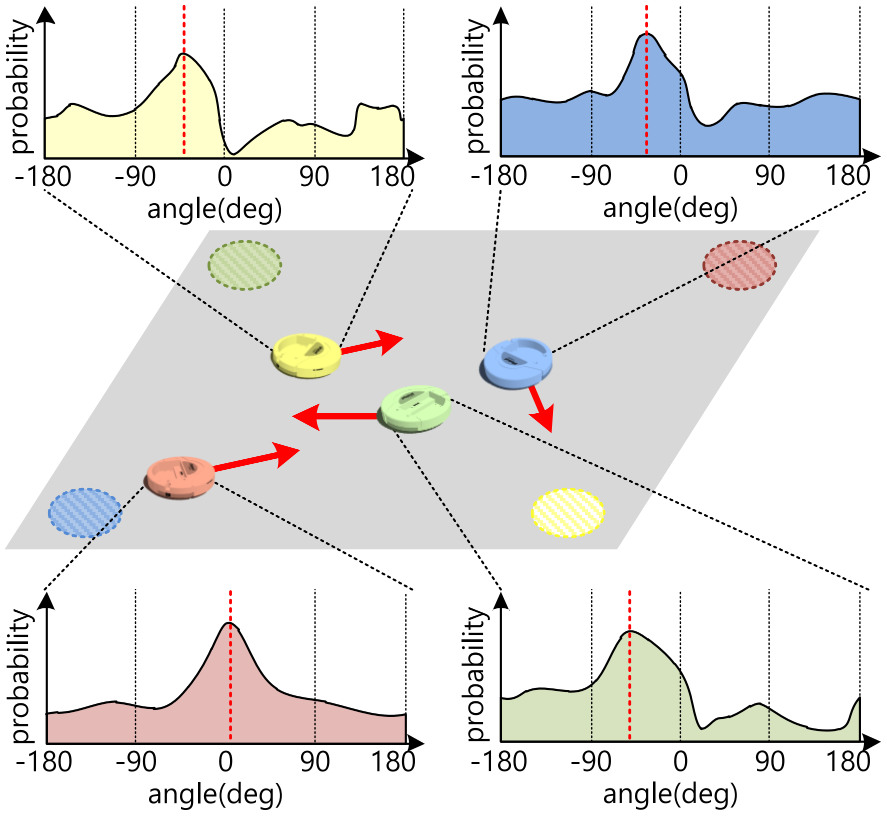

Figure 5.1: An overview of the Multi-robot collision avoidance. The robots should reach their goals separately without colliding with other robots. Each robot has its own probability distribution that indicates the optimal direction for achieving two tasks: to avoid collision and to reach a goal. The red arrows indicate the desired actions for each robot and dotted circles are goal points. The goal points are illustrated with same colors of the agent robots.

5.3 Experimental Results

5.3.1 Simulated Experiment

In the simulated training experiments, CALC is implemented on a Gazebo simulator with Robot Operating System (ROS) [35] and RosPy. For the agent and obstacles, round shaped robot model, iRobot Create mobile robot [38], whose radius equals to 0.18m is used. The iRobot Create mobile robot is a non-holonomic robot which can move by two velocities, linear and angular. Even though the robot is a non-holonomic model, holonomic motion is assumed during the training process. That is, iRobot Create mobile robot is not controlled by linear and angular velocity in the training algorithm. The purpose of the agents is to travel from their initial points to goal points without colliding with other agent robots. All agent robots are initially located on the edge of a rectangle shaped boundary and their goal points are diagonal vertices of the rectangle. Some scenarios include interfering robots with random dynamics inside the environment. The overall scenarios are described in Fig 5.2. The only thing that the agent robot can observe is the current position of other robots inside the sight range and the local goal. An episode is terminated at the time when all agent robots reached their goal points or one of the agent collide with other robots including interfering robots and other agent robots.

The proposed algorithm is compared with the state-of-the-art collision avoidance method, Optimal Reciprocal Collision Avoidance (ORCA) [20], that can perform on the planning environment. ORCA is implemented in Python by adapting the RVO2 library [39]. Since ORCA is for holonomic robots, holonomic to non-holonomic conversion approach is applied to the algorithm. Also, additional information about the velocity of other agent robots is provided to ORCA. In other words, communication

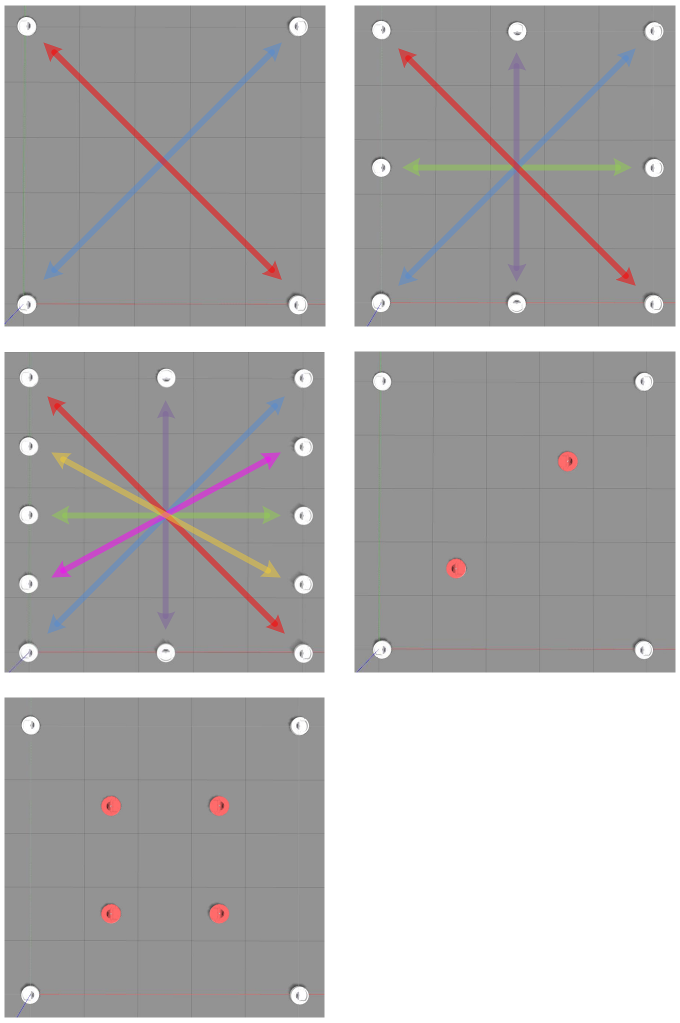

Figure 5.2: Three scenarios are conducted only with agent robots which are white iRobot Create model. Other two scenarios are conducted with agent robots and interfering robots with random dynamics which are described as red robots. The goal points of each agent robot are the initial point of the counter robots. For example, on the 4 robots scenario (Left), the robot on the upper left should go to the lower right position which is the initial point for counter robot. Simultaneously, the counter robot on the lower right should move to the upper left position.

