Policy Integration for Person-Following Robot Training Using Deep Reinforcement Learning
(深層強化学習を用いた人物追従ロボットの学習のための方策の統合)

July, 2021

Doctor of Philosophy (Engineering)

Chandra Kusuma Dewa
(チャンドラ　クスマ　デワ)

Toyohashi University of Technology

別紙４－１（課程博士（英文））

Date of Submission（month day, year）: July , 2021

Department of Computer Science and Engineering	Student ID Number	D189301	Supervisors	Jun Miura, Shigeru Kuriyama
Applicant's name	Chandra Kusuma Dewa			

Abstract（Doctor）

Title of Thesis	Policy Integration for Person-Following Robot Training Using Deep Reinforcement Learning

Approx. 800 words

Recently, robots are widely used in numerous fields to support humans in many tasks. Thanks to the rapid growth of technology, the automated machines are becoming more powerful and more beneficial in their roles to help performing some tasks that humans even cannot do. Likewise, a particular type of partner robots which can interact closely to humans is also required to support their daily needs. In such cases, there are also demands for specialized robots which have the main ability to follow and attend specified humans in a safe close distance continuously. However, making them able to perform that kind of skills is not trivial since person-following cannot be considered as a simple task to be performed by robots.

In this thesis, we tried to train a mobile robot for performing the person-following task. Here, we see that the task is a complex task which can be broken down into several simpler sub tasks. When the robot is away from the target person, it should be able to perform the navigation task safely in order to make its position is close enough to him. Afterwards, the robot should also be able to perform the attending task properly once its position is close to the target person. In our study, we consider several previous studies which tried to make the robot able to accompany the target person at his left side or at his right side instead of following him from behind. In order to make the robot able to master the complex person-following task appropriately, we utilize deep reinforcement learning (DRL) approach for both obtaining the optimal policy for each sub task and integrating all those optimal policies into one strong optimal person-following meta-policy.

To obtain the optimal navigation policy, we employ the soft actor-critic (SAC) learning algorithm for making the robot able to approach the target person well. Moreover, we propose a specific framework which is intended to train a mobile robot to navigate quickly but safely. The framework utilizes a novel state transition checking method to ensure that the training environment provided for the robot always follows the Markov decision process properly. Furthermore, it also employs a novel velocity increment scheduling technique during the training process. The technique follows a curriculum learning strategy by setting a small value of velocity for the robot at the beginning of the training episode. As the number of episodes increases, the robot's velocity is increased gradually so that the robot can gradually learn the complex task of fast but safe navigation in the

training environment form the easiest level, such as the one with the slow movement, to the most difficult level, such as the one with the fast movement.

To obtain the optimal attending policies, we also use the SAC algorithm to make the robot able to attend the target person when its position is close to the target person. During the training process, we propose the U-shaped reward function which can guide the robot to attend the target person at his left side or at his right side. Moreover, we propose a novel weight-scheduled action smoothing technique so that the robot can generate smooth and safe trajectories for the attending task. To make the robot can better portray the surroundings we also propose a novel policy network architecture which employs one dimensional convolutional neural network to extract features from laser scans automatically.

Finally, to integrate all the optimal navigation and attending policies, we also propose a framework which employs the double deep Q-network which can make the robot learn to choose the most appropriate policy given the current state of the person-following environment. Inside our framework, we introduce the action generator module which can adjust the state of the person-following environment for each sub policy appropriately. Furthermore, the module is also able to smooth the actions generated by the robot using the action smoothing strategy to prevent the robot hitting the target person when it is close to him and when the robot's actions are generated from changing policies.

From all experiments that we conduct in our study, we can conclude that the proposed navigation training framework is able to make the robot navigate approaching the target person faster with lower collision rate compared to other DRL-based navigation baseline frameworks. We confirm that the U-shaped reward function and the weight-scheduled action smoothing that we propose can make the robot attend the target person both at his left and right side with smooth and safe trajectories. We also confirm that our proposed framework can integrate all the navigation and attending policies for obtaining the meta-policy for the person-following task. For future work, we consider using dynamic environments for the training of the navigation and the attending tasks so that more robust policies can be obtained. We also plan to propose another method so that the robot can switch its attending position easily when it is close to the target person. Furthermore, we also will use computer vision techniques for detecting and tracking the target person so that the policies can be deployed for real robots.

Contents

Chapter 1 Introduction 1

1.1 Research Background 1

1.2 Person-Following Task for Robots 2

1.3 Research Objective and Main Contribution 4

1.4 Thesis Organization 5

Chapter 2 Related Work 6

2.1 Reinforcement Learning 6

2.2 Curriculum Learning 7

2.3 DRL for Person-Following Robots 7

2.4 DRL for Environments with Multiple Tasks 8

Chapter 3 Navigation Task Training with DRL 10

3.1 Background 10

3.2 Proposed DRL-Based Navigation Training Framework 12
3.2.1 DRL-Based Navigation Environment 13
3.2.1.1 Navigation State Space 13
3.2.1.2 Navigation Action Space 14
3.2.1.3 Navigation Reward Function 15
3.2.2 Soft Actor-Critic 16
3.2.3 State Transition Checking 17
3.2.4 Velocity Increment Scheduling 19

3.3 Experiment Results and Discussions 21
3.3.1 Experiment Setup 21
3.3.1.1 Implementation 21
3.3.1.2 Baselines 23
3.3.2 Results and Analysis 24

3.4 Conclusion on Navigation Task Training 27

Chapter 4 Attending Task Training with DRL 29

4.1 Background 29

4.2 Proposed DRL-Based Attending Training Framework 30
4.2.1 Attending Task Training System Architecture 30
4.2.1.1 Gazebo Data Helper 30
4.2.1.2 Reward Calculator 30
4.2.1.3 Observation Generator 32
4.2.1.4 Action Performer 32
4.2.1.5 Soft Actor-Critic Agent 32
4.2.2 Weight-Scheduled Action Smoothing 32

4.3 Experiment Results and Discussion 33

4.4 Conclusion on Attending Task Training 36

Chapter 5 Multiple Policies Integration with DRL 37

5.1 Background 37

5.2 Proposed DRL-Based Polices Integration Framework 38
5.2.1 General Framework for Policies Integration with DQN 38
5.2.2 Person-Following Task Training with Policies Integration 41
5.2.2.1 Person-Following Robot Environment 41
5.2.2.2 Policies Integration in the Person-Following Task 43

5.3 Experiment Results and Discussion 44
5.3.1 Experiment Setup 44
5.3.2 Results and Discussion 47

5.4 Conclusion on Multiple Policies Integration 49

Chapter 6 Policies Improvement Trials 52

6.1 Utilization of Symmetric Nature in the Attending Task 52
6.1.1 Proposed Method to Invert the Attending Policies 53
6.1.2 Experiments on Inverting the Attending Policies 54

6.2 Policies Integration in a Dynamic Environment 55
6.2.1 Dynamic Environment Implementation with APF 55

6.2.2 Experiment Setup of the Policies Integration 57
6.2.3 Results and Discussion of the Policies Integration 58

6.3 Conclusion on Policies Improvement Trials 59

Chapter 7 Conclusion and Future Works 60

7.1 Thesis Main Conclusion 60

7.2 Future Works 60
7.2.1 Plan to Improve All Policies 60
7.2.2 Plan to Deploy The Learned Policies to Real Robots 61

References 62

Abbreviation

AI:	artificial intelligence
ANN:	artificial neural networks
APF:	artificial potential field
API:	application programming interface
CNN:	convolutional neural networks
DDPG:	deep deterministic policy gradient
DL:	deep learning
DRL:	deep reinforcement learning
DQN:	deep Q-networks
MDP:	Markov decision process
ML:	machine learning
PER:	prioritize experience replay
RL:	reinforcement learning
ROS:	robot operating system
SAC:	soft actor-critic
SL:	supervised learning
TD3:	twin delayed deep deterministic
VIS:	velocity increment scheduling

Chapter 1

Introduction

1.1 Research Background

With the rapid growth in information technology and artificial intelligence (AI), there has been a lot of research on a specific topic in the field of robotics which has the main focus on developing particular robots that have a main duty to assist humans while performing their daily tasks. In order to accomplish the tasks appropriately, the robots must be able to interact socially with their surrounding environments [1]. More importantly, they are also required to be able to interact safely and properly when delivering their services around humans [2]. Despite several challenges on the development of service robots [3], there are also increasing demands toward particular robots which are able to stay around and help a specified target person by always following and attending him continuously.

Accordingly, typical person-following robots are needed in such scenarios when a specific person and a robot are required to finish some common tasks altogether by letting the robot to follow the human [4]. In more details, this kind of special robots are equipped with cameras so that they can properly recognize the correct target person to be followed and are usually assigned to help carrying some loads [5, 6]. Moreover, they are must also be equipped with other specific sensors so that obstacle avoidance can also be performed well. Figure 1.1 depicts two examples of person-following robot while helping target humans. Figure 1.1 (a) depicts a person-following robot is carrying some personal belongings inside for a young lady. Figure 1.1 (b) shows a robot is helping an elderly bring several shopping goods while following him behind.

(a) Gita cargo robot [7]

(b) CompaRob assistance robot [8]

Figure 1.1: Examples of person-following robots help carrying some stuffs

However, developing person-following robots is not just a trivial task since there are still many challenges that have to be faced with regard to the special characteristics of the task that have to be performed by the robots [4,9]. Therefore, it can be broken down into other several simpler tasks that have to be completed by the robot one by one [10, 11]. Hence, appropriate methods are required in order to properly develop the robots so that all of the sub tasks inside the substantial person-following task can be accommodated well and properly.

One of possible options that should be considered for developing person-following robots is robot learning, which is the intersection research field between robotics and machine learning (ML) [12]. Instead of manually program the robots in details to be able to perform several tasks, the approach lets robots to automatically learn to perform those tasks by implementing machine learning techniques [13]. In most cases, artificial neural networks (ANN) are implemented inside the robots so that they can infer some meaningful outputs continuously given streamed data obtained from various attached sensors. In order to make the robots can infer the outputs correctly, training procedures which follow some ML approaches for the ANN are required [14, 15].

Reinforcement learning (RL) may become one of the ML approaches that can be chosen when we want to solve tasks to be performed by robots [16]. Recently, this approach has been gaining some interest and has been widely implemented in various fields, including robotics since robots are required to perform end-to-end mapping from sensors data to appropriate actions [17]. In contrast to supervised learning (SL) approach which requires enormous amount of labeled data to perform the update the ANN, RL uses a different mechanism which employs environments that provide particular rewards related to some specific tasks for the RL agents inside the robots while interactions are performed during the training process. In order to update the ANN inside the robots, all experiences during the training process are used to form optimal polices which capable of mapping states of the environments to suitable actions which can maximize expected future rewards [18].

Furthermore, with the advancements and potentials of deep learning (DL) [19], the concept of deep reinforcement learning (DRL) then emerges which extends traditional RL framework with DL methods. DRL is able to make the agent to generate more appropriate actions since it enables better feature extractions of the environment by leveraging deeper neural network architectures with wider scope and variety of state representations [20, 21].

1.2 Person-Following Task for Robots

In this thesis, we focus on a specific task to be performed by a robot which is the person-following task. The task involves an environment $\mathcal{E}$ of a specific area in which a robot $\mathcal{R}$ becomes a follower of a specific target person $\mathcal{P}$ as the leader. In order to make the robot able to perform the task successfully, it should be able to obtain data of its surroundings $\mathcal{D}$ which are gained from sensors attached to the robot. Also, it

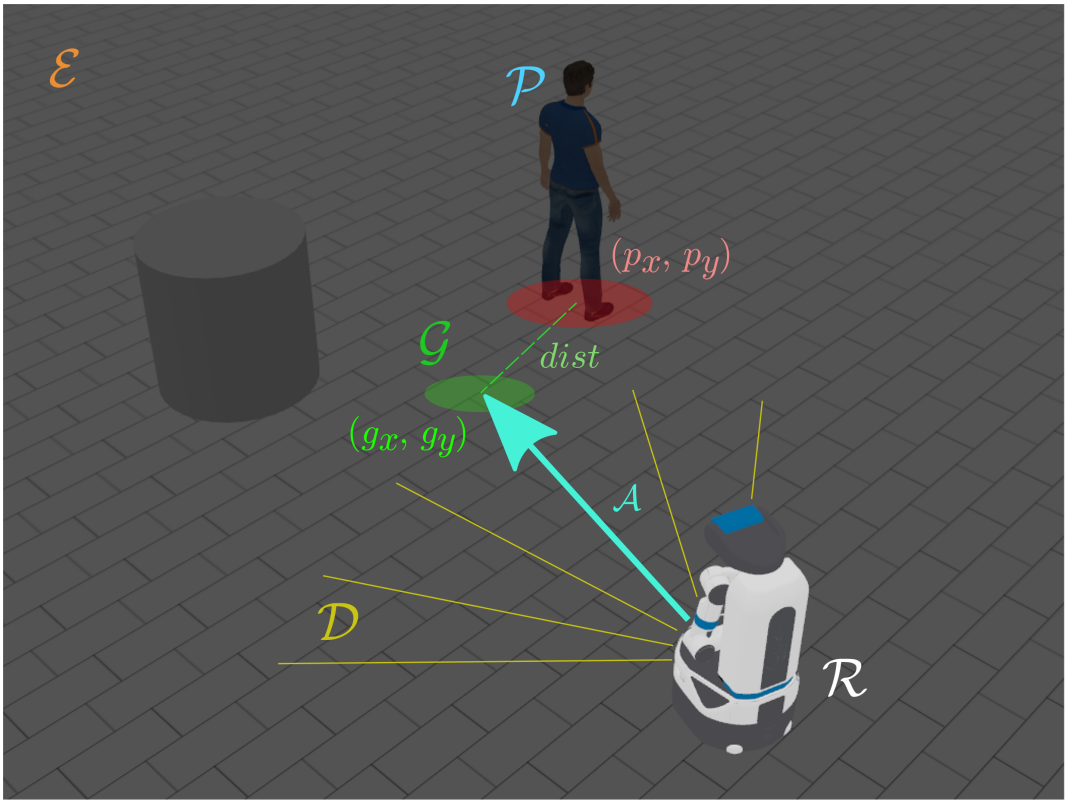


Figure 1.2: The illustration of a Gazebo-based simulated environment containing all required variables for the person-following task to be performed by the robot.

should be able to identify the correct person to be followed and then able to predict his coordinate (p_x, p_y) accurately. Subsequently, the robot should also be able to calculate the appropriate goal position $\mathcal{G}$ that must be close enough to the target person while still taking into account the safe close distance *dist* between the robot and the target person. The detailed illustration of an environment for the person-following task is depicted in Fig. 1.2

After obtaining all data related to the person-following task as the inputs, the robot must perform the path planning for generating the appropriate actions $\mathcal{A}$ to achieve the goal coordinate (g_x, g_y) and avoid surrounding obstacles safely. Suppose that all data related to the person-following task can be considered as the states of the person-following environment $\mathcal{S}$, in this thesis, we focus on obtaining a specific person-following policy π which able to map the given states of the environment to specific actions to be performed by the robot as follows

$$\pi : \mathcal{S} \rightarrow \mathcal{A}. \qquad \text{(Eq. 1.1)}$$

Furthermore, the policy can be represented as a neural networks function which can be deployed inside the robot. Therefore, a learning procedure for the neural networks can be performed to train the robot for mastering the person-following task.

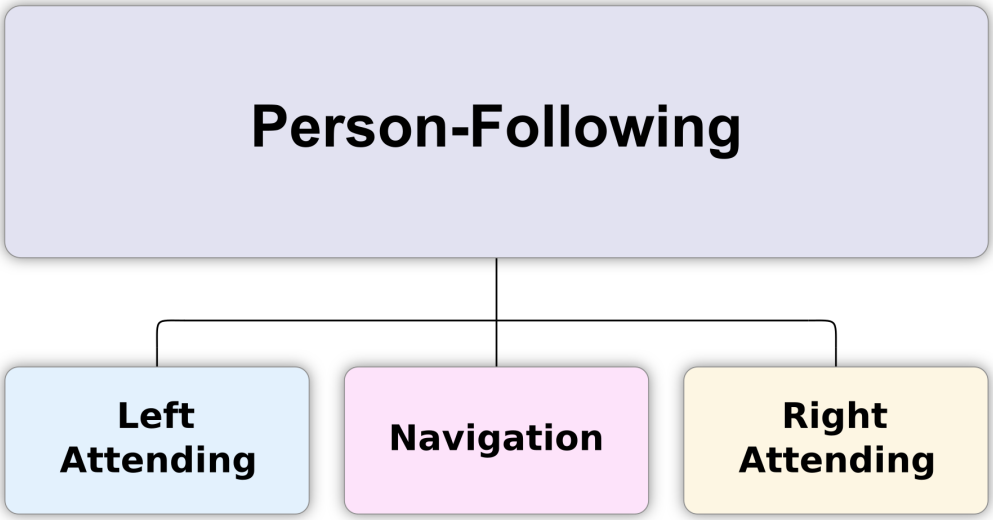


Figure 1.3: The illustration of person-following task to be performed by robots in this dissertation. We see that person-following is a complex task which can be hierarchically decomposed into left attending task, navigation task, and right attending task.

1.3 Research Objective and Main Contribution

The main objective of this research is to perform training procedures for the person-following robots development using DRL approach. Along with the potentials in training and acquiring optimal policies for robots using deep reinforcement learning (DRL) [17, 22], applying the approach to develop a person-following robot can be one of great options to be considered. Using a combination of DL and RL, several studies such as [23] and [24] have shown that DRL can be successfully implemented to train agents for obtaining the policy for a robot to generate appropriate actions to follow a specified target person. However, the person-following task to be performed by the robot in [24] is not seen as an integrated substantial task so that another important sub task, namely the obstacle avoidance is neglected. On the other hand, since the policy in [23] is only focused on the attending task, the robot often fails to minimize collisions while it is away from the target person. Thus, to make the robot able to perform all sub tasks in the person-following problem well, the task is seen as a complex task to be learned by robots using DRL approach. In this dissertation, the person-following task is defined as a task that can be decomposed hierarchically from several sub tasks as shown in Figure. 1.3.

One of abilities a person-following robot must have is the navigation skill. Suppose that the robot is in the position of away from the target person, it has to be able to perform a motion planning for finding safe paths in order to navigate approaching him while also avoiding surrounding obstacles. Once it's position is near to the target person, the robot subsequently has to find the most appropriate position towards

him to perform the attending task appropriately. Furthermore, Instead of making the robot able to continuously follow behind the target person, we consider several previous studies [23, 25] which tried to make the robot able to accompany him at his left side or at his right side. Therefore, the person-following task can be described hierarchically as a task which includes left attending task, navigation task, and right attending task. Hence, multiple policies which are correlated to each task in the person-following robots development are required to be obtained during the training process.

The main contribution in this dissertation is the specific proposed novel method which is intended for performing the person-following task training for a mobile robot with DRL using bottom-up approach. In the proposed method, an RL agent is trained for each sub task in the particular person-following task to obtain a set of sub optimal policies. Subsequently, all of those sub optimal polices are then integrated altogether to form an optimal meta-policy, which is the person-following policy. By following our proposed method, all of sub tasks in the person-following task can be accommodated and a strong optimal meta-policy can be obtained during the training process.

1.4 Thesis Organization

In accordance to the research objective in this dissertation which decomposes the particular person-following task into several sub tasks, we organize the thesis as follows: We first discuss some previous works which are related to the main method which is proposed in this dissertation in Chapter 2. Subsequently, the discussion regarding how we perform the training process for the navigation task is presented in Chapter 3. In this chapter, we present our proposed framework which is intended to train a mobile robot using DRL so that it is able to navigate safely and quickly. Next, we present our proposed method to perform the attending task training in Chapter 4. Furthermore, how we integrate the optimal navigation and attending policies is presented in Chapter 5. Afterwards, our trials to improve the polices in the person-following task is presented in Chapter 6. Finally, Chapter 7 concludes the thesis and discusses the future work.

Chapter 2

Related Work

2.1 Reinforcement Learning

Reinforcement learning (RL) is one of the approaches in the machine learning techniques which aims to build models from training data. In contrast to other approaches, RL approach leverages the interaction between an agent and an environment to collect the data to train the model. However, compared with the supervised learning approach, RL does not use label for the data. Instead, it uses rewards for all actions generated by the agent from the states of the environment. In order to make the agent able to learn an appropriate policy, RL algorithms aim to make it able to maximize the training reward. Furthermore, RL can be combined with deep learning (DL) to form better agents called deep reinforcement learning (DRL).

DRL is part of machine learning techniques which is the intersection field between DL and RL [26]. By leveraging DL in RL, it is possible to train an agent to have better understanding of the environment where it interacts. By using DRL, the training process for a specific policy which is represented as a DL model can be done directly without having to previously collect enormous labeled datasets. Instead, the policy is updated through rewards obtained from the interaction of an RL agent with its environment during the training process. Mnih et al. [27] demonstrated that DRL can be used to train an agent to have a human-level control skill in playing Atari games. Moreover, Silver et al. [28] showed that the trained DRL agent even can beat human skills in playing Go. In addition, several studies such as reported in [29–31] and [23] had demonstrated that DRL can also be successfully applied in robotics.

In order to train a DRL agent, an environment which follows Markov Decision Process (MDP) is needed. As described in Bellman [32], the environment should be able to be defined in tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}$ is state space, $\mathcal{A}$ is action space, $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is state transition probability function, $r : \mathcal{S} \times \mathcal{A} \rightarrow [r_{min}, r_{max}]$ is bounded reward function, and $\gamma \in [0, 1)$ is the discount rate of reward. In the training process, the DRL agent will interact with the environment which always provides immediate reward r_t whenever the agent performs an action $a_t \in \mathcal{A}$ based on current state $s_t \in \mathcal{S}$. The policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ is then learned by the agent by maximizing the sum of expected future reward using a specified DRL algorithm.

In our study, we focus on the details of the implementation of the DRL environment. As small changes in the environment may influence the resulting navigation policy [33], we need a mechanism which ensures that the environment will always follow MDP. In the case of navigation task, our proposed state transition checking method verifies that the environment follows MDP by inspecting terminal state conditions which may occur while actions are performed in the environment. Moreover, our method examines whether each action has been successfully executed in the environment by setting the timestep size in the environment dynamically.

2.2 Curriculum Learning

Bengio et al. [34] introduced curriculum learning as a training strategy in the context of machine learning which presents organized examples based on meaningful order, gradually from simple to complex. As reported in Hacohen and Weinshall [35] and also Narvekar [36], it is possible to make the training process of deep neural networks and RL becomes faster with improved final performance using this strategy. However, generating appropriate examples following a specified curriculum is considered to be an open problem to be solved [37].

In the context of DRL, curriculum learning strategy is used to generate suitable experiences to be sampled from the replay buffer in the update process of neural networks. Schaul et al. [38] introduces Prioritize Experience Replay (PER) which priorities important experiences to be sampled from the replay buffer to generate the examples following curriculum learning scheme. Subsequently, Ren et al. [39] combined self-paced prioritize function and coverage penalty function which could select samples with appropriate difficulty with penalty when samples are replayed frequently. Another studies, such as [40] and [41] use curriculum learning to schedule ordered list of task and maps to be solved by the RL agent.

We consider that generating actions for a mobile robot to navigate fast yet safe is a complex task. From the perspective of curriculum learning, we see that the task can be gradually learned from simple tasks. Furthermore, we relate the difficulty of the navigation task to the robot's velocity range settings. We first set a small range in the robot velocity setting at the initial training stage. Subsequently, we follow the curriculum learning strategy by expanding the velocity range during the training process gradually.

2.3 DRL for Person-Following Robots

With the advancements which are offered by DRL to train agents capable of producing policies which can map states of a particular environment to suitable actions, there have been many studies that employ the method in the field of robotics [17, 42]. Accordingly, previous studies such as Dewantara and Miura [31], Liu et al. [43], and Guldenring et al. [44] showed that the method can also be successfully applied to train robots for generating safe and appropriate actions while they are interacting with humans. Moreover, several studies have tried to apply DRL for obtaining specific policies that able to generate actions to be performed by robots which are intended for the person-following task as well.

Pang et al. [24] applied a hybrid method of supervised learning (SL) and DRL to train an agent with deep Q-network (DQN) for developing a robot which able to continuously follow a specified target person from behind. In order to generate the robot's actions, the study uses images obtained from an RGB camera attached to the robot as states which are then fed to the neural networks inside the agent. Even

though they demonstrate that the transfer learning from SL to DRL can make the training for DRL become much faster, important modules in the person-following task namely obstacle avoidance and distance measurement towards the target person are excluded in the study.

On the other hand, a prior study conducted by Kohari et al. [23] considered that the obstacle avoidance should also be included in the DRL training process for the person-following task. In the study, a simulated dynamic environment is prepared for the robot so that it is able to avoid obstacles while performing the attending task. When the agent generates actions, states of the target person are also observed and considered as part of the environment's states as the input for the policy. Subsequently, the study is then extended by Dewa and Miura [45] who introduced a reward function which is designed based on the pose of the robot with respect to the pose of the target person. Nevertheless, the learned policies still give high value of collision rate in the validation since the main concern of both studies is mostly focused on how the robot performs the attending task towards the target person. Therefore, we concern that person-following is a complex task and come out with the idea to provide a multi-task environment for the RL agent during the training process.

