Addressing Multi-Objective and domain adaptation Challenges in Reinforcement Learning through Case Studies in Multi-Agent Navigation and Visual Servoing

by

Salar Asayesh Ghalehseyf

M.Sc.,Isfahan University of Technology, 2013
B.Sc., Isfahan University of Technology, 2009

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Engineering Science
Faculty of Applied Sciences

© Salar Asayesh Ghalehseyf 2023
SIMON FRASER UNIVERSITY
Summer 2023

Declaration of Committee

Name:	**Salar Asayesh Ghalehseyf**
Degree:	**Doctor of Philosophy (Engineering Science)**
Title:	**Addressing Multi-Objective and domain adaptation Challenges in Reinforcement Learning through Case Studies in Multi-Agent Navigation and Visual Servoing**
Committee:	**Chair:** Michael Hegedus Lecturer, Engineering Science

Abstract

Reinforcement learning (RL) has gained a lot of attention in recent years due to its potential to solve complex control problems. However, RL faces various challenges in multi-agent settings and high-dimensional action and observation spaces. This thesis addresses some of these challenges through two case studies: Multi-Agent Reinforcement Learning and Robotic Arm Manipulation.

The first case study focuses on multi-agent navigation and proposes a novel class of RL-based controllers called least-restrictive controllers for multi-agent collision avoidance problems. This study aims to implement a high-level safe RL policy that provides safe navigation for multi-agent navigation in a shared environment with or without static obstacles. The proposed policy works in different tasks containing a different number of agents and different task objectives.

The second case study proposes a novel visual servoing (VS) algorithm using sequential stochastic latent actor-critic and reinforcement learning. This study aims to overcome domain adaptation and control challenges in high-dimensional action and observation spaces. The proposed algorithm can adapt to a real robot through only single-shot transfer learning in representation learning parts.

Keywords: Reinforcement Learning; Optimal Control; Multi-agent navigation; Visual Serving; Sim2Real Transfer

Dedication

To my parents (Mohammed and Sheida), wife (Soheila), daughter (Sorme), sister (Sara) and my soon-to-arrive son.

Acknowledgements

I would like to express my heartfelt gratitude to all those who have supported me throughout my journey toward completing this thesis. This achievement would not have been possible without the unwavering support and encouragement of my supervisors, family, friends, and colleagues.

Undertaking a PhD is a daunting and challenging path, and it is not one that can be successfully navigated alone. It requires a level of dedication, hard work, and perseverance that can be extremely difficult to maintain over the years. However, with the right support network, the road becomes easier to travel.

First and foremost, I would like to thank my supervisors, Dr. Kamal Gupta, Dr. Mehran Mehrandezh and Dr.Mo Chen, for their guidance, expertise, and unwavering support throughout the entire research process. Their commitment to my success, their patience, and their encouragement have been invaluable. They have provided me with the tools and resources necessary to conduct my research and have always been there to help me overcome any obstacles.

I would also like to express my gratitude to my family for their constant support, love, and encouragement. They have been a source of inspiration, motivation, and strength throughout my academic journey, and I cannot thank them enough. Their unwavering support has helped me to persevere through the most challenging times and has made this accomplishment possible.

Finally, I would like to express my gratitude to my friends and colleagues who have been there for me throughout this journey. Their support, encouragement, and companionship have made the process much more enjoyable and have helped me to maintain my sanity during this period.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recent advancements in deep reinforcement learning (deep RL) methods have shown quantum success in solving various challenging tasks from raw input data [3]. The challenging tasks include video games [76, 100], continuous control [65, 94], and manipulation tasks from high-dimensional image observations [63, 32]. However, most current works rely on specific domains and simulations with a series of assumptions that are hard to satisfy in real settings [26]. Therefore, deploying the RL policies to an industrial application is still a big challenge in pioneer industries and scientific communities. My research motivation is to explore how we can implement RL-based controllers[1] in real robots. Since several challenges contribute to the implementation of RL in real robotic applications, I will try to explore a part of them through two case studies.

1.1 Challenges In Robotics RL

There are several challenges in using Rl in real robotic systems. In this thesis, I will focus on two of them and try to suggest solutions using case studies.

Domain Adaptation

One way to implement RL in a real robot is to perform the learning procedure on the physical system directly; however, it is slow and expensive. In addition, due to safety rules and physical constraints, the RL policy cannot perform aggressive exploration during training. Therefore, the RL learning algorithm should be highly sample efficient. One remedy could be using samples previously gathered through running the physical system and learning from offline data. Offline reinforcement learning [62] tries to address the problem of learning from previously collected data and assumes the RL policy can no longer execute during training before the final deployment. The offline RL suffers from the Out of Distribution (OOD) challenge. The policy may fail in OOD if it visits the part of the

[1]In this thesis, the terms 'policy' and 'controller' are used interchangeably, and both refer to the decision-making agent.

state-space distribution that isn't directly inside the training data distribution. Another big challenge in direct learning or adaptation in real systems is safety constraints [105]. Since any safety violation in safety-critical systems can be costly and catastrophic, the RL policy optimization should consider the safety constraints. One remedy is to perform the training in simulation and then deploy the trained policy in real robots. This problem is usually known as a domain adaptation in the RL community. In domain adaptation, the agent learns from available data in the source domain (e.g. simulation). Then it hopes to generalize the trained policy in the target domain (e.g. real-world) [43]. Sadly, this method also creates several difficulties that prevent a robust adaptation. One major issue in the domain adaptation problem is that the real systems are partially observable. For instance, the sensors, noises, and actuator delays may not be directly measured or estimated. In addition, the source and target domain also may have some differences that are not obvious. Due to non-stationary and partial observability issues, the RL problem should use raw high-dimensional sensory observation; however, it increases the problem's complexity. Another issue is that slight changes to the reward/cost function in different domains can cause significant failure and demand re-training on the new domain.

Multi-Objective Learning

Most RL methods intend to solve one objective by engineering specific reward functions for that objective; however, actual word problems deal with multiple objective tasks. For example, an autonomous mobile robot must trade off between tasks, Energy consumption and safety, and in human-robot interaction, a manipulator first needs to perform a grasping task. Then, it must generate safe collision-free path planning and a robust object handover. This problem is often called Multi-Objective RL (MORL) [67]. Compared to single-objective RL, the MORL needs to simultaneously optimize a policy for more than one objective [67].

One approach to tackle these problems is to cast the multiple-objective into a single reward scalar and solve it using the standard reinforcement learning approach [79, 89]. However, the challenge arises when objectives are not entirely aligned with each other. For example, the objective may not be in the same unit (energy vs speed); therefore, it needs careful tuning and normalization between objectives. Moreover, objectives may conflict with each other, and some may have a sparse reward. In this case, the policy, at some points, may only be able to optimize one behaviour or finds a trade-off between objectives [114].

One alternate approach is to use multiple policies (one for each objective) [114, 88]. However, one of the significant challenges in this method is the learning procedure itself. Since the RL objective optimizes the behaviour through the entire trajectory, learning with multiple policies can cause OOD in the buffer, making the learning procedure difficult. In addition, a supervisor algorithm needs to decide which approach should be executed in each specific time step. The latter challenge is often addressed in hierarchical reinforcement learning. Hierarchical reinforcement learning is widely used in two objectives RL problems to decompose objectives into high-frequency and low-frequency controllers. This method is suitable for situations that demand low-frequency controllers for complicated locomotion tasks and a high-frequency control that provides guidance through a longer horizon [41].

Moreover, once the environment and tasks related to a trained policy change, the whole training procedure should be repeated. For example, a mobile robot may need to execute different controllers for different tasks. Learning each control strategy is time-consuming; it may also be challenging to reach the level of accuracy that classical control approaches can offer.

1.2 Case Studies

Multi-Agent Collision Avoidance

In the first part of the thesis, I focused on the problem of navigation and collision avoidance for multiple mobile robots that are working in a shared environment. In this problem, I assume each agent has its own objective (and default controller) responsible for task completion or reaching goals. I also assume that the default controllers don't have any avoidance capabilities in a multi-agent setting. The target is to train a single safe RL policy that can adapt all the objectives (i.e., default controllers) and provides safe navigation for all agents in the presence of static obstacles. This is a challenging problem because objectives vary from one agent to other agents. In addition, the default policy might be a human operator in the loop, making it even impossible to learn the correct behaviour. There are also more challenges in implementing the trained policy in a real setting. First, multi-agent RL training on real systems is slow and expensive. Due to the sample efficiency of current RL approaches, it requires multi-robots to work over the clock to perform data collection and policy rollout. Second, a practical algorithm has to use raw sensory observation, and each agent needs to act independently. Lastly, it also faces domain adaptation issues because there might be differences between simulation and real robots, such as sensor noises and actuator delays that are hard to measure and simulate in the source domain. Please note that in Chapter 3 and Chapter 4, I made the assumption that agents are always moving with their maximum linear velocity using a fixed-wing aircraft model. However, in Chapter 5, I relaxed this assumption and considered additional maneuvers for the agents, including stopping, slowing down, and even backing maneuvers.

Visual navigation of Robotic Manipulator

Visual Servoing (VS), or controlling a robotic manipulator using visual data, is an old classical problem in active vision. Over the last two decades, several approaches developed over time to solve this problem. As I will discuss in Chapter. 6, the classical approaches have some limitations, namely a small convergence domain and needing servoing in an environment with dense image features. Recent advancements in computer vision using Deep learning approaches caused researchers to revisit this problem. Some recent DL-based approaches try to mitigate classical approaches' limitations, such as light variation, partially occluding and servoing in featureless environments. However, they are prone to a lack of generalizability and require the network to be re-trained on a small set of images. In this part of the research, I attempted to deploy reinforcement learning (RL) to address this problem in a scalable setting. In addition, Learning VS for real robots raises a lot of additional challenges, such as learning on high dimensional image data, representation learning bottleneck, and partial observability of states.

1.3 Thesis Overview

The work presented in this thesis tries to explore the mentioned RL challenges through two main case studies: Multi-Agent Reinforcement Learning and Robotic Arm Manipulation. The first case study makes the following main contributions:

- Proposing a novel learning-based Least-restrictive (LR) controller that enhances the existing controllers by providing an additional safety layer.
- Introducing a novel reward framework for learning least-restrictive safety behaviour.
- Developing a learning framework capable of achieving zero-shot sim2real transfer.

The second case study's primary contributions include the following:

- Proposing a novel learning-based Vision System (VS) algorithm that operates on pixel data, allowing scalability to a wide range of scenes and objects.
- Achieving robust domain adaptation by performing a single-shot fine-tuning on the representation learning part.

Please refer to each chapter for the detailed list of contributions. This thesis lies in the intersection of RL, optimal control and robotic applications. The thesis is organized as follows:

- **Chapter 2** We will review the fundamental concepts of optimal control (Hamilton-Jacobi reachability), RL and Variation Auto Encoders (VAE) that are extensively used in the thesis.
- **Chapter 3** We present a new class of RL-based controllers called least-restrictive controllers for multi-agent collision avoidance problems to address the multi-objective challenges in reinforcement learning. This study aims to implement a high-level safe RL policy that provides safe navigation for multi-agent navigation in a shared environment with or without static obstacles. We assume each agent in a multi-agent setting has its objective, and they use the same trained RL policy for safe interaction with other dynamic agents and static obstacles. We want the RL policy to work in different tasks containing a different number of agents and different task objectives.

Agents' objectives can vary from different goal-reaching controllers or even human-operated motion. The main contribution of this chapter is proposing a state-based version of our LR policy. In addition, we propose a novel reward function based on the HJ-reachability theory to train our safety policy with a rich notion of safety. This chapter is reproduced with permission from Springer Nature from Salar Asayesh, Mo Chen, Mehran Mehrandezh, and Kamal Gupta (2023). Least-Restrictive Multi-agent Collision Avoidance via Deep Meta Reinforcement Learning and Optimal Control. In: et al. Robot Intelligence Technology and Applications 7. RiTA 2022. Lecture Notes in Networks and Systems, vol 642. Springer, Cham. https://doi.org/10.1007/978-3-031-26889-2-19. [4]. It was nominated for the best paper award.

- **Chapter 4** The main contribution of this chapter is that we addressed the limitations of our proposed method in Chapter 3 by directly using raw Lidar observation data and also considering static obstacles. We introduced novel reward functions based on HJ reachability and local cost maps, making our algorithm capable of learning multi-agent navigation in various scenarios and accommodating a variable number of agents. This chapter is based on ©2021 IEEE. reprinted, with permission, from [Salar Asayesh, Mo Chen, Mehran Mehrandezh, and Kamal Gupta (2021). Toward Observation Based Least Restrictive Collision Avoidance Using Deep Meta Reinforcement Learning. IEEE Robotics and Automation Letters (IEEE RAL 2021)] [3].

- **Chapter 5** In this chapter, we addressed another limitation of previous approaches by introducing the least-restrictive control policy using continuous control. Continuous control provides more robust maneuvers in obstacle/other agents avoidance by stopping, reducing or reversing the speed. To mitigate the complexity of the reward function for this controller, we proposed a hybrid approach using Generative Adversarial Imitation Learning (GAIL). GAIL is an inverse reinforcement learning approach that provides a reward signal for mimicking an expert policy. We re-purposed an existing classical policy to provide expert demonstrations and fed the GAIL with the pair of Lidar observations/actions. We also add a naive task reward to help the learning procedure.

- **Chapter 6** Next, we present a novel visual servoing (VS) algorithm using sequential stochastic latent actor-critic and reinforcement learning. This case study aims to overcome domain adaptation and control challenges in high-dimensional action and observation spaces. This chapter proposes an RL-based method for VS tasks by decomposing the task into representation learning and manipulation learning. We showed that by using this decomposition and domain randomization in training using photo-realistic images, the algorithm could adapt to a real robot through only single-shot transfer learning and manipulation parts. This work was submitted as: Salar Asayesh, Hossein Sheikhi Darani, Mo Chen, Mehran Mehrandezh and Kamal Gupta, VSLS:

visual servoing in sequential latent space. IEEE International Conference Conference on Intelligent Robots and Systems (IROS 2023)

- **Chapter 7** We will conclude the work and discuss future works.

Chapter 2

Background

This chapter aims to establish the necessary mathematical foundations and concepts for integrating learning and control. It encompasses crucial principles in reinforcement learning, introduces the Hamilton-Jacobi reachability theory, and variational autoencoder and establishes the notation that will be utilized in the upcoming chapters.

2.1 Optimal Control

At its core, optimal control theory focuses on the study of controlling a system to achieve desired behaviour in an optimal manner. For example, consider the task of driving a car to a particular destination while minimizing fuel consumption and travel time and ensuring safety compliance. To mathematically define optimal control, we begin by defining the system of interest as a dynamic system. Subsequently, we formulate the optimal control problem, which allows us to derive concepts related to Hamilton-Jacobi reachability.

2.1.1 System Dynamics

Let's define $x \in \mathbb{R}^n$ as the system state. The state is a vector with n dimensions that encompasses all the parameters necessary to monitor the system's evolution over time. The system's states evolve according to the following ordinary differential equation (ODE)

$$\begin{gathered} \dot{x} = f(x(t), u(t), d(t)), \\ t \in (-\infty, 0], u(t) \in \mathcal{U}, d(t) \in \mathcal{D}. \end{gathered} \tag{2.1}$$

In the given system, the variable t represents time, while $u(t)$ and $d(t)$ represent the control and disturbance inputs, respectively. The control function $u(\cdot)$ and the disturbance function $d(\cdot)$ are assumed to be drawn from compact sets: $u(\cdot) \in \mathcal{U} \subseteq \mathbb{R}^{n_u}$ and $d(\cdot) \in \mathcal{D} \subseteq \mathbb{R}^{n_d}$ [13]. This assumption is often true since the amount of control we can exert over a system is typically limited. Furthermore, the disturbance is also subject to limitations. However, it can become problematic if the limits are excessively large, as the disturbance can significantly impact the system. Conversely, if the limits are excessively small, they may not accurately represent the real disturbances encountered in practice [42]. For a fixed u and d, the system dynamics $f : \mathbb{R}^n \times \mathcal{U} \times \mathcal{D} \to \mathbb{R}^n$ are assumed to be uniformly continuous, bounded, and Lipschitz continuous in t. Hence, for a given $u(\cdot) \in \mathcal{U}$ and $d(\cdot) \in \mathcal{D}$, there exists a unique solution trajectory for solving the system dynamics described in Equation (2.1) [18]. This solution trajectory is denoted as $\xi(\cdot)$ [13]

$$\zeta(x; \tau; t; u(\cdot); d(\cdot)) : [\tau, 0] \to \mathbb{R}^n. \tag{2.2}$$

Which starts from state s at time τ and evolves under control $u(\cdot)$ and disturbance $d(\cdot)$ across $[\tau, 0]$. The trajectory ζ satisfies (2.1) an initial condition:

$$\begin{aligned} &\frac{d}{dt}\zeta(t; x, \tau, u(\cdot), d(\cdot)) = f\Big(\zeta\Big(t; x, \tau, u(\cdot), d(\cdot)\Big), u(t), d(t)\Big) \\ &\zeta(\tau; x, \tau, u(\cdot), d(\cdot)) = x \end{aligned} \tag{2.3}$$

2.1.2 Optimal Control Formulation

Mathematically, optimal control means minimizing some cost functions while taking into account the system's dynamics, its initial state, and the problem's constraints. The cost function is defined as [42]:

$$J(x, t, u(\cdot)) = \int_t^0 c(x(\tau), u(\tau))d\tau + l(x(0)) \tag{2.4}$$

Where $c(x_t, u(t))$ is a continuous-time running cost, and $l(x(0))$ is a terminal cost computed at the last time step.

The goal is to use the control signal optimally in order to minimize the total cost. This leads to the value function, written as:

$$\begin{aligned} V(x(t), t) = \inf_{u(\cdot)} &J(x, t, u(\cdot)) \\ \text{s.t} \quad &\dot{x(t)} = f(x(t), u(t)), \quad \forall t \in [\tau, 0] \\ &u(t) \in \mathcal{U} \end{aligned} \tag{2.5}$$

The value function indicates the minimum amount of cost at each state under optimal control. The optimal control problem (2.5) can be solved using two principal methods: dynamic programming and calculus of variations. "Calculus of variations" expands the objective of the optimization problem by bringing the constraints into the optimization problem using the Lagrangian method. Next, it deploys the optimization tools to solve the unconstrained problem by finding the stationary point. Note that the "calculus of variations" doesn't provide a global solution unless the optimization problem is convex with a zero duality gap.