between the agent robots is assumed in ORCA. When all agent robots reached their own goal points, the episode is counted as a successful episode. If one of the agent robots collides with the other robots including the interfering robots, the episode is regarded as fail. The overall results are demonstrated in Table 5.1. The result indicates that CALC shows similar or higher performance than ORCA even though ORCA uses the whole information about the environment. Additionally, there are two environments that contain interfering robots, one with two interfering robots and the other with four interfering robots. The only information about the interfering robots that the agent robots can receive is their position. The agent robots of both CALC and ORCA cannot know the velocity data of the interfering robots. Even though the dynamics of the interfering robots are unknown, CALC performed well compared with ORCA. The success rates of the experiments are shown in Table 5.1.

Table 5.1: Comparison of Success Rate of CALC and ORCA

Method	# of Robots	Success Rate		
		Constant	Linear	Quadratic
CALC (Proposed)	4(0)	97.0%	100%	100%
	8(0)	89.1%	98.0%	98.5%
	12(0)	63.2%	91.0%	87.6%
	4(2)	97.0%	95.0%	95.5%
	4(4)	91.0%	86.6%	87.1%
ORCA [20]	4(0)	100%	100%	100%
	8(0)	90.1%	98.5%	92.5%
	12(0)	56.7%	86.6%	86.5%
	4(2)	92.5%	91.0%	86.6%
	4(4)	88.6%	83.6%	77.6%

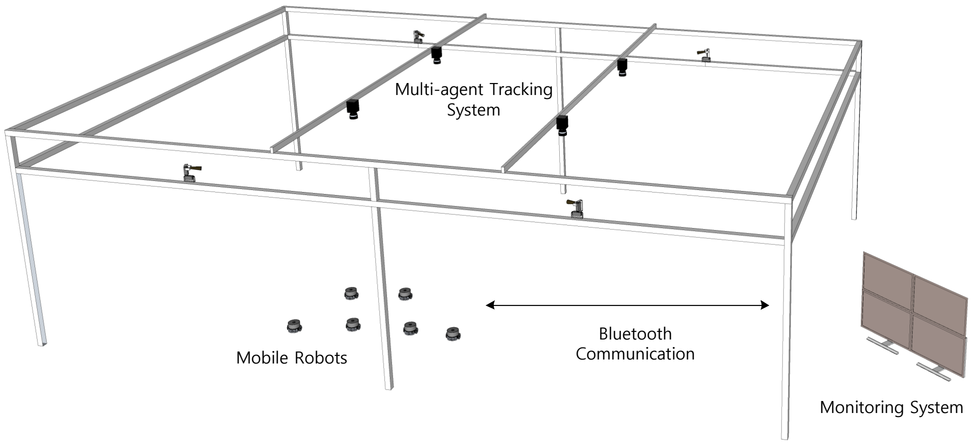

Figure 5.3: An overview image of the intelligent space. Positions and orientations of mobile robots can be tracked and monitored by multi-agent tracking system. The robots can be controlled via bluetooth communication.

5.3.2 Real-World Experiment

To show the applicability of the proposed method to the real-world scenarios, additional experiments are conducted by using a real robot, e-puck [36]. E-puck is a 7cm sized mobile robot with 2 wheels, 10 Light-Emitting Diodes (LED), 8 infrared sensors, 3 microphones, and 1 speaker. Fig. 5.5 is the picture of e-puck robot that is used in the

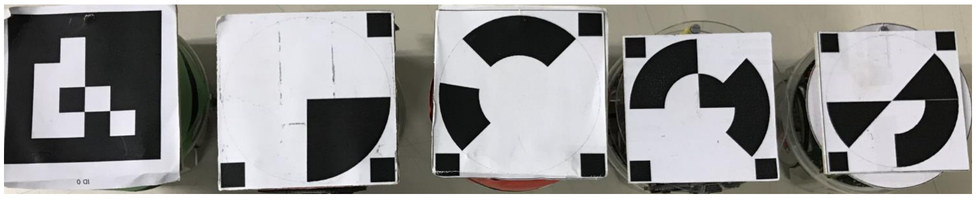

Figure 5.4: Five tags used for catching position and orientation information of the robots.

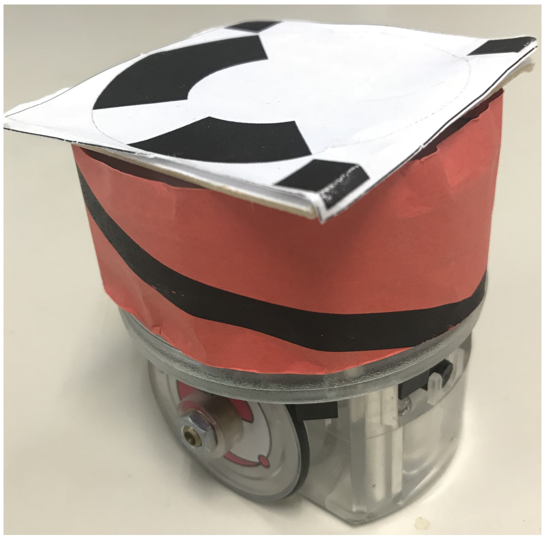

Figure 5.5: An image of e-puck mobile robot used for the real world experiment. The robot is two wheel-based mobile robot with 2 wheels, 10 light-emitting diodes, 8 infrared sensors, 3 microphones, and 1 speaker.

experiments. The experiments are done under the intelligent space (iSpace) [37] as described in Fig. 5.3. In the iSpace, positions of the robots can be tracked by overhead cameras and sensors. In the proposed method, one overhead camera is used for tracking e-puck robot. All e-puck robots have different tags on its head for identification as described in Fig. 5.4. With the tags, iSpace can catch the position and orientation of the robots.