2.4 DRL for Environments with Multiple Tasks

In order to make DRL can be successfully applied to train agents inside complex environments, several studies have considered multi-task learning in the training process [46]. Previously, Yang et al. [47] introduced a neural networks architecture inside a DRL agent capable of learning multiple tasks simultaneously. The study extends the standardized deep deterministic policy gradient (DDPG) learning algorithm [48] by applying multiple actors to accommodate various tasks. In addition, it also modifies the training environment so that the agent is given a vector of rewards which corresponds to each task whenever an action is performed.

Accordingly, other studies apply transfer learning strategy to deal with multi-task environments. Teh et al. [49] applied transfer learning strategy to distill a base policy among all tasks in the environment. In the study, several agents are trained simultaneously for dissimilar tasks inside different environments and share the same base policy. Along with the progress in the training, the base policy and all task-specific policies are optimized and regularized so that the learning process becomes more robust. Similarly, Espeholt et al. [50] proposed a framework for performing distributed DRL mechanism over several dissimilar environments using multiple actors that interact with the same critic which is called the learner. Subsequently, the study is then extended by Hessel et al. [51] who introduced a method to stabilize the learning by normalizing the impact for all tasks so that there is no single task which becomes more salient compared to other tasks.

Apart from the aforementioned methods, various studies have considered that a single complex task can be broken down into other simple sub tasks resulting a new approach called hierarchical RL [52]. In this type of approach, an RL agent

learns a meta-policy for a substantial task and sub policies which are correlated to each sub task simultaneously [53]. In order to make the learning mechanism can be realized, Sutton et al. [54] previously proposed a specific framework that is intended for hierarchical RL called Options which then influences other related studies such as reported in [55–57]. Accordingly, the framework is also extended by Frans et al. [58] who introduced a shared state space over multi-task setting in the learning process. The study then becomes the most closely related work to our proposed method.

Even though in our proposed method the agent also shares the same states from the complex environment with other sub policies for each sub task, we do not perform updates for those sub policies since we have already obtained a set of optimal sub policies from the previous training. In our proposed method, we consider several other studies which employ ensemble learning along with the RL mechanism to integrate all sub policies in our environment. Previously, Marivate and Littman [59] introduced a method which employs a weighted linear combination over Q-values from several agents while interacting with different environments. Similarly, Carta et al. [60] proposed an ensemble methodology with agreement threshold technique for several RL agents which are trained at different epochs. Other than that, Elliot et al. [61] proposed a crowd ensemble learning technique with weight sharing over a modified DQN agent for one specific task to speed up the learning process. Liu et al. [62] introduced a method which uses an RL agent to integrate three types of deep neural network models that have been trained beforehand. The integration process in the study is performed by making the agent learn to set the appropriate weight for each model in order to produce the final output.

Most of the ensemble methods described in the aforementioned studies apply weighting for each sub task since the main objective is to integrate several similar classifiers. However, the condition is quite different with the person-following task because all sub tasks have dissimilar goals. Since the the task can be decomposed into several different sub tasks, we let an RL agent learn to choose the most appropriate optimal policy which corresponds to the most suitable sub task for generating the most applicable action given the current state of the complex environment.

Chapter 3

Navigation Task Training with DRL

To train a mobile robot to navigate using end-to-end approach which maps sensors data into actions, we can use deep reinforcement learning (DRL) method by providing training environments with proper reward functions. Although some studies have shown the success of DRL in navigation task for mobile robots, the method needs appropriate hyperparameter settings such as the environment's timestep size and the robot's velocity range to produce a good navigation policy. The previous existing DRL framework has proposed the use of odometry sensor to generate dynamic timestep size in the environment to solve the mismatch problem between the timestep size and the robot's velocity. However, the framework lacks a procedure for checking terminal conditions which may occur during action executions resulting inconsistency in the environment and poor navigation policies. In the case of navigation task, terminal conditions may happen when the robot achieves the navigation goal position or collides with obstacles while performing an action in one timestep. To cope with this problem, we propose a state transition checking method in the DRL environment which is specific for navigation task that leverages odometry and laser sensor to ensure that the environment follows Markov Decision Process with dynamic timestep size. We also introduce a velocity increment scheduling to stabilize the mobile robot during training. Our experiment results show that state transition checking along with the velocity increment scheduling are able to make the robot navigate faster with higher success rate compared to other existing DRL frameworks.

3.1 Background

In navigation tasks, DRL is proven to be successfully implemented to train agents to perform path planning [63], [64]. It also offers some advantages in training a mobile robot to avoid obstacles and also to achieve navigation destination points. Provided training environments which follow MDP, we only need suitable state representation and also appropriate rewards as the input for the DRL agent to generate the velocity of the robot to navigate in the training process [65]. We also do not have to collect enormous labeled data to train navigation policies, as in the training procedure in supervised learning since the robot will learn from interactions with the training environments [66]. As mentioned in several studies [67–69], it is also possible to use the learned policies for the mobile robot to navigate in new unknown complex environments without having to retrain the DRL agent.

However, challenges in DRL method do exist [70], [22]. In order to get an optimal policy to be applied for mobile robots to navigate using the method, we need to appropriately implement the training environment and also carefully set the proper hyperparameters [57]. One of the hyperparameters that should be correctly set in the DRL environment is the timestep size which indicates the duration of a single action

(can be selected and performed) in the environment [71]. Wrongly choosen timestep size will cause poor performance since the Q-function may collapse to V-function so that the DRL agent will not get enough information to select actions resulting higher reward [72]. In the case of navigation task training with DRL, the second problem which may occur is how we can train the DRL agent to generate velocities which can make a robot to navigate quickly but still considered as safe since faster navigation may cause higher collision rate.

A new way to solve the timestep size problem in DRL environment has been introduced by openai_ros [*1] which sets dynamic timestep size for each action to be performed in the DRL environment. The openai_ros package sets the duration of each action by comparing the output velocity from the DRL agent to the current robot's velocity from odometry sensor so that the timestep size will always match with the robot's velocity. Nevertheless, the package always assumes that all actions will always be successfully executed in the environment and does not put much attention on terminal conditions which may occur in the middle of performing an action. In the case of robot navigation, terminal conditions may happen when the robot achieves the navigation goal position or collides with obstacles during an action execution. When the terminal conditions occur, the environment should move to terminal state and a training episode should be ended. Otherwise, the condition may lead to inconsistency in the environment which can make the training process fail.

To solve the second problem, several studies such as [73] and [74] include the DRL agent's output action, which is the velocity, as the magnitude for the reward function to make it able to generate high velocity value for autonomous outdoor vehicles. In the context of DRL based robot navigation task, although the agent is forced to generate higher velocity value as in [75] and [76], only small values are set for the robot's maximum velocities which prevent it from navigating quickly. Even though [73] demonstrates that the DRL agent can generate smaller velocity towards obstacles, the aforementioned studies mostly focus on how to make the agent generate faster velocity rather than consider how the robot can navigate safely in the environment using the learned policy.

In the navigation task, the goal of our study is to train a mobile robot to navigate fast yet safe in indoor environments using DRL method. In our study, we focus on DRL navigation for a mobile robot which is equipped with laser sensor to detect obstacles and odometry sensor to estimate the robot's velocity. In order to accomplish our objective, we propose a novel framework for DRL based navigation which extends the ordinary RL mechanism and previously existing frameworks by applying state transition checking in the DRL environment to ensure that it will always follow MDP. Moreover, we also apply velocity increment scheduling which is based on curriculum learning for the mobile robot during the training process of the DRL agent as depicted in Fig. 3.1 [77].

In our proposed state transition checking method, we extend openai_ros package so that it becomes aware of terminal conditions which may occur in the middle of

[*1] http://wiki.ros.org/openai_ros

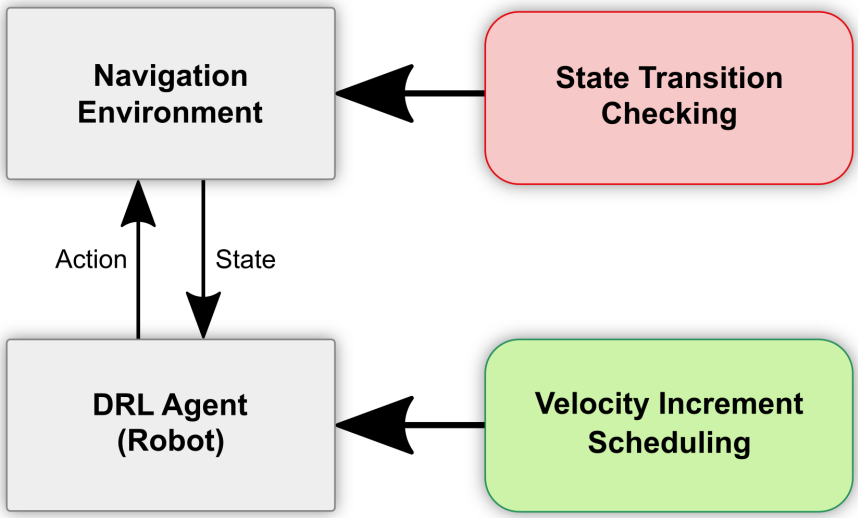


Figure 3.1: The block diagram of our proposed framework in the navigation training. We extend the ordinary RL mechanism and previously existing frameworks by applying state transition checking in the navigation environment. Moreover, we also implement velocity increment scheduling for the DRL agent during the training process.

performing an action in the environment. The method continuously checks whether the current state should move to terminal state whenever the DRL agent meets terminal conditions which may happen in the navigation task, such as arriving at the navigation goal position, colliding with obstacles, or reaching the maximum steps. The current state will then be immediately changed to terminal state when one of those conditions occurs. By doing so, we can ensure that the DRL environment will always follow MDP. We also do not have to manually set the timestep size in the environment since the duration of each action to be performed will be dynamically set not only by using odometry but also by using laser sensor.

In contrast to other methods, our proposed framework schedules the range of the robot's velocity using a velocity increment scheduling mechanism to make a DRL agent learn to navigate fast. The scheduler will set the range of the robot's velocity with a small value at the beginning of the training. It then will expand the range gradually along with the increasing number of the training episode. By using the mechanism, the DRL agent is given the opportunity to gently learn the skill to navigate fast but safe from the easiest level to the hardest level.

3.2 Proposed DRL-Based Navigation Training Framework

The details of our proposed framework are presented in this section. We first describe the properties of the DRL environment which we use to train the agent for the navigation task. Subsequently, we describe the learning algorithm which is used to train the agent. Afterwards, we describe our proposed state transition checking in

the environment and our proposed velocity increment scheduling for the mobile robot in the training process.

3.2.1 DRL-Based Navigation Environment

We follow the definition of MDP to describe our environment for training a mobile robot to navigate in an indoor setting as outlined in the following subsections.

3.2.1.1 Navigation State Space

We need to carefully design the state space in the DRL environment since it will greatly affect the generated actions by the agent. To make the DRL agent able to generate appropriate actions to navigate, it needs information related to the goal position and the surrounding obstacles from the laser sensor attached to the mobile robot. Additionally, it also needs information related to the current robot's state which is represented as the previous velocity generated by the agent which has been successfully executed in the environment. Therefore, the state space in this study consists of laser scan data from the laser sensor, the goal coordinate position, and also the robot's previous velocity [78].

Fig. 3.2 depicts variables that we use to formulate the state space. Suppose that the robot's current state is at the linear and angular velocity of v_{t-1} and ω_{t-1}, we represent the navigation goal position $\mathcal{P}_{goal}$ as the relative coordinate of p_x and p_y from the robot's current position. In contrast to [79], we do not directly use all of the laser scan values to represent the surrounding obstacles around the robot. We follow [80] and [81] who divide the laser scans into sectors and then only include values which represent all sectors in the state space. Therefore, the state space $s \in \mathcal{S}$ is defined as,

$$\mathcal{S} = \{\mathcal{L}_{sctr}, p_x, p_y, v_{t-1}, \omega_{t-1}\}, \tag{Eq. 3.1}$$

where $\mathcal{L}_{sctr} = \{l_{sctr,i} \mid i \in 1 \dots M\}$ denotes all of the sector's laser scans and M denotes the number of sector.

To compute $\mathcal{L}_{sctr}$, as also depicted in Fig. 3.2, we first adjust N laser scans obtained from the laser sensor $\mathcal{L} = \{l_j \mid j \in 1 \dots N\}$ by limiting the values to a radius constraint of l_{max} so that the RL agent can more easily portray the state space in the environment. The function to compute the adjusted laser scan values l_{adj} is given by

$$l_{adj,k} = \begin{cases} l_j, & l_j \leq l_{max} \\ l_{max}, & l_j > l_{max}, \end{cases} \tag{Eq. 3.2}$$

where $k \in \{1, 2, \dots, N\}$. Subsequently, we compute the average values of the adjusted laser scans for all sectors using the following formula:

$$l_{sctr,i} = \frac{\sum_{z=i(N/M)-(N/M)+1}^{i(N/M)} l_{adj,z}}{N/M}. \tag{Eq. 3.3}$$

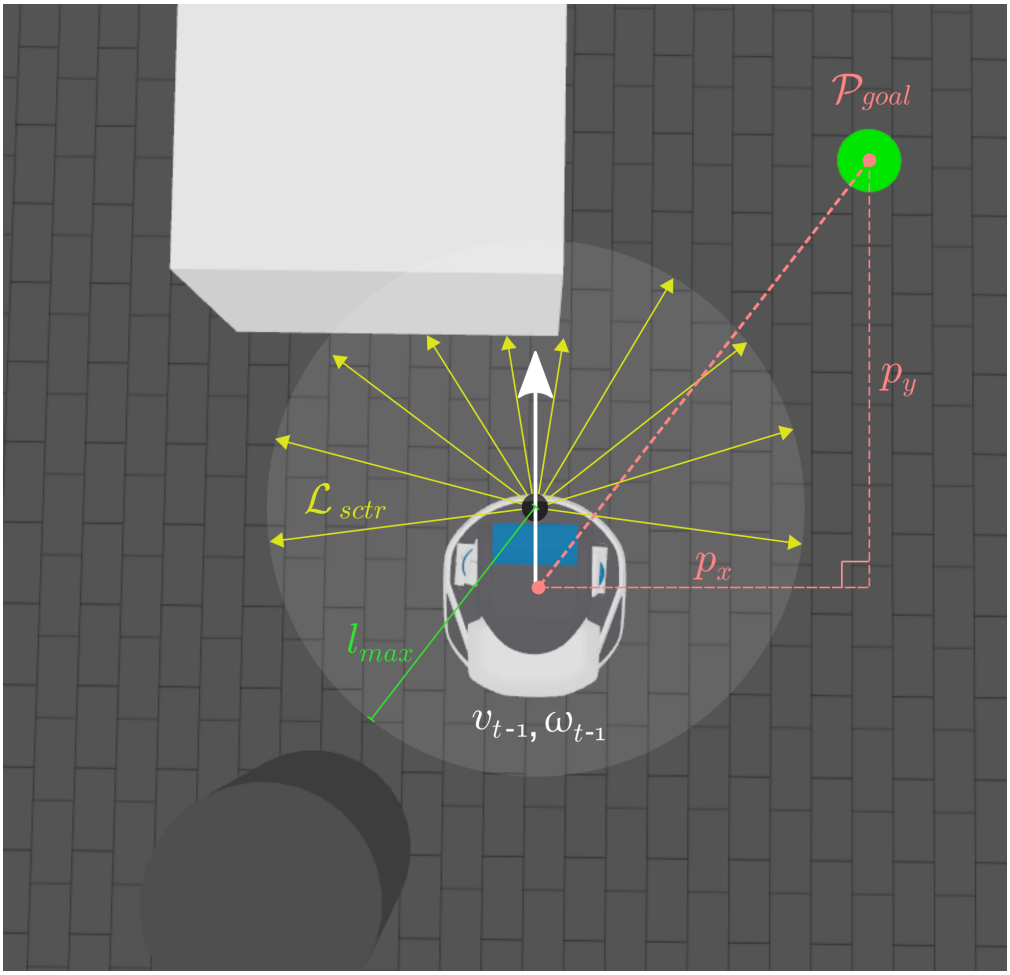


Figure 3.2: An illustration of a robot's position towards obstacles and a goal in our study. The state space in our study is composed of sector's laser scan, navigation goal coordinate, and the robot's previous velocity. We limit the laser scan values to a specified radius constraint so that the agent can more easily portray the state space.

We also define a terminal state which ends an episode in the training or validation process. Current state will move to terminal state s_T whenever the robot successfully arrives at a defined goal position, collides with barriers, or reaches the allowed maximum steps in each episode.

3.2.1.2 Navigation Action Space

We use continuous action space [27] so that the agent will generate more dimension output actions resulting more smooth robot's trajectories when it navigates [24]. The continuous action space $a \in \mathcal{A}$ is defined as

$$\mathcal{A} = \{v, \omega\}, \tag{Eq. 3.4}$$

where v and ω denote linear and angular velocity to be published to the mobile robot

respectively. Here, we assume that the robot in the environment is a differential drive robot which can be controlled by publishing v and ω.

To control the generated output action so that it will always follow ranges of $v \in [v_{min}, v_{max}]$ and $\omega \in [\omega_{min}, \omega_{max}]$ for time step t, we adjust the velocities as follows

$$v_{adj} = v_{min} + \frac{(v_t^{net} + 1)(v_{max} - v_{min})}{2}, \tag{Eq. 3.5}$$

$$\omega_{adj} = \omega_{min} + \frac{(\omega_t^{net} + 1)(\omega_{max} - \omega_{min})}{2}, \tag{Eq. 3.6}$$

where v_{adj} and ω_{adj} denote the adjusted velocities, whilst v_t^{net} and ω_t^{net} denote the velocities generated from the RL agent (the policy networks). Finally, we clip the action using the following functions:

$$v_t = \begin{cases} v_{max}, & v_{adj} > v_{max} \\ v_{adj}, & v_{min} \leq v_{adj} \leq v_{max} \\ v_{min}, & v_{adj} < v_{min}, \end{cases} \tag{Eq. 3.7}$$

$$\omega_t = \begin{cases} \omega_{max}, & \omega_{adj} > \omega_{max} \\ \omega_{adj}, & \omega_{min} \leq \omega_{adj} \leq \omega_{max} \\ \omega_{min}, & \omega_{adj} < \omega_{min}. \end{cases} \tag{Eq. 3.8}$$

3.2.1.3 Navigation Reward Function

We follow [81] which designs the reward function for the navigation task based on the artificial potential field method [82]. Using the function, the RL agent will be given a positive reward k_{goal} when the robot is able to arrive at the specified destination point. A negative penalty reward $-k_{crash}$ will also be given to the agent whenever the robot collides with obstacles. Additionally, the reward obtained by the agent will be computed based on the attractive potential field U_{attr} towards the goal position and the repulsive potential field U_{rep} towards surrounding obstacles.

The function to calculate the immediate reward for the time step t is defined as,

$$r_t = \begin{cases} k_{goal}, & d_{goal} < \epsilon_{goal} \\ -k_{crash}, & d_{obs} < \epsilon_{obs} \\ -U_{attr} - U_{rep} - r_{t-1}, & \text{otherwise}, \end{cases} \tag{Eq. 3.9}$$

where d_{goal} and d_{obs} denote the current robot's distance to the goal position and the nearest distance to an obstacle. Here, ϵ_{goal} and ϵ_{obs} denote small distance tolerance values toward the goal position and toward an obstacle. Note that the function also includes reward from the previous time step r_{t-1} to force the robot moving faster to reach the goal position.

We use the following formula to compute U_{attr} in the reward function,

$$U_{attr} = k_{attr} * d_{goal}, \tag{Eq. 3.10}$$

where k_{attr} denotes the constant for the attractive field. On the other hand, we use the following formula to compute the repulsive force:

$$U_{rep} = k_{rep} \sum_{i=1}^{N} \left(\frac{1}{c + d_i^{map\_obs}} \right) - \frac{k_{rep} * N}{c + l_{max}}, \tag{Eq. 3.11}$$

where k_{rep} denotes the constant for the repulsive field and N denotes the number of surrounding obstacles. Here, $d_i^{map\_obs}$ denotes the distance of the robot to the detected obstacle i in the local map whilst c and l_{max} denote a constant to limit the repulsive field and the maximum laser scan range value respectively.

3.2.2 Soft Actor-Critic

Soft Actor-Critic (SAC) is one of model-free, off-policy DRL algorithms which was introduced by Haarnoja et al. [83]. Dissimilar to other DRL algorithms which only consider maximizing expected future reward in the training process, SAC also aims to maximize entropy along with the future reward by utilizing a stochastic actor. As a result, the algorithm gives better performance since it is able to make the agent explores more in the environment compared to other algorithms which use deterministic actor architecture, such as DDPG [48] and Twin Delayed Deep Deterministic (TD3) policy gradient algorithm [84].

As depicted in Fig. 3.3, we need three types of functions to implement SAC which are all implemented as neural networks, namely policy networks, value networks, and Q networks. Since SAC is included as one of actor-critic frameworks in DRL algorithms, the policy network becomes the actor, whilst the value network along with the Q network become the critic. Additionally, SAC structure uses two Q networks to solve positive bias problem in the policy improvement step as has been previously described in [85]. It also uses a target value network in the structure so that the training process becomes more stable [27].

For each step iteration in time step t, the policy network generates and performs an action a_t based on the current state of the environment s_t to obtain the next state s_{t+1} and the immediate reward r_t. Subsequently, an experience which consists of tuple (s_t, a_t, r_t, s_{t+1}) is stored in the replay buffer $\mathcal{D}$. At the end of each training episode, the policy network and all of the critic networks are all updated by previous experiences sampled from $\mathcal{D}$. The updates are sequentially performed for 1) the value networks, 2) both Q networks, 3) the policy networks, and 4) the target value networks.

We choose to employ SAC in our study since it is one of the leading state-of-the-art methods which may improve the convergence ability and may able to avoid the high sample complexity problem in DRL training. We also choose to apply SAC since the algorithm can be appropriately used to train DRL agents performing task

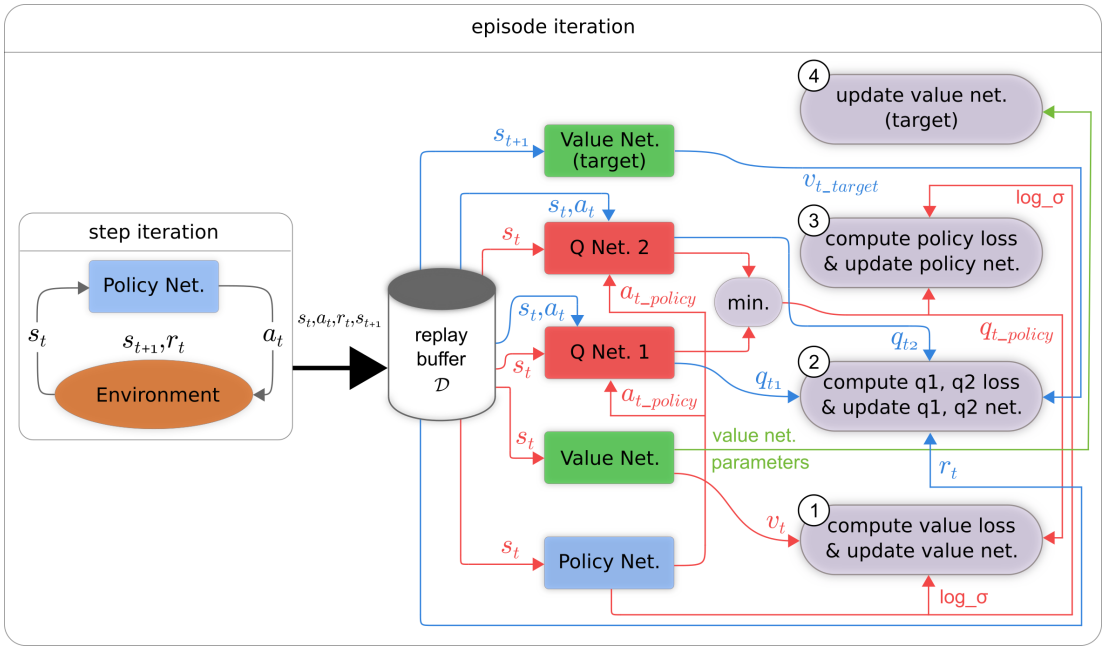


Figure 3.3: The structure of Soft Actor-Critic (SAC) algorithm proposed by Haarnoja et al. [83]. SAC employs five neural network functions which composed of policy network, Q network, and value network. The update process of all networks are done by using data which are sampled from the replay buffer for each episode iteration.

in environments with continuous action space. In the case of navigation task with differential drive robot, the output of the policy networks is the linear and the angular velocity $a_t = \{v_t^{net}, \omega_t^{net}\}$ to be published to the mobile robot. However, SAC does not provide a mechanism to limit the range of v_t^{net} and ω_t^{net} in the environment. Therefore, we need an additional clipping function to ensure that the final output actions from the RL agent always lay in a specified range.