On the other hand, "Dynamic Programming" can mitigate the local optima problem at the cost of higher computational complexity. The main idea behind dynamic programming is to think backwards in time. This recursive update equation is called the Bellman equation (or Bellman backup) in discrete time. In continuous time, it is called the Hamilton-Jacobi-Bellman (HJB) partial differential equation (PDE) [42]. The next section will deploy the dynamic programming approach to derive the HJ partial differential equation (PDE).

2.1.3 Derivation of Hamilton-Jacobi PDEs

Based on the Bellman principle of optimality, we can recursively calculate the value of the system backward if we know the value of the system at the last time step. In other words, we can divide the problem into subproblems that we can solve. Then we solve the original problem recursively from the subproblem.

Based on (2.4) and (2.5):

$$V(x(t),t) = \inf_{u}(\cdot) \int_{t}^{T} c(x(\tau),u(\tau))d\tau + l(x(T)) \tag{2.6}$$

T = 0 denotes the final time step, and the value of time T equals the terminal cost, $V(x(T),T) = l(x(T))$. According to the bellman optimality (dynamics programming), the target is to write the value of the current time as a function of some value for a future time step. Therefore, we break down the (2.6) into two-time intervals:

$$V(x(t),t) = \inf_{u}(\cdot) \Big\{ \int_{t}^{t+\delta} c(x(\tau),u(\tau))d\tau + \int_{t+\delta}^{T} c(x(\tau),u(\tau))d\tau + l(x(T)) \Big\}, \tag{2.7}$$

Next, we will write down the second interval part in the form of a value function at a time $(t+\delta)$,

$$V(x(t),t) = \inf_{u}(\cdot) \Big\{ \int_{t}^{t+\delta} c(x(\tau),u(\tau))d\tau + V(x(t+\delta),t+\delta) \Big\}. \tag{2.8}$$

For very small δ we can re-write (2.8) as:

$$V(x(t),t) = \inf_{u(t)\in\mathcal{U}} [c(x(t),u(t))\delta + V(x(t+\delta),t+\delta)]. \tag{2.9}$$

To expand the expression further, we use Taylor's expansion to expand $V(x(t+\delta),t+\delta)$:

$$V(x(t+\delta),t+\delta) \approx V(x(t),t) + D_t V(x(t),t)\delta + D_x V(x(t),t) \cdot \frac{dx}{dt}\delta \tag{2.10}$$

By plugging the Taylor approximation into (2.9) and factoring out all the terms that don't depend on the control variable from inf we have

$$V(x(t),t) = V(x(t),t) + D_t V(x(t),t)\delta + \inf_{u\in\mathcal{U}} \Big[c(x(t),u(t))\delta + D_x V(x(t),t) \cdot \frac{dx}{dt}\delta \Big] \tag{2.11}$$

, Since $\delta \neq 0$ doesn't depend on u can come out of inf expression and plugging in $\frac{dx}{dt} = f(x, u)$, we have:

$$\begin{aligned} &D_t V(x(t), t) + \inf_{u \in \mathcal{U}} [c(x(t), u(t)) + D_x V(x(t), t) \cdot f(x, u)] = 0, \\ &V(x, T) = l(x(T)). \end{aligned} \tag{2.12}$$

This equation is called Hamilton-Jacobi-Bellman (HJB) equation. Having the terminal value function, we can propagate the value function backward through time.

In general, the continuity and differentiability of the value function may not be valid. However, it can be shown there exists a unique viscosity solution to (2.12), which is uniformly continuous in x and t [21].

Lastly, we can add disturbance in (2.5) and derive the robust optimal control problem. In this case, we can assume the disturbance is acting adversarially, and the control problem seeks to minimize the cost in that scenario. The resulting equation is called the Hamilton-Jacobi-Issacs (HJI) equation and is written as:

$$\begin{aligned} &D_t V(x(t), t) + \inf_{u \in \mathcal{U}} \sup_{u \in \mathcal{D}} \Big[c(x(t), u(t), d(t)) + D_x V(x(t), t) \cdot f(x, u, d) \Big] = 0, \\ &V(x, T) = l(x(T)). \end{aligned} \tag{2.13}$$

The optimal control and disturbance can be given by:

$$\begin{aligned} u* &= \arg_{u \in \mathcal{U}} \inf \sup_{d \in \mathcal{D}} \Big[c(x, u, d) + D_x V(x(t), t) \cdot f(x, u, d) \Big], \\ d* &= \inf_{u \in \mathcal{U}} \arg_{d \in \mathcal{D}} \sup \Big[c(x, u, d) + D_x V(x(t), t) \cdot f(x, u, d) \Big]. \end{aligned} \tag{2.14}$$

Note that (2.13) can also be interpreted as differential game theory. In this scenario, Player 2 (disturbance) maximizes the cost function while player 1 (control) minimizes it. In robust control, we often assume player 2 uses non-anticipative control strategies $\Gamma(.)$ [119, 42], defined as follows:

$$\begin{aligned} d \in \Gamma_t^T := \{ \mathcal{N} : \mathbb{U}_t^T \to \mathbb{D}_t^T : u(t') = \hat{u}(t') \text{ a. e. } t' \in [t, T] \\ \Rightarrow \mathcal{N}[u](t') = \mathcal{N}[\hat{u}](t') \text{ a. e. } t' \in [t, T] \} \end{aligned} \tag{2.15}$$

According to (2.15), player 2 has an instantaneous informational advantage. In other words, player 2 chose its action after player 1's choice of input to act adversely.

2.1.4 Hamilton-Jacobi Reachability Analysis

In HJ-reachability, we assume the problem doesn't have running costs and only cares about a target set $\mathcal{G}$. It can be either a set of goal states or dangerous states. HJ-reachability analysis aims to find the initial states set that will lead to the target set under worst-case

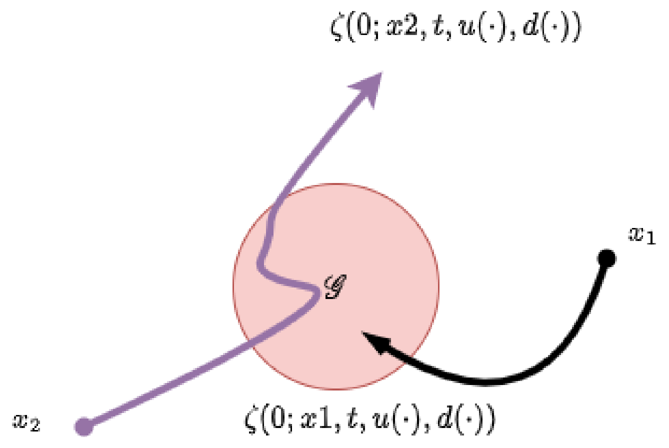

$\zeta(0; x2, t, u(\cdot), d(\cdot))$
x_1
$\mathcal{G}$
x_2
$\zeta(0; x1, t, u(\cdot), d(\cdot))$

Figure 2.1: Difference between BRS and BRT. The state x_1 is in both BRS and BRT; however, state x_2 is only in BRT

disturbances and optimal control. Depending on the nature of target set $\mathcal{G}$, the optimal behaviour is to either enter the set of initial states that can result in a target set (goal satisfaction) in case of worst-case disturbance or to avoid them (safety problem). If we care about entering the target set at a specific time, the problem is referred to as Backward Reachable Set (BRS). If we consider entering within a temporal horizon, the problem is referred to as Backward Reachable Tube (BRT). Fig. 2.1 depicts the difference between BRS and BRT. There are other types of reachability problems that this thesis does not utilize; however, we briefly mention them here for completeness. The first one is forward reachability. In this problem, we calculate all the final states that can be reached from a set of initial states at a given time (FRS) or within a given time horizon (FRT). In addition, the reach-avoid problem arises when both goal and danger sets are considered. The reach-avoid problem seeks to identify all initial sets that can reach a given target set while avoiding hazardous states [42].

Backward Reachable Set (BRS)

As we discussed, a BRS represents a set of initial states $x \in \mathbb{R}^n$ that the system can lead to a target set $\mathcal{G} \subseteq \mathbb{R}^n$ at the end of time horizon $|t|$ [13]. The BRS can be defined for both goal satisfaction and danger avoidance scenarios. In the former, $\mathcal{G}$ is the set of goal states, and BRS represents the set of states that are guaranteed to reach $\mathcal{G}$ in the presence of worst-case disturbance. This problem is often called "Maximal BRS" and is mathematically defined as:

$$\mathcal{R}(t) = \{x : \forall d(\cdot) \in \Gamma(t), \exists u(\cdot) \in \mathbb{U}, \zeta(0; x, t, u(\cdot), d(\cdot)) \in \mathcal{G}\}, \tag{2.16}$$

where $\Gamma(.)$ denotes the feasible set of disturbances. The expression in (2.16) reads as a set of all states x for which there exists a control strategy such that the trajectory starting at time t and states x will end up in target set $\mathcal{G}$ at the end of time horizon $|t|$.

To calculate the maximal BRS, we define the target set $\mathcal{G} \in \mathbb{R}^n$ to represent the sub-zero level set of an implicit surface function $g(x)$ defined as:

$$\mathcal{G} = \{x : g(x) \leq 0\}. \tag{2.17}$$

Note that $g(x)$ is a bounded and Lipschitz continuous function that is positive outside the target set and negative inside[42]. One choice of $g(x)$ can be the signed distance function from $\mathcal{G}$. The optimal control objective is the value of $g(x)$ at the end of the time horizon, i.e.

$$J_{\mathcal{R}}(x, t, u(\cdot), d(\cdot)) = g(\zeta(0; x, t, u(\cdot), d(\cdot)). \tag{2.18}$$

The corresponding value function can be obtained using the following optimization problem,

$$R(x, t) = \inf_{u(\cdot)} \sup_{d(\cdot)} \Big\{ J_{\mathcal{R}}(x, t, u(\cdot), d(\cdot) \Big\}, \tag{2.19}$$

and the optimal control is given by:

$$u^*_{\mathcal{R}}(.) = \arg_{u(\cdot)\in\mathcal{U}} \inf \sup_{d(\cdot)\in\mathcal{D}} \Big\{ J_{\mathcal{R}}(x, t, u(\cdot), d(\cdot) \Big\}. \tag{2.20}$$

The value function can be computed using the modified version of (2.5) written as:

$$\begin{aligned} &D_t R(x, t) + H(x, \nabla R) = 0, \quad t \in [T, 0], \\ &R(x, 0) = g(x). \end{aligned} \tag{2.21}$$

Where $H(x, \nabla)$ is Hamiltonian and depends on the system dynamics and the optimal control given by:

$$H(x, \nabla R) = \inf_{u(\cdot)\in\mathcal{U}} \sup_{d(\cdot)\in\mathcal{D}} D_x R(x, t) \cdot f(x, u, d). \tag{2.22}$$

Finally, the maximal BRS is the subzero level set of the value function (2.16):

$$\mathcal{R}(t, x) = \{x : R(x, t) \leq 0\}. \tag{2.23}$$

Intuitively, suppose the value function of a specific state is negative. In that case, the optimal trajectory starting from that state will achieve a negative cost value at the end of the trajectory, which according to (2.17), means it will end up inside the target set.

Similarly, we could define the $\mathcal{G}$ as the dangerous (unsafe) states we want to avoid. The avoid problem is often called "minimal BRS", defined as:

$$\mathcal{A}(t) = \{x : \exists d(\cdot) \in \Gamma(t), \forall u(\cdot) \in \mathcal{U}, \zeta(0; x, t, u(\cdot), d(\cdot)) \in \mathcal{G}\}. \tag{2.24}$$

The cost function for the avoid problem is defined the same as:

$$J_{\mathcal{A}}(x, y, u(\cdot), d(\cdot)) = g(\zeta(0; x, t, u(\cdot), d(\cdot)), \tag{2.25}$$

and to define the corresponding value function, we will have to change the optimization variable. In minimal BRS, the optimal controller seeks to maximize the cost function to avoid unsafe states while the disturbance acts the opposite, i.e.

$$A(x, t) = \sup_{u(\cdot)} \inf_{d(\cdot)} \Big\{ J_{\mathcal{A}}(x, t, u(\cdot), d(\cdot)) \Big\}. \tag{2.26}$$

This value function can be computed similar to (2.21). According to (2.24), all the states with negative values could lead to unsafe states in case of worst-case disturbance and should be avoided. Therefore the minimal BRS is computed as the subzero level set of the minimal value function:

$$\mathcal{A}(x, t) = \{x : A(x, t) \leq 0\}. \tag{2.27}$$

Note that one can assume $t \to -\infty$ and derive the time-invariant infinite-horizon value function as:

$$\begin{aligned} R(x) &= \lim_{t \to -\infty} R(x, t) \\ A(x) &= \lim_{t \to -\infty} A(x, t) \end{aligned} \tag{2.28}$$

Backward Reachable Tube (BRT)

Computing BRT is very important in the avoid problem in which we are interested in avoiding any states that could lead to danger at any time within the time horizon duration of $|t|$ [13]. Compared to BRS, if a trajectory enters the unsafe states and leaves the target set before the end of the time horizon, it will still receive a negative cost. Similar to the previous section, we will define BRT for both the avoid and reach problems.

Minimal BRT refers to calculating avoid reachable tube and is defined as:

$$\hat{\mathcal{A}}(x, t) = \{x : \exists d(\cdot) \in \Gamma(.), \forall u(\cdot) \in \mathcal{U}, \exists \tau \sim [t, 0] \zeta(\tau; x, t, u(\cdot), d(\cdot)) \in \mathcal{G}\}. \tag{2.29}$$

To capture whether the trajectory enters the target set, instead of considering the terminal cost function, we keep track of minimum cost during the trajectory i.e.

$$J_{\hat{\mathcal{A}}}(x, t, u(\cdot), d(\cdot)) = \min_{\tau\sim[t,0]} g(\zeta(\tau; x, t, u(\cdot), d(\cdot)). \tag{2.30}$$

Similarly, the value function can be obtained from the following optimization problem:

$$\hat{A}(x,t) = \sup_{u(\cdot)} \inf_{d(\cdot)} \Big\{ J_{\hat{\mathcal{A}}}(x, t, u(\cdot), d(\cdot)) \Big\}. \tag{2.31}$$

Note that computing the value function is not as straightforward as (2.13) because it involves minimization over time. This process requires updating the value function using the HJI PDE and minimizing the cost function to receive minimum cost over that time instant [42].

To begin with, we re-write the minimal BRT value function (2.31) as [42]:

$$\hat{A}(x,t) = \sup_{u(\cdot)} \inf_{d(\cdot)} \min_{\tau\sim[t,T]} g(x(\tau), \tau), \tag{2.32}$$

next, we break the expression into two-time intervals $[t, t+\delta]$ and $[t+\delta, T]$:

$$\hat{A}(x,t) = \sup_{u(\cdot)} \inf_{d(\cdot)} \min \left\{ \min_{\tau\sim[t,t+\delta]} g(x(\tau), \tau),\ \min_{\tau\sim[t+\delta,T]} g(x(\tau), \tau) \right\}. \tag{2.33}$$

To create a recursive equation, we add an extra supremum-infimum into the second inner minimum equation:

$$\hat{A}(x,t) = \sup_{u(\cdot)} \inf_{d(\cdot)} \min \left\{ \min_{\tau\sim[t,t+\delta]} g(x(\tau), \tau), \sup_{u(\cdot)} \inf_{d(\cdot)} \min_{\tau\sim[t+\delta,T]} g(x(\tau), \tau) \right\}. \tag{2.34}$$

$$\hat{A}(x,t) = \sup_{u(\cdot)} \inf_{d(\cdot)} \min \left\{ \min_{\tau\sim[t,t+\delta]} g(x(\tau), \tau), \hat{A}(x(t+\delta), t+\delta) \right\}. \tag{2.35}$$

We can simplify further by approximating the trajectory as a single time step for small δ as:

$$\hat{A}(x,t) = \sup_{u(\cdot)} \inf_{d(\cdot)} \min \left\{ g(x(t), t), \hat{A}(x(t+\delta), t+\delta) \right\}. \tag{2.36}$$

Next, we approximate $\hat{A}(x(t+\delta), t+\delta$ using Taylor's expansion:

$$\hat{A}(x,t) = \sup_{u(\cdot)} \inf_{d(\cdot)} \min \left\{ g(x(t), t), \left[\hat{A}(x(t), t) + D_t \hat{A}(x(t), t)\delta + D_x \hat{A}(x(t), t) \cdot f(x, u, d)\delta \right] \right\}. \tag{2.37}$$

By Applying the supremum-infimum inside the minimum expression, we have:

$$\hat{A}(x,t) = \min\left\{ g(x(t),t), \left[\hat{A}(x(t),t) + D_t\hat{A}(x(t),t)\delta + \sup_{u(\cdot)}\inf_{d(\cdot)} D_x\hat{A}(x(t),t)\cdot f(x,u,d)\delta\right]\right\}. \tag{2.38}$$

We can simplify more by subtracting the term $\hat{A}(x,t)$ from both sides of the equation:

$$0 = \min\left\{ g(x(t),t) - \hat{A}(x,t), \left[D_t\hat{A}(x(t),t) + \sup_{u(\cdot)}\inf_{d(\cdot)} D_x\hat{A}(x(t),t)\cdot f(x,u,d)\right]\delta\right\}. \tag{2.39}$$

Please note that both sides of the minimum expressions are non-negative at each time step. However, only one of the expressions is activated. Scaling a non-negative expression by a positive number ($\delta > 0$) doesn't affect the minimization; therefore, we can remove δ from the expression and derive the Hamilton-Jacobi-Isaacs Variational Inequality (HJI VI):

$$0 = \min\left\{ g(x(t),t) - \hat{A}(x,t), \left[D_t\hat{A}(x(t),t) + \sup_{u(\cdot)}\inf_{d(\cdot)} D_x\hat{A}(x(t),t)\cdot f(x,u,d)\right]\right\}. \tag{2.40}$$

According to (2.40), the HJI PDE must respect the inequality constraint that the BRT value function should not be greater than the $g(x(t),t)$ [42].

Finally, the minimal BRT will be the subzero level set of the value function:

$$\hat{\mathcal{A}}(x,t) = \{x : \hat{A}(x,t) \leq 0\} \tag{2.41}$$

If the BRT of a state has a negative value, it means the optimal trajectory will achieve a negative cost (entering the target set) at some point in its trajectory starting from that state and if we apply the optimal control expression,

$$u* = \arg\sup_{u(\cdot)}\inf_{d(\cdot)} D_x\hat{A}(x(t),t)\cdot f(x,u,d), \tag{2.42}$$

we can guarantee that the trajectory will avoid unsafe states.