The agent robot is trained from a simulation based training environment with simple game library named PyGame. Since PyGame is a pixel-based library, it is easy to generate any shape of the robots. Circular shaped agent and obstacle which is the same as e-puck robot is considered in the PyGame environment. Except for the shape and the radius, other processes and parameters are as same as Chapter 3. The learning process only took about 475 seconds with 3600 episodes. The trained policy of simulated environment is well applied to the real-world environment without any re-training or calibration. Since the method does not require learning process in the real world, the robot does not need to be damaged during the trial and error. Also, it is independent from any physical constraints such as battery level.

In the planning experiment, perfect sensing for local information is assumed. The positions of each robot are collected by overhead video camera with matrox imaging library (MIL). Each robot has independent tag on its head. MIL camera system detects the tag and finds the position and orientation of the robots. Even though all positions of the robots are collected, the restricted information, which are positions of the other robots within limited sight range, are provided to each robot. The collected data are transmitted to main computer via Transmission Control Protocol/Internet Protocol (TCP/IP) communication. With the data, the main computer derives the action for each robot. The robots are controlled separately with linear and angular velocity instructions via Bluetooth communication. Additionally, reliability scores, which indicate whether the perceived positions are reliable or not, are conveyed to the main computer. If the reliability score of a robot does not exceed 80%, the main computer refuses to move the robot. The overall settings of the experiments are expressed in Fig. 5.6.

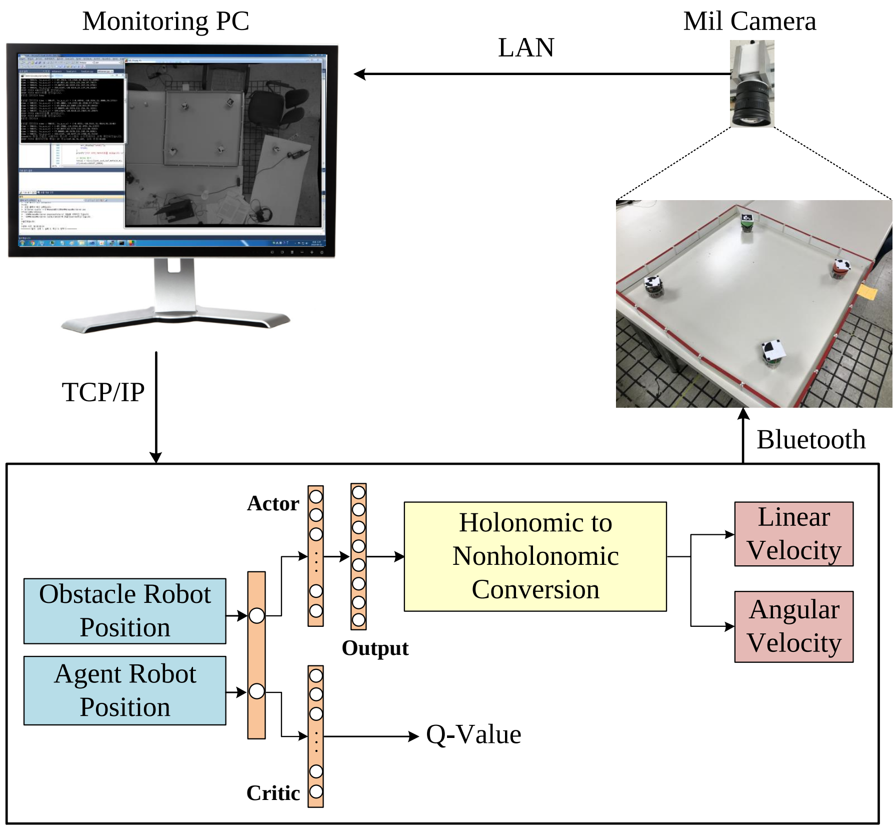

Figure 5.6: Real-world planning experiment environment setting with e-puck mobile robot. The obstacle and agent robot positions are collected from mil camera. With the collected data, the actor-critic network generates linear and angular velocity. Lastly, generated velocities are conveyed to the each e-puck robot via Bluetooth communication.

The experiment is conducted with different number of agents, 2, 3, 4, and 5. The resulting paths of real-world experiment are described in Fig. 5.7. Since the method is a decentralized local planner, the agent robot changes its direction only when the other robots are detected in the sight range. The resulting paths show that the agents have safely reached their goal points without any collision.