3.2.3 State Transition Checking

We propose a state transition checking procedure inside an RL environment which is specifically intended for navigation task by leveraging odometry and laser sensor attached to the mobile robot. In our proposed method, the odometry sensor is utilized to automatically determine the dynamic timestep size which is needed to successfully perform each action in the environment. Furthermore, the laser sensor is employed to detect whether the current state should move to terminal state while an action is performed in the environment. In addition, the data from the laser sensor and the odometry sensor are directly used in our method instead of modeling them. We notice that, specific for the navigation task, current state may move to terminal state before an action can be successfully performed in the environment. This condition may happen since the robot may arrive at the goal position or may collide with obstacles in the middle of performing an action and before the action is considered as done

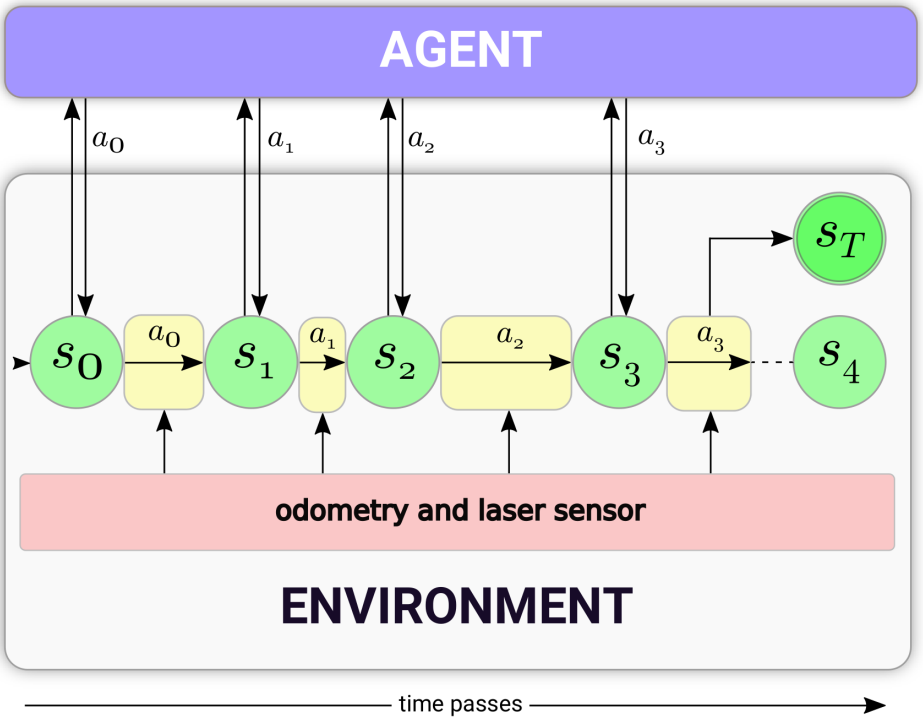


Figure 3.4: An illustration of the proposed state transition checking method in the DRL environment. The state transition checking is represented as yellow rounded rectangles inside an RL environment. It leverages odometry sensor to determine the dynamic timestep size and also laser sensor to immediately stop the current action whenever terminal conditions occur in the environment.

to be executed in the environment. When the condition occurs, the action should be cut and the environment's state should be moved to terminal state to keep the consistency in the environment.

Fig. 3.4 depicts the illustration of our proposed state transition checking method in the RL environment. The proposed checking function is represented as yellow rounded rectangles which will continuously check whether an action has been successfully performed in the environment or whether one of the terminal conditions occurs by using data obtained from odometry and laser sensor attached to the mobile robot. Note that the timestep size in the environment will be dynamically assigned as the width of the yellow rounded rectangle may vary for each action as shown in the case of action a_0, a_1, and a_2. In the case of action a_3, the state transition checking function is able to cut the action and capable to prevent the environment to move from state s_3 to state s_4. Instead, the environment moves from state s_3 to terminal state s_T.

The detailed formulation of our proposed state transition checking method is given by

$$s_{t+1} \sim \begin{cases} s_T, & d_{obs} < \epsilon_{obs} \text{ or } d_{goal} < \epsilon_{goal} \text{ or } t = N \\ \kappa(s_t), & v_{odom} \approx v_t \text{ and } \omega_{odom} \approx \omega_t \\ s_t, & \text{otherwise.} \end{cases} \quad \text{(Eq. 3.12)}$$

Here, the next state s_{t+1} becomes the terminal state s_T whenever the robot collides with obstacles ($d_{obs} < \epsilon_{obs}$), arrives at the navigation goal position ($d_{goal} < \epsilon_{goal}$), or when the maximum step in one episode is reached ($t = N$). Suppose that we train the RL agent using SAC, the environment will move to the next state if the robot's current velocity obtained from odometry sensor (v_{odom}, ω_{odom}) is close to the output action from the RL agent (v_t, ω_t), as described in the following formula:

$$\kappa(s_t) = p(s_{t+1}|s_t, \{v_t, \omega_t\})\pi_\phi(\{v_t, \omega_t\}|s_t), \qquad \text{(Eq. 3.13)}$$

where π_ϕ denotes the stochastic policy which generates the probability of taking action $a_t = \{v_t, \omega_t\}$ in state s_t. Moreover, p denotes the transition probability function in the RL environment, given the current state s_t and current action a_t.

In more details, we define Algorithm 3.1 to implement our proposed state transition checking procedure in an RL environment. The proposed algorithm extends openai_ros package and the standardized step function in OpenAI Gym framework [86] which performs the generated output action from the RL agent to the environment. As shown in line 3 through line 27, the algorithm utilizes a main looping procedure which will publish the velocity of the robot (line 4), obtain data from odometry and laser sensor (line 5 and line 6), and check the terminal conditions (line 9) continuously. Whenever one of the terminal conditions occurs based on the data from laser sensor, the looping procedure will be stopped and the environment will move to terminal state (line 16). Otherwise, the looping will only be stopped if the current robot's velocity obtained from the odometry sensor $\{v_{odom}, \omega_{odom}\}$ is equal to the output action generated by the RL agent $a_t = \{v_t, \omega_t\}$ with a velocity difference tolerance ξ (line 19). In addition, the robot will be stopped whenever the allowed maximum steps in one episode is reached (line 20).

3.2.4 Velocity Increment Scheduling

During the training process, we propose the velocity increment scheduling for the mobile robot along with our proposed state transition checking method to make the robot can navigate faster while still maintaining high value of success rate. Our proposed scheduling technique is one implementations of curriculum learning which manages the range value of the robot's velocity. In the early stage of the training process, a small velocity range is initially set. Subsequently, as the number of training episodes rises, we gradually increase the range of the robot's velocity linearly. Therefore, the complex task of generating appropriate actions which enables the mobile robot to navigate fast yet safe can be gradually and gently learned by the RL agent.

Suppose that we want to schedule the robot's velocity from lower speed A to higher speed B during N training episodes. Additionally, speed A has range of linear velocity of $v_a \in [v^a_{min}, v^a_{max}]$ and range of angular velocity of $\omega_a \in [\omega^a_{min}, \omega^a_{max}]$, whilst speed B has range of linear velocity of $v_b \in [v^b_{min}, v^b_{max}]$ and range of angular velocity of $\omega_b \in [\omega^b_{min}, \omega^b_{max}]$. Using our velocity increment scheduling technique, the range of

Algorithm 3.1: Step procedure in an RL environment with the proposed state transition checking method

Input: action $a_t = \{v_t, \omega_t\}$
Output: observation *observ*, reward r, done *done*
Data: goal coordinate $\{p_x^{goal}, p_y^{goal}\}$, robot coordinate $\{p_x^{robot}, p_y^{robot}\}$, obstacle distance threshold ϵ_{obs}, goal distance threshold ϵ_{goal}, velocity difference threshold ξ, current step t, maximum step in one episode N

$crash \leftarrow goal \leftarrow$ FALSE
unpause the environment
while TRUE **do**
  PUBLISHVELOCITY(v_t, ω_t)
  $\mathcal{L} \leftarrow$ GETLASERSCAN()
  $\mathcal{O} \leftarrow$ GETODOMETRY()
  $d_{goal} \leftarrow \sqrt{(p_x^{goal} - p_x^{robot})^2 + (p_y^{goal} - p_y^{robot})^2}$
  $d_{obs} \leftarrow$ MIN$(\mathcal{L})$
  if $(d_{obs} < \epsilon_{obs})$ ***or*** $(d_{goal} < \epsilon_{goal})$ **then**
    if $d_{obs} < \epsilon_{obs}$ **then**
      $crash \leftarrow$ TRUE
    else
      $goal \leftarrow$ TRUE
    end
    stop the robot
    break
  else
    $(v_{odom}, \omega_{odom}) \leftarrow$ GETCURRENTVELS$(\mathcal{O})$
    if $\mid v_{odom} - v_t \mid < \xi$ ***and*** $\mid \omega_{odom} - \omega_t \mid < \xi$ **then**
      if $t = N$ **then**
        stop the robot
      end
      break
    end
  end
  sleep for some time
end
pause the environment
observ $\leftarrow$ GETOBSERVATION$(\mathcal{L})$
$r \leftarrow$ COMPUTEREWARD$(crash, goal, \mathcal{L})$
done $\leftarrow$ *crash* **or** *goal* **or** $t = N$

the linear velocity for the current episode *eps* is given by

$$v_{eps} \in [v^a_{min}, v^a_{max} + eps * \Delta v_{max}], \quad \text{(Eq. 3.14)}$$

where $\Delta v_{max} = (v^b_{max} - v^a_{max})/N$ denotes the increment of the higher bound of the linear velocity for each episode. Here, we do not expand the range of the lower bound of the linear velocity since it is always set to zero and we do not want the robot to move backward while navigating. Similarly, the range of the angular velocity for the current episode *eps* is given by

$$\omega_{eps} \in [\omega^a_{min} - eps * \Delta\omega_{min}, \omega^a_{max} + eps * \Delta\omega_{max}], \quad \text{(Eq. 3.15)}$$

where $\Delta\omega_{min} = (|\omega^b_{min}| - |\omega^a_{min}|)/N$ denotes the increment of the lower bound of the angular velocity for each episode and $\Delta\omega_{max} = (\omega^b_{max} - \omega^a_{max})/N$ denotes the increment of the higher bound of the angular velocity for each episode.

3.3 Experiment Results and Discussions

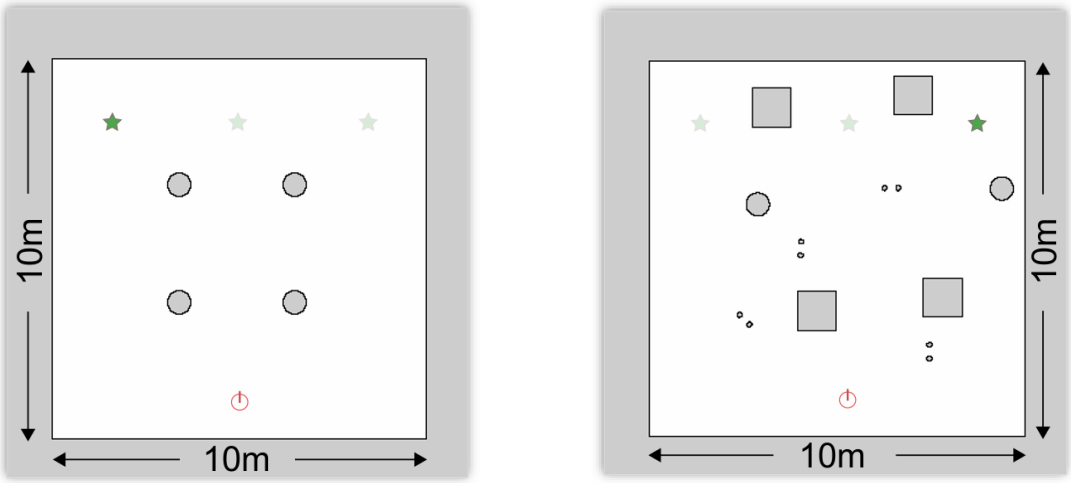
In this section, detailed analysis and discussion of the experiment results for our proposed framework is presented. We first describe the experiment setup including the implementation and the baselines for comparison. Subsequently, we present our analysis towards the experiment results.

3.3.1 Experiment Setup

3.3.1.1 Implementation

In order to validate our proposed framework, we conduct several experiments to train DRL agents to generate actions for a robot to navigate in an indoor environment. We implement our framework inside the Gazebo simulator [87] with a differential drive based Fetch robot [88] along with the Robot Operating System (ROS) [89] to subscribe data from the laser sensor and the odometry sensor. By using the ROS application programming interface (API), we directly feed the data from the sensors to the state transition checking method in our proposed framework. All of the navigation training environments used in this study are coded in Python which follow the standardized OpenAI Gym framework [86] and run on a workstation with an Nvidia Titan RTX graphic processor.

Fig. 3.5 depicts the maps which we use to train and to validate all of the DRL agents for the navigation task in this study. In the training process, we utilize a very simple map (Fig. 3.5 (a)) containing only four static cylinder obstacles. Additionally, we employ a more complex map (Fig. 3.5 (b)) in the validation process containing new static obstacles namely several boxes and several static human models. For both maps, we set the same one initial robot position and a goal position which is randomly picked from three candidates for each episode.



(a) The map for the navigation training

(b) The map for the navigation validation

Figure 3.5: Maps used in the navigation task experiments. Red circles represent the initial position of the mobile robot whilst green stars denote the navigation goal positions. For each episode, we set a random goal position for both maps.

Table 3.1: The hyperparameter settings of the SAC algorithm and the RL environment for the navigation task training.

Hyperparameter	Setting
SAC Algorithm Parameter	
Learning rate of all networks (η)	0.0003
Discount rate of reward (γ)	0.99
Soft update coefficient (τ)	0.01
Batch size	128
Replay buffer size	100,000
RL Environment Parameter	
Number of laser scan sectors (M)	10
Reward for arriving at the goal position (k_{goal})	100
Distance to goal threshold in meter (ϵ_{goal})	0.6
Penalty for collisions (k_{crash})	100
Min. distance to obstacle threshold in meter (ϵ_{obs})	0.4
Const. for attractive field (k_{attr})	100
Const. for repulsive field (k_{rep})	2.0
Const. to limit the repulsive field (c)	0.04
Const. to limit the max. laser scan range in meter (l_{max})	2.0
Velocity difference threshold (ξ)	0.3

All of the training process for the DRL agents are performed with SAC algorithm which is trained using Adam optimizer [90] and implemented using Pytorch library [91]. Moreover, we use network architectures which are shown in Fig. 3.6. We use the same two hidden layers with the same number of hidden neurons for each layer for all network architectures. In addition, the hyperparameters used in our experiments are listed in Table 3.1. We refer to [83] and [81] to define the hyperparameters for the SAC algorithm and for the RL environment with some additional adjustments. Furthermore, we refer to [78] to define the resolution of laser scans to compute values which represent each sector around the robot using Eq. 3.3.

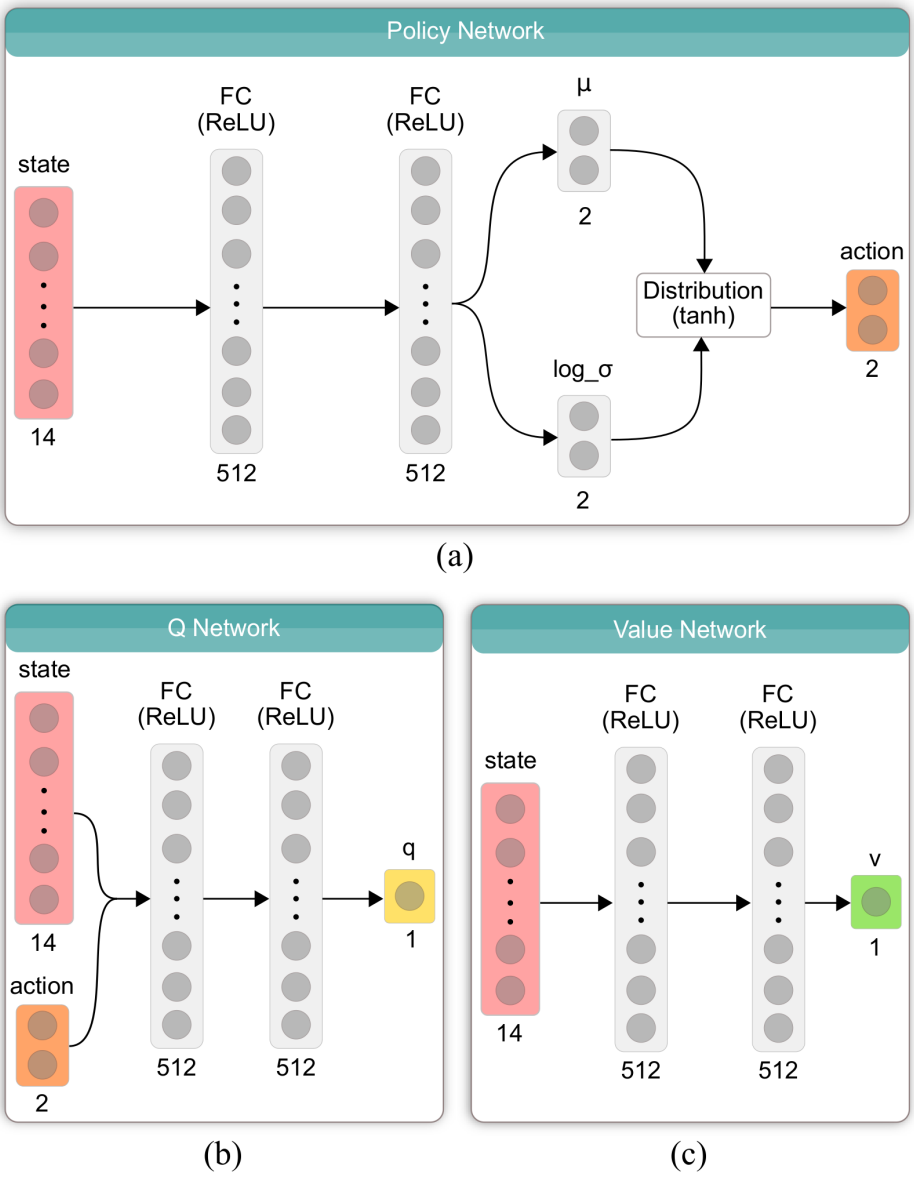


Figure 3.6: Network architectures for the SAC algorithm used in this study. (a) Policy network. (b) Q Network. (c) Value network. The policy network serves as the actor, whilst the Q network and the Value network serve as critics.

3.3.1.2 Baselines

For comparison purpose, we train several DRL agents using other previously existing frameworks with SAC algorithm and with the same map. We compare our proposed framework with openai_ros and gym-gazebo which are a framework and a toolkit intended for training robots inside Gazebo simulator with ROS framework. Moreover, we also train DRL agents using our framework without the velocity increment scheduling. The followings are the baselines which we use for the comparison:

1. openai_ros. In contrast to our proposed framework which uses both odometry and laser sensor, the openai_ros package only uses odometry sensor to determine the dynamic timestep size in the DRL environment.
2. gym-gazebo. Unlike our proposed framework and the openai_ros package, the gym-gazebo toolkit [92] uses fixed time step in the DRL environment. In our experiments, we set the same timestep size of 0.1s in the environment for all robot's velocity setting scenarios.
3. Ours without VIS. We only use the state transition checking method in the environment without the velocity increment scheduling for the robot in the training process.

3.3.2 Results and Analysis

The success rate values of training process in various speed settings are presented in Fig. 3.7. As depicted in Fig. 3.7 (a), our proposed framework which combines the state transition checking and the velocity increment scheduling gives high value of success rate. In addition, the training which schedules speed A to speed D gives better performance than the training which schedules speed A to C. Similarly, as shown in Fig. 3.7 (b), the success rate of our framework without the velocity increment scheduling with speed D is close to the success rate of the framework with speed A which has the lowest linear and angular velocity range value setting. Here, we confirm that lower range value of angular velocity can make the robot to navigate more stable.

Furthermore, we can see that the higher we set the velocity range value of the robot, the lower the success rate that we can get. As shown in Fig. 3.7 (b), we get the highest success rate when we set the robot at the lowest speed (speed A). When we increase the robot's velocity setting to speed B or speed C, the success rates then decrease gradually. On the other hand, Fig. 3.7 (c) shows that the training process with the openai_ros package fails and stops at the very early stage when we set the robot with higher velocity range values since the robot never reaches the desired velocity obtained from odometry sensor when it collides with obstacles. As also shown in Fig. 3.7 (d), the mismatch problem of the timestep size and the robot's velocity occurs with the gym-gazebo toolkit which uses fixed timestep size, resulting very unstable training process.

The progress of the time to goal value of the training process in various speed settings is presented in Fig. 3.8. By comparing Fig. 3.8 (a) to other sub figures, we can see that our proposed framework with the velocity increment scheduling is able to make the robot learn to move faster along with the increasing number of episodes. Moreover, the scheduling mechanism is proven able to reduce the average time to goal value. We get lower average time to goal values at the end of the training process for speed A in Fig. 3.8 (a) compared to the average time to goal value for speed A in Fig. 3.8 (b) and in Fig. 3.8 (c). Additionally, we also can see from Fig. 3.8 (a) that our proposed framework which schedules speed A to speed D is able to make the robot to have more stable average time to goal value compared to the training process which

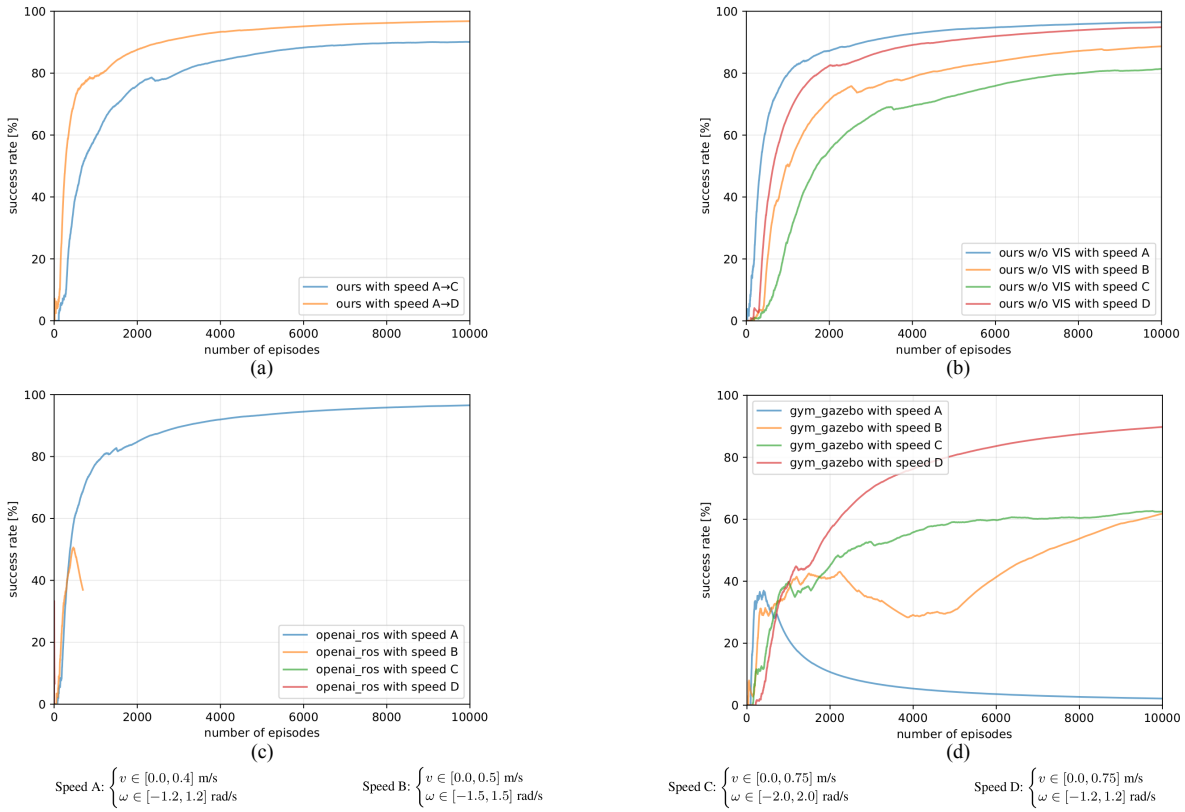


Figure 3.7: The success rate of the navigation training process in various speed settings. (a) Our proposed framework. (b) Our proposed framework without the velocity increment scheduling. (c) The openai_ros package. (d) The gym-gazebo toolkit.

schedules speed A to speed C.

Table 3.2 shows the details of all experiments conducted in this study. We can see that our proposed framework which combines the state transition checking along with the velocity increment scheduling gives the best success rate both in the training and in the validation process compared to other frameworks in all robot's velocity setting scenarios. On the other hand, the gym-gazebo toolkit gives the lowest average time to goal value among other frameworks. However, it results in lower success rate compared to our proposed framework both in the training and in the validation process. Therefore, we confirm that our framework can make the robot to navigate faster yet safe in the environment since it gives lower average time to goal values compared to other frameworks with speed A and speed B.