Similarly, we can write the BRT expression for the reach problem that is called maximal BRS, defined as:

$$\hat{\mathcal{R}}(x,t) = \{x : \exists u(\cdot) \in \mathcal{U}, \forall d \in \Gamma(\cdot), \exists \tau \sim [t,T], \zeta(\tau; x, t, u(\cdot), d(\cdot)) \in \mathcal{G}\}. \tag{2.43}$$

The cost function is the same as (2.30), and for the value function expression, we flip the role of control and disturbance:

$$\hat{R}(x,t) = \inf_{u(\cdot)}\sup_{d(\cdot)}\{\min_{\tau\sim[t,T]} g(\zeta(\tau; x, t, u(\cdot), d(\cdot)\}. \tag{2.44}$$

Once again, the maximal BRT is the subzero level set:

$$\hat{\mathcal{R}}(x,t) = \{x : \hat{R}(x,t) \leq 0\}. \tag{2.45}$$

2.1.5 Running Example

This section will introduce the collision avoidance problem for two identical robots as a differential game to illustrate the concepts. In this problem, a "pursuer" tries to enter a danger zone around the "evader". In the end, we also introduce the agent-obstacle avoidance example.

Moreover, this example is extensively used in Ch.3 for reward calculation. For each robot, we assume planar dynamics that can be approximated using an Ordinary Differential Equation (ODE):

$$\dot{x_i} = f_i(x_i, u_i, d_i), \tag{2.46}$$

Where x_i represents the state of each agent i, u_i denotes the control input, and d_i denotes the disturbance. The joint relative dynamic of a pair of agents can be derived as

$$\dot{x}_{ij} = g_{ij}(x_{ij}, u_i, u_j), \tag{2.47}$$

where x_{ij} is the relative state of agent j with respect to agent i and u_i, u_j are control inputs for agents i and j.

All of the ODEs are assumed to be uniformly continuous, bounded, and Lipschitz continuous in their domain for fixed control inputs and disturbances. Moreover, u_i, u_j d_i are chosen from a set of measurable functions $\mathbf{U_i}$, $\mathbf{U_j}$ and $\mathbf{D_i}$ [14], [12].

For numerical calculation, we assume the system dynamics of each agent approximate as a Dubin's car model with a planar position (p_x, p_y) and a heading (θ):

$$\begin{bmatrix} \dot{p_x} \\ \dot{p_y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix}, \tag{2.48}$$

Where $u = (v, \omega)$ represents linear and angular velocities. Then the pair-wise relative dynamic is given by:

$$\begin{bmatrix} \dot{p}_{x,ij} \\ \dot{p}_{y,ij} \\ \dot{\theta}_{ij} \end{bmatrix} = \begin{bmatrix} -v + v\cos\theta_{ij} + \omega_i p_{y,ij} \\ v\sin\theta_{ij} - \omega_i p_{x,ij} \\ \omega_j - \omega_i, \end{bmatrix} \tag{2.49}$$

where joint state $x_{ij} = (p_{x,ij}, p_{y,ij}, \theta_{ij})$ represents relative coordinate of j w.r.t. i, $v = v_{i_{\max}} = v_{j_{\max}}$ is linear velocity and ω is angular velocity in which $|\omega_i|, |\omega_j| \leq \omega_{\max}$.

We define the Backward Reachable Set (BRS) as the set of dangerous relative states between agent i and another agent j that will result in an inevitable collision if agent j

applies the worst-case control policy to cause a collision, regardless of agent i control policy. The BRS of agent i with respect to agent j defines as

$$\begin{aligned}\mathcal{R}_{ij}(t) = \{x_{ij} : \forall u_i \in \mathbb{U}_i, \exists u_j \in \mathbb{U}_j, x_{ij}(.) \text{ satisfies } (2.47),\\ \exists s \in [0,t], x_{ij} \in \mathcal{G}_{ij}\},\end{aligned} \tag{2.50}$$

where $\mathcal{G}_{ij}$ denotes the danger zone between any agents and is equal to a set of all states where a pair of agents are within r unit of other defined as:

$$\mathcal{G}_{ij} = \{x_i, x_j : (p_{x,i} - p_{x,j})^2 + (p_{y,i} - p_{y,j})^2 \leq d^2\} \tag{2.51}$$

We assume the danger zone is defined as the zero sub-level sets of a bounded and Lipschitz continuous cost function $G(x_i, x_j)$. The BRS $\mathcal{R}_{ij}(t)$ is represented as the zero sub-level set of the safety value function $R_{ij}(t, x_{ij})$, which is the viscosity solution of the associated HJI PDE (2.13) solved backward through time for $t \in [-T, 0]$. The HJ PDE in (3) is solved numerically using specialized finite difference methods like the Lax-Friedrichs method [81]. Fig. 4.5-right shows the resulting BRS for the pair-wise collision avoidance.

Note we assume $t \to \infty$ and derive the time-invariant infinite time horizon safety level value function for dynamic agents ($R_{ij}(x_{ij})$) as follows:

$$R_{ij}(x_{ij}) = \lim_{t\to\infty} R_{ij}(t, x_{ij}) \tag{2.52}$$

The interpretation is that if x_{ij} be outside of the $R_{ij}(x_{ij}) \geq 0$ for j, it can be guaranteed that there exists a control u_i for agent i such that the pair of i, j will never collide with each other within an infinite time horizon. On the other hand, if x_{ij} falls inside $R_{ij}(x_{ij})$, the collision is inevitable if agent j decides to apply the optimal control policy towards agent i. Please refer to [4] and [75] for more details.

Similar to the dynamic agents, we define a BRS for avoiding static obstacles as a set of undesirable states for an agent that can result in a collision as [14]:

$$\mathcal{R}_i(t) = \{x_i : \exists d_i \in \mathbf{D}_i, \forall u_i(.) \in \mathbf{U}_i, \exists s \in [t, 0], f_i(x_i, u_i, d_i) \in \sigma\}, \tag{2.53}$$

Where σ represents the target set, defined as the collection of dangerous states between an obstacle and an agent. We can also solve the associated Hamilton-Jacobi partial differential equation (HJ PDE) to compute the time-invariant safety level value function for static obstacles, denoted as $R_i(x_i^r, \theta_i)$. Here, x_i^r refers to the relative 2D position of agent i with respect to the obstacle, and θ_i represents the orientation of agent i. Figure 4.5-left illustrates the resulting backward reachable set (BRS) for the agent-obstacle scenario.

2.2 Reinforcement Learning

Reinforcement Learning (RL) aims to develop a method to learn how to think, make sequential decisions to achieve a goal and map the observations into actions. The early ideas of RL were proposed in 1950 by Alen Turing [113] to answer the question "Can machines think?". In this paper, Alan Turing proposes the idea of an autonomous algorithm that can learn itself:

> "In the process of trying to imitate an adult human mind we are bound to think a good deal about the process which has brought it to the state that it is in. We may notice three components,
> a. The initial state of the mind, say at birth,
> b. The education to which it has been subjected,
> c. Other experience, not to be described as education, to which it has been subjected.
> Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain. Presumably the child-brain is something like a note-book as one buys it from the stationers. Rather little mechanism, and lots of blank sheets. (Mechanism and writing are from our point of view almost synonymous.) Our hope is that there is so little mechanism in the child-brain that something like it can be easily programmed".

This section will provide a compact overview of important RL concepts used in this thesis.

2.2.1 RL Problem Formulation

RL usually proposes algorithms that learn by interacting with the environments. The agent-environment interaction is depicted in Fig. 2.2 In this model, the agent is the learner or decision-maker interacting with the environment. At each instance, the agent executes an action in the environment, and it will receive an observation and a reward signal from the environment.

RL agent-environment interaction is commonly modelled as a Markov Decision Process (MDP). MDP is a mathematical framework used to model decision-making situations in which an agent faces a sequence of choices or actions that affect the system's future state. The key characteristic of an MDP is that the agent's choices and the resulting outcomes are governed by the Markov property, which means that the probability of transitioning to a particular future state depends only on the current state and the action taken and not on the sequence of events that led to the current state.

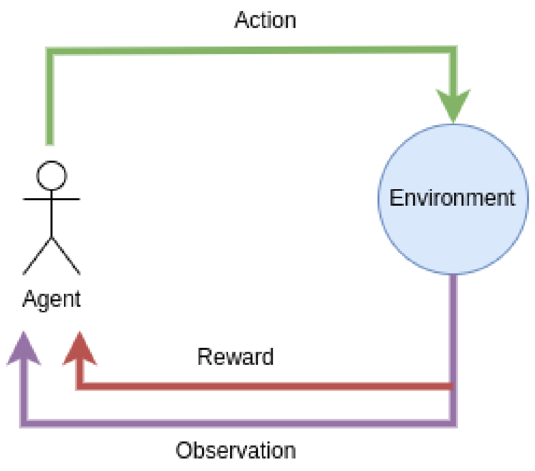


Figure 2.2: Agent-Environment Interaction in reinforcement learning. An agent executes an action in the environment, and the environment responds with an observation indicating the change in the agent/environment and a reward signal.

Fig.2.3 illustrates the RL problem in the MDP framework. An MDP defines with a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma, p(s_0))$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $p(s_{t+1}|s_t, a_t)$ is the transition probability, r is the reward function, $\gamma \; in[0, 1]$ denotes the discount factor and $p(s_0)$ is the initial state distribution.

The agent begins at an initial state, denoted as s_0, which is sampled from the state space $\mathcal{S}$ according to the distribution $p(s_0)$. At each time step, the agent observes the current state, denoted as s_t, from the state space $\mathcal{S}$, and takes an action, denoted as a_t, from the action space $\mathcal{A}$. The action is selected based on the policy $\pi^*(a_t|s_t)$. The environment then responds by providing the next state, denoted as s_{t+1}, which is sampled from the transition distribution $p(s_{t+1}|s_t, a_t)$, and a scalar reward signal, denoted as r_t.This cycle (which is called an episode) continues until some finite or infinite time horizon T is reached. The RL objective is to maximize the sum of discounted expected return:

$$J(\pi) = \mathop{\mathbb{E}}_{\tau \sim p(\tau|\pi)} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right], \tag{2.54}$$

where $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_T)$ is the episode trajectory and, $p(\tau|\pi)$ is the distribution of the trajectory induced by policy π i.e.

$$p(\tau|\pi) = p(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t). \tag{2.55}$$

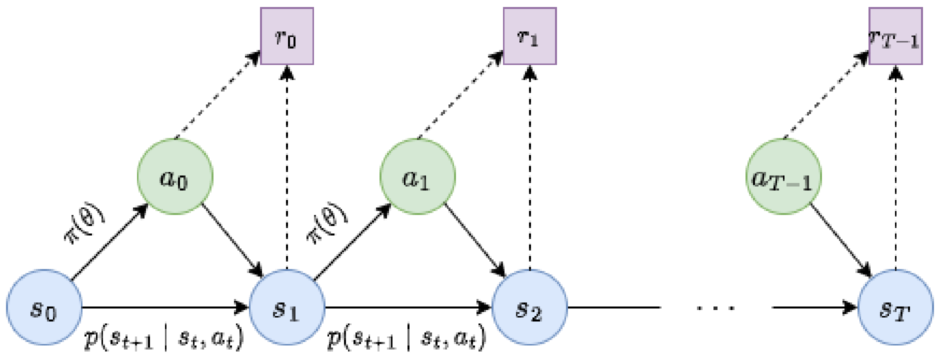


Figure 2.3: Reinforcement Learning framework in MDP form. At each time step t, the agent took action based on observation s_t using policy $\pi_\theta(a_t|s_t)$. The executed action leads to a transition to a new state s_{t+1} based on state-transition probability $p(s_{t+1}|s_t, a_t)$ and a reward r_t. The sequential decision-making continued up to a time horizon of T.

Finally, the optimal policy can be obtained by solving the following optimization problem:

$$\pi^* = \arg_\pi \max J(\pi) \tag{2.56}$$

.

2.2.2 Policy Iterations

Policy Iteration is an iterative Dynamic Programming (DP) based approach to optimize the policy behaviour according to the RL objective defined in (2.54). In this process, we evaluate policy and then improve its performance until it converges to the optimal policy. The remaining part of this section will explain policy evaluation and policy improvement steps.

Policy Evaluation

It is essential to evaluate the policy's performance before optimizing its behaviour. This also refers to the policy prediction problem [107]. We start with RL objective (2.54) and define two important concepts in RL: State value function and State-Action value function. The State value function of a given policy π at state s estimates how well that policy can perform in terms of expected returns [107] i.e.

$$V^\pi(s) = \mathop{\mathbb{E}}_{\tau \sim p(\tau|\pi, s_0 = s)} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right]. \tag{2.57}$$

Equation (2.57) provides an estimate of the expected discounted return when starting at state s and following policy π until the end of the episode. The inclusion of the discount factor γ is particularly important in scenarios with infinite episodes, as it prevents the state value function from diverging to infinity. The state value function serves as a valuable metric for comparing the performance of two policies that begin from the same initial state. It allows us to assess and contrast their expected returns over the long term.

The state-action value function (called Q-function) is another helpful identity to evaluate the performance of one action to another in a given state and then follow the policy. The value of state s and action a defines as:

$$Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{\tau\sim p(\tau|\pi,s_0=s,a_0=a)}\left[\sum_{t=0}^{T-1}\gamma^t r_t\right]. \tag{2.58}$$

Similarly, the (2.58) estimates how much we expect to receive the discounted reward by starting at state s, taking action a and following policy π.

By defining the value functions, we can define a partial ordering over policies as:

$$\pi \geq \pi' \Longleftrightarrow V^{\pi}(s) \geq V^{\pi'}(s), \tag{2.59}$$

then we can define the optimal state $V^*(s)$ and state-action $Q^*(s)$ value functions that are induced by optimal policy π^* according to the following theorem [118]:

Theorem 1. $\forall$ *MDPs* $,\forall\pi, \exists\pi^* \Rightarrow \pi^* \geq \pi.$

Once optimal value functions are obtained, the optimal action at every step can be recovered by taking actions that maximize the Q^* as:

$$\pi^*(a|s) = \begin{cases} 1 & \text{if} a = \arg_{a\in\mathcal{A}}\max Q^*(s,a) \\ 0 & \text{otherwise} \end{cases} \tag{2.60}$$

To develop RL algorithms, it is often essential to write (2.57) and (2.58) in recursive forms known as bellman equations:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{a\sim\pi(a|s)}\mathop{\mathbb{E}}_{s'\sim p(s'|s,a)}[r_t + \gamma V^{\pi}(s')] \tag{2.61}$$

$$Q^{\pi}(s) = \mathop{\mathbb{E}}_{s'\sim p(s'|s,a)}\mathop{\mathbb{E}}_{a'\sim\pi(a'|s')}[r_t + \gamma Q^{\pi}(s')], \tag{2.62}$$

and the relation between state and state-action value functions are:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{a\sim\pi(a|s)}[Q^{\pi}(s,a)] \tag{2.63}$$

To evaluate the policy's value, if the transition dynamic is well defined, we could use the dynamic programming (DP) algorithm to calculate the policy's state and state-action values. We start with an arbitrary value and then iteratively use the Bellman equations (2.61) and (2.62) until the convergence. Using the word "convergence" might not be accurate for all cases; however, we can show the algorithm converge to a fixed point for each state with proper assumptions.

It is also possible to estimate the value functions if the transition dynamic is unknown using Monte Carlo (MC) estimation or Bootstrap methods. In the former, we sample the episode with the same initial state distribution $p(s_0)$ multiple times by following policy π and then take the average to calculate the expectation. We can use the recursive bellman equations to calculate the value functions in the latter. The method is often called temporal-difference (TD) learning. The TD method has different variations depending on how many steps are used to estimate the returns. Compared to MC estimation, the TD method doesn't require waiting for the episode's outcome and can be calculated using single/multiple transitions.

To estimate the state and state-action value function using TD(0), we first roll out the policy π and record the dataset of transitions $\mathcal{D}$. We initialize the value function with an initial arbitrary guess V^0 and then use the bellman equations in an interactive procedure to estimate the V^π by optimizing the following objectives:

$$V^k = \arg\min_V \mathop{\mathbb{E}}_{(s_t, r_t, s_{t+1}) \sim D} [V(s_t) - (r_t + \gamma V^{k-1}(s_{t+1})]^2, \tag{2.64}$$

$$Q^k = \arg\min_Q \mathop{\mathbb{E}}_{(s_t, r_t, a_t, s_{t+1}, a_{t+1}) \sim D} [Q(s_t) - (r_t + \gamma Q^{k-1}(s_{t+1})]^2. \tag{2.65}$$

Where V^k and Q^k are the state and state-action value functions at iteration k, respectively. It can be shown for $k \to \infty$, $V^k \to V^\pi$ and $Q^k \to Q^\pi$.

Note that value functions are often approximated with function approximators such as Neural Networks (NNs), and the optimization is done using Gradient Descent Methods. Furthermore, we can compare the value functions properly with an infinite number of iterations.

Policy Improvements

Once we estimate the values of a policy, we can act greedy with respect to estimated values and improve policy behaviours. For any given state and greedy policy is calculated as:

$$\begin{aligned} \pi_{\text{new}}(s) =& \arg\max_{a \in \mathcal{A}} Q^\pi(s, a) \\ &= \arg\max_{a \in \mathcal{A}} \mathbb{E}[r_t + \gamma V^\pi(s_{t+1})] \end{aligned} \tag{2.66}$$

2.2.3 Policy Gradient

Policy Gradient (PG) is a class of RL approaches that try to optimize policy behaviour directly by solving the RL objective optimization through gradient ascent. We assume the policy is described using a function approximator such as NN with some parameter θ, which can be obtained using

$$\theta^* = \arg\max_{\theta} \mathop{\mathbb{E}}_{\tau\sim p_\theta(\tau)}[r(\tau)], \tag{2.67}$$

where $r(\tau) = \sum_{t=0}^{T-1}[\gamma^t r_t] = Q^\pi$. Next, we expand the expectation into the integral form to calculate the gradient of the policy objective as:

$$J(\theta) = \mathop{\mathbb{E}}_{\tau\sim p_\theta(\tau)}[r(\tau)] = \int p_\theta(\tau) r(\tau) d\tau. \tag{2.68}$$

By taking gradient of objective w.r.t to parameter θ and assuming $p_\theta(\tau) \neq 0$ we have:

$$\begin{aligned}\nabla_\theta J(\theta) &= \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau \\ &= \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) r(\tau) d\tau \\ &= \mathop{\mathbb{E}}_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)].\end{aligned} \tag{2.69}$$

Recalling (2.55) we have:

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau\sim p_\theta(\tau)}[\nabla_\theta[\log p(s_0) + \sum_{t=0}^{T-1} \log \pi(a_t|s_t) + \log p(s_{t+1}|s_t, a_t)] r(\tau)]. \tag{2.70}$$

Since the initial state and transition dynamic distributions don't depend on policy parameter θ:

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathop{\mathbb{E}}_{\tau\sim p_\theta(\tau)}[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t) \sum_{t=0}^{T-1} \gamma^t r_t] \\ &= \mathop{\mathbb{E}}_{\tau\sim p_\theta(\tau)}[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t) Q^\pi]\end{aligned} \tag{2.71}$$

Finally, we use the gradient descent to update the parameters

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta). \tag{2.72}$$

Alg. 1 summarizes the simple PG algorithm, often called Reinforce algorithm [59].