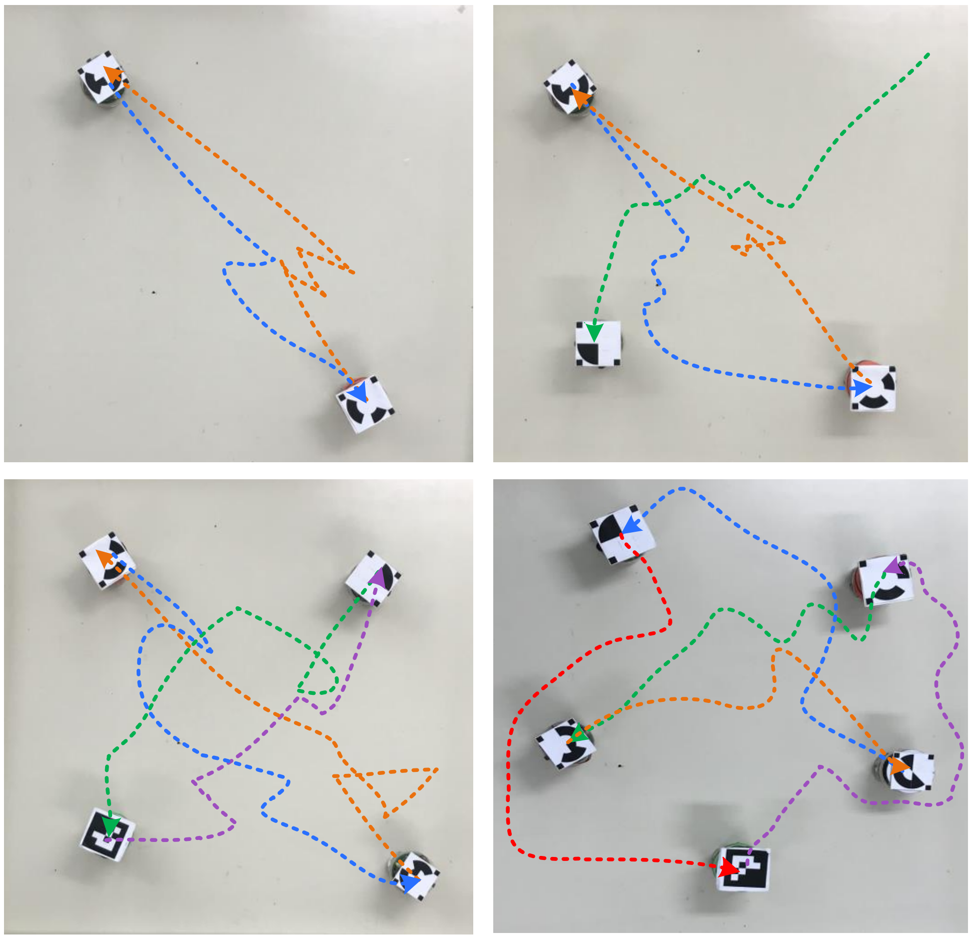

Figure 5.7: The resulting paths for 2, 3, 4, and 5 agent robots experiment. With the trained policy from the simulated environment, real e-puck robots can successfully avoid other robots and reach to the goal without any re-training.

Table 5.2: Consuming Time for One Successful Episode

Velocity Model	Constant	Linear	Quadratic
Average	54.14s	113.71s	$\mathbf{37.12}s$
Median	54.01s	96.49s	$\mathbf{35.88}s$
Standard Deviation	20.52	75.44	$\mathbf{8.90}$

5.3.3 Holonomic to Non-Holonomic Conversion Experiment

For holonomic to non-holonomic conversion approach, three different linear velocity (v_{lin}) models are tested. The overall system is described in Fig. 5.8. First model is a constant model as follows:

$$v_{lin} = \begin{cases} v_{max}, & \text{if } |(\theta_h - \theta_*)| \leq \frac{\pi}{2} \\ -v_{max}, & \text{else} \end{cases} \tag{5.3}$$

Second model is a linear model as described as follows:

$$v_{lin} = \begin{cases} -\frac{2v_{max}}{\pi}(\theta_h - \theta_*) + v_{max}, & \text{if } 0 \leq \theta_h - \theta_* \\ \frac{2v_{max}}{\pi}(\theta_h - \theta_*) + v_{max}, & \text{else} \end{cases} \tag{5.4}$$

Final model is a quadratic model as demonstrated in equation (5.2). All equations are designed to drive a robot with v_{max} when $\theta_h - \theta_* = 0$ and $-v_{max}$ in $|\theta_h - \theta_*| = 2\pi$. The success rates of three models are listed on Table 5.1. Even though the constant model results in the highest success rate in some scenarios, it is not appropriate for both CALC and ORCA since it does not guarantee the steady performance.

In the quadratic model, forward movement covers bigger range ($|\theta_h - \theta_*| \leq \pi/\sqrt{2}$) than backward movement ($|\theta_h - \theta_*| \geq \pi/\sqrt{2}$) whereas in the other two models, forward and backward movement covers equally ($|\theta_h - \theta_*| \leq \pi/2$ and $|\theta_h - \theta_*| \geq \pi/2$).

Therefore, with linear and constant models, the robot used to oscillate back and forth a lot and this increases consuming time per one episode. However, since quadratic model prefers forward moving, the model did not show the oscillating movement and this results in less consuming time. Data about consuming time are listed in Table 5.2. It is obvious that the quadratic model consumes less time than other two models and it has less standard deviation. This indicates that the quadratic model is appropriate for CALC even though the success rate of the linear and the quadratic model are similar. The experiment was conducted by 8 multi-robot system scenario with no interfering robots.

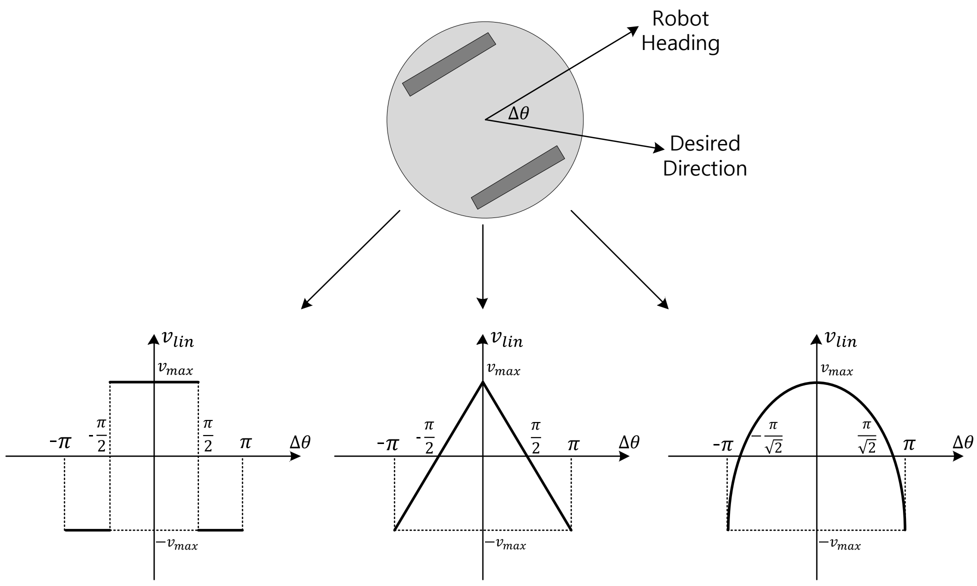

Figure 5.8: The angular velocity is decided proportional to the difference between robot heading and desired direction ($\Delta\theta$). Also, linear velocity is arranged with respect to three models: constant, linear, and quadratic.