The stability and robustness of our proposed framework is also shown in Table 3.2. Our proposed state transition checking in the RL environment is proven able to stabilize the training process with various robot's velocity settings. The second block of the table lists the training results of our proposed framework without the velocity increment scheduling. From the list in the second block, we can clearly see that the success rate decreases and the average time to goal increases if we rise the

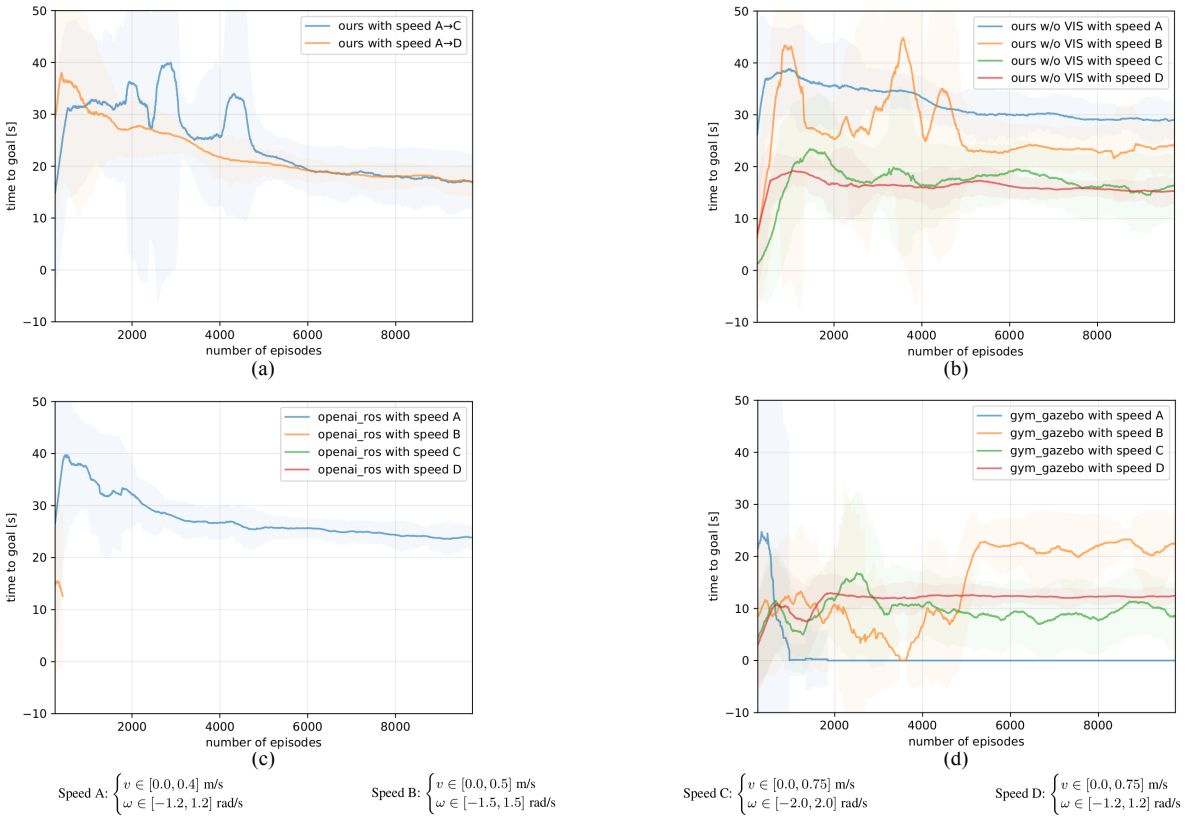


Figure 3.8: The average time to goal of the navigation training process in various speed settings. (a) Our proposed framework. (b) Our proposed framework without the velocity increment scheduling. (c) The openai_ros package. (d) The gym-gazebo toolkit.

robot's velocity (speed A through speed C). On the other hand, experiment results with the gym-gazebo toolkit do not show the same pattern due to the mismatch problem between the timestep size in the environment and the robot's velocity setting. Similarly, experiment results with the openai_ros package show that it does not robust to various robot's velocity settings. In our experiments, the robot gets stuck on some obstacles and cannot finish the training for speed B, speed C, and speed D. From the results of our proposed framework, we can confirm that the framework can make the training process more stable and also able to be successfully conducted in various speed settings compared to other existing frameworks.

We also present plots of some generated navigation trajectories of the robot which is trained with our proposed framework in the validation map. As depicted in Fig. 3.9, we confirm that the higher we set the angular velocity range value of the robot, the more shaky the robot is. We can obviously see that we get very oscillate trajectories (denoted in blue lines) when we set the robot velocity with speed C in the training process. Including the velocity increment scheduling strategy in the training process gives better trajectories which can minimizes the collision risk (denoted in orange lines). Nevertheless, we still get oscillating trajectories as shown in the orange lines

Table 3.2: Performance of our proposed framework in various speed settings. In both training and validation process, we confirm that the success rate of our proposed framework outperforms success rates of the baselines. Furthermore, we show that our framework can shorten the average time to goal while still maintaining a high success rate.

Framework	Speed	Training			Validation	
		Success Rate (%)	Avg. Time Goal (s)	Eps. Ends	Success Rate (%)	Avg. Time Goal (s)
Ours	A→C	90.08	26.48	10,000	44.39	19.08
	A→D	**96.79**	23.19	10,000	**91.10**	16.63
Ours w/o VIS	A	96.49	32.92	10,000	74.50	27.47
	B	88.67	29.56	10,000	62.80	23.79
	C	81.38	20.53	10,000	38.55	19.18
	D	94.85	16.77	10,000	58.35	14.20
openai_ros	A	96.55	28.10	10,000	73.09	21.16
	B	36.95	30.38	693	n/a	n/a
	C	0.0	0.0	12	n/a	n/a
	D	6.66	30.03	15	n/a	n/a
gym-gazebo	A	2.14	68.83	10,000	0.0	n/a
	B	61.83	23.8	10,000	67.85	21.89
	C	62.41	15.16	10,000	2.20	15.99
	D	89.79	**12.83**	10,000	32.25	**12.89**

Speed A: $\begin{cases} v \in [0.0, 0.4] \text{ m/s} \\ \omega \in [-1.2, 1.2] \text{ rad/s} \end{cases}$ Speed B: $\begin{cases} v \in [0.0, 0.5] \text{ m/s} \\ \omega \in [-1.5, 1.5] \text{ rad/s} \end{cases}$ Speed C: $\begin{cases} v \in [0.0, 0.75] \text{ m/s} \\ \omega \in [-2.0, 2.0] \text{ rad/s} \end{cases}$ Speed D: $\begin{cases} v \in [0.0, 0.75] \text{ m/s} \\ \omega \in [-1.2, 1.2] \text{ rad/s} \end{cases}$

from Fig. 3.9 (b) and Fig. 3.9 (c).

On the contrary, we get more smooth trajectories if we set the robot with lower angular velocity in the training process. As we can see from Fig. 3.9, the trajectories generated from our proposed framework give better results when we schedule the velocity of the robot from speed A to speed D. Here, we only schedule the linear velocity in the training process. We can obviously see that the trajectories denoted as red lines are very similar to the trajectories denoted as green lines for all target settings in the validation map. This confirms that our proposed velocity increment scheduling strategy is able to make the robot to generate smooth trajectory as when we set the robot with speed A, but with higher linear velocity setting which can give better average time to goal value for the robot to navigate.

3.4 Conclusion on Navigation Task Training

In this chapter, we proposed a framework for DRL based navigation for mobile robots with state transition checking in the RL environment and velocity increment scheduling for the robots during the training process. Based on the results in this study, we can conclude that the proposed framework performs well both in the training and in the validation process which enables the RL agent to learn a policy which can make the robot able to navigate fast yet safe in the environment. Moreover, we show that our proposed state transition checking in the RL environment is crucial to make the training process robust to various robot's velocity settings. In addition, we show that our proposed velocity increment scheduling strategy is able to assist the RL agent

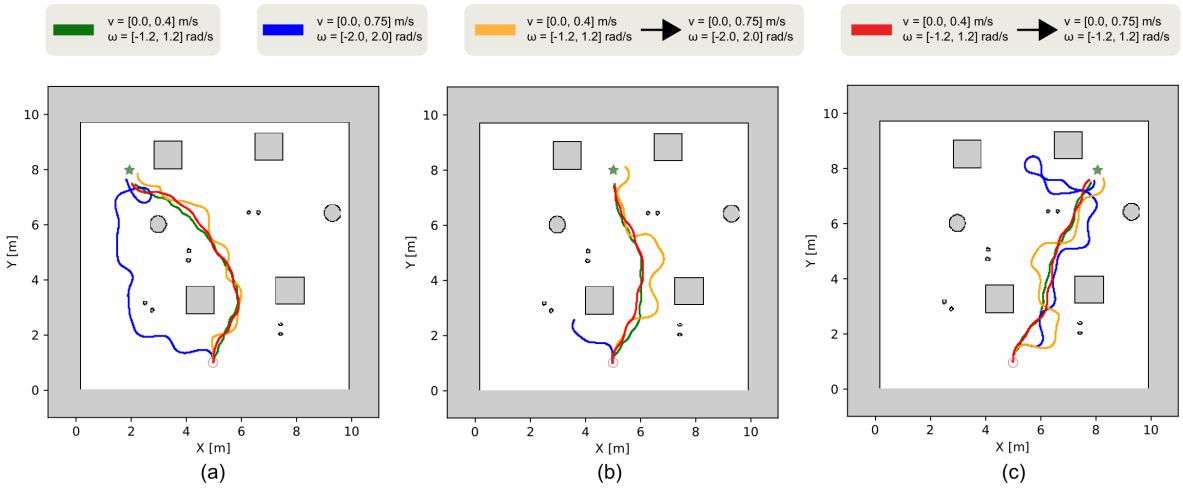


Figure 3.9: The plot of some robot's trajectories over several speed settings in the validation process. The robot's initial positions are represented as red circles whilst the target locations are represented as green stars. (a) The target is at the position of (2, 8). (b) The target is at the position of (5, 8). (c) The target is at the position of (8, 8). We demonstrate that the robot is able to generate more smooth trajectory using our proposed framework.

to gently and gradually learn the complex task of navigating in an environment in high speed and with high success rate. The proposed framework has been validated in a simulated environment with various velocity settings which confirms that our proposed framework has better performance compared to other frameworks.

Chapter 4

Attending Task Training with DRL

In this chapter, we present the application of soft actor-critic (SAC) learning algorithm to train a mobile robot to attend a target person at specific locations inside a Gazebo simulator. Since our previous study confirmed that the appropriate attending position while the target person is standing or walking is at his left or his right side, we design a novel U-shaped reward function behind the target person's position with respect to the robot's position. To make the robot can better portray the surroundings, we also propose a novel SAC architecture which employs one dimensional convolutional neural networks (CNN) to extract features from laser scans automatically during the training process. We also introduce the use of weight-scheduled action smoothing which able to stabilize actions generated by the agent in the attending task training. Our experiment results show that the robot is able to attend the target person at the designed location using our proposed reward function and SAC architecture. Furthermore, the weight-scheduled action smoothing which is introduced in this paper is able to make the agent generate more smooth actions. Most importantly, the proposed method does not prevent exploration, which becomes one of crucial aspects to make the agent can learn appropriately during the training process.

4.1 Background

Attending a specific target person is one of required basic tasks for an autonomous mobile service robot. To make the robot able to generate appropriate actions to attend the target person based on various conditions, we can train the robot (agent) using DRL by providing training environments which follow MDP and letting the robot to interact with them. In addition, we also need well designed reward functions inside the environments for guiding the robot to learn appropriate policies which will map the given state of the environments to suitable actions.

The position of the robot towards the target person is a very important aspect in the case of training a service robot for attending a specific person using DRL [25]. Therefore, we need to carefully formulate a reward function based on the appropriate position of the robot towards the target person. In our previous study [23], we confirmed that the appropriate attending positions for a mobile robot to the target person while he is standing or walking are at his left or his right side. Hence, we should design a reward function in the environment which can force the robot to position itself at the left side or at the right side of the target person.

In this study, we design a new reward function in the training environment by relating the reward distribution with the relative position of the robot towards the target person which can guide the robot to the appropriate attending positions. Furthermore, we employ SAC [83], which is one of the state-of-the-art DRL algorithms, to train the RL agent (robot). We also design a new architecture inside

the SAC agent which uses one dimensional CNN which extract features from all of the laser scans automatically so that the robot can better portray the surroundings.

In addition, we introduce a novel method called *weight-scheduled action smoothing* for performing the attending task training which does not prevent the exploration for the RL agent and able to make the robot generate more smooth actions while attending the target person. Since smoothing the robot actions may prevent the RL agent to find the right or the left attending goals around the target person, we modify the action smoothing strategy [74] and follow the curriculum learning strategy [34] to schedule the smoothing weights for the current action and for the previous action during the attending task training procedure.

4.2 Proposed DRL-Based Attending Training Framework

4.2.1 Attending Task Training System Architecture

The overview of our proposed training system is depicted in Fig. 4.1 which are divided into several components. As shown in the figure, we create a module which represents the training environment of the Gazebo simulator called the person attending environment. Moreover, we also develop another module called the soft actor-critic agent which represents the learning algorithm which we employ in the attending task training procedure. Each of the module in the architecture will be described briefly in the following subsections.

4.2.1.1 Gazebo Data Helper

One of modules inside our person attending environment is the Gazebo data helper. This module is responsible for subscribing the Gazebo model states, laser scans, and odometry data from the Gazebo simulator continuously. Moreover, the module passes the data to the reward calculator, the observation generator, and also the action performer module.

4.2.1.2 Reward Calculator

The module calculates relative robot's position rewards for the agent based on the relative position of the robot towards the target person's position. Fig. 4.2 (a) shows the U-shaped rewards distribution plot of the robot's relative position behind the target person, which is the extended version of our previous work [23]. By using our proposed reward formula, we do not want the robot to follow the target person from behind. Instead, we want the robot to be able to accompany the target person at his left or his right side. Additionally, the module also calculates the orientation reward which is shown in Fig. 4.2 (b) based on the orientation difference angle between the robot and the target person which is denoted as α, and the error tolerance ϵ. If terminal conditions do not occur, the final rewards received by the agent are the sum of the position rewards and the orientation rewards. Otherwise, the agent will receive penalties of big negative values.

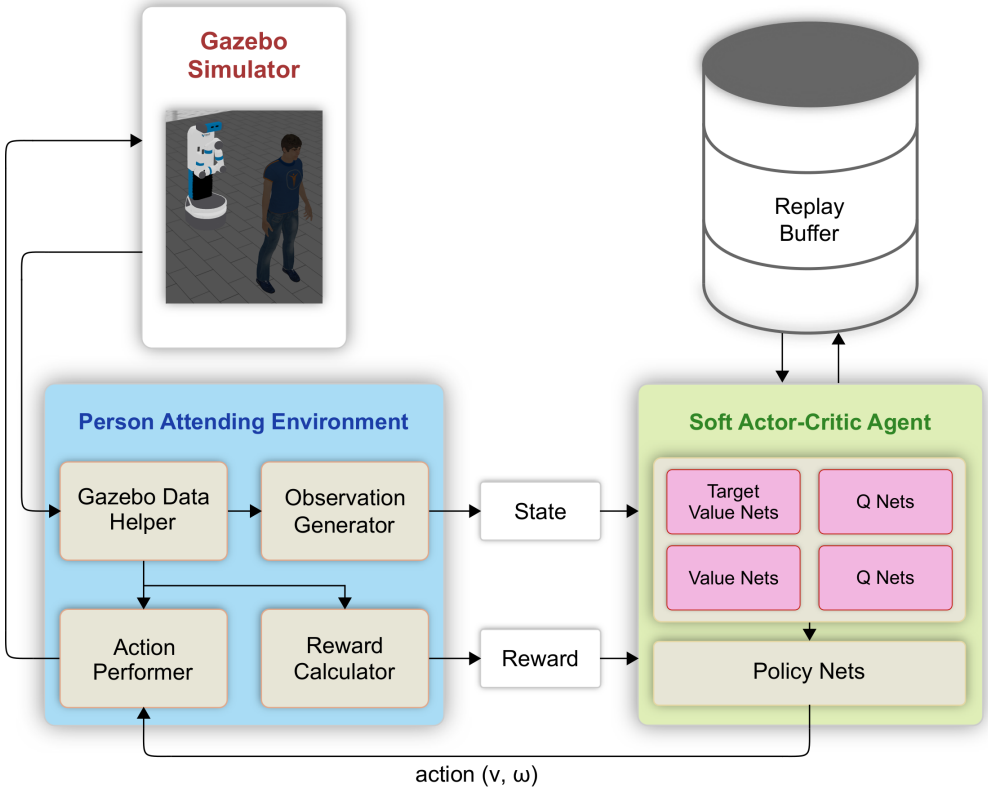


Figure 4.1: The overall architecture of the system in this study. We created the person attending environment which consists of several Python modules based on the OpenAI Gym framework that relates the Gazebo simulator and the soft actor-critic [83] agent using ROS application programming interface.

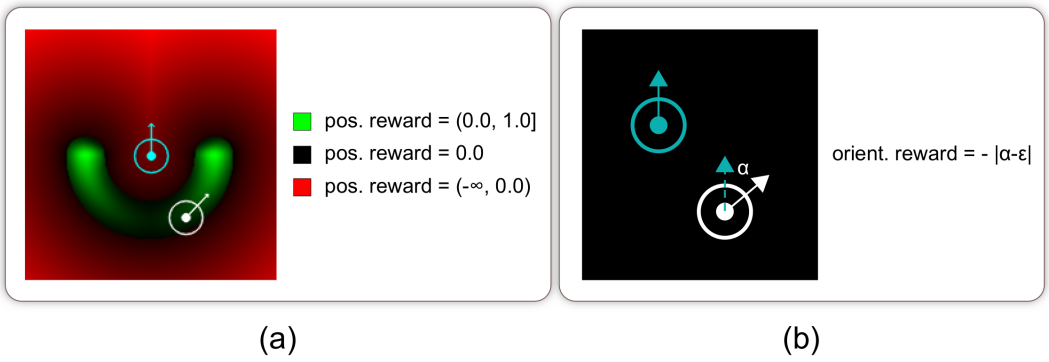


Figure 4.2: The position and the orientation reward of our person attending environment. (a) The reward distribution of the robot's relative position towards the position of the target person. (b) The calculation of the orientation reward which is based on target person's relative orientation towards the robot's orientation. Blue circles represent the target person whilst white circles represent the robot.

4.2.1.3 Observation Generator

The observation generator module is responsible for generating the state which consists of: a) laser scans (662 values), b) target person's relative coordinate from the robot (2 values), c) robot's previous action (2 values), and d) the target person's relative orientation (1 value). In our environment, we use all of the laser scans to form the state instead of just using several laser scan values, such as in [78].

4.2.1.4 Action Performer

The module is in charge of performing the given action from the policy nets to the robot inside the Gazebo simulator. It is also responsible for checking terminal conditions which may occur during the performance of actions in the environment. Hence, we implement the action performer module based on our state transition checking method [77].

4.2.1.5 Soft Actor-Critic Agent

We propose a novel architecture inside our SAC agent. As depicted in Fig. 4.3, given the state from the observation generator module, the policy nets extracts features from the laser scans using 1D convolutional neural networks. Afterwards, it generates the action by concatenating the learned features with other state values.

4.2.2 Weight-Scheduled Action Smoothing

To make the robot able to generate smooth and safe actions while attending the target person at his left side or at his right side, we propose the weight-scheduled action smoothing mechanism during the training process inside the attending environment. We modify the action smoothing strategy [74] which able to maintain the continuity of actions generated by the robot by adjusting the current action to the previous action using fixed weights. Nevertheless, instead of using fixed weights, our proposed weight-scheduled action smoothing follows the curriculum learning strategy [34] to schedule the weights during the training process so that the RL agent (the robot) can explore the environment sufficiently.

Suppose that we want to perform N training episodes for the robot inside the attending environment to obtain the left attending policy and the right attending policy. In order to generate the current action a_t to be performed in the environment within time step t, we propose the following equation

$$a_t = \begin{bmatrix} 1-\Delta_v & 0 \\ 0 & 1-\Delta_\omega \end{bmatrix} a_t^\pi + \begin{bmatrix} \Delta_v & 0 \\ 0 & \Delta_\omega \end{bmatrix} a_{t-1}, \qquad \text{(Eq. 4.1)}$$

where a_t^π denotes the action generated by the attending policy network within time step t and a_{t-1} denotes the previous action which has been executed in the environment. Here, $\Delta_v = eps * (C_v/N)$ denotes the scheduled weight for the linear velocity and $\Delta_\omega = eps * (C_\omega/N)$ denotes the scheduled weight for the angular velocity

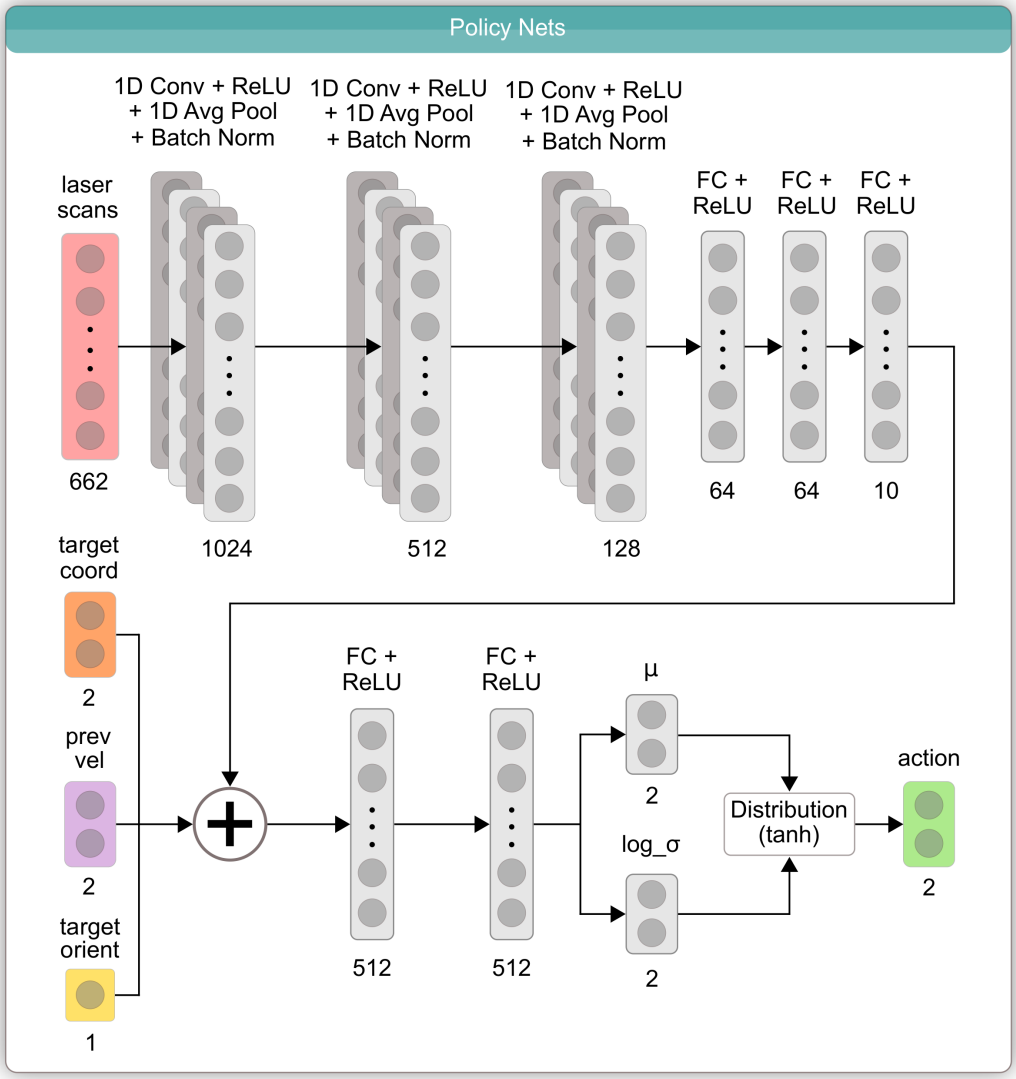


Figure 4.3: The structure of the proposed policy networks inside our SAC agent. In our structure, we employ a 1D convolutional neural networks which extracts features from the laser scan during the training process.

at the current episode *eps*. Moreover, C_v and C_ω denote final weights for the previous linear and angular velocity at the end of the training respectively which are set to 0.7 and 0.9 following Cai et al. [74].