Algorithm 1 Reinforce Algorithm

```
Require: initialize policy parameters $\theta$ and learning rate $\alpha$
1: while not done do
2:     roll out the policy $\pi_\theta$ and sample trajectory $\{\tau_i\}$
3:     calculate $\nabla_\theta J(\theta)$ according to (2.71)
4:     update $\theta$ using gradient ascent (2.72)
5: end while
```

For an unknown dynamic, we have to estimate the expectation (2.71) by sampling from multiple trajectories and taking the average. Therefore, the PG objective has high variance and is often unstable.

One remedy to reduce the variance is subtracting the estimated return by a state-dependant value. One of the popular choices is to use the value function itself [59],

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau \sim p_\theta(\tau)} [\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t)(Q^\pi - V^\pi)] \tag{2.73}$$

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau \sim p_\theta(\tau)} [\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t) A^\pi] \tag{2.74}$$

where A^π is called the advantage function, which shows how much a given action performs better than average i.e.

$$\begin{aligned} A^\pi(s_t, a_t) &= Q^\pi(s_t, a_t) - V^\pi(s_t) \\ &= r(s_t, a_t) + \gamma V^\pi(s_{t+1}) - V^\pi(s_t) \end{aligned} \tag{2.75}$$

2.2.4 Actor-Critic

One must estimate the policy state and state-action value function to estimate the PG objective. Instead of estimating the return at each step using numerical methods, one practical implementation is to use another function approximator to estimate the PG objective (Q^π, V^π or A^π). This boils down to another class of RL algorithm called actor-critic, which is used in this thesis to train our methods. Alg. 2 summarize a simple actor-critic algorithm [59].

Algorithm 2 Actor-Critic Algorithm

```
Require: initialize policy $\pi^\theta$, value function $V^\phi$ and learning rate $\alpha$
1: while not done do
2:     roll out the policy $\pi_\theta$ and sample trajectory $\{\tau_i\}$
3:     update $V^\phi$ using (2.64)
4:     evaluate $A^\pi$ using (2.75)
5:     calculate $\nabla_\theta J(\theta)$ according to (2.71)
6:     update $\theta$ using gradient ascent (2.72)
7: end while
```

In the following chapters, we use two well-known actor-critic algorithms to train our algorithm. First, in Chapters 3-5, we used Proximal Policy Optimization (PPO), which has an additional modification to stabilize the PG training algorithm using Generalized Advantage Estimation (GAE) to update the value function [96]. Second, in Chapter 4, we used Soft Actor-Critic (SAC) algorithm to train our agent [39]. SAC objective has an additional term that also tries to maximize the policy's entropy and the advantage of encouraging exploration in the policy learning procedures.

2.2.5 Meta Reinforcement Learning

Meta Reinforcement Learning (meta RL) refers to learning-to-learn problems. In regular RL problems, the objective is to learn a task from scratch by trial-and-error directly on a physical or simulated agent. On the other hand, the meta RL proposes an approach to leverage what the policy learns on one task and use that knowledge in another task. Therefore, the learning algorithm can be more efficient and effective. In Meta RL, we assume learning n different multiple tasks, where each task is an MDP $\mathcal{M}_i$. Then meta RL optimization can be written as: [59]:

$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{n} \mathop{\mathbb{E}}_{\pi_{\phi_i}} [R(\tau)], \tag{2.76}$$

where $\phi_i = f_\theta(\mathcal{M}_i)$ and $f(.)$ is a function approximator that estimates policy parameter π that is parameterized by θ.

MAML (Model Agnostic Meta-Learning) is one of the popular gradient-based meta-RL approaches [33]. In MAML, policy parameter optimizes so that they can be easily adapted to a new task using a few examples from the new task. MAML training procedure consists of two phases. The first phase, called meta-learning, is the policy parameter optimized to behaviour fair in several tasks. In the second phase, called meta adaptation, the policy parameters are adapted to a specific task by performing a few fine-tuning on new task transitions. In MAML, the function f is given by [59]:

$$f_\theta(\mathcal{M}_i) = \theta + \alpha \nabla_\theta J_i(\theta) \tag{2.77}$$

Where α is the learning rate, and $J_i(\theta)$ is the RL objective for a task $\mathcal{M}_i$. Furthermore, the policy update parameter (2.72) change to:

$$\theta \leftarrow \theta + \beta \sum_{i=1}^{n} \nabla_\theta J_i(\theta + \alpha \nabla_\theta J_i(\theta)). \tag{2.78}$$

where β and α are the meta-training and meta-adaptation learning rates, respectively.

2.2.6 Multi-Agent RL Problem Formulation

As we mentioned in Chapter 1, one of our case studies is a multi-agent collision avoidance problem that can be formulated as a multi-agent meta RL problem. We consider a meta RL formulation with task distribution $p(\mathcal{T})$, where task $\mathcal{T}$ involves multiple random agents ranging from $N = 3, ..., n$ in a meta world in the presence of random obstacles. Each of the tasks is assumed to be a Markov Decision Process (MDP) with states $s_t \in \mathcal{S}$, discrete actions $a_t \in \mathcal{A}$, rewards r_t, initial state distribution $p(s_1)$ and stochastic state transition

$p(s_{t+1}|s_t, a_t)$. Note that each task involves a different number of agents, resulting in a different MDP; thus, we use meta RL to solve all tasks simultaneously. The objective is to meta-learn a policy $\pi(a_t|o_t)$ that maximizes the sum of expected discounted returns $R = \mathbb{E}[\sum_{t=t'}^{T}(\gamma^{T-t})r_t]$ for all agents in all tasks. We also assumed that states, transition function and reward function are unknown to the learning agents.

2.3 Variational AutoEncoder

In Chapter 3 and Chapter 6, we used Variational Auto Encoder (VAE) as a useful dimensionality reduction tool in the RL training framework. Variational Autoencoder (VAE) is a generative model that learns to represent high-dimensional data in a lower-dimensional latent space. The VAE learns a probabilistic model of the input data in the latent space, which can be used for data compression, image generation, and data augmentation tasks. The VAE consists of two main components: an encoder and a decoder. The encoder maps the input data to a latent representation, while the decoder generates the output data from the latent representation. The key idea behind the VAE is to learn a probabilistic model of the input data in the latent space, which allows for efficient sampling and manipulation of the data in the latent space.

To understand the VAE, let's start with the basic equations. The VAE aims to learn a distribution over the latent variables z that best explains the input data x. The encoder parameterizes the approximate posterior distribution $q_\phi(z|x)$ over the latent variables, and the decoder parameterizes the conditional distribution $p_\theta(x|z)$ over the input data given the latent variables.

The VAE is trained by maximizing the evidence lower bound (ELBO), which is given by:

$$\begin{aligned}\mathcal{L}_{\text{ELBO}} &= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \text{KL}(q_\phi(z|x)||p(z)) \\ &= \mathbb{E}_{q\phi(z|x)}[\log p_\theta(x|z)] - \text{KL}(q_\phi(z|x)||\mathcal{N}(0, I))\end{aligned} \tag{2.79}$$

where $\mathbb{E}_{q\phi(z|x)}$ denotes the expected value over the posterior distribution, KL denotes the Kullback-Leibler divergence between the posterior distribution and the prior distribution, and $\log p_\theta(x|z)$ is the reconstruction loss of the decoder. The first term of the ELBO represents the reconstruction loss of the decoder. In contrast, the second term encourages the posterior distribution to be similar to the prior distribution, which is typically chosen as simple as the standard normal distribution.

The VAE tends to produce blurry and low-quality reconstructions compared to other generative models, such as Generative Adversarial Networks (GANs). This is because the VAE maximizes the ELBO objective, which can result in the model focusing more on accu-

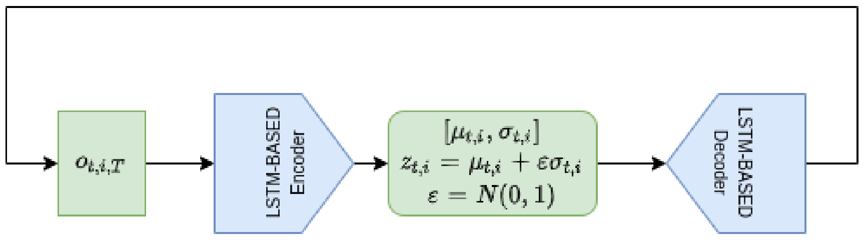

$o_{t,i,T}$
LSTM-BASED Encoder
$[\mu_{t,i}, \sigma_{t,i}]$
$z_{t,i} = \mu_{t,i} + \varepsilon\sigma_{t,i}$
$\varepsilon = N(0,1)$
LSTM-BASED Decoder

Figure 2.4: Proposed LSTM-based VAE

rately reconstructing the input data than on capturing the underlying structure of the data in the latent space.

Another challenge is the difficulty in choosing an appropriate prior distribution for the latent space. The prior distribution can affect the quality of the generated samples and the expressiveness of the model. In practice, the standard normal distribution is often used as the prior distribution due to its simplicity and ease of use.

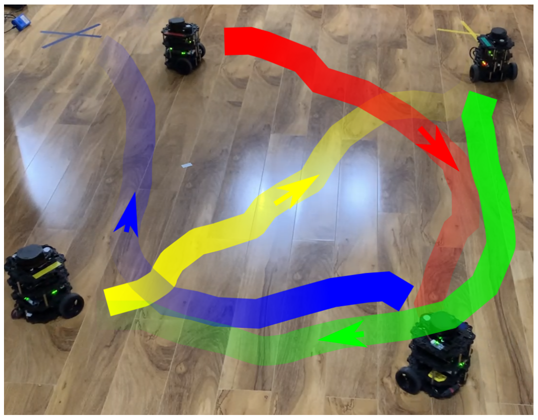
2.3.1 Running Example

In Chapter 3, we utilize a Variational Autoencoder (VAE) to handle inputs with varying size dimensions and map them into a fixed-size latent space. Specifically, in the tasks involving other agents, the observations are defined by the relative position and orientation of these agents with respect to the ego-agent. Fig. 2.4 provides a visualization of the proposed VAE architecture. To convert an input observation $o_{i,t,\tau}$ into the latent space, we employ the encoder network, which utilizes the reparameterization trick to enable sampling. Subsequently, the sampled latent space is transformed back into an observation via the decoder network. Both networks are meta-trained by maximizing the Evidence Lower Bound (ELBO) as defined in Equation (2.79).

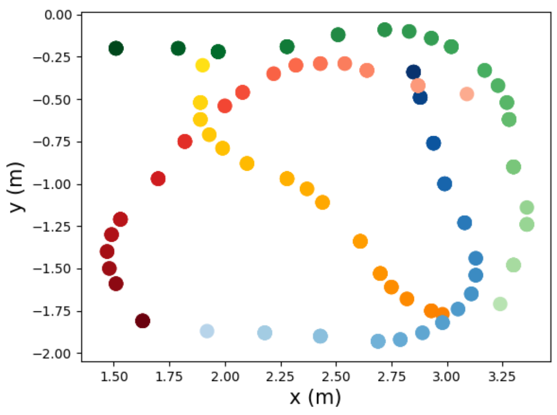
It is worth noting that, in order to handle varying size inputs, we employ a Long-Term Short-Memory (LSTM) based network architecture. LSTM is a type of recurrent neural network that is capable of handling sequential input.

Chapter 3

State-Based Collision Avoidance Module



(a) Trajectory visualization on real implementation - colours became lighter as time goes



(b) Generated trajectory for a 4-agent task through real implementation

Figure 3.1: The visualization of generated trajectory in a real-world experiment.

With the increasingly widespread use of autonomous vehicles such as mobile robots and unmanned aerial vehicles (UAVs), control and trajectory planning for multi-agent systems are gaining attention within the scientific community. Especially considering the current pandemic situation, the use of multiple autonomous vehicles in Giga-Factories and hospitals, where physical distancing prevents human workers from performing their regular jobs, is essential. In all these applications, multiple agents need to operate in the same workspace carrying out their respective tasks while not interfering with others. Many of these multi-agent systems are also safety critical.

Recent deep reinforcement learning (RL) methods have produced a quantum leap in solving difficult tasks. Successful applications include video games [76]-[100], continuous control [65]-[94], and manipulation tasks from high-dimensional image observations [63][32]. However, there are still some key challenges in the end-to-end training of precise controllers based on sensory observations using deep RL (DRL), which prevents their use in some real-world applications. Moreover, DRL tends to need lots of training episodes to learn a single task, and it is usually not trivial to transfer the learned policy to other similar tasks. Even though recent deep meta-RL methods are capable of adapting to new similar tasks, they usually are on-policy and require a massive amount of data from different tasks [33], [25], [74].

Our motivation is to design a Least-Restrictive Collision Avoidance Module (LR-CAM) that can be added on top of any autonomous agent conducting tasks in an environment that consists of multiple other agents. Our approach adds a layer of safety to all agents co-existing in a shared environment, each of whom typically operates based on some default objective/controller. The least-restrictive policy has several benefits compared to other "do-it-all" policies that try to both achieve a goal as well as maintain safety. For example, do-it-all policies can require a lot of training data and resources to learn both objective and safety, and therefore unable to maintain both performance and safety at the same time. In addition, if the goal or objective changes, a new policy or complicated meta-trained policy needs to be trained. In contrast, the least-restrictive policy enables agents to have any classical or learning-based default controller, such as a goal-reaching controller, while providing a higher level of safety for all agents. To achieve this, we propose a distributed control approach using a single deep policy network for all agents in which the LR-CAM of each agent, based on the agent's observations, decides whether it is safe to follow its default controller. If not safe, LR-CAM takes control of the agent and attempts to resolve potential conflicts that may lead to collisions. In our algorithm, first, a Long Short Term Memory (LSTM) [45] variational auto-encoder (VAE) maps variable-length observations from a varying number of observed agents into a fixed-length latent space. Then our meta-trained policy supervises the navigation based on the latent space. To remedy the need for a complicated hand-designed reward function, we propose a novel reward function based on Hamilton-Jacobi (HJ) reachability theory [75] to help LR-CAM effectively learn when to take control over each agent.

The key contributions of the proposed work are as follows: (1) We propose LR-CAM, which encompasses any goal-oriented policy of autonomous vehicles to monitor whether collision avoidance maneuvers are needed and if so, take over control to perform the avoidance maneuvers. (2) We propose a model-free sample efficient meta-RL algorithm for training the LR-CAM; through the use of an LSTM-VAE, we map observations to a latent space to enable LR-CAM to be used in different environments with a varying number of agents. (3) We propose a novel reward function based on a safety value function computed from HJ reachability. We conduct our training and experiments within a ROS-based controller. We show that the LR-CAM outperforms the baseline classical approach.

3.1 Related Works

Multi-agent collision avoidance Research pertinent to multi-agent collision-free trajectory planning is vast. The authors in [35] proposed using the dynamic obstacle method in collision avoidance problems; here, the assumed dynamics are used to predict agents' future behaviours. In another work, [115] proposed an Optimal Reciprocal Collision Avoidance (ORCA) method, which is a well-known classical approach for multi-agent collision avoidance. The ORCA achieves collision avoidance by optimizing a related constrained cost in a short time horizon. The problem of trajectory planning and collision avoidance is also studied widely in safety critical systems under the differential game framework. In [75], collision avoidance has been studied using HJ reachability theory in small-scale problems such as those found in a two-agent setup; the authors provide safety and goal-reaching guarantees. However, HJ analysis becomes intractable as the number of agents increases. The authors in [12] proposed using Mixed Integer Programming (MIP) to establish a higher-level logic around pair-wise collision avoidance using HJ-reachability to alleviate the intractability.

The problem of multi-agent collision avoidance is also addressed in many previous works within the RL research community [15, 28, 27]. The authors in [15] proposed using RL for a two-agent collision avoidance problem. They extend their previous work in [27] and proposed an RL-based method for single agent motion planning among moving objects, treated as obstacles, without any assumptions on the agents' behaviour. They used an actor-critic algorithm to train an agent with an LSTM in the first layer of their policy network. In our work, we meta-trained a separate embedding-VAE to alleviate the "representation learning bottleneck" [99] and varying input sizes. The authors in [28] proposed a decentralized policy, trained using an actor-critic RL method which is capable of mapping raw sensory observations to control outputs that result in collision-free trajectories. In comparison to the existing RL methods, which propose end-to-end solutions that consider both the problem of goal-reaching and avoidance simultaneously, our proposed LR-CAM takes inspiration from HJ reachability by only interrupting the default controller when needed.

Meta Reinforcement Learning Meta RL has gained attention recently because it enables an agent to easily adapt to new tasks with some shared structures. Meta RL comprises two main stages. In the first stage, a meta-policy is trained with a huge amount of data across different tasks. In the second stage, the trained policy is adapted to a specific task with relatively few training iterations. However, most of the proposed meta-RL algorithms are on-policy, which means that they often need a huge amount of training data in the meta-training phase. In general, Meta-RL methods can be divided into two main categories [86]: (1) context-based and (2) gradient-based. In the context-based approach, the experiences of tasks are mapped into a latent representation, and then a conditional policy is used to adapt to previously seen tasks [25], [120]. The authors in [86] proposed an off-policy context-based approach by representing the context using probabilistic latent variables. Compared to our method, it needs an additional latent representation layer to embed the varying dimension of inputs to the latent state, which could make the training procedure inefficient. In gradient-based methods, a policy network [33] or a loss function [106] is meta-trained to capture the task experience in model parameters. Our method can be considered as a gradient-based method built on top of the Model Agnostic Meta-Learning (MAML) by [33].

3.2 Approach

We are interested in learning the LR-CAM policy that allows all agents to follow their default controllers as long as no intervention is needed to prevent the collision. The default controller could be used for goal-reaching or simply be given by a human controller. However, when all or a subset of the agents are in potential conflict with each other, the LR-CAM will intervene and take over control to attempt to allow vehicles to revert back to safety. As a result, our policy not only decides whether each agent can follow its default controller or avoid danger but also detects possible dangers from a latent-space observation.