Chapter 6

Conclusion

In this thesis, a method for single robot and multi-robot collision avoidance problem by using the concept of reinforcement learning is suggested. First, a new idea of learning collision is suggested in the thesis. The method learns collision instead of avoiding an obstacle in the training environment which comprise of one agent robot, one obstacle, and training range. In the training environment, the agent robot learns how to collide with the obstacle and derives the colliding policy. The experimental result shows the efficiency of the collision-learning method.

Second, a method for expanding the pre-trained policy into high-dimensional problem is proposed. The pre-trained policy from the restricted and small-sized environment can be applied to the problems with multiple obstacles and robots such as single robot collision avoidance and multi-robot collision avoidance. In other words, by learning collision with only one obstacle, the agent can successfully solve the problems with multiple dynamic obstacle or multiple agent robots. To avoid obstacles and other robots, the actions derived from the policy should be reversed. Simultaneously, to pursue a goal point, the actions for the goal should be applied directly. By this process, the agent robot can successfully reach the goal point without collision.

Third, the method is successfully applied to both holonomic and non-holonomic robots. The simulated experiments with PyGame and Gazebo simulator with ROS are conducted to prove the performance of the proposed algorithm. Holonomic robot experiments are conducted with PyGame simulator. Non-holonomic robot experiments are conducted with Gazebo simulator with iRobot Create mobile robot. The experimental results show that the proposed method can guarantee higher performance in avoiding collision compared with other methods such as RVO, PEARL and ORCA.

Finally, the experiment with real e-puck mobile robot shows the applicability of the proposed method into real-world robots such as Unmanned Ground Vehicle (UGV) and car. It also shows that the learned policy from the simulation can be directly applied to the real robots without re-training. This enables the system to avoid damaging of robots during the trial and error in the reinforcement learning process. Furthermore, the method is successfully implemented into two different non-holonomic robot types: iRobot Create and e-puck. This shows the scalability of the proposed method that the algorithm can be applied to any robot regardless of its type.

In future work, the proposed method can be applied to the robots with high degree of freedom (DOF) such as quadrotors and UAVs. Mobile robots used in the experiments of the proposed method move on the 2D plane. However, quadrotors and UAVs should move in 3D space which needs additional actions compared with the mobile robots. Therefore, expansion of the action space should be required. Also, current action covers 8 cardinal directions and one static. This can be specified into more detailed action space such as 16 directions or more. With the expansion, the robot can move more accurately and smoothly.

Additionally, more real-world experiments should be conducted in order to prove the applicability of the proposed method. In the thesis, only one real-world robot, e-puck, is used for the experiment. However, additional experiments with other mobile robots such as Turtlebot and Pioneer robot can be proceeded. Also, not only two-wheeled mobile robot, but also other type of non-holonomic robots such as car-like robot or bicycle-like robot can be used for the experiment.

Moreover, it is possible to improve the holonomic to non-holonomic conversion method with more precise and detailed function. In this thesis, only three functions, constant, linear and quadratic, are used for the conversion. However, other functions such as logarithmic function can make an improved performance. It should be proved with more precise experiments. Therefore, detailed analysis for the conversion method should be proceeded. Also, comparison between existing holonomic to non-holonomic conversion method such as Non-holonomic ORCA (NH-ORCA) [40] and the proposed method should be conducted.

Bibliography

[1] E. Yoshida, C. Esteves, T. Sakaguchi, J. P. Laumond, and K. Yokoi, “Smooth collision avoidance: Practical issues in dynamic humanoid motion,” in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, October 2006, pp. 827-832.

[2] D. Gandhi and E. Cervera, “Sensor covering of a robot arm for collision avoidance,” in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics,* Washington D.C., USA, October 2003, pp. 4951-4955.

[3] M. Hoy, A. S. Matveev, and A. V. Savkin, “Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey,” *Robotica*, vol. 33, no. 3, pp. 463-497, 2015.

[4] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, “A survey of motion planning and control techniques for self-driving urban vehicles,” *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33-55, 2016.

[5] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, “Deep reinforcement learning that matters,” *arXiv preprint arXiv:1809.06560*, 2017.

[6] Y. Li, “Deep reinforcement learning: An overview” *arXiv preprint arXiv: 1701.07274*, 2017.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529-533, 2015.

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484-489, 2016.

[9] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Grapel, and D. Hassabis, “Mastering the game of Go without human knowledge,” *Nature*, vol. 550, no. 7676, pp. 354-359, 2017.

[10] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, F. -F. Li, and A. Farhadi, “Target-driven visual navigation in indoor scenes using deep reinforcement learning,” in *Proceedings of IEEE International Conference on Robotics and Automation*, Singapore, May 2017, pp. 3357-3364.

[11] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data

collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421-436, 2018.

[12] R. S. Sutton, A. G. Barto, and F. Bach, *Reinforcement learning: An introduction*, MIT press, 1998.

[13] A. Y. Ng, "Shaping and policy search in reinforcement learning," Ph. D. Dissertation, CS Department, Univ. of California, Berkeley, 2003.

[14] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *Proceedings of IEEE International Conference on Robotics and Automation*, St. Paul, MN, USA, May 2012, pp. 477-483.

[15] D. Connel and H. Manh La, "Extended rapidly exploring random tree-based dynamic path planning and replanning for mobile robots," *International Journal of Advanced Robotic Systems*, vol. 15, no. 3, p. 1729881418773874, 2018.

[16] M. Otte and E. Frazzoli, "Rrtx: Asymptotically optimal single-query sampling-based motion planning with quick replanning," *The International Journal of Robotics Research*, vol. 35, no. 7, pp. 797-822, 2016.

[17] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 760-772, 1998.

[18] J. Van Den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacle for real-time multi-agent navigation," in *Proceedings of IEEE International Conference*

on Robotics and Automation, Pasadena, California, USA, May 2008, pp. 1928-1935.

[19] J. Snape, J. Van Den Berg, S. J. Guy, and D. Manocha, “The hybrid reciprocal velocity obstacle,” *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 696-706, 2011.