4.3 Experiment Results and Discussion

For the training and the validation procedure, we prepare a world inside the Gazebo simulator as depicted in Fig. 4.4. To obtain the left and the right attending policies, we perform the attending task training with the weight-scheduled action smoothing for

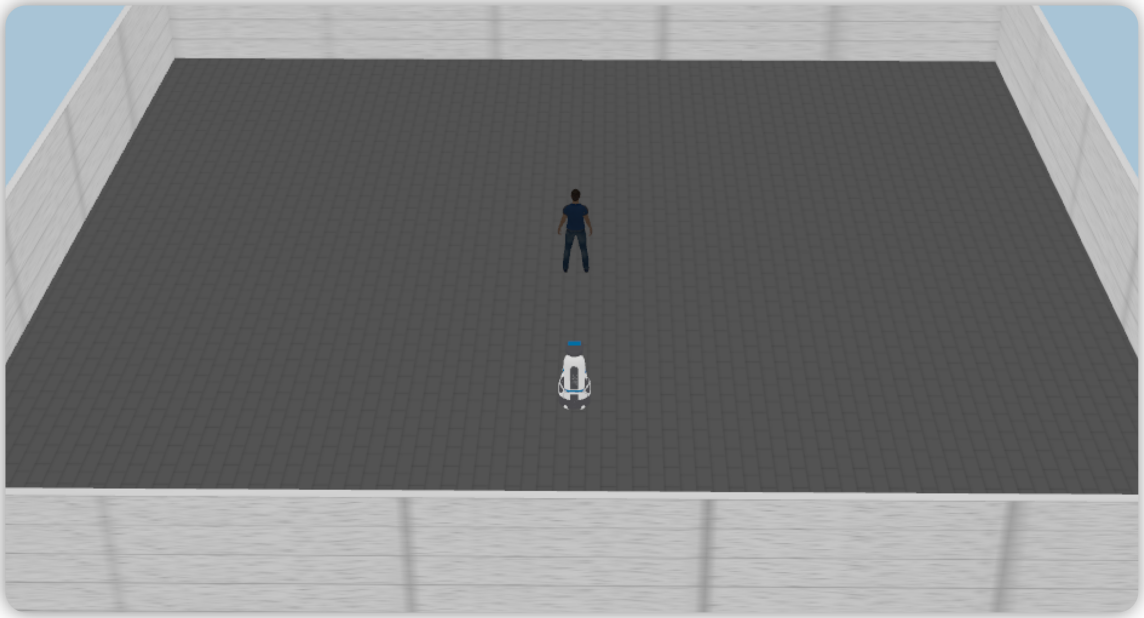
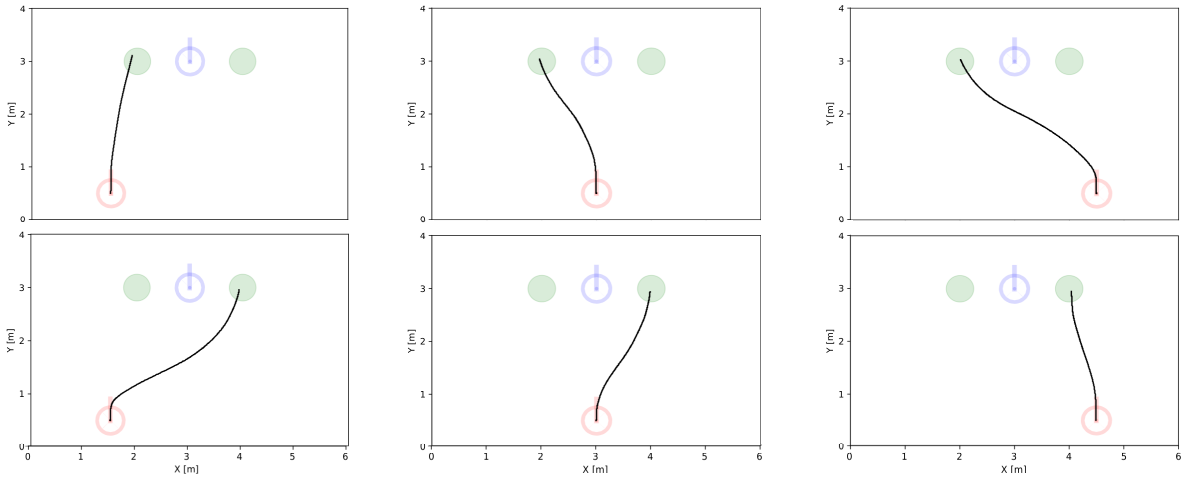


Figure 4.4: The simulated Gazebo environment for the training and the validation process of the attending task. We prepared an empty world filled with a target person to be attended and a Fetch robot.

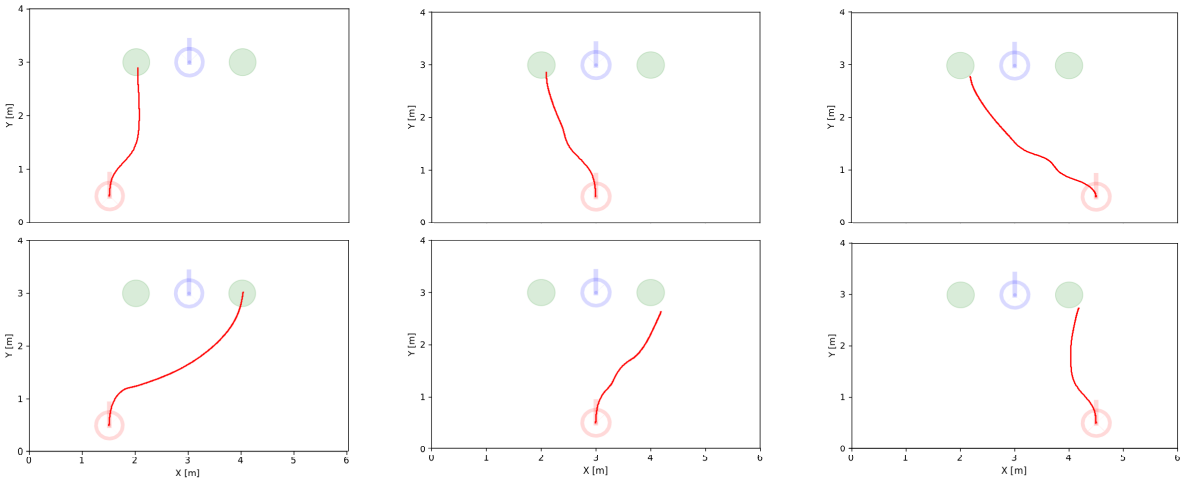
several agents using the SAC learning algorithm [83] along with the Adam optimizer [90]. We use PyTorch library [91] to implement all of the neural networks which represent all of the agents. We then set 5,000 episodes of 1,000 steps for the attending task training. We only end the episode whenever the robot collides with obstacles or hits the target person.

The plot of trajectories generated by the robot in the validation process for the attending task is depicted in Fig. 4.5. As shown in Fig. 4.5 (b), the generated trajectories without applying any action smoothing techniques are shaky and unstable, especially when the robot's position is a little bit far from the target person. As a result, the robot cannot reach the attending goals for some initial positions. Similarly, the condition also occurs when we employ the action smoothing strategy [74] in the training process. As shown in fig. 4.5 (c), the robot seems only try to get closer to the target person and ignores both left and right attending goals. This is due to the fact that the strategy uses fixed and small weights for the current actions generated by the agent compared to the weights for previous actions performed in the environment. Although the strategy is able to make the robot maintain the continuity of the actions and generate smooth trajectories, it prevents the agent finding the attending goals around the target person.

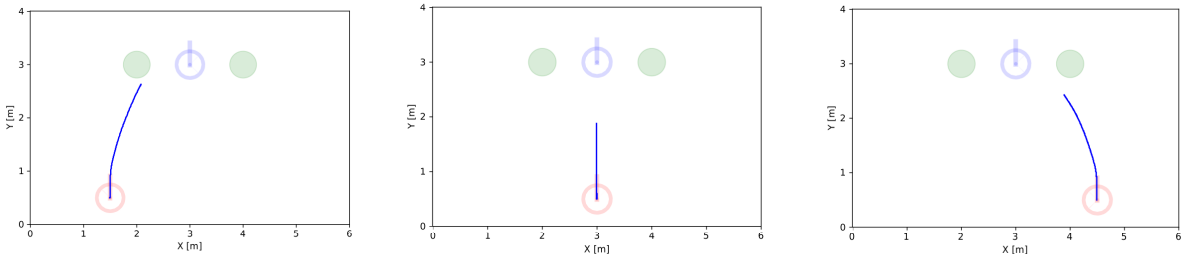
On the contrary, Fig. 4.5 (a) shows very smooth trajectories of the robot while attending the target person. Since our proposed weight-scheduled action smoothing method sets big values of weights for current actions and gradually decrease them, it



(a) Robot's attending trajectories of our proposed weight-scheduled action smoothing



(b) Robot's attending trajectories without applying action smoothing



(c) Robot's attending trajectories for applying the action smoothing strategy

Figure 4.5: Attending trajectories of the robot at various initial positions behind the target person in the validation process. Blue circles represent the position of the target person while red circles represent the initial positions of the robot. Green circles represent the left and the right attending goal positions toward the target person.

gives the opportunity for the agent to explore more in the environment to find the left and the right attending goals. As a result, the robot is capable of generating smooth actions at the end of the training and also able to reach both attending goals at the left and at the right side of the target person. Nevertheless, we also face the same problem mentioned in [58]. Since we set two goal positions for the attending task, the agent tends to choose only one goal and ignores another goal for each training session. Therefore, we need several training sessions to produce the optimal left and right attending policies. The main reason for this problem is that both attending goals are equal and give the same rewards for the agent. Moreover, as also mentioned in [93], the agent will only choose one same goal in each training session because of the catastrophic forgetting problem which is commonly occurs in RL-based training. In the following subsection, we show that the problem can be solved using the method which is proposed in this article.

4.4 Conclusion on Attending Task Training

From our experiment results, we show that our proposed U-shaped reward function in the environment can guide the robot to attend the target person at the desired location using our proposed architecture in the SAC agent. Furthermore, the weight-scheduled action smoothing which is introduced in this paper is able to make the agent generate more smooth actions. Most importantly, the proposed method does not prevent exploration, which becomes one of crucial aspects to make the agent can learn appropriately during the training process.

Chapter 5

Multiple Policies Integration with DRL

Given a training environment which follows MDP for a specified task, a DRL agent is able to find possible optimal policies which map states of the environment to appropriate actions by repeatedly trying various actions to maximize training rewards. However, the learned policies cannot be reused directly in the training process for other new tasks resulting wasted precious time and resources. To solve this problem, we propose a DRL-based method [94] for training an agent capable of selecting the appropriate policy for current state of the environment from a set of previously trained optimal policies for a given task which can be decomposed into other sub tasks. We implement our proposed method to a person-following robot task training that can be broken down into three sub tasks, namely: navigation, left attending, and right attending. Using the proposed method, the previously trained optimal navigation policy obtained from our previous work is integrated to attending policies which are trained in this study. Our experiment results show that the proposed method is able to integrate all sub policies using the action smoothing method even though the navigation and the attending policies have dissimilar input structures, unalike output ranges, and are trained in different ways. Moreover, our proposed method shows better results compared to training from scratch and training using transfer learning strategy.

5.1 Background

In line with the rapid growth in the fields of robotics, there are also increasing demands for service robots which have a main objective to assist and stay close with humans for supporting their daily needs. In order to successfully help finishing some tasks, robots are required to maintain close contacts when they interact and also be able to generate safe actions at the same time. For that reason, developments for particular robots which have the main ability to follow and attend a specified target person are also desired. Nevertheless, there are still many challenges that have to be dealt with when developing this type of partner-robots since person-following is not just a simple and ordinary task [4, 9]. Accordingly, the task can be seen as a complex duty to be performed by robots which can be decomposed into other several simpler tasks [10, 11].

Since adjusting the previously learned policy by retraining the agent for the navigation task may lead to catastrophic forgetting effect which may ruin the prior optimal attending policy [93], one of possible solutions to make the agent able to learn both tasks well is by extending the standard RL procedures so that temporally abstract actions can be accommodated [95, 96]. Hence, an RL agent is trained inside a complex environment to select among subroutines or sub goals instead of primitive actions [97]. By following this approach, Frans et al. [58] proposed a method to

perform the RL training for a substantial task in a hierarchical way by breaking down it into several sub tasks. In more details, a sub agent is assigned for each sub task and then a meta agent is set for all sub agents by sharing the same state space from the environment. However, the meta agent and all sub agents are trained simultaneously to form all of the sub policies and the meta policy altogether. Therefore, the method is not intended to be used directly in which a set of optimal sub policies have already been obtained from the previous training.

In this chapter, we propose a novel and general DRL-based method which is intended to perform a training procedure for an agent inside a complex environment by integrating multiple optimal policies from the previous training. In our proposed method, we modify the prior work of Frans et al. [58] by dividing the training process into two sequential stages for obtaining the optimal policy for each sub task and for acquiring the optimal meta policy. Moreover, we also introduce a module capable of integrating generated actions from those policies by applying the action smoothing strategy [74] which uses weighting for the current action and the previous action so that the robot can generate smooth and safe actions while it is around the target person. Furthermore, we show the implementation of our proposed method in the case of person-following robot training. To the best of our knowledge, this is the first work that applies DRL for training a robot to follow a specified target person by integrating the navigation and the attending optimal policies which have been learned beforehand.

5.2 Proposed DRL-Based Polices Integration Framework

5.2.1 General Framework for Policies Integration with DQN

Suppose that we have several RL environments which follow Markov decision process (MDP) and are associated with particular tasks E_{task} with specific reward functions r_{task} and state space $\mathcal{S}_{task}$. We then can train several agents to maximize rewards using any reinforcement learning algorithms inside those environments to obtain a set of optimal policies

$$\prod = \{\pi_{task} \mid task \in 1 \dots N\}, \tag{Eq. 5.1}$$

where π_{task} denotes the optimal policy which maps $\mathcal{S}_{task}$ into corresponding actions for each task and N denotes the number of environments. Furthermore, we can create a new complex combined environment $\mathcal{E}_{comb} = \{E_{task} \mid task \in 1 \dots N\}$ which has multiple objectives and can be decomposed hierarchically into other sub-environments. Moreover, the new combined environment also has a combined reward function $r_{comb} = \{r_{task} \mid task \in 1 \dots N\}$ which consists of multiple reward functions that represent the corresponding objective for each sub-task.

In order to obtain the optimal policy for the environment $\mathcal{E}_{comb}$, we can train an RL agent which able to reuse optimal policies $\prod$ from the previous training processes by assuming that we can convert the corresponding state space from the

combined environment $\mathcal{S}_{comb}$ into the state space for a specific sub-task using the state conversion function $\sigma_{task} : \mathcal{S}_{comb} \rightarrow \mathcal{S}_{task}$. Therefore, given the current state of the combined environment, we then can obtain a set of actions for each task in $\mathcal{E}_{comb}$ as follows

$$\mathcal{A} = \{\pi_{task}(\sigma_{task}(\mathcal{S}_{comb})) \mid task \in 1 \dots N\}. \quad \text{(Eq. 5.2)}$$

Thus, the optimal policy for the RL agent is then given by the following equation:

$$\pi^*(\mathcal{S}_{comb}) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q^{\pi}(\mathcal{S}_{comb}, a). \quad \text{(Eq. 5.3)}$$

The optimal policy will make the RL agent choose the most appropriate action to be executed in the environment from $\mathcal{A}$ given the current state $\mathcal{S}_{comb}$. Here, Q^{π} denotes a neural network function which estimates the score for each possible action given the current state of the environment. Q^{π} follows the Bellman equations [18]:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{k}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right], \quad \text{(Eq. 5.4)}$$

where γ denotes the discount factor for the future reward which has range value of $[0, 1]$.

Fig. 5.1 shows the general overview of our proposed method. Since our goal is to reuse optimal policies obtained from previous training processes, we divide our method into two main stages. At the first stage, we let each RL agent to interact with the corresponding environment in the *initial policies training*. We then use all optimal policies obtained from the previous stage in the *policies combination training* stage. At the second stage, we train an RL agent with the Double DQN [98] learning algorithm so that it becomes able to select the most appropriate policy for the given current state of the environment. Here, we use the Double DQN since it has a simple structure and it is easy to be implemented [24]. Moreover, it employs two Q networks which can solve the overestimation bias problem during the training and can make the learning process becomes more stable [85, 98].

Algorithm 5.1 shows the detailed procedure in our proposed method to integrate multiple optimal policies using the Double DQN learning algorithm. In contrast to Frans et al. [58] who perform the training for all task and the training for the policy selection agent simultaneously, we first perform the initial policy training steps which are shown in line 6 through 13. Once we obtain a set of optimal policies for all tasks, we then move to the second stage of our proposed method which is the policies integration training steps that are shown in line 15 through 31. In the second stage, a set of actions generated from the previously trained policies are obtained in line 17 through line 19 given the current state $s_t \in \mathcal{S}_{comb}$ from the combined environment $\mathcal{E}_{comb}$ at time step t. Subsequently, in line 20 and 21, the agent will select the most appropriate policy for the current state to generate the proper action a_t to be executed in the environment as shown in line 22. Finally, the update steps for all

Algorithm 5.1: Policies Integration with Double DQN

Prepare $\mathcal{E}_{comb} = \{E_{task} \mid task \in 1 \dots N\}$
Initialize networks Q^{π} with random weight θ
Initialize networks $\hat{Q}^{\pi}$ with random weight $\theta^{-} = \theta$
Initialize a replay buffer for the policy integration $\mathcal{D}$
Initialize policies $\prod = \{\pi_{task} \mid task \in 1 \dots N\}$
repeat
  for *task = 1, N* **do**
    for *t = 1, T* **do**
      Select task action a_t^{task} generated from π_{task}, given current task state s_t^{task}
      Execute a_t^{task} in E_{task}, observe task reward r_t^{task} and new task state s_{t+1}^{task}
      Update π_{task} using j transitions $(s_j^{task}, a_j^{task}, r_j^{task}, s_{j+1}^{task})$
    end
  end
until *convergence*
repeat
  for *t = 1, T* **do**
    for *task = 1, N* **do**
      Generate action a_t^{task} from π_{task}, given current state $s_t \in \mathcal{S}_{comb}$ from $\mathcal{E}_{comb}$
    end
    With probability ϵ, select random a_t^{task} as a_t
    Otherwise, select $a_t = \underset{a^{task}}{\operatorname{argmax}}\, Q^{\pi}(s_t, a^{task}; \theta)$
    Execute a_t in $\mathcal{E}_{comb}$, observe reward $r_t \in r_{comb}$ and new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    Sample a random transition batch (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$
    Set $a' = \underset{a'}{\operatorname{argmax}}\, Q^{\pi}(s_{t+1}, a'; \theta)$
    Set $r' = r_j + \gamma\, \hat{Q}^{\pi}(s_{t+1}, a'; \theta^{-})$
    Set $y_j = \begin{cases} r_j, & \text{eps. terminate at step } j+1 \\ r', & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q^{\pi}(s_j, a_j; \theta))^2$ w.r.t. θ
    Every C steps, reset $\hat{Q}^{\pi} = Q^{\pi}$
  end
until *convergence*

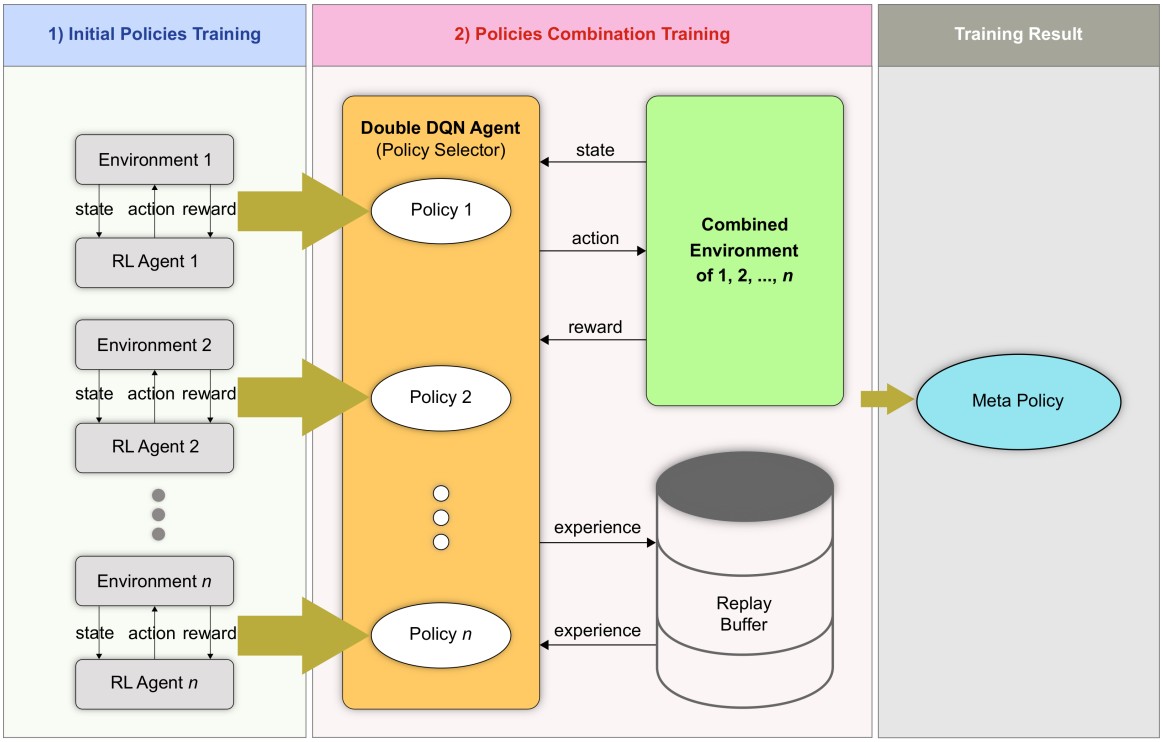


Figure 5.1: The general overview of our proposed method in this article. We extend the previous work from Frans et al. [58] so that optimal policies obtained from the previous training can be reused for another new task. In the initial policies training, we let several RL agents to form optimal policies from the data (state, action, reward) obtained from the environment. Subsequently, the optimal policies are used in the policies combination training for obtaining the optimal meta policy.

neural networks which follow the Double DQN learning algorithm using experiences stored inside a replay buffer $\mathcal{D}$ are shown in line 23 through 29.

5.2.2 Person-Following Task Training with Policies Integration

In this section, the implementation of our proposed method to perform the training procedure for a person-following robot is presented. We first describe the details of the person-following robot environment. Afterwards, we show the application of the action smoothing strategy [74] for integrating actions which are generated from different policies inside our RL agent.

5.2.2.1 Person-Following Robot Environment

We define the person-following robot environment as an environment which combines the navigation environment and the attending environment. Here, we want the robot able to perform the navigation task by approaching the target person and then attending him within a safe close distance. Since the attending position of the robot towards the target person is such an important aspect [25], we want the robot able to attend the target person at his left side or at his right side [23, 45]. Therefore,

Table 5.1: Comparison of training and validation environments in our study. We use different state space and action space since each environment has dissimilar objective.

Environment	State Space						Action Space	
	$\mathcal{L}$	v_{t-1}	ω_{t-1}	x_{tgt}	y_{tgt}	α_{tgt}	Range of v (m/s)	Range of ω (rad/s)
Attending	✓	✓	✓	✓	✓	✓	$[-0.75, 0.75]$	$[-1.2, 1.2]$
Navigation	✓	✓	✓	✓	✓	✗	$[0.0, 0.75]$	$[-1.2, 1.2]$
Person-Foll.	✓	✓	✓	✓	✓	✓	$[-0.75, 0.75]$	$[-1.2, 1.2]$

the person-following robot environment $\mathcal{E}_{follow}$ can be written as

$$\mathcal{E}_{follow} = \{E_{left}, E_{nav}, E_{right}\}, \tag{Eq. 5.5}$$

where E_{left}, E_{nav}, and E_{right} denote left attending environment, navigation environment, and right attending environment respectively. The left attending and the right attending environment share the same state space and action space. Nevertheless, as listed in Table 5.1, the navigation environment has different state space and action space from the attending environments since they have completely different goals.

In the navigation environment, to make the robot able to navigate safely, we include distance to obstacles data $\mathcal{L}$ around the robot in the state space which are obtained from a laser sensor attached to it. Moreover, to make the robot able to approach the target person's position and move steadily, we include the relative coordinate of the target person from the robot (x_{tgt}, y_{tgt}) and the previous velocity of the robot (v_{t-1}, ω_{t-1}) as well. In this study, we assume that the robot can get the accurate position of the target person and it can be controlled by publishing linear velocity v and angular velocity ω within time step t. On the other hand, we add the relative orientation of the target person from the robot α_{tgt} in the state space for the attending environment since the data is required by the robot in order to attend the target person at correct positions. Thus, the state space for the person-following robot environment $s \in \mathcal{S}_{follow}$ can be defined as the union between state space of those environments as follows

$$\mathcal{S}_{follow} = \{\mathcal{L}, v_{t-1}, \omega_{t-1}, x_{tgt}, y_{tgt}, \alpha_{tgt}\}. \tag{Eq. 5.6}$$

We define the terminal episode S_T for the agent whenever the robot collides with obstacles or hits the target person. Additionally, we also define the same continuous action space for the navigation environment and for the attending environment. Therefore, the action space for the person-following robot training environment $a \in \mathcal{A}_{follow}$ is defined as

$$\mathcal{A}_{follow} = \{v, \omega\}. \tag{Eq. 5.7}$$

Here, we set the same range value of angular velocity ω for each environment but different range value of the linear velocity v for the navigation environment.