3.2.1 Observation, Action, and Latent Spaces

We start by assuming each agent operates according to the following approximate ordinary differential equation (ODE) model:

$$\dot{x}_i = f_i(x_i, u_i) \tag{3.1}$$

Where $x_i \in \mathcal{X}_i = (p_{x,i}, p_{y,i}, p_{\theta,i})$ and $u_i \in \mathcal{U}_i = (v, \omega)$ are the states and control inputs (linear and angular velocities) of agent i respectively. The dynamic in (3.1) induces the following relative dynamic between each pair of agents:

$$\dot{x}_{ij} = g_{ij}(x_{ij}, u_i, u_j), i \neq j \tag{3.2}$$

where u_i, u_j are control inputs to agent i, j respectively, $x_{ij} = (p_{x,ij}, p_{y,ij}, p_{\theta,ij})$ is the relative states of agent j with respect to agent i and calculated as follows:

$$\begin{bmatrix} p_{x,ij} \\ p_{y,ij} \\ p_{\theta,ij} \end{bmatrix} = \begin{bmatrix} \cos\theta_j & -\sin\theta_j & 0 \\ \sin\theta_j & \cos\theta_j & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_{x,i} - p_{x,j} \\ p_{y,i} - p_{y,j} \\ p_{\theta,i} - p_{\theta,j} \end{bmatrix} \tag{3.3}$$

Using the relative coordinate not only reduces the dimension of observation space but also can be extracted using onboard sensors of agents. However, as we will later explain in an ablation study, using information from only the current time step as observation is not enough to learn to predict upcoming dangers efficiently. As a result, we define the observation based on a history of states and actions as follows:

$$O_{i,t,\mathcal{T}} = (o_{i,t,\mathcal{T}}, o_{i,t-1,\mathcal{T}}, o_{i,t-2,\mathcal{T}}, o_{i,t-3,\mathcal{T}}, a_{i,t-1}, a_{i,t-2}, a_{i,t-3}) \tag{3.4}$$

where $a_{i,t}$ is the action of agent i at time t and $o_{i,t,\mathcal{T}}$ is defined by

$$o_{i,t,\mathcal{T}} = (x_{i1}, x_{i2}, \ldots, x_{i,i-1}, x_{i,i+1}, \ldots, x_{iN}) \tag{3.5}$$

The action space of each task MDP with N agents defines as follows:

$$A_{t,\mathcal{T}} = (a_{t,0},\ \ldots,\ a_{t,N}) \tag{3.6}$$

where the action of each individual agent defines as

$$\begin{aligned} a_{t,i} \in \mathcal{U}_i = \{ \\ 0 : \text{default-controller}, \\ 1 : \text{turning-right}, \\ 2 : \text{turning-left} \\ \}. \end{aligned} \tag{3.7}$$

In (3.7), “default-controller” means the agent is allowed to execute any default controller, such as a goal-reaching controller. This controller can be different for each agent inside a task. The other two classes of actions refer to avoidance actions that agents should execute to prevent safety violations with the other agents. Taking inspiration from reachability theory [75], the least restrictive controller lets agents execute their own default controllers unless some upcoming danger is inevitable, in which case agents should avoid danger with their maximum actuation capacity. As a result, the avoidance actions are translated to

$$\begin{aligned} \text{turning-right} &:= [v_{max},\ \omega_{max}], \\ \text{turning-left} &:= [v_{max},\ -\omega_{max}]. \end{aligned} \tag{3.8}$$

where v and ω are the linear and angular velocities respectively.

To handle the varying size of observation space in (3.4) across different tasks, we use an encoder part of an LSTM-based VAE to encode the observation to a fixed-size latent space. The VAE consists of two parts: an LSTM-based encoder network $q_\phi(z_{i,t}|o_{i,t,\mathcal{T}})$ parametrized by ϕ, which encodes the observation o (3.5) to a fixed size latent space; and an LSTM-based decoder network $q_\phi(o_{i,t,\mathcal{T}|z_{i,t}})$ parametrized by ϕ. Given latent variables $z_{i,t}$, we define the fixed-size augmented latent state as follows:

$$Z_{i,t} = (z_{i,t}, z_{i,t-1}, z_{i,t-2}, z_{i,t-3}, a_{i,t-1}, a_{i,t-2}, a_{i,t-3}) \tag{3.9}$$

3.2.2 Reward Engineering

In a multi-agent system with distributed control, where each agent acts independently, each agent needs a notion of safety margin with respect to other agents. While a naive distance function is widely selected as the base metric for setting up a reward function, we instead propose using a value function derived from the HJ reachability. It encodes the degree of safety as a function of all relative state variables rather than just relative position, as defined in (3.3). HJ reachability also accounts for the system's dynamics. We calculate the time-invariant safety value function $R_{ij}(x_{ij})$ as described in (3.2) to (2.28) and we define the reward function of agent i as follows:

$$r_{i,t,\mathcal{T}}(o_{i,t,\mathcal{T}}, a_{t,i}) = \begin{cases} -C, & \text{if } \forall j, R_{ij} \geq 1 \text{ AND } a_{t,i,\mathcal{T}} \neq 0 \\ k \times \min_j(R_{ij}), & \text{if } \forall j, \exists R_{ij} \leq 0 \\ -K, & \text{if collision with any agent } j \\ K, & \text{finish the task without collision} \end{cases} \tag{3.10}$$

where $j = 1, ..., N, i \neq j$ and k is a constant factor. This reward function is derived from the following logic. The first row implies that when there is not any possible danger (all safety values are greater than 1), the agent will receive a big punishment (C) if it does not follow its default controller. The second row implies that if the pairwise safety between the agent and other agents is less than zero, then the reward will be the most negative safety value across all agents j with a scaling factor k. The other parts of (3.10) are the sparse rewards (K) that the agent will receive at the end of the episode depending on whether it finishes its own task or collides with another agent. We are not considering any continuous reward if all safety values are in the range $(0, 1]$. However, since we are defining discount factor γ in our MDP definition, decreasing the discount factor motivates agents to finish their tasks sooner.

3.2.3 Learning Algorithm

In this section, we explain how we combine our proposed LSTM-VAE with a sample efficient RL algorithm, and how we train them all in a meta-training loop. Our meta RL makes the learning procedure sample efficient in both training and adaptation loops. However, efficient off-policy meta-RL has two main challenges [86]. First, the same distribution of data should be used for both meta-training and meta-testing. This indicates that since the meta-testing phase is usually done on on-policy data, the meta-training is also should be done on on-policy data as well. Second, the meta policy should be able to reason about the distribution of experience by optimizing the distribution of visited states. This means the policy gradient-based methods are more appealing for meta-reinforcement learning.

To remedy these challenges, we propose using Proximal Policy Optimization (PPO) as a more sample-efficient actor-critic method [96] and we meta-train our algorithm using MAML [33]. The PPO algorithm enables multiple epochs of update from the replay buffer and is more sample efficient compared to other on-policy methods. Moreover, directly learning a policy can be more effective in stochastic exploration rather than deriving the policy from a value network, as is done in value-based methods. The LSTM VAE is meta-trained via MAML [33] across different tasks using reparameterization trick by maximizing the resulting Evidence Lower Bound (ELBO) [50]:

$$\mathbb{E}_{O\sim\mathcal{T}}[\mathbb{E}_{z\sim q}[\log p_\phi(O|z)] - D_{KL}(q_\phi(z|O)\ ||\ p(z)) \tag{3.11}$$

where $p(z)$ is a prior distribution over z. To prevent LSTM blocks from forgetting critical data, for the observation of each agent i in (3.5), we sort the (x_{ij}) in ascending order of HJ safety value function R_{ij}, and feed them to LSTM block such that the relative state of most critical agent – the agent j with minimum R_{ij} – is fed last. The critic is meta-trained via the following bootstrap update rule:

$$L_{critic} = \mathbb{E}_{s,r,s'\sim\mathcal{T},Z\sim q_\phi(z|s),Z'\sim q_\phi(Z'|s)}[Q_\psi(Z) - (r + \gamma\bar{Q}_\psi(Z'))]^2 \tag{3.12}$$

where $\bar{Q}$ is the target network. The policy which predicts the categorical distribution over actions defined in (3.7) is meta-trained using the following clipped surrogate loss with KL penalty term:

$$\begin{aligned} L_{policy} = \mathbb{E}_{s,r\sim\mathcal{T},Z\sim q_\phi(Z|s)}[\min(r_t(\theta)\bar{A}_t, \text{clip}(r_t(\theta), 1-\epsilon,\\ 1+\epsilon)\bar{A}_t - \beta D_{KL}(\pi_{\theta_{old}}(.|Z)||\pi_\theta(.|Z))] \end{aligned} \tag{3.13}$$

where the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|z_t)}{\pi_{\theta_{old}}(a_t|z_t)}$, ϵ is the clipping hyperparameter value, β is a constant coefficient, D_{KL} is shorthand of KL-divergence between new and old policy and $\bar{A}_t$ is the Generalized Advantage Estimation (GAE) which is computed as in [95]:

$$\begin{aligned} \bar{A}_t = \delta_t + (\gamma)\delta_{t+1} + ... + (\gamma)^{T-t+1}\delta_{T-1}\\ \text{where } \delta_t = r_t(s,a) + \gamma Q_\psi(Z_{t+1}) - Q_\psi(Z_t) \end{aligned} \tag{3.14}$$

3.3 Simulated and Real-World Experiments

In this section, we will explain the implementation, training and performance of our proposed method in both simulation and experimentation. We compare our method with a classical sub-optimal approach and evaluate the specific choice of our design through an ablation study. It should be noted that since the LR-CAM objective is to be least restrictive by only interrupting the default controller to avoid danger, it cannot be compared with previous collision avoidance algorithms such as ORCA [117] that solve the goal-reaching and avoidance simultaneously. As a result, we chose the method presented in [12] as the classical baseline because similar to LR-CAM it tries to maintain safety by interrupting the default controller.

3.3.1 Implementation

Since there are no standard benchmark tests set for multi-agent RL, we created our own simulation environments using the Gazebo and ROS open-source software. We created four different tasks, with three to six agents working simultaneously. We considered a default controller for each agent, in all the aforementioned four tasks, with the objective of reaching a specific location within the environment or simply following a path. To show the performance of our algorithm in the presence of static obstacles in the environment, we make the agents stationary under two scenarios: (1) we randomly freeze some of the agents, or (2) we let agents stop and stay in the environment when they finish their default tasks.

In simulation We localize agents based on their odometry data and, in real experiments, we localize agents using their on-board sensors. We make the localization information available to all other agents. To make the simulation/training runs a better reflection of reality, we also add white noises to all states. The LR-CAM module is implemented in TensorFlow 2.0. For real-world experiments and simulations, we used the TurtleBot3 Burger robots which are equipped with an onboard 360-degree 2D LiDAR and Gyro (Fig. 3.1-a). We also upgrade their raspberry pi development board to Nvidia Jetson Nano to increase their computational capacity.

To calculate the reward function, we approximate the Turtlebot dynamic with simple Dubins car, where the pairwise relative dynamic of (3.2) is written as the following planar kinematic model:

$$\begin{aligned} \dot{p}_{x,ij} &= -v + v\cos(\theta_{ij}) + \omega_i p_{y,ij} \\ \dot{p}_{y,ij} &= v\sin(\theta_{ij}) - \omega_i p_{x,ij} \\ \dot{p}_{\theta,ij} &= \omega_j - \omega_i, |\omega_i|, |\omega_j| \leq \bar{\omega} \end{aligned} \tag{3.15}$$

For training and testing, we consider moderate and difficult scenarios. In the former, initial and target locations for each agent are randomly selected between two concentric circles with a radii r_1 and r_2 ($r_1 \leq r_2$), respectively. Then, we add random perturbation to the agents' initial states. Under this setting, one agent usually comes into conflict with a maximum of two other agents. In the latter, however, agents' locations are randomly initialized around a circle with a radius r and their objectives are chosen such that agents would pass through the center of this circle to reach their goal locations. Under this scenario, each agent can be in conflict with two or more agents. To evaluate the performance, we calculate the success rate to be the fraction of agents that succeeded in reaching a goal location without colliding with other agents over 100 trials for each scenario.

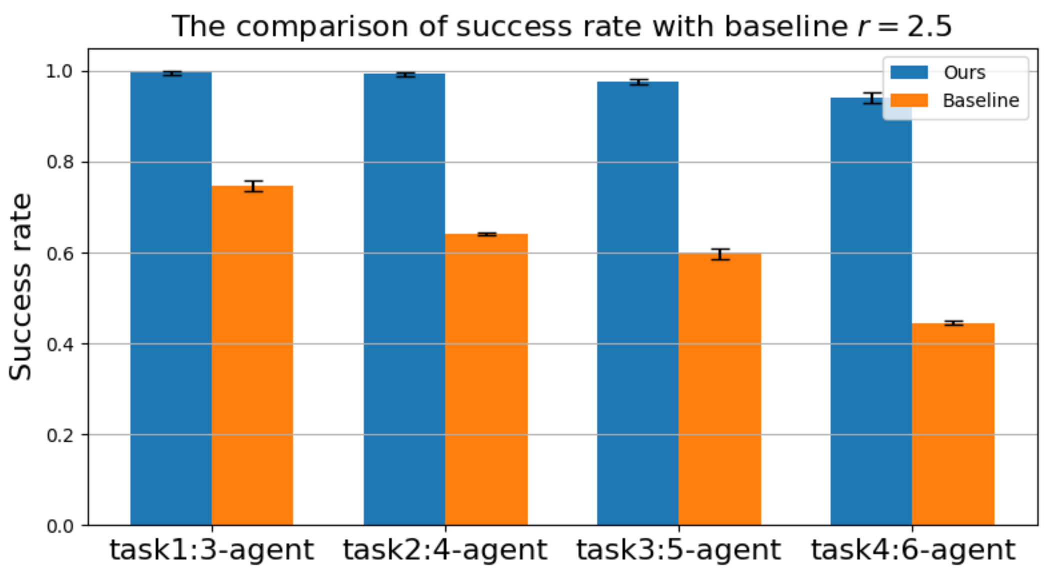


Figure 3.2: Comparison with Baseline

3.3.2 Results

Fig. 3.2 compares the success rate of our proposed approach with baseline across different tasks in a difficult test case scenario. To calculate the variance of the success rate we repeat each success rate calculation 5 times. As the number of agents in a task increased, the success rate is decreased for both algorithms; however, the variation is negligible for our method (more than 90 percent success rate for all cases). In addition, our proposed method outperforms the baseline approach at least by 30 percent in all cases. we also evaluate the performance of our proposed method in the case in which only some of the agents use LR-CAM. To this end, we designed the following experiment. First, in a 5-agent task, we start with no agent and increase it all the way to all agents using the LR-CAM. Results for each case in visualized in Fig. 3.3. At each point, the red graph indicates the success rate for the portion of agents that don't have LR-CAM, the green shows the average success rate for all agents and the blue one shows the success rate for the agents with LR-CAM. The results indicate that even only if some of the agents use LR-CAM, the success rate for agents who use LR-CAM and also for the rest of the agents increased significantly. We also tested our approach in an unseen environment involving $N = 8$ agents. The first row of Table 3.1 shows the success rate in a test case where $N = 8$ in (3.5). In the second row, we only feed each agent the six lowest safety values while ignoring the remaining 2 safety values. Fig. 3.4 and Fig. 3.5 show trajectory visualization for a 6-agent task and an 8-agent task respectively. In Fig. 3.4, we chose a crowded initial condition with $r = 1.5$ and in Fig. 3.5, agents are initialized a bit farther from each other. In both figures, the agents' trajectories are color coded with the opacity proportional to the elapsed time.

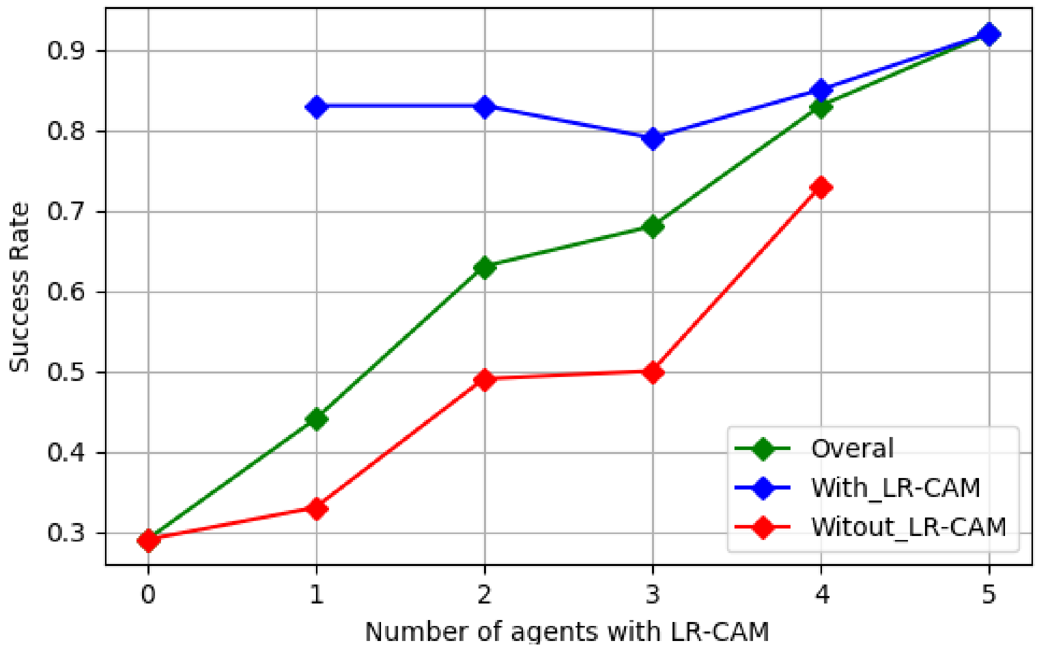


Figure 3.3: comparison of success rate when a subset of agents in a difficult-task scenario(with $r = 1.7$) uses LR-CAM.

To visualize how exactly LR-CAM intervenes the default controller we overlay yellow dots inside the colored tubes for all time steps in which LR-CAM interrupts the default controller to perform avoidance actions. Based on these trajectories, when the agents are in dangerous configurations, LR-CAM takes over control and, when the safety value is relatively high, it will let the agent execute its default controller. Fig. 3.1-b shows the trajectories of four TurtleBots in a real-world 4-agent task. Additional experiments involving two to four agents can be found at `https://bit.ly/34K8YKB` .