[20] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, “Reciprocal n-body collision avoidance,” in *Robotics research*, ser. Springer Tracts in Advanced Robotics, C. Pradalier, R. Siegwart, and G. Hirzinger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, vol. 70, no. STAR, pp. 3–19.

[21] O. Montiel, U. Orozco-Rosas, and R. Sepúlveda, “Path planning for mobile robots using bacterial potential field for avoiding static and dynamic obstacles,” *Expert Systems with Applications*, vol. 42, no. 12, pp. 5177-5191, 2015.

[22] D. Gandhi, L. Pinto, and A. Gupta, “Learning to fly by crashing,” in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vancouver, Canada, September 2017, pp. 3948-3955.

[23] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. L. Cun, “Off-road obstacle avoidance through end-to-end learning,” in *Advanced in neural information processing systems*, pp. 739-746, 2006.

[24] L. Tai, S. Li, and M. Liu, “A deep-network solution towards model-less obstacle avoidance,” in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Daejeon, Korea, October 2016, pp. 2759-2764.

[25] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, “From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots,” in *Proceedings of IEEE International Conference on Robotics and Automation*, Singapore, May 2017, pp. 1527-1533.

[26] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, “Learning monocular reactive uav control in cluttered natural environments,” in *Proceedings of IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany, May 2013, pp. 1765–1772.

[27] D. K. Kim and T. Chen, “Deep neural network for real-time autonomous indoor navigation,” *arXiv preprint arXiv:1511.04668*, 2015.

[28] J. Sergeant, N. Sünderhauf, M. Milford, and B. Upcroft, “Multimodal deep autoencoders for control of a mobile robot,” in *Proceedings of Australasian Conference for Robotics and Automation*, Canberra, Australia, Dec. 2015.

[29] 전호웅, 이범희, “저차원 학습 환경을 이용한 고차원 환경의 동적 장애물 회피 방법,” *제어로봇시스템학회 2018년 제33회 학술대회*, 2018년 05월. pp. 214-215.

[30] A. Faust, H. T. Chiang, N. Rackley, and L. Tapia, “Avoiding moving obstacles with stochastic hybrid dynamics using PEARL: PrEference Appraisal Reinforcement Learning,” in *Proceedings of IEEE International Conference on Robotics and Automation*, Stockholm, Sweden, May 2016, pp. 484-490.

[31] W. Ding, S. Li, and H. Qian, “Hierarchical reinforcement learning framework towards multi-agent navigation,” *arXiv preprint arXiv:1807.05424*, 2018.

[32] P. Shinners, “Pygame,” 2011.

[33] V. R. Konda and J. N. Tsitsiklis, “Actor-critic algorithms,” in *Advances in Neural Information Processing Systems*, 2000, pp. 1008–1014.

[34] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *International Conference on Learning Representations*, vol. 5, 2015.

[35] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs,R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” in *IEEE International Conference on Robotics and Automation Workshop on open source software*, vol. 3, no. 3.2, Kobe, Japan, 2009, p. 5.

[36] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli, “The e-puck, a robot designed for education in engineering,” in *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, vol. 1, no. LIS-CONF-2009-004. IPCB: Instituto Polite cnico de Castelo Branco, 2009, pp. 59–65.

[37] Doojin Kim, “Operation of an intelligent space with heterogeneous sensors,” Ph. D. Dissertation, EECS Department, Seoul National University, 2014.

[38] M. Dekan, F. Duchon, *et al.*, “irobot create used in education,” *Journal of Mechanics Engineering and Automation*, vol. 3, no. 4, pp. 197-202, 2013.

[39] J. Van Den Berg, S. J. Guy, J. Snape, M. C. Lin, and D. Manocha, “Rvo2 library: Reciprocal collision avoidance for real-time multi-agent simulation,” http://gamma.cs.unc.edu/RVO2/, 2011

[40] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, "Optimal reciprocal collision avoidance for multiple non-holonomic robots," in *Distributed Autonomous Robotic Systems*, ser. Springer Tracts in Advanced Robotics, vol.83, 2013, pp. 203-216.

초 록

본 논문에서는 강화 학습 기반의 충돌 회피 방법을 제안한다. 충돌 회피란 로봇이 다른 로봇 또는 장애물과 충돌 없이 목표 지점에 도달하는 것을 목적으로 한다. 이 문제는 단일 로봇 충돌 회피와 다개체 로봇 충돌 회피, 이렇게 두 가지로 나눌 수 있다. 단일 로봇 충돌 회피 문제는 하나의 중심 로봇과 여러 개의 움직이는 장애물로 구성되어 있다. 중심 로봇은 랜덤하게 움직이는 장애물을 피해 목표 지점에 도달하는 것을 목적으로 한다. 다개체 로봇 충돌 회피 문제는 여러 대의 중심 로봇으로 구성되어 있다. 이 문제에도 역시 장애물을 포함시킬 수 있다. 중심 로봇들은 서로 충돌을 회피하면서 각자의 목표 지점에 도달하는 것을 목적으로 한다. 만약 환경에 예상치 못한 장애물이 등장하더라도, 로봇들은 그것들을 피해야 한다.

이 문제를 해결하기 위하여 본 논문에서는 충돌 회피를 위한 충돌 학습 방법 (CALC) 을 제안한다. CALC는 강화 학습 개념을 이용해 문제를 해결한다. 제안하는 방법은 학습 그리고 계획 이렇게 두 가지 환경으로 구성 된다. 학습 환경은 하나의 중심 로봇과 하나의 장애물 그리고 학습 영역으로 구성되어 있다. 학습 환경에서 중심 로봇은 장애물과 충돌하는 법을 학습하고 그에 대한 정책을 도출해 낸다. 즉, 중심 로봇이 장애물과 충돌하게 되면 그것은 양의 보상을 받는다. 그리고 만약 중심 로봇이 장애물과 충돌 하지 않고 학습 영역을 빠져나가면, 그것은 음의 보상을 받는다. 계획 환경은 여러 개의 장애물 또는 로봇들과 하나의 목표 지점으로 구성되어 있다.