Furthermore, we use different reward functions for the navigation and for the attending environment since all environments have dissimilar objectives. We employ a reward function which is based on the artificial potential field (APF) for the navigation environment so that the robot able to avoid obstacles and able to reach the target person [77, 81, 82]. On the other hand, we apply U-Shaped reward function [45] for the attending environment so that the robot able to attend the target person at his left or at his right side. Since in the person-following environment we want the robot able to switch between performing the attending task and performing the navigation task based on the distance of the robot from the target person, we then formulate the combined reward function for the person-following environment as follows

$$r_{follow} = \begin{cases} r_{att} & d_{tgt} < \xi \\ r_{nav} & \text{otherwise,} \end{cases} \qquad \text{(Eq. 5.8)}$$

where r_{att} and r_{nav} denote the U-Shaped attending reward function and the APF navigation reward function respectively. Here, d_{tgt} denotes the distance between the target person and the robot while ξ denotes the distance threshold value which is set to 2 meters in this study.

5.2.2.2 Policies Integration in the Person-Following Task

Fig. 5.2 depicts the application of our proposed DRL-based policies integration for the person-following task in this study. Since weight-scheduled action smoothing was applied to obtain both left attending policy π_{left} and right attending policy π_{right} but not for obtaining the navigation policy π_{nav}, we then propose the implementation of action smoothing mechanism to integrate actions which are generated from those different policies. In our system, we develop an action generator module (represented as the gray rounded rectangle) which has the main objective to produce final actions to be performed. Inside our module, the current state of the person-following environment s_t is converted into the appropriate state form for each policy using the navigation-state conversion function σ_{nav} and the attending-state conversion function σ_{att}. Subsequently, all generated actions from each policy are fed into the action adjustment function along with the selected policy idx from the Double DQN module.

We show the detailed procedure inside the action adjustment function in Fig. 5.3. First, the Euclidean distance between the target person and the robot is computed using the following formula:

$$d_{tgt} = \sqrt{(p_x^{tgt} - p_x^{rbt})^2 + (p_y^{tgt} - p_y^{rbt})^2}, \qquad \text{(Eq. 5.9)}$$

where $(p_x^{tgt},\ p_y^{tgt})$ denotes the coordinate of the target person, and $(p_x^{rbt},\ p_y^{rbt})$ denotes the coordinate of the robot. Afterwards, the current person-following action a_t is set based on the given selected policy idx. In order to make the robot able to generate smooth and safe actions, the adjusted action a_{adj} is then computed based on the action smoothing strategy [74] which follows the equation below:

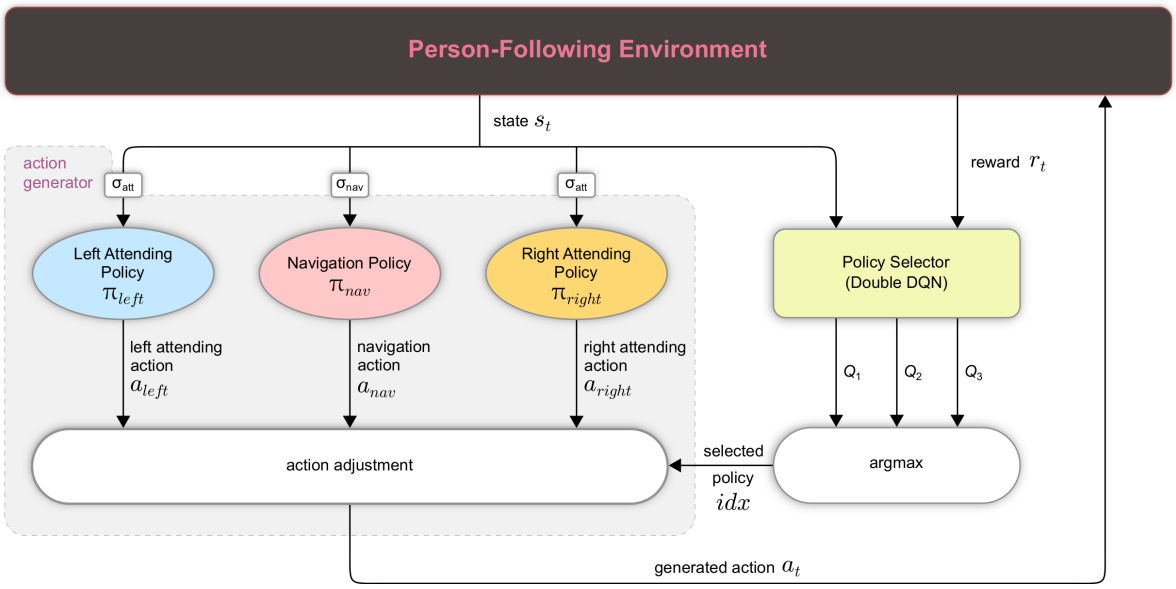


Figure 5.2: The implementation of our proposed policies integration method for the person-following environment. We apply the action smoothing mechanism inside the action generator module in our system for generating actions to be performed in the environment.

$$a_{adj} = K_1\, a_t + K_2\, a_{t-1}, \qquad \text{(Eq. 5.10)}$$

where a_{t-1} denotes the previous person-following action which has been performed in the environment. K_1 and K_2, respectively, denote the constant weights of the current action and the previous action which are given in the following matrices:

$$K_1 = \begin{bmatrix} 0.3 & 0.0 \\ 0.0 & 0.1 \end{bmatrix}, \; K_2 = \begin{bmatrix} 0.7 & 0.0 \\ 0.0 & 0.9 \end{bmatrix}. \qquad \text{(Eq. 5.11)}$$

Since we applied the weight-scheduled action smoothing inside the attending environment in the previous training, the action smoothing strategy will be used if the selected action is generated from the left attending policy ($idx = 0$) or from the right attending policy ($idx = 2$). Additionally, we also apply the smoothing mechanism so that the robot will not hit the target person by considering a safe distance threshold ξ which is set to 2 meters. Finally, the previous action is updated and the current person-following action a_t is generated.

5.3 Experiment Results and Discussion

5.3.1 Experiment Setup

All experiments conducted in this study are performed inside the Gazebo simulator [87] with a differential drive based Fetch robot [88] which can be controlled

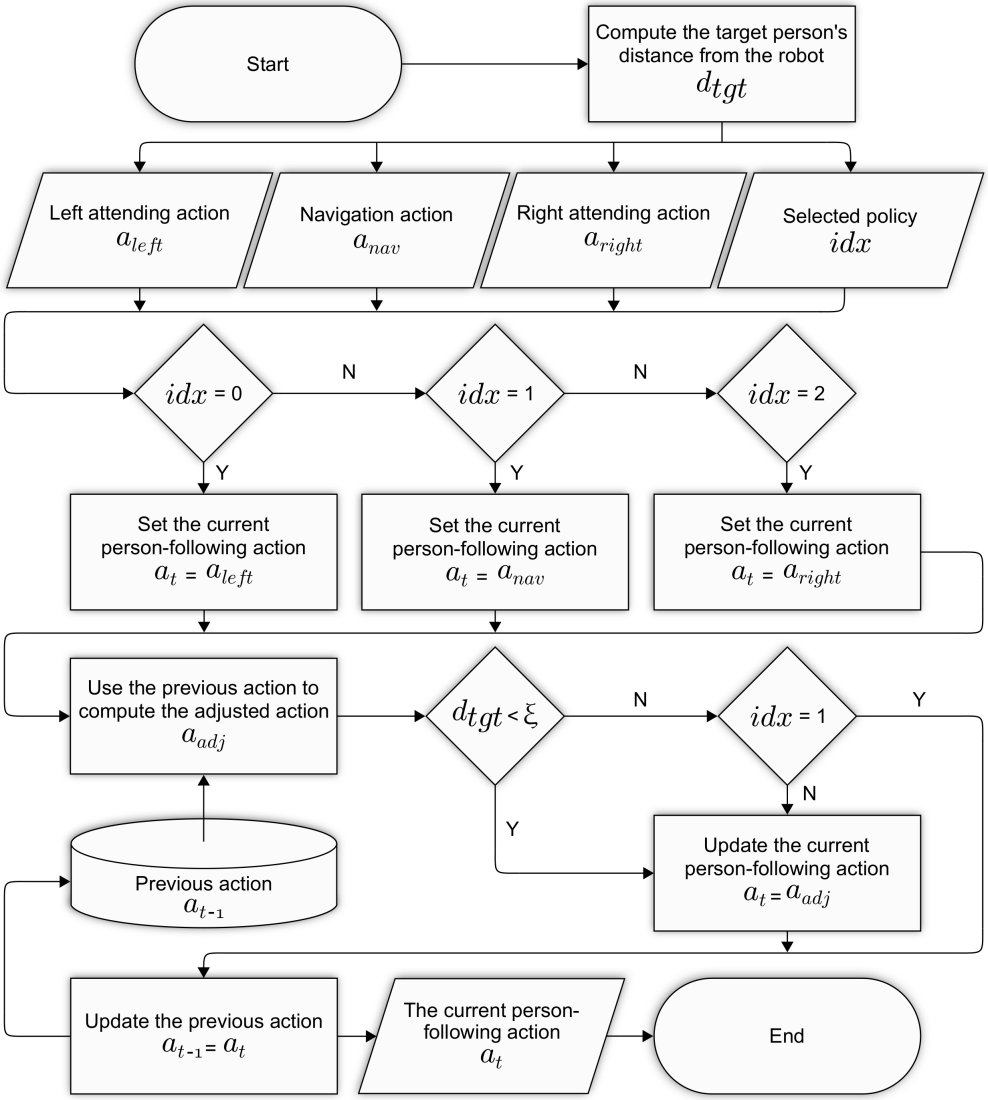
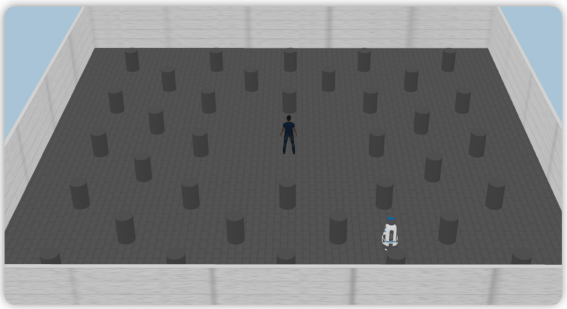


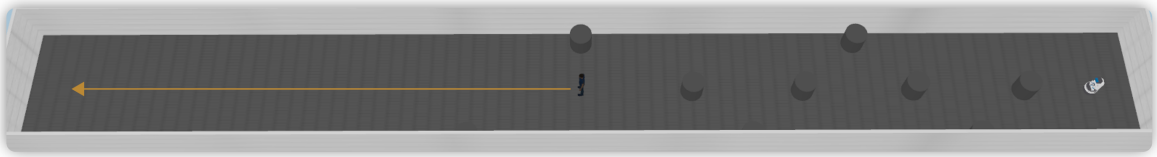
Figure 5.3: A flowchart which represents the detailed procedure inside the action adjustment function of our proposed action generator module. The function generates the current person-following action from four inputs which consist of all actions and the selected policy by the Double DQN agent.

by publishing linear velocity v and angular velocity ω using ROS application programming interface [89]. Fig. 5.4 depicts the simulated training and validation environments for the attending task and the person-following task which are developed in this study. In addition, all environments in this study are developed in Python which follow the standardized OpenAI Gym framework [86] and the state transition checking method [77].

We use PyTorch library [91] to implement all of the neural networks which represent all of the agents. In addition, we set 1,000 episodes of 1,500 steps for the person-following task training. Moreover, we set 100 episodes with maximum step of 1,500



(a)



(b)

Figure 5.4: Simulated environments developed in this study. (a) The training and validation environment for the person-following task. (b) The validation environment for the person-following task with a moving target person. We use Gazebo simulator [87] and load a Fetch robot [88] along with the ROS application programming interface [89] to implement all environments. The goal of the person-following environment is to make the robot able to navigate towards the target person and then attend the target person.

for the person-following validation process. In both training and validation, we only end the episode whenever the robot collides with obstacles or hits the target person.

Furthermore, we also train other agents for the comparison purpose. We train agents with the modification of action smoothing mechanism in our proposed method. Moreover, we also train agents using the SAC algorithm from scratch and also with transfer learning strategy. The followings are the details of the baselines which we use for the comparison:

1. Proposed method with full action smoothing. In this method, no matter what policy are selected, we always apply the action smoothing strategy for all actions generated by the Double DQN agent.
2. Proposed method with half action smoothing. In contrast to our original proposed method, we only apply the action smoothing strategy for all actions generated by the Double DQN agent if the left attending policy or the right attending policy are selected.
3. SAC from scratch with attending mode. We use the SAC learning algorithm to train agents in the person-following environment from scratch by including the relative target person's orientation from the robot (α_{tgt}) in the state space.
4. SAC from scratch with navigation mode. We also train other agents in the person-following environment from scratch using the SAC learning algorithm by excluding the relative orientation of the target person from the robot in the state space.

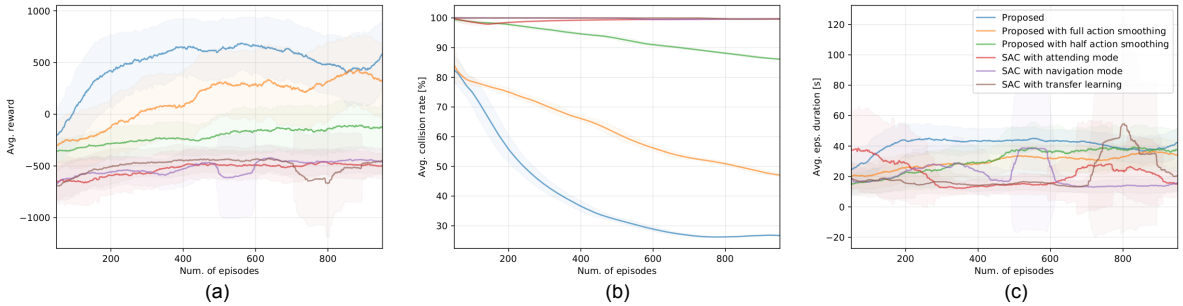


Figure 5.5: Plots of agents' performances in the person-following task training across three trials. (a) Average reward per episode. (b) Average collision rate per episode. (c) Average episode duration per episode. A rolling average of 100 episodes is considered to depict each curve in this figure.

5. SAC with transfer learning. In this method, we transfer the weights of the navigation policy to the SAC agent at the initial stage of the training so that the agent only has to learn the attending policies.

5.3.2 Results and Discussion

Fig. 5.5 depicts performances of all agents during the person-following task training across three trials. Overall, the proposed method shows better performance compared to all baselines. We can obviously see from Fig. 5.5 (a) that all agents which are trained using SAC algorithm from scratch can only obtain average rewards at around -500, whilst the agents which are trained using the proposed method (depicted as the blue line) can gain nearly 500 rewards in average. As also shown in Fig. 5.5 (b), only agents which are trained using the proposed method (depicted as blue, orange, and green lines) can learn to avoid obstacles and not to hit the target person during training. Moreover, from Fig. 5.5 (c), we can clearly see that not only do agents which are trained using the proposed method (depicted as the blue line) can attend the target person longer, but they also can have very stable average episode duration compared to other agents.

In more details, the summary of all agents' performances during the training and the validation process are listed in Table 5.2. As shown in the first row, agents which are trained using our proposed method show the best results both in the training and in the validation process. Moreover, as also listed in the first until the third row, our proposed method and its variations show higher average of episode duration compared to agents which are trained from scratch and agents which are trained using transfer learning strategy. Therefore, the results show the effectiveness of the action smoothing method which is employed while integrating generated actions from different policies in the action generator module inside our proposed method.

If we compare the collision rate of the proposed method with its variations, we can see that agents which are trained using half action smoothing show the highest collision rate among others. The result is reasonable since there are possibilities

Table 5.2: Summary of performances of all agents during the training and the validation process for the person-following task across three trials. The best results in each column are highlighted in bold font.

Method	Training			Validation	
	Reward	Collision Rate (%)	Episode Duration (s)	Collision Rate (%)	Episode Duration (s)
Proposed	**462.49 ± 573.4**	**26.14 ± 5.26**	**40.29 ± 16.3**	**14.01 ± 6.0**	**30.78 ± 7.71**
Proposed with full action smoothing	127.42 ± 629.33	46.37 ± 11.52	29.45 ± 16.82	38.34 ± 8.39	21.41 ± 11.45
Proposed with half action smoothing	-210.86 ± 345.23	85.54 ± 1.88	30.57 ± 24.5	78.0 ± 6.09	29.0 ± 16.98
SAC from scratch with attending mode	-533.72 ± 294.69	99.7 ± 0.52	21.76 ± 34.63	85.34 ± 18.91	13.13 ± 9.86
SAC from scratch with navigation mode	-523.45 ± 340.38	99.67 ± 0.58	20.45 ± 35.8	98.0 ± 3.47	12.88 ± 5.25
SAC with transfer learning	-509.45 ± 398.09	99.64 ± 0.56	20.9 ± 47.19	80.34 ± 23.03	16.27 ± 17.97

of changing policies from attending to navigation even when the robot is close to the target person. Since there is no smoothing mechanism over changing policies from the attending to navigation or vice versa, the robot tends to end up hitting the target person after striving to maintain its position on the attending goals. Accordingly, agents which are trained using full action smoothing show lower collision rate and shorter episode duration compared to agents which are trained using half action smoothing. This is due the fact that the robot is capable of smoothing actions generated from both navigation and attending policies but unable to generate responsive actions to avoid obstacles while its position is away from the target person.

The last three rows of Table 5.2 show the experiment results for agents which are trained from scratch and trained using transfer learning with SAC learning algorithm. Even though in the training process all agents seem to have similar poor performances, they show quite different results in the validation process. As listed in the last row, agents which are trained with transfer learning show the lowest collision rate and the longest episode duration among two other agents. The result indicates that, in general, the transfer learning strategy gives better results compared to training from scratch approach. In addition, agents which are trained using SAC from scratch with attending mode gives better performance than agents which are trained from scratch with the navigation mode. The result is also reasonable since we include the relative target person's orientation α_{tgt} in the state space as the input for the policy. As a result, the agent can generate better actions while attending the target person.

On the other hand, the validation results which show the effectiveness of our proposed method for the person-following task are depicted in Fig. 5.6. From the first

column of the figure, we can obviously see that the trained agent with our proposed method is capable of making the robot to move approaching the target person and also attending him at his left or at his right side. As shown in the second column of the figure, the robot is able to reach the attending goal positions after 15 seconds in average and also able to keep up its positions steadily after it reaches the correct attending goals until the end of an episode. In addition, we also show the plots of the selected policies by the agent in the last column of the figure. In general, we can see that the agent will select the navigation policy (policy 1) whenever the robot's position is far from the target person until it is able to reach the correct attending distance (1 meter). Subsequently, we can also see in general that the agent will select the right attending policy (policy 2) whenever the robot is at the right side of the target person and it will select the left attending policy (policy 0) whenever the robot is at the left side of the target person.

Moreover, the last column of Fig. 5.6 also shows that the application of the action smoothing strategy in our proposed method works well for the person-following robot task training problem. If we take a look into more details, the agent does not always completely choose policy 0 or policy 2 after the robot has reached the correct attending distance towards the target person. As shown in Fig. 5.6 (c) and Fig. 5.6 (f), although mainly the agent selects policy 2, it sometimes chooses policy 0 and at times selects policy 1. However, the robot still can maintain its position and does not end up hitting the target person when the agent switches the selected policy randomly. Accordingly, similar condition also occurs as shown in the third and the fourth rows of the figure. From Fig. 5.6 (i) and Fig. 5.6 (l), we can see that the agent does not produce smooth selected policy plots compared to plots from the previous rows. This is due to the fact that the robot is too close to the target person after 15 seconds as shown in Fig. 5.6 (h), and Fig. 5.6 (k). Therefore, the agent tries to correct the position of the robot by selecting other policies. Even though the selected policy changes frequently, the action smoothing strategy prevents the robot to hit the target person and makes the robot able to keep its position while attending the target person.

Additionally, we also validate the learned Double DQN policy inside a new environment in order to observe its generalization ability. Even though the robot was trained inside a different environment, as shown in Fig. 5.7, it is able to navigate approaching the target person in a completely new environment of a narrow corridor filled up with dense obstacles and a moving target person. Furthermore, from the first and the second row of Fig. 5.7, we can clearly see that the robot is able to attend the target person both at his left or at his right side once it reaches the waiting point. Moreover, we once again show that the proposed action smoothing can make the robot generates smooth and safe trajectories around the target person.

5.4 Conclusion on Multiple Policies Integration

In this chapter, we proposed a method based on deep reinforcement learning to train an agent capable of selecting the most suitable policy for the current state of the

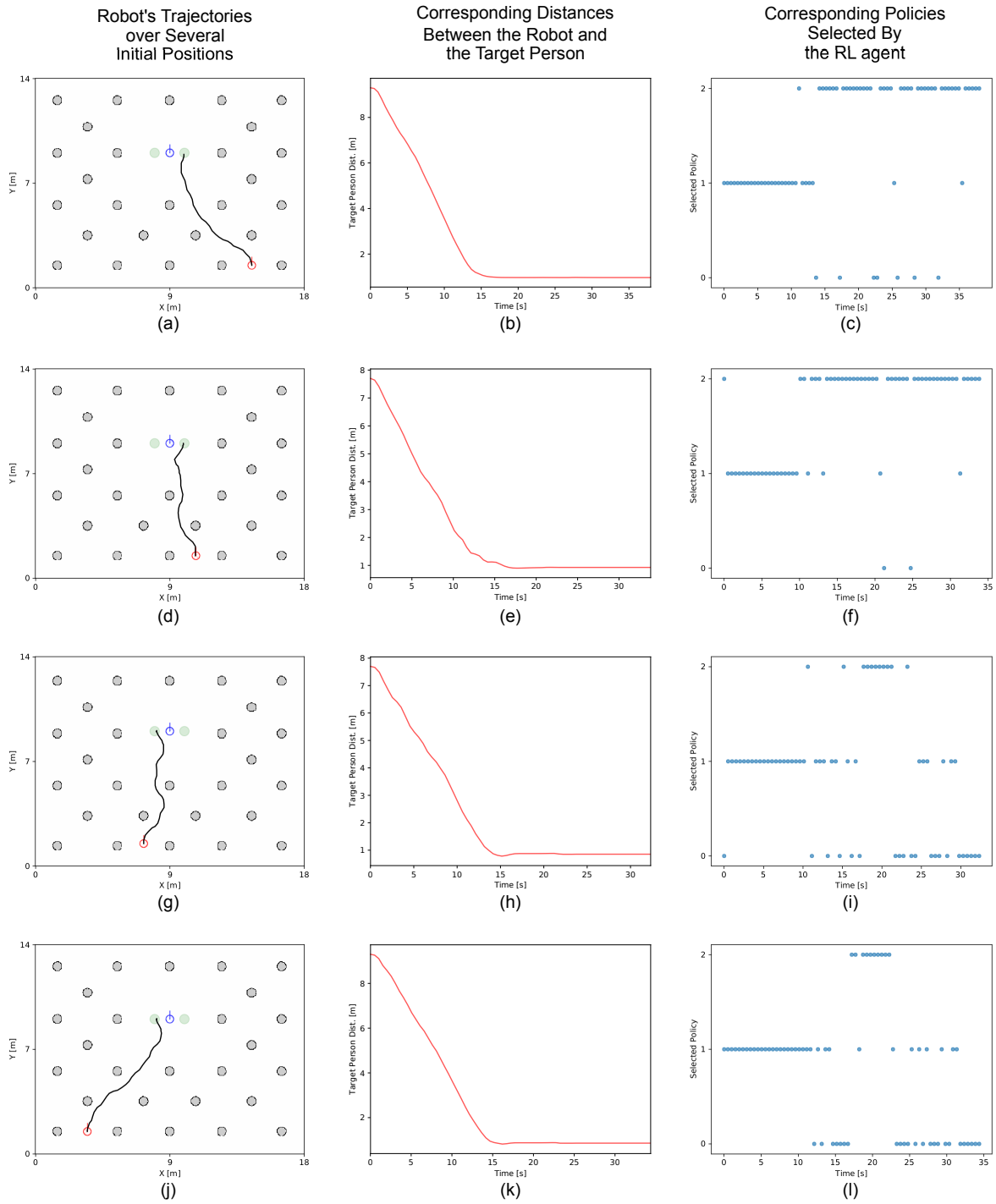


Figure 5.6: Validation results of the proposed method in the person-following task. First column depicts trajectories of the robot for several initial positions behind the target person. Second column depicts the corresponding distance of the target person from the robot. Third column depicts the corresponding policies which are selected by the agent which was trained using the proposed method. In the first column, blue circles represent the target person, red circles represent the initial positions of the robot, and green circles represent the left and the right attending goal positions toward the target person. In the last column, policy 0 represents the left attending policy, policy 1 represents the navigation policy, and policy 2 represents the right attending policy.