Table 3.1: Performance of LR-CAM in an unseen task with 8 agents

Test Case	Success Rate
Observation Space with all agents	0.85
Observation space with first six critical agents	0.91

3.4 Ablation Studies

Reward Design. To demonstrate the benefits of our reward function, we meta-trained a separate policy using a naive distance-function-based reward. In this setting, we modify

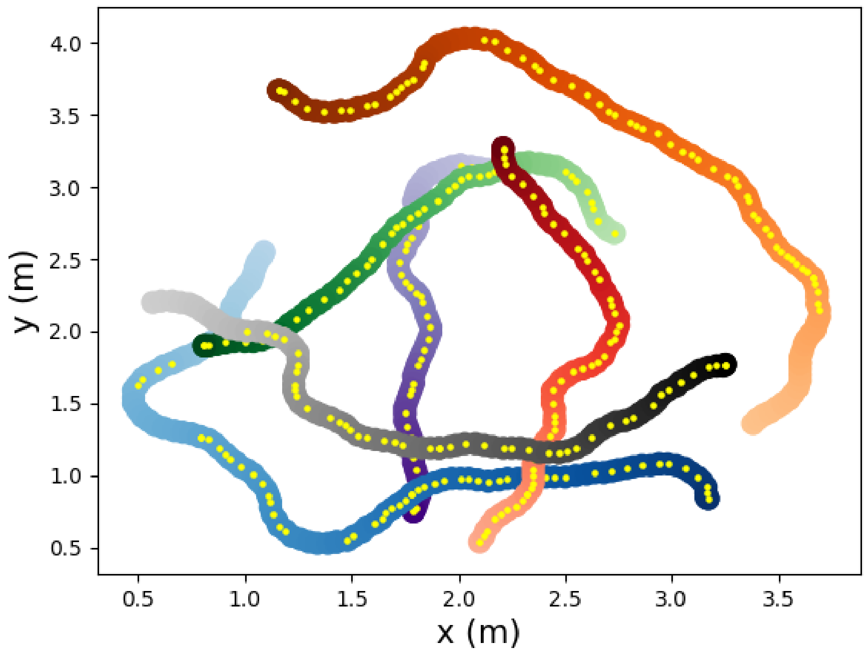


Figure 3.4: Trajectory visualization for 6-agent task in a crowded initial condition ($r = 1.5$). Agents are color coded and the level of opacity in the figures is proportional to the elapsed time. Yellow dots inside tubes indicated and all the time-steps that the LR-CAM of agent is interrupting the default controller.

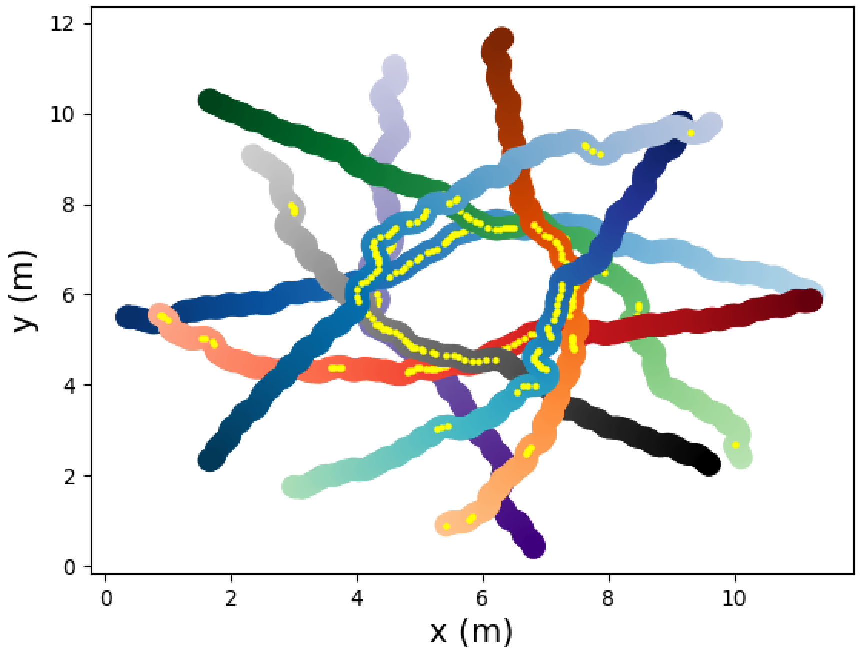


Figure 3.5: Trajectory visualization for an unseen 8-agent task in a long range initial condition ($r = 5.5$)

(3.10) using the Euclidean distance ($L2_{ij}$) instead of the safety value function (R_{ij}) as:

$$r_{i,t,\mathcal{T}}(o_{i,t,\mathcal{T}}, a_{t,i}) = \begin{cases} -C & \text{if } \forall i, L2_{ij} \geq 1 \text{ AND } a_{t,i,\mathcal{T}} \neq 0 \\ k \times \min_j((L2_{ij} - d)) & \text{if } \forall j,\ \exists L2_{ij} \leq d \\ -K & \text{if collision with any agent } j \\ K & \text{finish the task without collision} \end{cases} \tag{3.16}$$

where d is the collision radius defined in (2.51). Table 3.2 shows the comparison of success rate for using the HJ-based reward (first row) and the distance-based reward (second row) for difficult test scenarios with $r = 1.7$. We observed that not only did the success rate improve by approximately 10%, but also the training time reduced by 25% when using our proposed HJ-based reward. To compare how LR-CAM evaluates the safety and upcoming

Table 3.2: Ablation study for reward design - success rate comparison

Algorithm	4-agent	5-agent	6-agent
Ours with HJ reward	0.99	0.93	0.91
Ours with naive distance reward	0.89	0.86	0.75

dangers, we define the “restrictiveness factor” to be the fraction of interrupting actions over all actions in a total of 100 trials for each scenario/task. As it can be seen from Table 3.3, using our proposed HJ-based reward improved the restrictiveness factor by 10%, which means that our proposed reward is 10% less restrictive and allows the default controller to be used more often.

Table 3.3: Ablation study for reward design - restrictiveness comparison

Algorithm	4-agent	5-agent	6-agent
Ours with HJ reward	0.46	0.45	0.48
Ours with naive distance reward	0.54	0.54	0.48

Chapter 4

Observation-Based Collision Avoidance Module

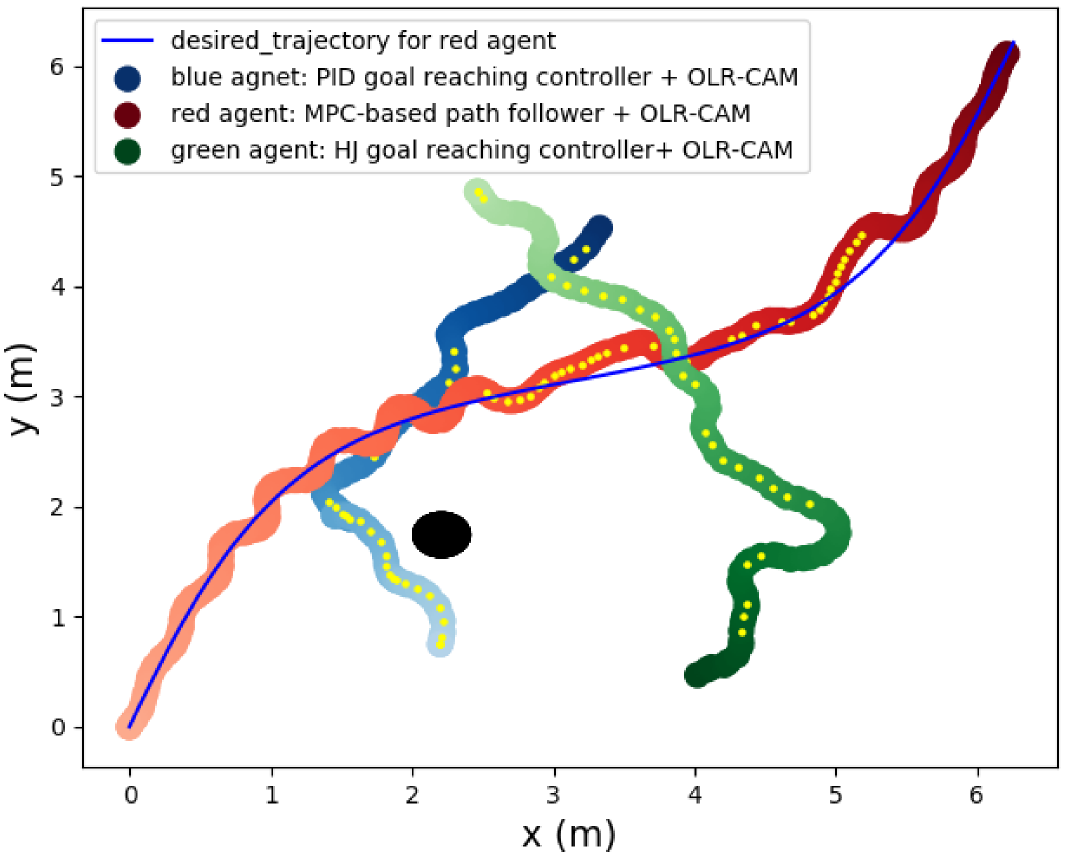


Figure 4.1: OLR-CAM application: Yellow dots inside agents trajectories show OLR-CAMs are intervening to safely guide three agents with different default controllers to navigate to their goals without collisions, as explained in Section. 4.4

Using multi-agent systems for automating tasks and services has gained attention recently. However, having a generalized algorithm that can be used in a wide range of tasks and environments remains a challenge. Difficulties arise when agents have different objectives to pursue and controllers to execute them. Furthermore, safe navigation in the presence of both dynamic and static obstacles without having prior knowledge about the global map of the environment and other agents' trajectories renders itself as a difficult problem both in the reinforcement learning (RL) and control domains. Following our previous research [4], our motivation is to design a fully distributed Observation Based Least Restrictive Collision Avoidance Module (OLR-CAM) that can accept any default controller or policy. The term "least-restrictive" is commonly used in safety-critical systems in the control community [16, 47, 20, 22]. It refers to a minimally restrictive control layer that allows agents to execute all possible control inputs that are guaranteed not to violate the defined constraints [47]. The least-restrictive set of safe control actions usually is determined by calculating a maximally-safe control set and a least-restrictive policy ensures that control inputs/actions will remain inside this set [20]. Since the agents in a multi-agent system may have non-linear dynamics, the environment and safety constraints can be non-convex [47], making calculating the maximally-safe control set computationally expensive (an NP-hard problem in collision avoidance cases) [47, 87]. A number of previous algorithms such as [87, 76] attempt to explicitly calculate the maximal set only for a small number of agents. However, the computational complexity increases exponentially with the number of agents [47, 14]. Our learning-based method aims to pave the way toward addressing these challenges. Fig. 4.1 shows the application of our least-restrictive policy in a multi-agent system. Three agents with different objectives and controllers were moving within a shared environment without any avoidance capability. The OLR-CAM is wrapped around the existing policy of each agent and leads to safe navigation for all.

In this section, we present the *observation*-based version of our LR-CAM, OLR-CAM, which takes direct sensory observations as input to provide a high-level safety layer to any existing default policy for agents. At each time step, the OLR-CAM takes an input LiDAR observation of the environment and evaluates whether the agent is safe with respect to nearby dynamic and/or static obstacles, relative to the ego-agent (self-agent). If the situation is safe, OLR-CAM will allow the agent to execute its default controller; otherwise, it will take over the control of the agent and attempt to resolve the potential conflicts with other agents or obstacles. OLR-CAM maintains safety by only interrupting the default policy, which can be executed by any classical or learning-based controller. In contrast to our previous work, LR-CAM, the OLR-CAM maintains safety in the presence of both dynamic and static obstacles.

To meta-train the OLR-CAM under a wide range of different environments, we designed a "2D Navigation Meta World" simulation environment and designed our novel reward function using Hamilton-Jacobi (HJ) reachability notion for effectively teaching the policy a rich and informative notion of safety margins. Our reward function, in contrast to traditional applications of HJ reachability, also leverages the local cost map data and remedies the need for prior knowledge about the environment and obstacles' positions, which allows robust tuning of the policy to adapt to any unknown environment at the meta-test stage.

We used Model Agnostic Meta-Learning (MAML) [33] to meta-train our algorithm. Most optimization-based meta-learning approaches that use MAML are on policy; however, the ORL-CAM uses an off-policy actor-critic algorithm to improve sample efficiency. To the best of our knowledge, the proposed approach is the first learning-based least-restrictive algorithm that leads to collision avoidance with other dynamic agents and/or static obstacles using raw sensory observations as input without having prior knowledge about the shape, or motion of other agents and/or obstacles.

The main contributions of this section are as follows:

- We extend our previous collision avoidance algorithm, LR-CAM, to directly use raw LiDAR observations as input and to consider both dynamic and static obstacles in the environment.
- We propose a new reward function, which uses local cost map data and a safety value function derived from HJ reachability theory to avoid collision with obstacles in any unknown environment.
- We design a 2D navigation Meta World simulation to meta-train a multi-agents system in random indoor environments with obstacles.
- We implement and show that the proposed approach achieves zero-shot sim2real transfer on a real multi-robot system.

4.1 Related Work

4.1.1 Classical Collision Avoidance

In [80], the structural potential function was used for multiple agents with a double-integrator dynamics to generate a collision-free path in a shared environment. Similarly, [17], used the potential field to generate a repulsive force for pairwise interactions; however, incorporated more complex dynamics into agents' behaviours. The authors in [35] proposed using the concept of Velocity Obstacles (VO) via selecting a set of avoidance maneuvers to avoid both static and dynamic obstacles in the velocity space. They assumed that agents' future behaviour could be predicted, using their known dynamic models, to calculate the VO. In VO-based methods, Optimal Reciprocal Collision Avoidance (ORCA) is known to be one of the most celebrated state-of-the-art collision avoidance classical approaches [115, 102]. In ORCA, each agent calculates the pairwise velocity obstacles and the sets of collision avoidance velocities induced by other agents. Via a linear programming, the algorithm adjusts the *preferred-velocity* to avoid the collision. They assume that each agent has perfect knowledge about other agents' shape and their position and velocity. Their algorithm remains susceptible to uncertainties, though. Authors in [125] proposed a piece-wise motion planning inside Buffered Voronoi Cells (BVS) within a receding horizon for collision avoidance problem. Compared to that in VO-based approaches, the BVS methods do not need velocity information of other agents. They also show better robustness to uncertainties. However, they assume a priori known dynamics for all the agents.

The collision avoidance problem is also widely studied in safety-critical systems using a differential game framework and HJ-reachability theory [75, 36]. However, these studies focus on small-scale problems due to the HJ-intractability computational burden as the number of agents increases. To mitigate the intractability, authors in [12] proposed a decision-based approach using Mixed Integer Programming (MIP) to reduce the problem into pairwise collision avoidance. Note that the MIP-HJ can be considered the state-of-the-art classical least restrictive multi-agent collision avoidance in the literature that can guarantee collision-free movement for three agents. In MIP-HJ, a centralized controller decides whether the agents can execute their default controller or be interrupted. As we showed in [4], the performance of our approach is at least 30 percent better. As a result, we didn't include the same study in this work.

4.1.2 Learning Based Collision Avoidance

While the Deep Reinforcement Learning (DRL) methods showed high performances in some application domains, solving the multi-agent collision avoidance problem still remains a challenge. Authors in [15] trained a pair-wise collision avoidance policy for a two-agent problem. In [27] they extended their initial work and trained a policy that enabled a single agent to have a collision-free path among other moving agents. To achieve this, they assumed that the relative position and velocity and the occupied space by other agents could be measured and be available to the learning agents. Authors in [28], proposed a decentralized trained policy capable of mapping raw sensory observations to the desired control action that would navigate the agent to its goal safely while avoiding obstacles. In comparison to this method, our approach only interrupts the default policy. It is also meta-trained in various dynamic environments using a novel reward function that does not need any information about the environment's states, making it desirable to adapt to any new environment in the meta-test phase. In our earlier work [4], we took inspiration from HJ reachability and proposed a least-restrictive policy that only interrupted the default controller to solve the collision avoidance navigation problem. This method took a history of the relative positions of all the agents as input and encoded them into a fixed-sized latent space, and a meta-trained policy was then used to predict the next action. While the method can handle a reasonable amount of uncertainty associated with estimating agents' positions, it still heavily relied on the agents' relative position information. Moreover, only dynamic obstacles were considered.

4.2 Method

We propose a meta RL algorithm to find a distributed optimal least-restrictive policy. Our algorithm is built on top of the MAML [33] with an informative reward function built upon HJ formulation to expedite the training procedure.

4.2.1 Least Restrictive Control Policy

The least-restrictive policy responsibilities in OLR-CAM are as follows: 1) Observe and detect potential dangerous joint configurations of the ego-agent, other dynamic agents, and static obstacles using raw sensory observations, 2) Allow the ego-agent to follow its own objective as long as it is safe to do so, and 3) In case of an upcoming danger, take control of the agent and navigate safely. In our previous research [4], we adopted three discrete actions:

$$\begin{aligned} a_t \in \mathcal{A} = \{&\text{action-1 : default-controller}, \\ &\text{action-2 : turning-right}, \\ &\text{action-3 : turning-left}\} \end{aligned} \tag{4.1}$$

Since the optimized behaviour would be to interrupt as least as possible, we assume the policy will use its maximum actuation capacity to prevent collision and translate the turning actions to

$$\begin{aligned} \text{turning-right} &:= (v_{\max},\ \omega_{\max}), \\ \text{turning-left} &:= (v_{\max},\ -\omega_{\max}). \end{aligned} \tag{4.2}$$

We also take the above approach in this letter. Having the defined action space, the OLR-CAM policy predicts three probability values for each class of (4.1), and the action that renders the maximum probability is chosen.

4.2.2 Reward Shaping

An informative reward function that is not only efficient for optimizing the agents' behaviour but also is easy to compute and interpret [24] is of paramount importance. As we showed in [4], using a reward based on a value function derived from HJ Reachability significantly reduced the training time, increased the success rate, and decreased the restrictiveness factor compared to that when using a reward based on the naive distance function. We define the restrictiveness factor to be the ratio of the number of avoidance actions taken (action-2 and action-3 in (4.1)) to the total number of actions. The least restrictiveness strives to minimize this ratio. In this letter, we modify our reward function design to take observations, rather than internal states, as input, and to account for both static and dynamic obstacles. To this end, we propose a systematic approach using the local cost map data to calculate the reward function in any random environments in the meta-world without knowing the global map or obstacles positions. Our reward design has three main parts:

$$r_t(s_t) = k_d \times r_{\text{dynamic-agents}} + k_s \times r_{\text{obstacles}} + r_{\text{sparse}} \tag{4.3}$$

where k_d and k_s are constant scaling factors. The sparse part is given at the end of the episode for each agent. The agent will receive a high positive reward for finishing its task and a big punishment for any safety violation that results in a collision. The episode for an agent will end if it violates safety or if it finishes its task within episode maximum time limit.