학습 환경에서 학습한 정책을 통해 중심 로봇은 여러 대의 장애물 또는 로봇들과의 충돌을 피할 수 있다. 본 방법은 충돌을 학습 했기 때문에, 충돌을 회피하기 위해서는 도출된 정책을 뒤집어야 한다. 하지만, 목표 지점과는 일종의 '충돌'을 해야하기 때문에, 목표 지점에 대해서는 도출된 정책을 그대로 적용해야 한다. 이 두 가지 종류의 정책들을 융합하게 되면, 중심 로봇은 장애물 또는 로봇들과의 충돌을 회피하면서 동시에 목표 지점에 도달할 수 있다. 학습 환경에서 로봇은 홀로노믹 로봇을 가정한다. 학습된 정책이 홀로노믹 로봇을 기반으로 하더라도, 제안하는 방법은 홀로노믹 로봇과 비홀로노믹 로봇 모두에 적용이 가능하다.

CALC는 다음의 세 가지 문제에 적용할 수 있다. 1) 홀로노믹 단일 로봇의 충돌 회피. 2) 비홀로노믹 단일 로봇의 충돌 회피. 3) 비홀로노믹 다개체 로봇의 충돌 회피. 제안된 방법은 시뮬레이션과 실제 로봇 환경에서 실험 되었다. 시뮬레이션은 로봇 운영체제 (ROS) 기반의 시뮬레이터인 가제보와 게임 라이브러리의 한 종류인 PyGame을 사용하였다. 시뮬레이션에서는 홀로노믹과 비홀로노믹 로봇을 모두 사용하여 실험을 진행하였다. 실제 로봇 환경 실험에서는 비홀로노믹 로봇의 한 종류인 e-puck 로봇을 사용하였다. 또한, 시뮬레이션에서 학습된 정책은 실제 로봇 환경 실험에서 재학습 또는 별도의 수정과정 없이 바로 적용이 가능하였다. 이러한 실험들의 결과를 통해 제안된 방법은 Reciprocal Velocity Obstacle (RVO) 또는 Optimal Reciprocal Collision Avoidance (ORCA)와 같은 기존의 방법들과 비교하였을 때 향상된 성능을 보였다. 게다가, 학습의 효율성 또한 기존의 학습 기반의 방법들에 비해 높은 결과를 보였다.

주요어: 충돌 회피, 경로 계획, 모바일 로봇, 다개체 시스템, 강화 학습

학번: 2017-28406

감사의 글

졸업 논문을 마무리 하고 마지막 감사의 글만을 남겨두고 있을 때, 어떻게 하면 이 부분을 좀 더 인상 깊게, 진정성 있게 쓸 수 있을까 고민을 하다가 연구실에 비치되어 있는 선배들의 먼지 쌓인 학위 논문을 하나 하나 펼쳐보았습니다. 80년대 말 역사의 수레바퀴 한 중간을 걸어오신 선배님, 직장을 다니시다 늦은 나이에 학문의 길로 다시 돌아오신 선배님, 오랜 시간 몸 담아온 서울대학교를 떠나는 아쉬움을 담은 글 등 다양하고, 또 공감가는 발자취들이 논문 위의 먼지 만큼이나 소복히 쌓여 있었습니다. 이렇게 깊이 있고 유구한 역사를 지닌 연구실의 마지막 입학생이자, 마지막 석사로서 함께 할 수 있었던 것은 저에게는 너무나 큰 행운이자 영광이었습니다. 그러기에 이런 자랑스러운 연구실을 만들고 30년간 이끌어 주신 이범희 교수님께 가장 먼저 감사드립니다. 제가 어떤 선택을 하건 항상 격려해 주시고, 지원해 주시는 교수님의 모습에 용기를 얻고 많은 일을 도전할 수 있었습니다. 특히 마지막 학기를 교수님과 더욱 가깝게 보낼 수 있던 것은 석사 과정으로서는 흔치 않은 일이었다고 생각합니다. 자동화 연구소와 301동을 오가며 나눈 대화를 통해 교수님이 저뿐만 아니라 제자들을 얼마나 신경쓰시고, 걱정하시는지 직접 느낄 수 있었습니다.

그리고 부족한 석사학위 논문의 심사를 맡아 귀한 시간을 내주신 조동일 교수님께 감사드립니다. 2015년도에 교수님의 제어공학개론 수업을 통해 이쪽 분야에 처음 관심을 갖게 되었고, 그것이 지금 이 자리까지 이어지게 되었습니다. 또한 바쁘신

와중에도 심사를 맡아주신 심형보 교수님께도 감사드립니다. 지금까지도 학생들과 함께 후배 교수의 수업을 청강하시며 새로운 학문을 배우려는 교수님의 열정적인 모습을 보고 정말 많은 것을 느낄 수 있었습니다.

어떻게 보면 짧은 시간이지만 다른 누구보다도 짙게 지냈던 연구실 선배님들께도 감사 말씀 드립니다. 먼저 제가 입학함과 동시에 졸업을 해서 같이 지낸 시간은 길지 않지만 배려심이 느껴지는 재도형께 감사드립니다. 제가 이 석사학위 논문을 작성할 때에 가장 많이 참고한 논문이 형의 박사학위 논문입니다. 다행히 분야가 비슷해서 제게 많은 도움이 되었습니다. 가끔 연구실의 경조사가 있을때 종종 만났는데, 앞으로는 더욱 자주 만날 수 있으면 좋겠습니다. 회사와 학위를 동시에 진행하시느라 자주 마주치지 못했던 훈수형께도 감사드립니다. 끊임 없이 새로운 것에 관심을 가지시는 모습을 보고 많은 자극을 받았습니다. 조금 더 길게 함께했으면 좋았을텐데 아쉽습니다. 항상 유머를 잃지 않으시는 정현이형께도 감사드립니다. 형이 졸업하시고 썰렁한 농담을 하는 사람이 없어서 심심할 때가 많았습니다. 그리고 저의 진로에 대해 많이 조언해주시고, 좋은 자료도 제공해주셔서 정말 많은 도움이 되었습니다. 젠틀하고 멋지신 원석이형께도 감사드립니다. 처음에 키와 외모를 보고 깜짝 놀랐는데, 연구도 열심히 하시는 모습을 보고 너무 불공평한거 아닌가 하는 생각을 했습니다. 최근에 결혼하셨는데, 형수님과 앞으로 행복한 가정 꾸리시길 기원하겠습니다. 연구실의 어머니와 같았던 현기형께도 감사드립니다. 너무 세심하게 모든걸 잘 챙겨주시고, 봐주시는 모습에 도움을 정말 많이 받았습니다. 바쁘신 와중에도 제 논문들을 꼼꼼히 봐주셔서 제가 많이 발전할 수 있었습니다. 마지막 학기에 다행히 자동화연구소에서 함께 지낼 수 있어서 다행이라고 생각합니다. 저에게 선생님과 같았던 현우형께도 감사드립니다. 연구실 기간동안 저에게 많은 Insight를 주었고, 많은 것을 가르쳐준 점 정말 감사드립니다. 석사학위 논문도 형이 조언해준 것으로부터 출발했다고 해도 과언이 아닐정도로 많은 도움을 받았습니다. 지금 그