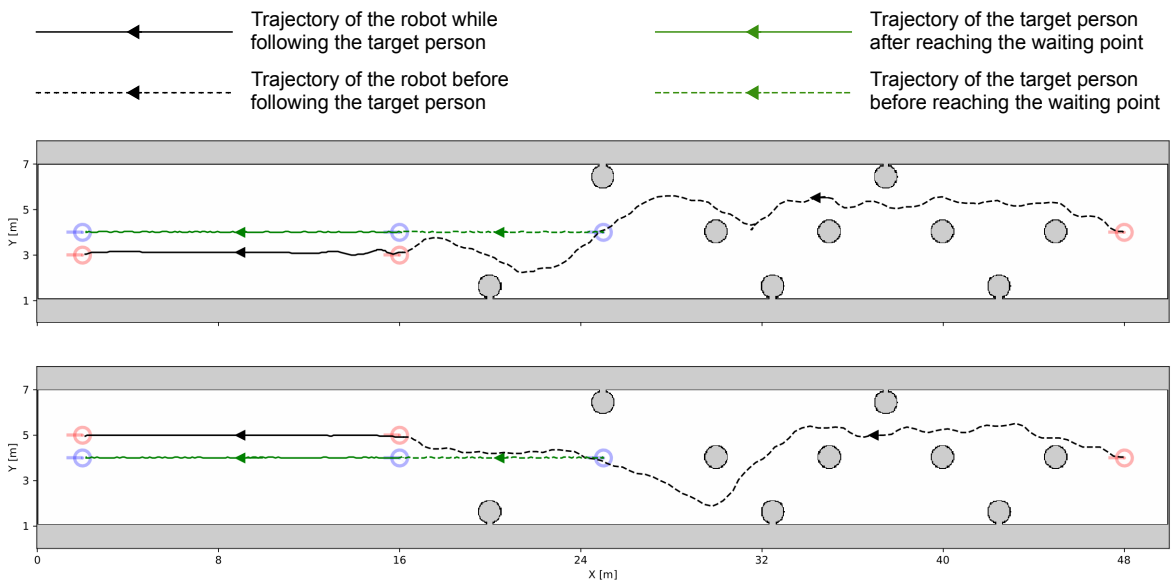


Figure 5.7: Plot of validation trajectories when the robot is following a moving target person. Initially, the target person's position is set at x = 25 m and the robot's position is set at x = 48 m. Subsequently, the target person is moving to the waiting point (x = 16 m). After the the robot is able to reach the target person, it starts attending the target person. First row depicts the trajectory when the robot attends the target person at his left side, whilst second row depicts the trajectory when the robot attends the target person at his right side.

environment given a set of previously trained optimal policies for the person-following robot training problem. Based on the results in this study, we conclude that the proposed method is capable of reusing previously trained optimal policies and integrating those policies to another new task. It also has been validated in a simulated person-following task environment which confirms that our method gives better performance compared to training from scratch and also training using transfer learning strategy. Moreover, we also conclude that the application of the action smoothing strategy in our proposed method is essential for the agent while integrating different optimal policies to generate final actions to be performed in the environment. Furthermore, the weight-scheduled action smoothing which is introduced in this paper is able to make the agent generate more smooth actions. Most importantly, the proposed method does not prevent exploration, which becomes one of crucial aspects to make the agent can learn appropriately during the training process.

Chapter 6

Policies Improvement Trials

In this chapter, we present our trials to improve the meta-policy and the attending policies in the person-following task. We focus on the potentials to improve the attending policies that can be obtained through utilizing symmetric nature in the attending task training. Moreover, we also consider the utilization of a dynamic environment for integrating all policies in the person-following task.

6.1 Utilization of Symmetric Nature in the Attending Task

Since the left and the right attending goals are set symmetrically around the target person, there is a possibility that we can convert the left attending policy to the right attending policy and vice versa. One way to do so is by inverting some data of the input and output of the trained attending policy network.

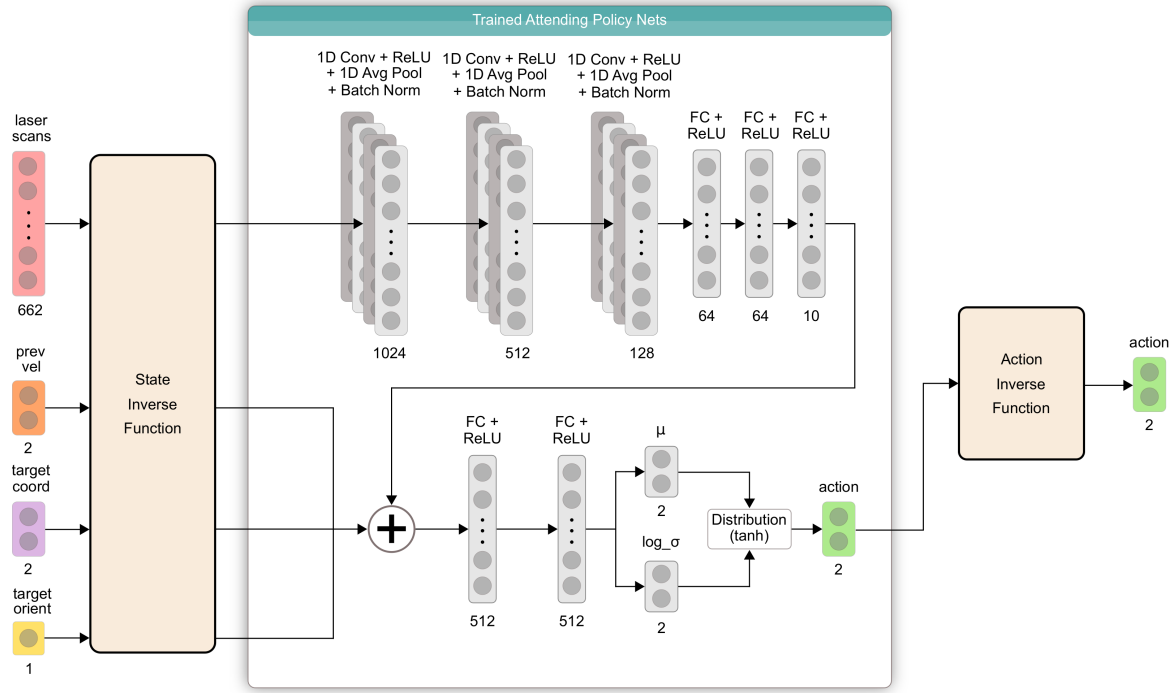
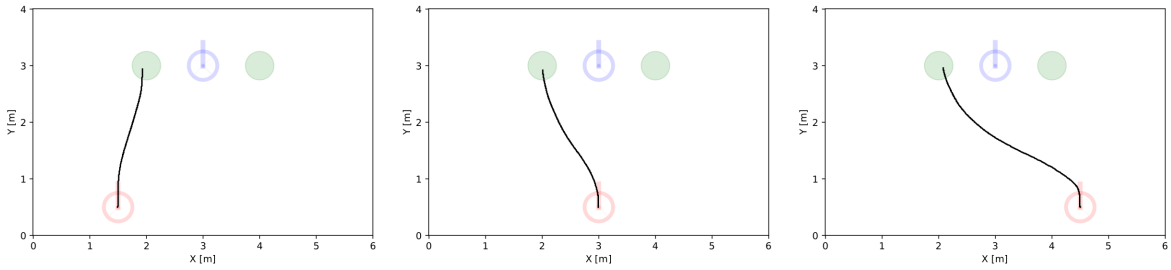
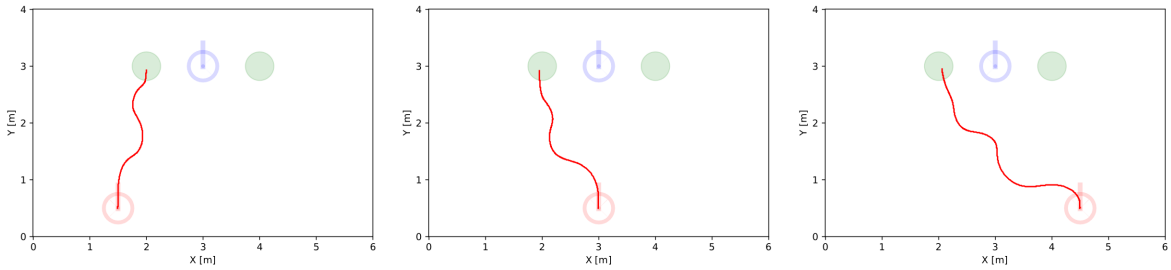


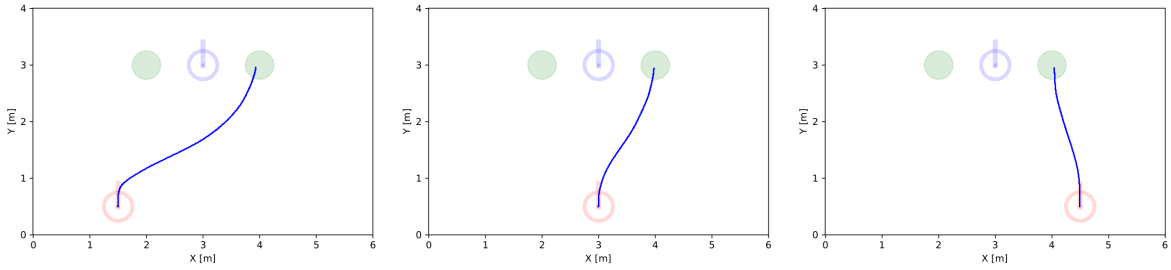
Figure 6.1: The proposed method to invert the left attending policy to the right attending policy and vice versa. We employ two inverse function for inverting the environment's state as the input and for inverting the robot's action and the output.



(a) Trajectories of the proposed method to invert the right attending policy



(b) Trajectories of inverting the target's pose and the robot's action

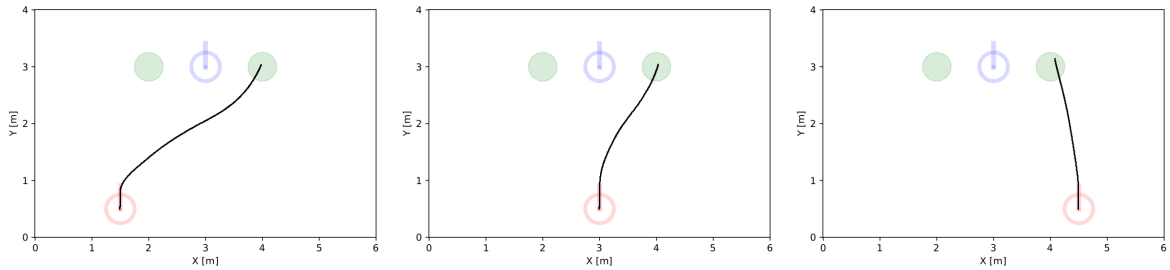


(c) Trajectories of the original right attending policy

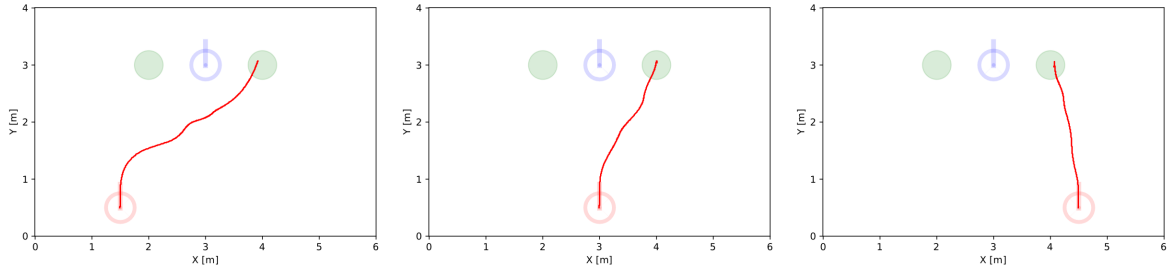
Figure 6.2: The trajectories of our first experiment for inverting the right attending policy to the left attending policy. Blue circles represent the target person, red circles represent the robot, and green circles represent the left and right attending goals.

6.1.1 Proposed Method to Invert the Attending Policies

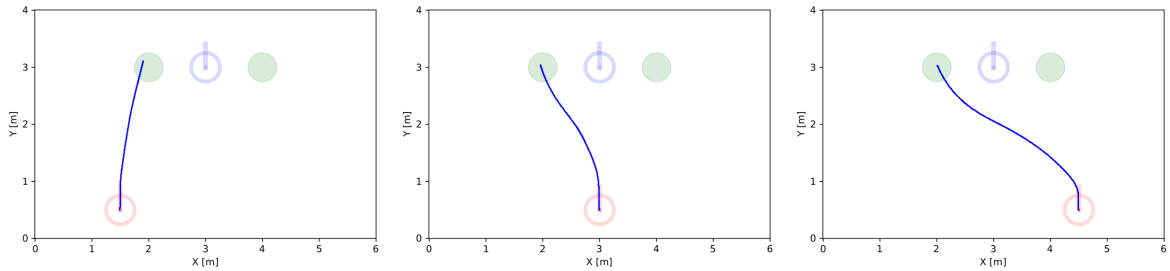
We show our proposed method to invert the attending polices in Fig. 6.1. In our proposed method, we include two inverse functions for both the input and output of the attending policies. For the input inverse function, we flip the laser scans, invert the y-coordinate of the relative target person's position and orientation and also invert the angular velocity of the previous action. Moreover, for the output inverse function, we invert the angular velocity of the robot's action to be performed in the environment.



(a) Trajectories of the proposed method to invert the left attending policy



(b) Trajectories of inverting the target's pose and the robot's action



(c) Trajectories of the original left attending policy

Figure 6.3: The trajectories of our second experiment for inverting the left attending policy to the right attending policy. Blue circles represent the target person, red circles represent the robot, and green circles represent the left and right attending goals.

6.1.2 Experiments on Inverting the Attending Policies

We conducted two experiments to invert the left and right attending policies. In the first experiment as shown in Fig. 6.2, we tried to invert the right attending policy to the left attending policy using our proposed method. As we can see from Fig. 6.2 (a), our proposed method is successfully able to invert the original right attending policy which is shown in Fig. 6.2 (c). Moreover, we also perform an experiment using the modification of our proposed method. As depicted in Fig. 6.2 (b), only inverting the target person's pose for the input of the attending task policy is not sufficient to invert the policy since it will generate oscillating attending trajectories. Subsequently,

we also show the result of our second experiment of inverting left attending policy to the right attending policy in Fig. 6.3. However, in this experiment, the result of inverting the target's pose and the robot's action of our proposed method gives less oscillating trajectories compared to the previous experiment as depicted in Fig. 6.3,

6.2 Policies Integration in a Dynamic Environment

In this section, experiments for integrating all policies for the person-following task are performed inside a dynamic environment. We think that this procedure is required so that the obtained meta-policy can become more robust and able to perform better obstacle avoidance while following the target person. We implement the dynamic environment inside a Gazebo simulator by loading several human models which are controlled using the Artificial Potential Field (APF) [82]. Three training scenarios are prepared, namely: a) policy integration training from scratch, b) policy integration training using transfer learning, and c) policy integration training using a hybrid strategy, which is the combination of all policies from the previous training and the APF method.

6.2.1 Dynamic Environment Implementation with APF

We show the design of the dynamic environment for the training procedure for integrating the polices for the person-following task in Fig. 6.4. In order to make the obtained policy becomes more robust, we prepare a simulated dynamic world inside the Gazebo simulator and then load several human models that can be moved dynamically. To control all those models, several Gazebo plugins are prepared for all human models so that a single human model can be controlled by publishing the given linear velocity v and the given angular velocity ω. Moreover, through the ROS plugins, a ROS topic for publishing the velocities for each model is also prepared as depicted in Fig. 6.5.

To simulate the movement of all human models, we use the APF method [82] so that they can navigate to some goals based on the attractive potential field which is defined in the following equation:

$$U_{attr}(x) = \frac{1}{2} k_p (x - x_d)^2, \qquad \text{(Eq. 6.1)}$$

where x denotes the current position of a human model, k_p denotes the position gain, and x_d denotes the goal position. Moreover, in order to make the human model able to avoid obstacles as well, the repulsive potential field is also computed using the following formula:

$$U_{rep}(x) = \begin{cases} \frac{1}{2}\eta \left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)^2, & \text{if } \rho \leq \rho_0 \\ 0, & \text{if } \rho > \rho_0, \end{cases} \qquad \text{(Eq. 6.2)}$$

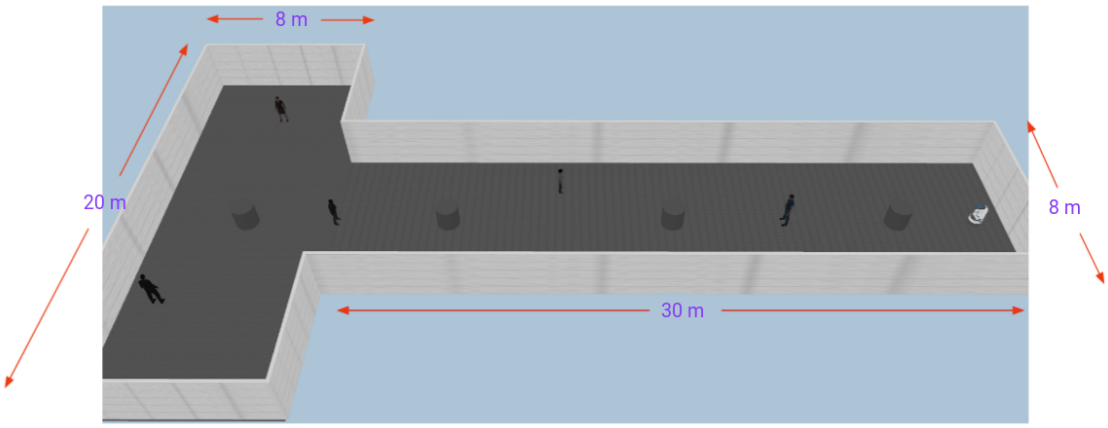


Figure 6.4: The design of the dynamic environment for the policies integration training for the person-following task. Along with static obstacles, we also load several human models which are controlled using the APF method.

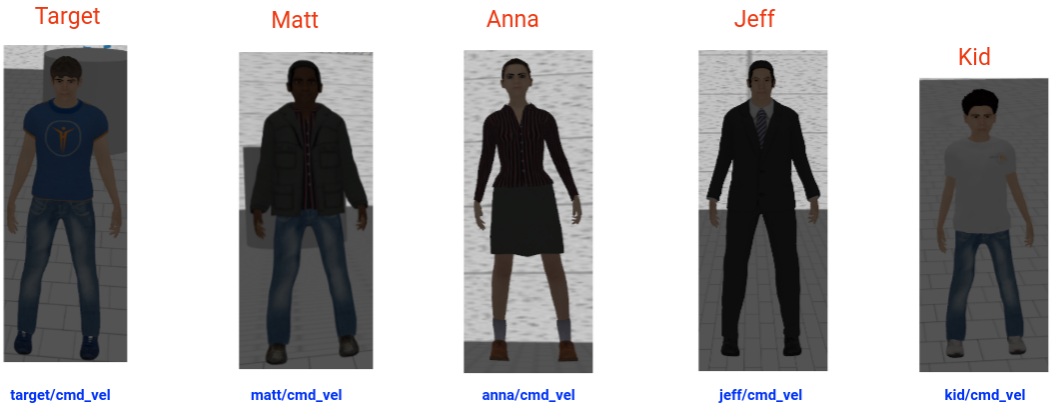


Figure 6.5: All the human models inside the dynamic environment can be controlled by publishing the linear velocity v and angular velocity ω through specific ROS topics.

where η denotes the constant gain, ρ denotes the shortest distance to the obstacle, and ρ_0 denotes the limit distance of the potential field influence. Finally, the final actions which are generated by the human model inside the simulated environment are computed based on the formula below

$$U_{apf}(x) = U_{attr}(x) + U_{rep}(x). \qquad \text{(Eq. 6.3)}$$

For each human model, for each time step t, we first generate nine intermediate goal candidates around it. Subsequently, the chosen one is the goal which has the minimum value of U_{apf} among others. Fig. 6.6 shows the illustration of an annotated local map of a human model inside the simulation which is used to generate the action.

Figure 6.6: The annotated local map of a human model named 'Jeff' for generating the current action using the APF method. Inside the local map, the human model is placed at the center of the map which is represented as a blue triangle. Moreover, the cyan line represents the direction towards the current goal (represented as a red rectangle inside a red circle) and a small red dot represents the intermediate goal which is calculated using the APF method.

6.2.2 Experiment Setup of the Policies Integration

We design two experiment scenarios inside the dynamic environment for training an RL agent for the person-following task. The followings are details of each experiment:

1. Training from scratch. In this experiment, policy integration of the left attending policy, the right attending policy and also the navigation policy for the person-following task is performed from scratch inside the dynamic environment.
2. Training using transfer learning strategy. In this scenario, at the beginning of the training episode inside the dynamic environment, the meta-policy of the person-following task from the static environment of the previous training which is described in the previous chapter is loaded to the RL agent.

For all experiment scenarios, we perform 3,000 episodes of 1,500 steps of training inside the simulator. We end the episode whenever the robot hits all obstacles and all human models. We also end the episode if the maximum step is reached. In addition, we perform three experiment runs for each scenario.

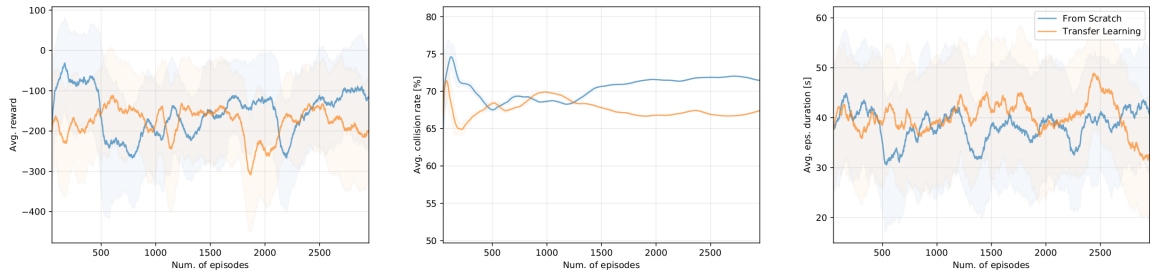


Figure 6.7: The plot of all training scenarios' performance over three trials in a dynamic environment. (a) Average reward. (b) Average collision rate. (c) Average episode duration.

Table 6.1: Summary of performances of all RL agents during the training process for the person-following task in a dynamic environment over three trials.

Experiment Scenario	Reward	Collision Rate (%)	Episode Duration (s)
Training from scratch	-155.12 ± 264.7	71.39 ± 6.92	38.35 ± 22.4
Training using transfer learning strategy	-179.2 ± 282.06	67.59 ± 3.87	40.07 ± 22.79

6.2.3 Results and Discussion of the Policies Integration

The plot of all RL agents performances over three trials is depicted in Fig. 6.7. Moreover, the summary of performance of all RL agent is listed in Table 6.1. As depicted in Fig. 6.7 (a) and Fig. 6.7 (c) the average reward and the average episode duration for both experiment scenarios are quite similar. However, Fig. 6.7 (b) clearly shows that applying the transfer learning strategy during the training process is definitely able to make the average collision rate lower. This means that the RL agent successfully transfers the knowledge of avoiding obstacles from the previous training inside the static environment to the new training inside the dynamic environment. Nevertheless, compared to the performances of the RL agents inside the static environment, experiment results inside the dynamic environment still give disappointing results. As listed in Table 6.1, we still get a very high value of collision rate even when the transfer learning strategy was applied during the training process.

We notice that the main reason why the performance of the RL agent in the dynamic environment drops significantly is because the robot tends to keep its current attending position when it is close to the target person. Our analysis is corroborated from the case scenario which is depicted in Fig. 6.8, especially in Fig. 6.8 (c) which shows the condition when there is another person around the target person. As shown in Fig. 6.8 (d), the robot is indeed tries to avoid to hitting another person while also strives to keep its position in the left attending goal.

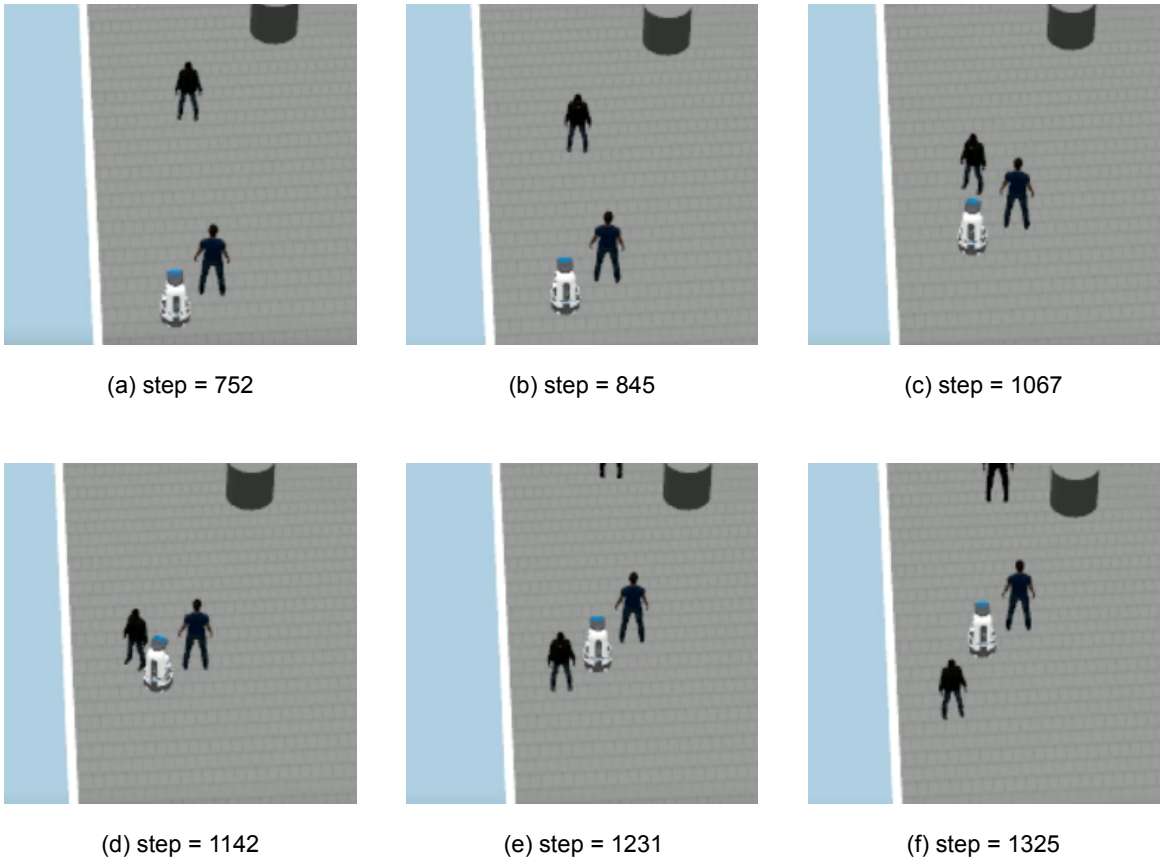


Figure 6.8: An episode when the robot attends the target person at the left side in the dynamic environment from step 752 through step 1325.