The dynamic-agent part of the reward is introduced and directly used from our previous work:

$$r_{\text{dynamic-agents}} = \begin{cases} -5 \quad \text{if } \forall j, R_{ij} \geq 1 \text{ AND } a_{t,i,\mathcal{T}} \neq \text{action-1} \\ k \times \min_j(R_{ij}) \qquad \text{if } \forall j, \exists R_{ij} \leq 0 \end{cases} \tag{4.4}$$

where R_{ij} is the time-invariant safety level value function defined in — and k is a constant scaling factor. This reward definition follows a simple logic. The first part implies that if all the joint configuration between the ego-agent and other agents is safe, the agent will receive a negative reward only if it does not follow the default controller. The second part aims to teach the agent how much margin it has from dangerous configuration until the collision. For all other configurations, the agent will receive a zero reward.

To produce a robust reward function for all environments (restricted to 2D navigation only), we built our method based on local cost map data and the safety level value function described earlier. The local cost map [38] can be obtained using online observations of the agent and provides compact information about the relative tracked position of obstacles in a grid-map format associated with an occupancy probability for each grid cell. Note that this cost map is locally observed and constructed and is therefore different from a global cost map that may require a SLAM algorithm to obtain [38]. To process the incoming cost map data, we first apply a non-Max suppression filter to thicken and make obstacle layers more sparse. Next, we calculate the time-invariant safety level value function for static obstacles ($R_i(x_i^r, \theta_i)$) for each occupied data point in the filtered cost map. It should be noted that any data point in cost map (x_n, y_n) can be mapped to a relative 2D position with respect to the agent as:

$$x_n^r = \begin{bmatrix} x_n \\ y_n \end{bmatrix} \times \text{res} \tag{4.5}$$

where res stands for cost map resolution. Finally the $r_{\text{obstacles}}$ is calculated by:

$$r_{\text{obstacles}} = \sum_{n=1}^{N} R_i(x_n^r, \theta) p(n) \delta(R_i(x_n^r, \theta)) \tag{4.6}$$

where N denotes the total number of cells in cost map, $p(n)$ denotes the probability of occupancy for cell n and $\delta(\cdot)$ is a binary function given by:

$$\delta(s) = \begin{cases} 0 & \text{if } s \geq 0 \\ 1 & \text{if } s < 0 \end{cases} \tag{4.7}$$

Like the dynamic part of the reward, we only give negative rewards for unsafe configurations, and safe scenarios are rewarded with zero. Fig. 4.2 shows the process of reward calculation using cost map data for a sample scene.

4.2.3 Meta-RL algorithm

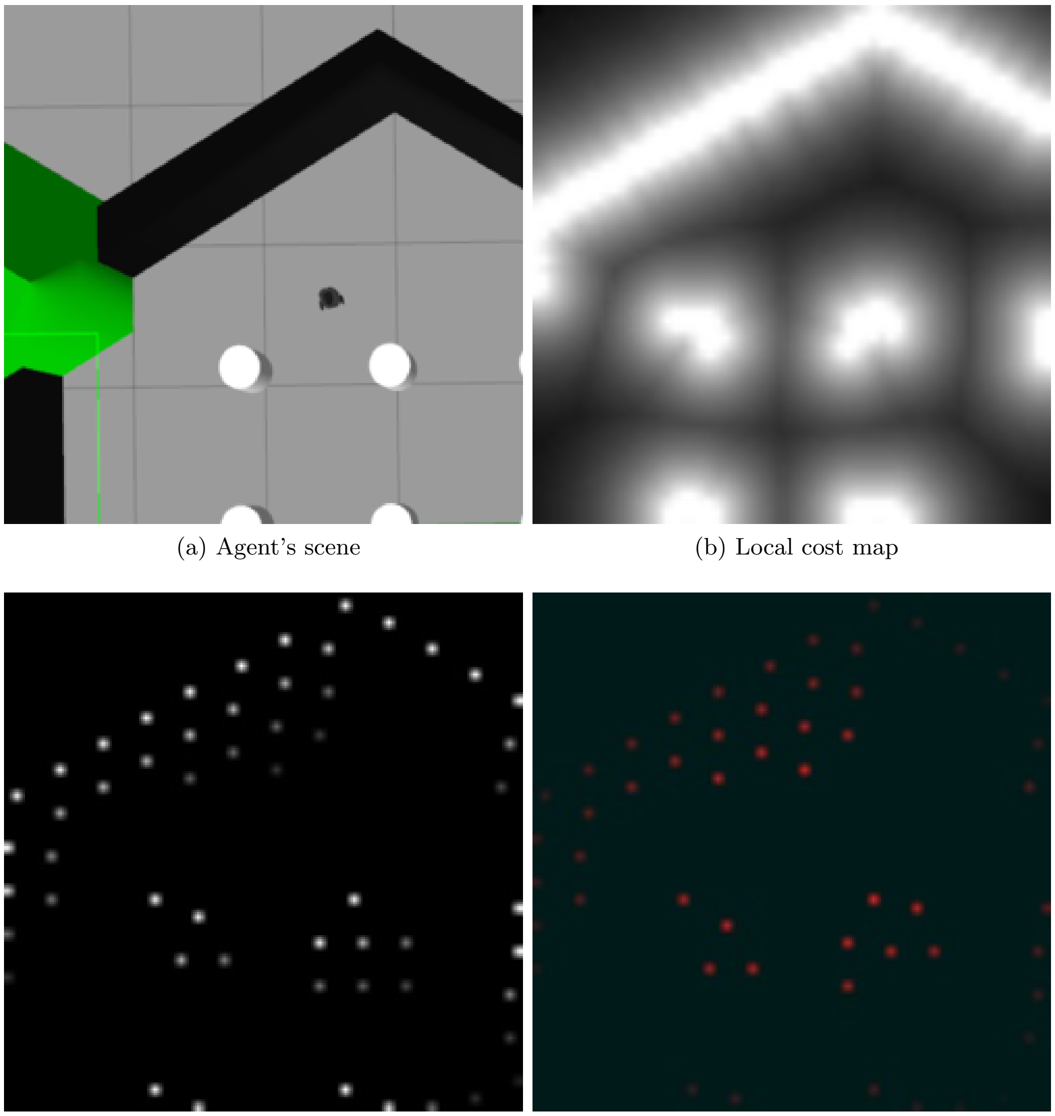
As we described earlier, we assume a distributed control structure in which an agent does not have any communication with other agents and only makes decisions based on its observation, without having access to other agents' pose and actions. To help OLR-CAM detect potential conflict with both dynamic and static obstacles, we propose using a history of observations and actions as the input of the OLR-CAM as:

$$o_t = (L_t, L_{t-1}, ..., L_{t-\mathbf{T}}, a_{t-1}, ..., a_{t-\mathbf{T}}) \tag{4.8}$$

where L_t is the raw LiDAR data at time t and $\mathbf{T}$ denotes the history length.

To improve the sample efficiency of our learning process, we use the Proximal Policy Optimization (PPO) [96] algorithm and meta-trained it using MAML [33]. Fig. 4.3 shows the architecture of both actor and critic networks. To prevent forgetting of the history of actions due to significant dimensional differences compared to LiDAR observations, we first feed the history of Lidar data to multiple 1-D convolution layers with Rectified Linear Activation (ReLU). After encoding the data to a lower dimension, we concatenate it with the history of actions and feed the result to Fully Connected layers and predicted the output. We train the critic with the following loss:

$$L_{\text{critic}} = \mathbb{E}_{o_t, r, o_{t+1} \sim \mathcal{T}}[Q_\psi(o_t) - (r + \gamma \bar{Q}_\psi(o_{t+1}))]^2 \tag{4.9}$$



(a) Agent's scene

(b) Local cost map

(c) Filtered cost map

(d) Safety map

Figure 4.2: The procedure of reward calculation from local cost map. (a) shows the sample view of agent's environment, (b) shows the inflated local cost map with res = 0.05, (c) shows filtered cost map by NonMax suppression algorithm for $w = 5 \times 5$ where w is the filtered window size, and (d) shows the calculated safety values for filtered data point. In (d) the level of opacity in red color is proportional to the magnitude of safety value

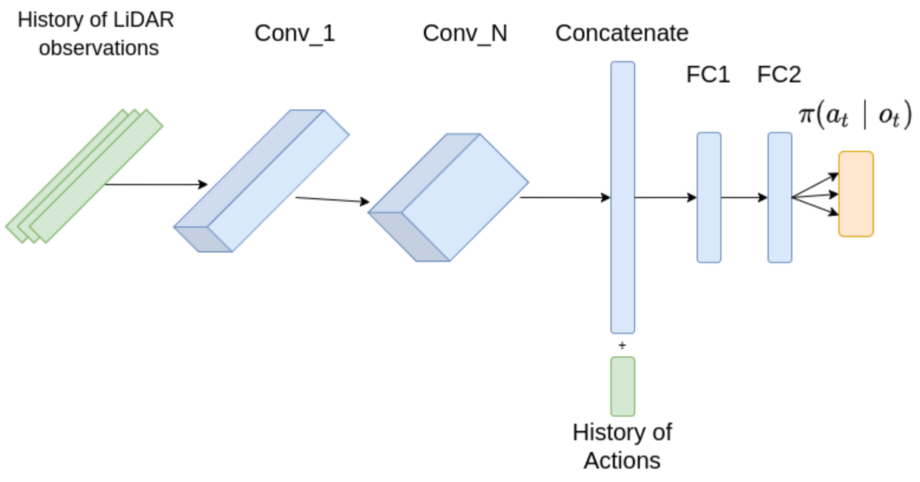


Figure 4.3: Proposed OLR-CAM architecture

where $\bar{Q}$ is the target network and ψ represents the critic parameters. We use the following clipped surrogate loss with KL term to train the actor:

$$\begin{aligned} L_{\text{actor}} = \mathbb{E}_{o,r\sim\mathcal{T}}[&\min(\lambda_t(\theta)\bar{A}_t, \text{clip}(\lambda_t(\theta), 1-\epsilon, 1+\epsilon)\bar{A}_t \\ &-\beta D_{KL}(\pi_{\theta_{old}}(.|o)||\pi_\theta(.|o))] \end{aligned} \tag{4.10}$$

where $\lambda_t(\theta) = \frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{old}}(a_t|o_t)}$ is the importance sampling ratio, ϵ is the clipping hyperparameter value, β is a constant coefficient, D_{KL} is shorthand of KL-divergence between new and old policy, θ is the actor parameters and $\bar{A}_t$ is the Generalized Advantage Estimation (GAE) which is computed as in [95]:

$$\begin{aligned} \bar{A}_t &= \delta_t + (\gamma)\delta_{t+1} + ... + (\gamma)^{T-t+1}\delta_{T-1} \\ \text{where } \delta_t &= r_t(s,a) + \gamma Q_\psi(o_{t+1}) - Q_\psi(o_t) \end{aligned} \tag{4.11}$$

Alg. 3 illustrates the meta-training procedure. We only sample from a single task during the meta-test time and run a few training loops to adapt to that specific task.

Algorithm 3 OLR-CAM training loop

```
Require: initialize parameters $\phi, \theta$
    initialize reply buffers $\mathcal{B}^i$ for each training task
 1: while True do
 2:     for k=1, ...,$K$ do
 3:         for each task $\{\mathcal{T}_i\}$ do
 4:             for each agent in task $\{\mathcal{T}_i\}$ do
 5:                 Roll out policy $\pi_\theta(a_t|o_t)$ to collect data $\{o_t, a_t, r_t, o_{t+1}\}$ and add to $\mathcal{B}^i$
 6:             end for
 7:         end for
 8:     end for
 9:     for each task $\{\mathcal{T}_i\}$ do
10:         pull data from $\mathcal{B}^i$ and calculate GAE and update buffer with $\{o_t, a_t, r_t, o_{t+1}, \bar{A}_t\}$
11:     end for
12:     for step in PPO off-policy training steps do
13:         for $\mathcal{T}_i$ do
14:             sample mini batch $b^i \sim \mathcal{B}^i$
15:             $\psi \leftarrow \psi - lr\nabla_\psi L_{critic}(b^i)$
16:             $\theta \leftarrow \theta - lr\nabla_\theta L_{actor}(b^i)$
17:         end for
18:     end for
19: end while
```

4.3 Training and Experiments

4.3.1 2D Navigation Meta World

Due to the lack of a standard 2D navigation simulation environment for meta training that can provide a robust sim2real transformation, we create our own meta world using Gazebo Simulation engine [52] and ROS [85]. In our simulation, we randomize the number of agents to be between 3 and 6 for training and up to 8 agents for testing. We used Turtlebot3 BURGER robots as our agents; each has a 2D LiDAR and Gyro. The position of the walls, obstacles, and agents' goals are randomized at the beginning of each episode. The agents' exact pose and local cost map data are only used for reward calculations, and goal positions are only provided to the default controller; the OLR-CAM has no knowledge of the agents' goals. We ran the Navigation Stack package for each agent in which each agent can have access to a wide range of ROS planners and controllers for the default controller. However, for the default controller *only during training*, we chose the globally optimal goal-reaching controller obtained by solving an associated HJ PDE; this controller is taking from [14], and does not perform any avoidance. Note that agents are always moving forward with maximum

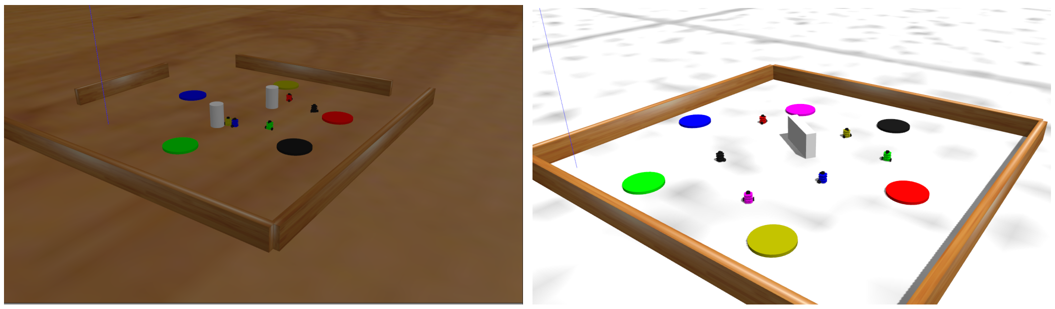

(a) (b)

Figure 4.4: Two random shots from proposed 2D navigation Meta World simulation. (a) difficult test case scenario with two obstacles (b) moderate test case scenario with one obstacle

velocity in both default controllers and interruption actions in training and evaluation. The simulation environment can run up to four parallel episodes simultaneously, and all of the environment processes such as sampling and reward calculations can be run in real-time using Intel® Core™ i9-10900 CPU and 32 GB RAM. Fig. 4.4 shows two sample views from the meta-world environment.

4.3.2 Implementation

The training algorithm is implemented using TensorFlow 2.0, and we ran training in simulation only. We tested our approach on simulation and a real multi-agent system, consisting of multiple TurtleBot3 Burgers. The real robots are equipped with onboard 360-degree 2D LiDAR and Gyro. Moreover, we equipped all of the robots with Nvidia Jetson Nano development boards. No additional training was done in the real world.

For reward calculation purposes, we approximate the agent's dynamic using Dubin's car model, and we rewrite (3.1) as:

$$\begin{bmatrix} \dot{p_x} \\ \dot{p_y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \tag{4.12}$$

where state $x = (p_x, p_y, \theta)$ represents the x position, y position and heading respectively, and control $u = (v, \omega)$ represents linear and angular velocities, and it is assumed $|\omega| \leq \omega_{\max}$. In the form of (4.12), the pair-wise relative dynamics of (3.2) is given by

$$\begin{bmatrix} \dot{p}_{x,ij} \\ \dot{p}_{y,ij} \\ \dot{\theta}_{ij} \end{bmatrix} = \begin{bmatrix} -v + v\cos\theta_{ij} + \omega_i p_{y,ij} \\ v\sin\theta_{ij} - \omega_i p_{x,ij} \\ \omega_j - \omega_i, \end{bmatrix} \tag{4.13}$$

where joint state $x_{ij} = (p_{x,ij}, p_{y,ij}, \theta_{ij})$ represents relative coordinate of j w.r.t. i, $v = v_{i_{\max}} = v_{j_{\max}}$ is linear velocity and ω is angular velocity in which $|\omega_i|, |\omega_j| \leq \omega_{\max}$. Having

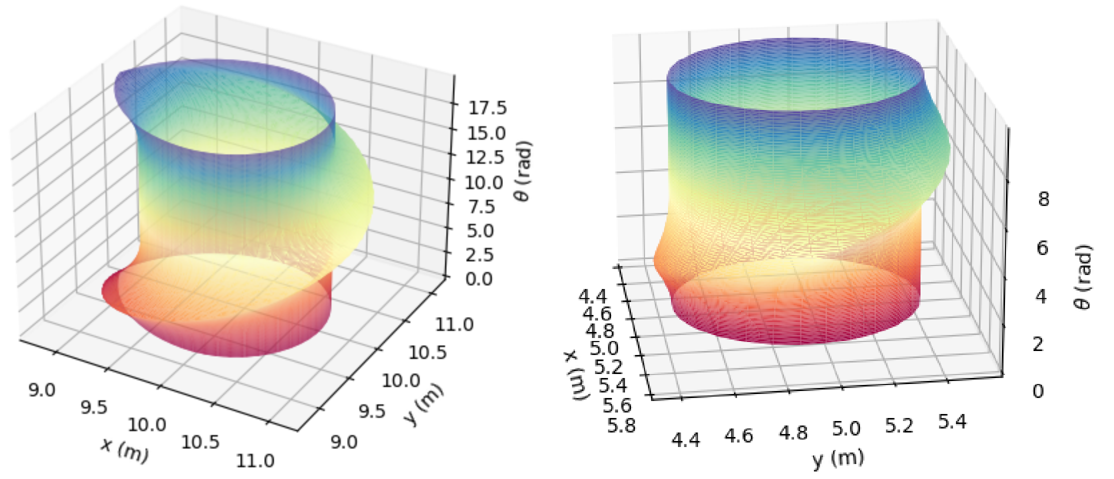


Figure 4.5: (Left) BRS for single agent-obstacle (Right) BRS for pairwise collision avoidance. Different colours indicate the different levels of orientation and relative orientation respectively for single agent-obstacle and pairwise collision avoidance.

the defined approximate dynamics, we calculate time-invariant safety value functions for dynamic and static agents offline and save them in a tabular format. During training, we interpolate the safety values from that table. Fig. 4.5 shows the BRS calculations for dynamic agents and static obstacles at the final time step $T = -5$.