열정과 능력으로 앞으로 하고자 하는 것을 꼭 이루시길 바랍니다. 연구실에서 가장 오랜 시간 함께 보낸 한준이형께도 감사드립니다. 301동 시절부터 마지막 학기 자동화연구소까지, 연구실 사람들 중에 가장 긴 시간을 형과 보냈습니다. 마지막 학기에 Intelligent Space를 사용할 때 많은 도움을 주셔서 연구를 잘 마무리할 수 있었습니다. 교수님의 신뢰만큼 좋은 성과 내셔서 잘 졸업 하실거라 믿습니다. 항상 밝은 모습의 지윤누나에게도 감사드립니다. 사실 같은 나이면서 누나이기도 한지라 무언가 호칭이 어색해서 그런지 선뜻 친해지지 못한 것 같습니다. 돌이켜보면 그게 조금 아쉽긴 합니다. 올해 논문 성과를 잘 내셨는데, 이 추세대로 졸업까지 잘 마무리하시길 바랍니다. 그리고 부지런한 원영이에게도 감사드립니다. 매일 새벽에 학교에 와서 아침 수영을 하고 가장 먼저 연구실 문을 여는 부지런한 모습을 보고 정말 대단하다는 생각을 많이 했습니다. 학문과 건강 모두 열정적으로 관리하는 모습을 보고 많이 반성했습니다. 연구실의 마지막 문을 아름답게 장식해줄 것이라고 믿습니다. 연구실에서는 짧게 스쳐갔지만 학부 시절부터 저를 잘 챙겨주신 진원이형께도 감사드립니다. 형이 연구실 신입생일 당시, 셔틀버스에서 저에게 여기 연구실을 소개해주신 기억이 있습니다. 덕분에 제가 이쪽 분야에 흥미를 가질 수 있었고, 여기까지 오게 되었습니다. 비록 연구실에서는 함께있지 못했지만, 앞으로 자주 볼 수 있으면 좋겠습니다. 또 같은 석사과정이었던 현일이에게도 감사의 인사를 전합니다. 입학 초창기에 저에게 스스럼 없이 말을 걸어주어서 편하게 금방 연구실에 적응할 수 있었습니다. 앞으로 깊은 신앙의 힘으로 많은 것을 이뤄낼 것이라고 믿습니다. 마지막으로 준혁이에게도 고맙다는 말을 전합니다. 워낙 박학다식하여 제가 자극도 많이 받았고, 또 모르는게 있을 때 많은 도움을 받았습니다. 301동의 마지막 학기에 둘이서 같이 가깝게 지낼 수 있어서 다행이었다고 생각이 듭니다. 뛰어난 능력 만큼 앞으로 어느 진로를 가더라도 잘 할 것이라 믿습니다.

그리고 매일 같이 운동 하고, 술도 자주 마신 전기·정보공학부 R반 10학번 동기

들에게도 감사의 인사를 전합니다. 덕분에 학부 생활도, 대학원 생활도 지루할 틈이 없었습니다. 그리고 저의 대학 생활을 아름답게 채워준 김경래밴드 형, 누나들에게 감사드립니다. 음악 뿐만 아니라 다양한 분야에서 각자 최선을 다하고 또 성취하시는 모습을 보고 많이 배웠습니다. 더불어 저를 많이 응원해주고 무엇을 하든 박수쳐주던 분당 대진고등학교 친구들에게도 감사드립니다. 응원 덕분에 자신감을 갖고 많은 것을 도전 해볼 수 있었습니다.

마지막으로 누구 보다도 저를 제일 지지해 주시고 응원해 주신 부모님께 감사드립니다. 제가 큰 부담 없이 제 앞길을 자유롭게 선택할 수 있었던 이유는 항상 저의 선택을 믿고 지지해주시는 부모님이 계셨기 때문입니다. 아직까지 제대로 된 효도 한 번 못해드려서 항상 죄송한 마음 뿐입니다. 저의 앞길이 어디로 흘러갈지 완전히 정해지지 않았지만, 어디를 가든 열심히 노력해서 하루라도 빨리 효도하는 아들이 되도록 하겠습니다.

석사 졸업하는 2019년은 제가 서울대학교에 입학한지 10년째 되는 해입니다. 그만큼 ‘서울대학교’ 라는 글씨만 봐도 여러 개의 추억의 구슬이 기억의 실에 하나, 하나씩 연결됩니다. 수업 프로젝트를 위해 밤새 301동 전산실에서 치열하게 컴퓨터와 씨름했던적도 있었고, 샤 모양의 정문 밑을 지나는 것이 너무 설레고 좋아서 한동안 일부러 정문쪽으로 돌아서 학교를 올라갔던 시절도 있었습니다. 또 첫 오리엔테이션을 마치고 학관에서 토큰 같은 1700원 짜리 식권으로 학식을 처음 먹은 때도 생각납니다. 이렇게 추억의 구슬을 하나 하나씩 짚어 올라가다 보면 가장 첫 구슬에는 제가 프린트 한 서울대 합격증을 보시며 기뻐하시던 할머니의 모습이 있습니다. 지금 저의 모습을 보셨다면 그때보다 더 기뻐해 주셨을 것이라고 믿습니다.