6.3 Conclusion on Policies Improvement Trials

From the experiments that we conduct in the first part of this chapter, we can conclude that our proposed method is able to take the advantage of utilizing the symmetric nature in the attending task to convert the attending policies easily. Moreover, we show that inverting all environment's state along with the robot's action is necessary for obtaining appropriate converted attending policies for the person-following task. By following the strategy to convert the attending task policies, we do not have to spend a lot amount of time for training the attending policies.

Furthermore, from the experiments that we conduct in the second part of the chapter, we can conclude that we still get poor performances of the person-following task policy under dynamic environment. Based on our analysis, the main reason for the substandard performance is that the robot is unable to switch the attending position easily when it is close to the target person. Therefore, for future work, we would like to propose another algorithm that can make the robot capable of switching the attending position efficiently.

Chapter 7

Conclusion and Future Works

7.1 Thesis Main Conclusion

In this thesis, we have described a method which is intended for obtaining an optimal policy for a robot to perform the person-following task through several simulated environments. Since we see that the person-following is a complex task for the robot, the training process is then divided hierarchically into several sub tasks for obtaining the optimal sub policy for each sub task, namely the navigation policy, the left attending policy, and the right attending policy. Subsequently, we follow the bottom up approach by integrating those optimal sub policies for obtaining the final meta policy for the person-following task.

For the navigation task training, we showed that our proposed novel framework is able to make the RL agent learn a specific policy which is able to produce fast but safe navigation actions for the robot. Inside our proposed navigation framework, we showed that our method which is called the state transition checking is able to guarantee that the training environment will always follow the Markov decision process appropriately. Moreover, we also showed that our method which is called the velocity increment scheduling is able to make the robot learn the fast-but-safe navigation strategy gradually during the training process.

For the attending task training, we showed that our proposed weight-scheduled action smoothing method is able to make the robot capable of generating safe and smooth actions while attending the target person at his left side or at his right side. In our proposed method, we showed that, by employing the curriculum learning strategy during the training process, the RL agent is able to explore the environment sufficiently while also learn to generate smooth and safe actions around the target person.

Finally, we also showed that our proposed method is able to integrate the previously learned navigation and attending policies into one comprehensive optimal meta policy for the person-following task. Furthermore, we also showed that the training results of our proposed method can outperform the training results obtained from training the RL agent for the person-following task from scratch or the training results obtained using the transfer learning strategy.

7.2 Future Works

7.2.1 Plan to Improve All Policies

We consider to propose another method so that the robot is able to switch easily between the left attending policy and the right attending policy when it is close to the target person. We are planning to propose another method to minimize the collision

rate inside the dynamic environment. Moreover, to obtain more robust policies, we consider to train all RL agent for each sub task in the person-following task inside dynamic environments. We also planning to propose another method so that the robot always try not to lose the target person. In addition, we also consider to combine our proposed method with other specific techniques such as the human sentiment analysis [99] so that the robot can better understand the conditions of the target person based on his biometric data and capable of generating more appropriate and comfortable actions toward him.

7.2.2 Plan to Deploy The Learned Policies to Real Robots

Deploying the policies to real robots is indeed becomes our main goal in the future. In order to do so, we are planning to include images obtained from camera which is attached to the robot. We are planning to apply computer vision methods for detecting the target person and estimating his relative coordinate and orientation from the robot. Moreover, we are planning to track the target person and handle the condition when he is occluded. Furthermore, to minimize the gap between the simulator to real world, we are planning to handle all noises related to the input of the policies including noise from sensors and the calculation of the target person's pose.

References

[1] T. Fong, I. Nourbakhsh, and K. Dautenhahn, "A survey of socially interactive robots," *Rob. Auton. Syst.*, vol. 42, no. 3-4, pp. 143–166, 2003.

[2] P. A. Lasota, T. Fong, and J. A. Shah, "A Survey of Methods for Safe Human-Robot Interaction," *Found. Trends Robot.*, vol. 5, no. 4, pp. 262–349, 2017.

[3] M. Ceccarelli, "Problems and Issues for Service Robots in New Applications," *Int. J. Soc. Robot.*, vol. 3, no. 3, pp. 299–312, 2011.

[4] M. J. Islam, J. Hong, and J. Sattar, "Person-following by autonomous robots: A categorical overview," *Int. J. Robot. Res.*, vol. 38, no. 14, pp. 1581–1618, 2019.

[5] S. Nishimura, H. Takemura, and H. Mizoguchi, "Development of Attachable Modules for Robotizing Daily Items - Person Following Shopping Cart Robot," in *Proc. IEEE Int. Conf. Robot. Biomim. (ROBIO)*, pp. 1506–1511, 2007.

[6] H. Masuzawa, J. Miura, and S. Oishi, "Development of a Mobile Robot for Harvest Support in Greenhouse Horticulture - Person Following and Mapping," in *Proc. IEEE/SICE Int. Symp. Syst. Integr. (SII)*, pp. 541–546, 2017.

[7] Piaggio-Fast-Forward. Person Following by Gita. URL `http://piaggiofastforward.com/`, 2017.

[8] J. Sales, J. V. Martí, R. Marín, E. Cervera, and P. J. Sanz, "CompaRob: The Shopping Cart Assistance Robot," *Int. J. Distrib. Sens. Netw.*, vol. 12, no. 2, 2016.

[9] S. S. Honig, T. Oron-Gilad, H. Zaichyk, V. Sarne-Fleischmann, S. Olatunji, and Y. Edan, "Toward socially aware person-following robots," *IEEE Trans. Cogn. Devel. Syst.*, vol. 10, no. 4, pp. 936–954, 2018.

[10] J. Miura, J. Satake, M. Chiba, Y. Ishikawa, K. Kitajima, and H. Masuzawa, "Development of a Person Following Robot and Its Experimental Evaluation," in *Proc. 11th Int. Conf. Intell. Auton. Syst. (IAS)*, pp. 89–98, 2010.

[11] K. Koide, J. Miura, and E. Menegatti, "Monocular person tracking and identification with online deep feature selection for person following robots," *Robot. Auton. Syst.*, vol. 124, p. 103348, 2020.

[12] S. Thrun and T. M. Mitchell, "Lifelong Robot Learning," *Rob. Auton. Syst.*, vol. 15, no. 1-2, pp. 25–46, 1995.

[13] J. H. Connell and S. Mahadevan, *Robot Learning*, vol. 233. Springer Science & Business Media, 2012.

[14] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[15] I. Goodfellow, Y. Bengio, and A. Courville, "Machine learning basics," *Deep Learning*, vol. 1, pp. 98–164, 2016.

[16] V. Gullapalli, J. A. Franklin, and H. Benbrahim, "Acquiring robot skills via reinforcement learning," *IEEE Control Syst. Mag.*, vol. 14, no. 1, pp. 13–24, 1994.

[17] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement Learning in Robotics: A Survey," *Int. J. Robot. Res*, vol. 32, no. 11, pp. 1238–1274, 2013.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: The MIT Press, second ed., 2018.

[19] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, *et al.*, "The limits and potentials of deep learning for robotics," *Int. J. Rob. Res.*, vol. 37, no. 4-5, pp. 405–420, 2018.

[20] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, 2017.

[21] S. S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: an overview," in *Proc. SAI Int. Syst. Conf.*, pp. 426–440, Springer, 2016.

[22] Z. Ding and H. Dong, "Challenges of Reinforcement Learning," in *Deep Reinforcement Learning* (H. Dong, Z. Ding, and S. Zhang, eds.), pp. 249–272, Springer, Singapore, 2020.

[23] Y. Kohari, J. Miura, and S. Oishi, "Generating Adaptive Attending Behaviors using User State Classification and Deep Reinforcement Learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots*

Syst. (IROS), pp. 548–555, 2018.
[24] L. Pang, Y. Zhang, S. Coleman, and H. Cao, "Efficient hybrid-supervised deep reinforcement learning for person following robot," *J. Intell. Robot. Syst.*, vol. 97, no. 2, pp. 299–312, 2020.
[25] S. Oishi, Y. Kohari, and J. Miura, "Toward a robotic attendant adaptively behaving according to human state," in *Proc. 25th IEEE Int. Symp. on Robot Hum. Interact. Commun. (RO-MAN)*, pp. 1038–1043, 2016.
[26] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An Introduction to Deep Reinforcement Learning," *Found. Trends Mach. Learn.*, vol. 11, no. 3-4, pp. 219–354, 2018.
[27] 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, pp. 529–533, Feb. 2015.
[28] 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, pp. 484–503, 2016.
[29] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1334–1373, 2016.
[30] D. Gandhi, L. Pinto, and A. Gupta, "Learning to fly by crashing," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, pp. 3948–3955, IEEE, 2017.
[31] B. S. B. Dewantara and J. Miura, "Generation of a socially aware behavior of a guide robot using reinforcement learning," in *Proc. Int. Electron.Symp. (IES)*, pp. 105–110, IEEE, 2016.
[32] R. Bellman, "A Markovian decision process," *J. Math. Mech.*, vol. 6, no. 5, pp. 679–684, 1957.
[33] A. Zhang, Y. Wu, and J. Pineau, "Natural environment benchmarks for reinforcement learning," *arXiv:1811.06032.*, 2018.
[34] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn. (ICML)*, pp. 41–48, 2009.
[35] G. Hacohen and D. Weinshall, "On The Power of Curriculum Learning in Training Deep Networks," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, pp. 2535–2544, 2019.
[36] S. Narvekar, "Curriculum Learning in Reinforcement Learning," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, pp. 5195–5196, 2017.
[37] S. Narvekar and P. Stone, "Learning Curriculum Policies for Reinforcement Learning," in *Proc. 18th Int. Conf. Auto. Agents MultiAgent Syst.*, pp. 25–33, 2019.
[38] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)* (Y. Bengio and Y. LeCun, eds.), ICLR, 2016.
[39] Z. Ren, D. Dong, H. Li, and C. Chen, "Self-paced prioritized curriculum learning with coverage penalty in deep reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2216–2226, 2018.
[40] S. Narvekar, J. Sinapov, and P. Stone, "Autonomous Task Sequencing for Customized Curriculum Design in Reinforcement Learning," in *Proc. 26 th Int. Joint Conf. Artif. Intell.*, pp. 2536–2542, 2017.
[41] M. Svetlik, M. Leonetti, J. Sinapov, R. Shah, N. Walker, and P. Stone, "Automatic Curriculum Graph Generation for Reinforcement Learning Agents," in *Proc. 31st AAAI Conf. Artif. Intell.*, pp. 2590–2596, 2017.
[42] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, 2017.
[43] L. Liu, D. Dugas, G. Cesari, R. Siegwart, and R. Dubé, "Robot Navigation in Crowded Environments Using Deep Reinforcement Learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, pp. 5671–5677, 2020.
[44] R. Guldenring, M. Görner, N. Hendrich, N. J. Jacobsen, and J. Zhang, "Learning Local Planners for Human-aware Navigation in Indoor Environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, pp. 6053–6060, 2020.
[45] C. K. Dewa and J. Miura, "Training a Robot to Attend a Person at Specific Locations using

Soft Actor-Critic under Simulated Environment," in *Proc. IEEE/SICE Int. Symp. Syst. Integr. (SII)*, pp. 837–838, 2021.
[46] N. Vithayathil Varghese and Q. H. Mahmoud, "A Survey of Multi-Task Deep Reinforcement Learning," *Electronics*, vol. 9, no. 9, 2020.
[47] Z. Yang, K. Merrick, H. Abbass, and L. Jin, "Multi-Task Deep Reinforcement Learning for Continuous Action Control," in *Proc. 26th Int. Joint Conf. Artif. Intell. (IJCAI)*, pp. 3301–3307, 2017.
[48] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, 2016.
[49] Y. W. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu, "Distral: Robust Multitask Reinforcement Learning," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst. (NeurIPS)*, pp. 4499–4509, 2017.
[50] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures," in *Proc. 35th Int. Conf. on Mach. Learn. (ICML)*, pp. 1407–1416, 2018.
[51] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt, "Multi-Task Deep Reinforcement Learning with PopArt," in *Proc. 33rd AAAI Conf. Artif. Intell.*, pp. 3796–3803, 2019.
[52] A. G. Barto and S. Mahadevan, "Recent Advances in Hierarchical Reinforcement Learning," *Discrete Event Dyn. S.*, vol. 13, no. 1, pp. 41–77, 2003.
[53] H. Sahni, S. Kumar, F. Tejani, and C. Isbell, "Learning to Compose Skills," *arXiv:1711.11289*, 2017.
[54] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning," *Artif. Intell.*, vol. 112, no. 1–2, pp. 181–211, 1999.
[55] K. Arulkumaran, N. Dilokthanakul, M. Shanahan, and A. A. Bharath, "Classifying Options for Deep Reinforcement Learning," *arXiv:1604.08153*, 2016.
[56] P.-L. Bacon, J. Harb, and D. Precup, "The Option-Critic Architecture," in *Proc. 31st AAAI Conf. Artif. Intell.*, pp. 1726–1734, 2017.
[57] P. Henderson, W. Chang, P. Bacon, D. Meger, J. Pineau, and D. Precup, "OptionGAN: Learning Joint Reward-Policy Options Using Generative Adversarial Inverse Reinforcement Learning," in *Proc. 32nd AAAI Conf. Artif. Intell.*, pp. 3199–3206, 2018.
[58] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta Learning Shared Hierarchies," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, 2018.
[59] V. Marivate and M. Littman, "An Ensemble of Linearly Combined Reinforcement-Learning Agents," in *Proc. 17th AAAI Conf. Late-Breaking Dev. Field Artif. Intell.*, pp. 77–79, 2013.
[60] S. Carta, A. Ferreira, A. S. Podda, D. R. Recupero, and A. Sanna, "Multi-DQN: An ensemble of Deep Q-learning agents for stock market forecasting," *Expert Syst. Appl.*, vol. 164, p. 113820, 2021.
[61] D. L. Elliott, K. C. Santosh, and C. W. Anderson, "Gradient boosting in crowd ensembles for Q-learning using weight sharing," *Int. J. Mach. Learn. Cybern.*, vol. 11, no. 10, pp. 2275–2287, 2020.
[62] H. Liu, C. Yu, H. Wu, Z. Duan, and G. Yan, "A new hybrid ensemble deep reinforcement learning model for wind speed short term forecasting," *Energy*, vol. 202, p. 117794, 2020.
[63] L. Lv, S. Zhang, D. Ding, and Y. Wang, "Path planning via an improved DQN-based learning policy," *IEEE Access*, vol. 7, pp. 67319–67330, 2019.
[64] X. Zhou, P. Wu, H. Zhang, W. Guo, and Y. Liu, "Learn to navigate: cooperative path planning for unmanned surface vehicles using deep reinforcement learning," *IEEE Access*, vol. 7, pp. 165262–165278, 2019.
[65] J. Choi, K. Park, M. Kim, and S. Seok, "Deep reinforcement learning of navigation in a complex and crowded environment with a limited field of view," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, pp. 5993–6000, 2019.
[66] A. G. Barto and T. G. Dietterich, *Handbook of Learning and Approximate Dynamic*

Programming, ch. Reinforcement learning and its relationship to supervised learning, pp. 47–63. Canada: Wiley-IEEE Press, 2004.

[67] C. Wang, J. Wang, X. Zhang, and X. Zhang, "Autonomous navigation of UAV in large-scale unknown complex environment with deep reinforcement learning," in *Proc. IEEE Global Conf. Signal Inf. Process. (GlobalSIP)*, pp. 858–862, IEEE, 2017.

[68] A. Ramezani Dooraki and D.-J. Lee, "An end-to-end deep reinforcement learning-based intelligent agent capable of autonomous exploration in unknown environments," *Sensors*, vol. 18, no. 10, p. 3575, 2018.

[69] J. Zeng, R. Ju, L. Qin, Y. Hu, Q. Yin, and C. Hu, "Navigation in Unknown Dynamic Environments Based on Deep Reinforcement Learning," *Sensors*, vol. 19, no. 18, p. 3837, 2019.

[70] P. Kormushev, S. Calinon, and D. G. Caldwell, "Reinforcement learning in robotics: Applications and real-world challenges," *Robotics*, vol. 2, no. 3, pp. 122–148, 2013.

[71] S. Ramstedt and C. Pal, "Real-Time Reinforcement Learning," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 3073–3082, Curran Associates, Inc., 2019.

[72] C. Tallec, L. Blier, and Y. Ollivier, "Making Deep Q-learning methods robust to time discretization," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, pp. 6096–6104, 2019.

[73] N. Deshpande and A. Spalanzani, "Deep Reinforcement Learning based Vehicle Navigation amongst pedestrians using a Grid-based state representation," in *Proc. IEEE Intell. Transp. Syst. Conf. (ITSC)*, pp. 2081–2086, 2019.

[74] P. Cai, X. Mei, L. Tai, Y. Sun, and M. Liu, "High-speed autonomous drifting with deep reinforcement learning," *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 1247–1254, 2020.

[75] L. Xie, S. Wang, A. Markham, and N. Trigoni, "Towards Monocular Vision based Obstacle Avoidance through Deep Reinforcement Learning," in *RSS 2017 Workshop on New Frontiers for Deep Learning in Robotics*, July 2017.

[76] F. Leiva, K. Lobos-Tsunekawa, and J. Ruiz-del Solar, "Collision avoidance for indoor service robots through multimodal deep reinforcement learning," in *RoboCup: RobotWorld Cup XXIII*, (Cham), pp. 140–153, Springer, 2019.

[77] C. K. Dewa and J. Miura, "A Framework for DRL Navigation with State Transition Checking and Velocity Increment Scheduling," *IEEE Access*, vol. 8, pp. 191826–191838, 2020.

[78] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, pp. 31–36, 2017.

[79] M. Pfeiffer, M. Schaeuble, J. I. 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 *Proc. IEEE Int. Conf. Robot.Autom. (ICRA)*, pp. 1527–1533, 2017.

[80] M. Pfeiffer, S. Shukla, M. Turchetta, C. Cadena, A. Krause, R. Siegwart, and J. Nieto, "Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 4423–4430, 2018.

[81] C. Sampedro, H. Bavle, A. Rodriguez-Ramos, P. de la Puente, and P. Campoy, "Laser-Based Reactive Navigation for Multirotor Aerial Robots using Deep Reinforcement Learning," in *Proc. IEEE/RSJ Int. Conf. Intell.Robots Syst. (IROS)*, pp. 1024–1031, 2018.

[82] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *Int. J. Robot. Res.*, vol. 5, pp. 90–98, Apr. 1986.

[83] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, pp. 1861–1870, 2018.

[84] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, pp. 1587–1596, 2018.

[85] H. van Hasselt, "Double Q-learning," in *Proc. 24th Int. Conf. Neural Inf. Process. Syst. (NeurIPS)*, pp. 2613–2621, 2010.

[86] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.

[87] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intell.Robots Syst. (IROS)*, vol. 3, pp. 2149–2154,

2004.
[88] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich, "Fetch and freight: Standard platforms for service robot applications," in *Proc. Workshop Autonomous Mobile Service Robots*, 2016.
[89] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *Proc. ICRA Workshop Open Source Softw.*, 2009.
[90] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, 2015.
[91] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.(NeurIPS)*, pp. 8026–8037, 2019.
[92] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, "Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo," *arXiv preprint arXiv:1608.05742*, 2016.
[93] A. Wasala, D. Byrne, P. Miesbauer, J. O'Hanlon, P. Heraty, and P. Barry, "Trajectory based lateral control: A Reinforcement Learning case study," *Eng. Appl. Artif. Intell.*, vol. 94, p. 103799, 2020.
[94] C. K. Dewa and J. Miura, "Integrating Multiple Policies for Person-Following Robot Training Using Deep Reinforcement Learning," *IEEE Access*, vol. 9, pp. 75526–75541, 2021.
[95] M. M. Botvinick, Y. Niv, and A. G. Barto, "Hierarchically organized behavior and its neural foundations: a reinforcement learning perspective," *Cognition*, vol. 113, no. 3, pp. 262–280, 2009.
[96] B. Hengst, "Hierarchical Approaches," in *Reinforcement Learning*, pp. 293–323, Springer, 2012.
[97] M. M. Botvinick, "Hierarchical reinforcement learning and decision making," *Curr. Opin. Neurobiol.*, vol. 22, no. 6, pp. 956–962, 2012.
[98] H. v. Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, pp. 2094–2100, 2016.
[99] M. N. Y. Ali, M. G. Sarowar, M. L. Rahman, J. Chaki, N. Dey, and J. M. R. S. Tavares, "Adam deep learning with SOM for human sentiment classification," *Int. J. Ambient Comput. Intell.*, vol. 10, no. 3, pp. 92–116, 2019.

Acknowledgments

First of all I would like to express my sincere thanks to Professor Jun Miura who has provided the comprehensive support for my study as the supervisor of my doctoral course. While interacting with him, he kept guiding me to improve my skills to become a professional researcher in the field of robotics. He taught me how to think comprehensively to analyze problems and guided me to propose systematic and reasonable solutions to solve them appropriately. He even taught me to think critically and also gave me valuable opportunities to become a reviewer for a notable international conference and a reputable journal. Moreover, he also gave me a great opportunity to hone my skills through working for one of great and prestigious companies in Japan during my internship at Life Creation Center, Honda R&D Co., Ltd. Other than that, he gave me the permission to perform Hajj during my study. Thank you so much, *sensei*.

I am very grateful to my colleagues in the Active Intelligent System Laboratory, for making a good academic environment. I would like to thank Mr. Yubao Liu, Ms. Liliana Villamar Gomez, Ms. Ufaq Rehman, Mr. Hoai Luu Duc, and Mr. Ziyad Tareq Nouri for becoming my close friends. Many thanks for Mr. Shigemichi Matsuzaki who always tried to help anyone in the lab and Mr. Hiroaki Masuzawa who helped me occasionally. I would like to thank Mr. Kazuki Mano for the fruitful discussion since we have a similar research topic. Many thanks to Mr. Masataka Inouchi and Mr. Yasunori Kawamata for becoming the supporters for my first year at Toyohashi University of Technology.

I wish to thank the Japanese Ministry of Education, Culture, Sports, Science and Technology (MEXT) that has provided the generous financial support for my school and living expenses through the Monbukagakusho Scholarship. Without which, I could not complete my doctoral program. I also would like to thank Mr. Atsuki Osanai and Mr. Yasuhiro Taniguchi for becoming my supervisors at Honda R&D. In addition, I would like to thank Professor Mineo Ikematsu and Mr. Hirotsugu Kamahara for helping me improve my teaching skills through the Global Rotation Program. Moreover, I would like to express my sincere thanks to Professor Yoshiteru Ishida and Professor Shigeru Kuriyama for examining and judging my doctoral dissertation.

Lastly, I wish to thank my wife Afiahayati, my son Muhammad Alfatih Fiandra, and my parents who always gave me continuous support and help during my study.

List of Publications

Journal Articles

1. Chandra Kusuma Dewa and Jun Miura, "Integrating Multiple Policies for Person-Following Robot Training Using Deep Reinforcement Learning," in *IEEE Access*, vol. 9, pp. 75526-75541, 2021, doi: 10.1109/ACCESS.2021.3082136.
2. Chandra Kusuma Dewa and Jun Miura, "A Framework for DRL Navigation With State Transition Checking and Velocity Increment Scheduling," in *IEEE Access*, vol. 8, pp. 191826-191838, 2020, doi: 10.1109/ACCESS.2020.3033016.

Conference Paper

1. Chandra Kusuma Dewa and Jun Miura, "Training a Robot to Attend a Person at Specific Locations using Soft Actor-Critic under Simulated Environment," *2021 IEEE/SICE International Symposium on System Integration (SII)*, 2021, pp. 837-838, doi: 10.1109/IEEE CONF49454.2021.9382716.