4.4 Results and Discussion

This section will discuss the results of our proposed algorithm in both simulation and real-world experiments. As described, OLR-CAM does not carry the main navigation responsibility and only tries to maintain safety with the minimum amount of interruptions. As a result, it cannot be compared with most state-of-the-art classical or learning-based approaches as they are not least-restrictive. However, we compared our method with a well-known classical approach ORCA-DD [102]. ORCA-DD is somewhat similar to a least-restrictive policy since it changes the preferred velocities to prevent collision. We emphasize that our proposed policy's inputs and objective are different from the ORCA-DD in that we only interrupt the default controller and use direct LiDAR observations, while ORCA-DD requires position, velocity, radius information of all agents and obstacles. Since the Adaptive Monte Carlo Localization (AMCL) is a common localization technique in mobile robots [111], we added the minimum re-localization error of AMCL that is reported in [82] in terms of white noises ($N\ (0, 0.01)$) to the position data fed to ORCA-DD; this makes the test scenario more realistic. We also consider the LiDAR noises in our simulation. In our tests, the episode for an agent will end if it either reaches a goal or if it violates safety and collides with other agents of obstacles. In both cases, the agent will become stationary and remain in the environment. In this case, ORCA-DD will treat these agents as static obstacles. Table. 4.1 shows the implemented hyper-parameters of the baseline approach.

Table 4.1: Baseline Hyper-parameters

Parameter	Value
agents' radius: r	105 mm (real agent size) + 15mm
prediction horizon:τ	5 sec
effective radius:R	$2 \times r$

We consider two test case scenarios: moderate and difficult. In the moderate case, we have one static obstacle with random position other than environments' boundaries defined by solid walls, and agents are randomly initialized around a circle a with random radius range in $[r_1, r_2]$. The agents' objective is to reach to diagonally opposite side of the circle. In the difficult case, everything is the same except we have two obstacles with random positions. sample scenes from these scenarios is visualized in Fig. 4.4.

We define two metrics to evaluate our algorithm:

- Success Rate: The fraction of agents that succeed to reach the goal without safety violation over 100 trials for each task/scenario.

- Restrictiveness Factor: The fraction of velocity deviation from preferred velocity (baseline) or default controller (OLR-CAM) over all actions average in total of 100 trial for each scenario/task.

Fig. 4.1 shows the application and uniqueness of our proposed approach. In this experiment, we consider three agents each with its own unique default controller. The blue agent is running a PID goal-reaching controller, which turns and moves toward the goal. The green agent has an optimal goal-reaching controller derived from solving an HJ PDE which tries to move the agent inside the target using maximal actuation capacity and moving forward only. The red agent has an MPC-based controller that follows a desired trajectory visualized using the blue curve. Trajectories become darker as time goes on. In addition, we also randomly put an obstacle inside the navigation area. All agents have their own OLR-CAMs on top of their default controller, which acts independently and provides safe navigation for the agent that carries OLR-CAM by interrupting the default controller. To visualize how OLR-CAM takes control of the agent in potential conflicts, we overlay yellow dots inside trajectories for all time steps that OLR-CAM interrupts the default policy to maintain safety. Due to the vehicle's high speed and low frequency of controller (the controller frequency is bounded with observation frequency, which is 5 HZ), One can see the oscillatory behaviour for MPC, especially at early time steps. The red agent is initiated farther than the other two agents;

thus, the OLR-CAM does not interrupt the default controller until the middle of the path, which has a potential collision with the green agent. Importantly, the agent is not going far from the desired trajectory, and OLR-CAM interrupts its default policy the least. The blue agent is initiated near an obstacle with a potential collision; thus, the OLR-CAM begins to interrupt at the early time steps. Interestingly, OLR-CAM is successful at maintaining collision avoidance, even though we only meta-trained the algorithm with the HJ goal-reaching controller.

Table 4.2 shows the success rate of our proposed approach in the moderate test case compared to the baseline. For ORCA-DD we calculate success rates from two scenarios for each task. In first scenario we fed the prefect localization data to the baseline and for the second one we added a Gaussian white noise ($N(0, 0.01)$) to the localization data of other agents.

Table 4.2: success rate comparison in moderate test case scenario

Moderate task	3-agent	4-agent	5-agent	6-agent
Ours with noise	0.91	0.91	0.9	0.89
baseline without noise	0.9	0.9	0.89	0.86
baseline with noise	0.84	0.85	0.88	0.79

Table. 4.3 shows the performance of our algorithm on unseen task during meta-training phase with 8-agent in a moderate test case scenario with and without meta test phase.

Table 4.3: OLR-CAM success rate in unseen tasks during training for moderate test case scenarios

Task	8-agent
without meta test phase	0.62
with meta test phase	0.74

To evaluate the robustness of OLR-CAM performance in change of default controller, we repeated the moderate test case scenario experiments and replaced the default controller with a PID goal reaching controller without any additional training. Based on the results of this experiment in Table. 4.4, the OLR-CAM performance is not affected by the change of default controller.

Table. 4.5 compares the success rate of our proposed approach with baseline in the difficult test case scenario. For the first task with four dynamic agents and two static obstacles, our approach's performance is almost the same as the classical sub-optimal baseline. However, as the number of agents increased, the success rate decreased for baseline while remaining the same for OLR-CAM. Our proposed method outperforms the baseline by

Table 4.4: comparison of OLR-CAM success rate with two different default controller in moderate test case scenario

Moderate task	3-agent	4-agent	5-agent	6-agent
with HJ goal reaching	0.91	0.91	0.9	0.89
with PID goal reaching	0.86	0.89	0.9	0.87

10 percent on difficult scenarios. Note that in our evaluations, agents are always moving forward, and they are dynamic unless they reached their target. As a result, some configurations may not have a feasible solution. In addition, due to the complexity of computations in a multi-agent system, the feasibility of the initial configuration cannot be determined tractably [14].

Table 4.5: Comparison of success rate with baseline for difficult test case scenario

Difficult task	4-agent	5-agent	6-agent
Ours	0.76	0.76	0.74
Baseline	0.77	0.68	0.62

Table. 4.6 compares the restrictiveness factor of our algorithm with baseline. It can be inferred that OLR-CAM can evaluate the potential conflict from LiDAR observations more accurately and interrupt the preferred velocity significantly less than ORCA-DD. Table 4.7

Table 4.6: Comparison of restrictiveness factor with baseline for difficult test case scenario.

Difficult task	4-agent	5-agent	6-agent
Ours	0.32	0.33	0.33
Baseline	0.39	0.43	0.55

compares the average time to goal for both algorithms in terms of the number of simulation time steps; each time step represents 0.2 seconds. Agents using the OLR-CAM can reach the destination faster because it interrupts less than ORCA.

Fig. 4.6 shows the generated trajectory for each agent in a complex environment in which two additional stationary agents were added to the difficult test case scenario. The agents are shown in different colours, and the shade of the colors become lighter as time goes on. In the complex scenario, the OLR-CAM of each agent found agents in immediate potential conflict; as a result, it intervenes the default policy mostly in the early time steps. As the situation became safer in the later time steps, the OLR-CAM of each agent allows the agent to them follow its default policy. If the agents be initiated far from each other

Table 4.7: time to goal comparison-unit is simulation step

Difficult task	4-agent	5-agent	6-agent
Ours	105	105	107
baseline	116	136	142

without immediate potential conflict such as red agent in Fig. 4.1, the OLR-CAM will not interrupt the default controller until it finds the agent in potential conflict.

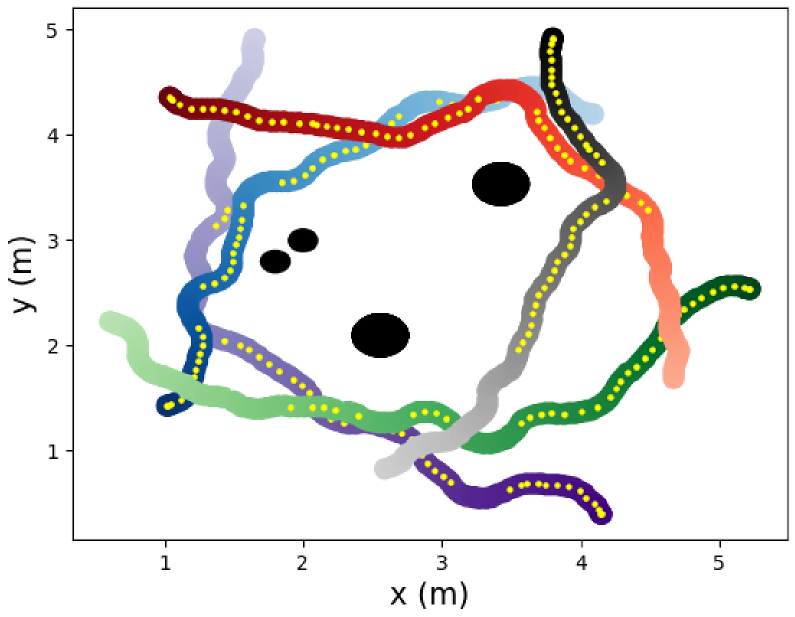
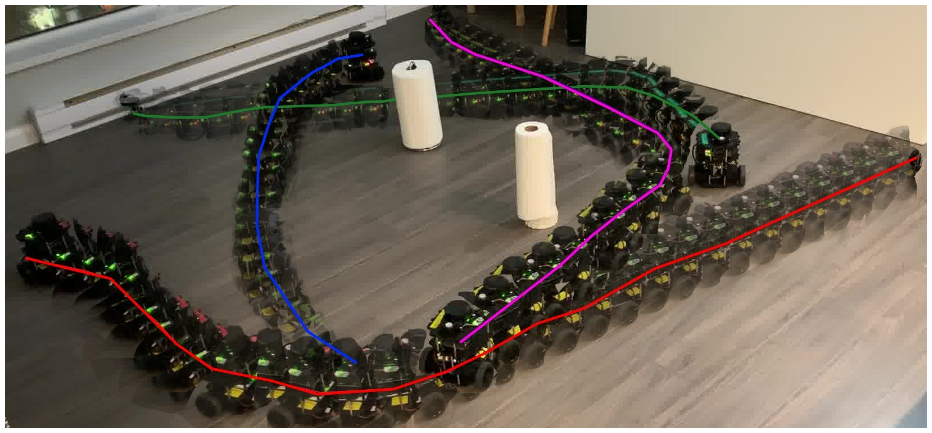
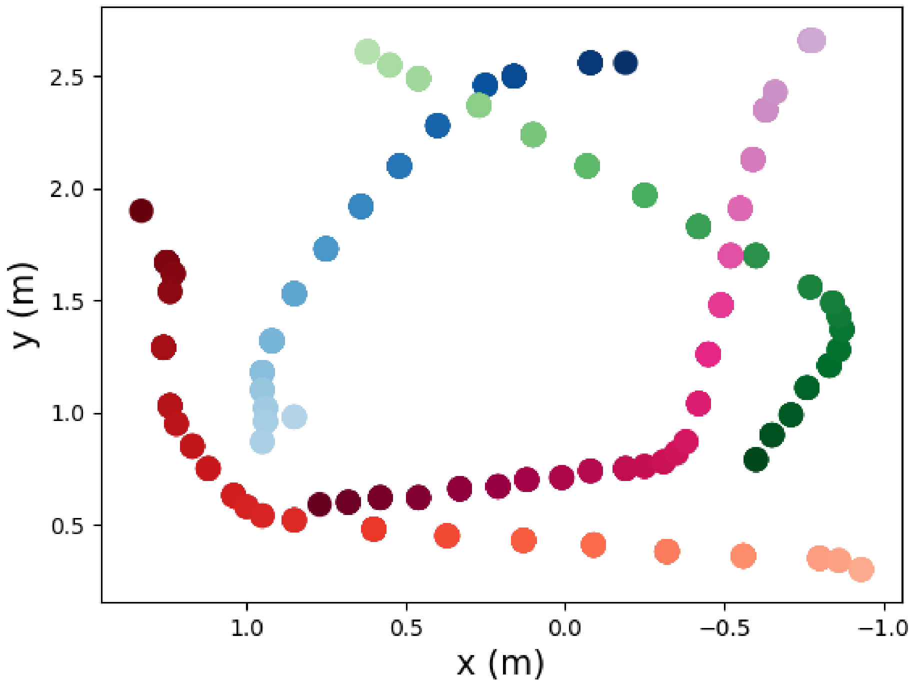


Figure 4.6: Trajectory visualization for a complex task.

Fig.4.7 shows the generated trajectory for real agents in a 4-agent task and difficult test case scenario. Additional videos of experiments can be found at `https://bit.ly/3pRydSw` and in the supplementary attached video.



(a)



(b)

Figure 4.7: (a) Trajectory visualization on real experiments.(b) The generated path. colours of real robots became darker as time goes.

Chapter 5

Continuous Control-Based Collision Avoidance Module

Recent advancement in Deep Reinforcement Learning (DRL) approaches enables the use of direct raw sensory observations in control problems. Playing complicated video games [77, 101], low-level control of various robot types [110, 55] and robotic manipulations [63, 32] are some of the significant works in this area. However, direct RL methods demand an informative hand-designed reward function which could not be easily achieved. In addition, as the uncertainty in the environment increases, the trained policy might not be able to adapt to new situations. Control of multi-distributed decision-makers is one of the challenging scenarios that suffers from domain adaptation difficulties [29]. Another issue that prevents industrial use of DRL methods on multi-distributed agents is that policies are not Least Restrictive (LR).

LR policies are initially introduced in safety-critical systems [20, 12] and refers to a control strategy that calculates a "maximal safe" control set and allows any control input that is within this control set to be executed by the agent [20]. LR policies are practical approaches that provide a high-level safety to the existing policies of an agent. LR policies can work jointly with fully autonomous, semi-autonomous and even human-operated robots and provide extra safety layers to all agents operating in the same environments. The default policy of the agent carries out the main navigation and performing objective responsibilities, and LR policies only intervene in the default policy to maintain safety. As a result, an existing agent in a specific industrial setting can be upgraded by a collision avoidance module without changing the robot controller and training the agent for different objectives. LR policies are a challenging problem as they need to balance safety and conservativeness. If the policy Addressing Multi-Objective and domain adaptation Challenges in Reinforcement Learning through Case Studies in Multi-Agent Navigation and Visual Servoiacts conservatively (i.e. interrupts), it does not allow the agent to execute the requested action) in unnecessary scenarios, it violated the "maximal" part of the definition. On the other hand, late interruptions will violate the safety criteria of the definition. From the computational complexity perspective, LR policies are NP-hard problems [47, 87] and for a large team of agents cannot be computed in real-time [13]. LR policies in the literature are in the developing phase. Some classical works tried to solve it explicitly for a small number of agents [87, 12]. In our previous works [3, 4], we also tried to address some of the LR challenges through the use of reinforcement learning.

This paper proposes a continuous control-based version of our least-restrictive collision avoidance module (CLR-CAM). Similar to OLR-CAM (Observation-based Least Restrictive Collision Avoidance Module) [3], the CLR-CAM integrates with the default policy of an existing agent, and it provides safe navigation among other decision-makers and static obstacles by looking at the raw sensory observation. However, compared to OLR-CAM, CLR-CAM gives continuous control to the interruption/correction actions and targets more complicated scenarios. A key challenge in training LR policies is the trade between conservativeness and task achievement. To control this trade-off, we designed a complicated reward function in the previous chapter, and the LR policies act as a classifier for three different actions. However, integrating continuous control on LR policies increases the reward function's complexity. We built our method using an Inverse RL (IRL) to remedy the need for a reward function. In this work, we got the idea from [84], and propose a combination of reward functions using expert demonstration data. The task reward gives a general reward for progressing toward the goal and discrete punishment for safety violations; however, the LR reward ensures the policy acts least-restrictive like the expert policy. First, we re-purposed the ORCA [102], a well-established classical collision avoidance module in the literature, to a least-restrictive controller. Then, we cloned the ORCA behaviour using a novel IRL framework to train multi-distributed decision-makers. Finally, we addressed ORCA deficiencies such as low robustness to uncertainties and the need for detailed information about other decision-makers and obstacles by changing the algorithm's input to the raw sensory observation data. We build out IRL on top of Generative Adversarial Inverse Reinforcement Learning (GAIL) [44], and we used Proximal Policy Optimization (PPO) [96] to train our learning agent.

The adversarial IRL methods are mostly used to train low-level control single agents with different robot dynamics ranging from ant to humanoid models in the literature [108, 112]. However, once GAIL is used for distributed systems, slight deviations in each decision-maker might significantly deviate from expert trajectories. In that case, the discriminator fails to provide an informative reward function for those parts of the observation space. In a related work by the authors of [103], they propose a multi-agent Generative Adversarial Imitation Learning (GAIL) framework to address some of these challenges. However, in our approach, we combine task-specific rewards and least-restrictive rewards and leverage several practical implementation tricks to train our agents while mitigating these challenges effectively. By integrating these components and employing suitable techniques, we aim to enhance the training process and improve the performance of our agents in complex environments.

The main contributions of this work are as follows:

- We Propose an LR policy with continuous action space.
- We re-purposed ORCA-DD to a least-restrictive controller and cloned the ORCA behaviour by using raw sensory observation.
- We propose a novel framework to use IRL for multi-distributed decision-makers.
- We mitigate ORCA deficiencies, namely, tedious tuning and low robustness to noise, and remove the need for detailed information about other decision-makers and static obstacles such as shape, position, and velocity.

5.1 Related Works

Multi-agent collision avoidance has been studied extensively, and the exciting approaches can be categorized into two main groups. The first group focuses on using centralized methods, in which a centralized algorithm plans and controls all other agents [109, 124]. While centralized approaches can guarantee the safety and completeness of the trajectory, they need complete knowledge about the entire system's state and are sensitive to uncertainties. On the other hand, the decentralized approaches address the problem of collision-free control for agents individually. Among the decentralized classical approaches, Optimal Reciprocal Collision Avoidance (ORCA) [116], and it's variants [102, 5] is one of the well-established algorithms in the robotic community. However, they suffer from several deficiencies preventing them from being used in wide industrial settings. First of all, ORCA demands perfect observation of the other decision-makers and obstacles such as positions, shape and velocities. In [5], the author proposed using a global positioning system to address the implementation limitation. Authors in [37], proposed an info-sharing protocol between agents. Both solutions imply a communication infrastructure that might not be practical in all scenarios. In addition, ORCA is controlled using multiple parameters, which should be carefully tuned for different scenarios. Lastly, while the ORCA, in theory, can accept any preferred velocity, the official implementation [48] doesn't accept any control input, and the preferred velocity calculator is hard-coded in the algorithm. In [68], authors proposed a pure imitation learning approach to clone ORCA behaviour but instead used LiDAR observation as the input. However, their method demands a large amount of expert data and suffers from compounding error caused by covariate shift [90]. In addition, they only considered other dynamic agents in their policy training. Our policy is the least restrictive, such that it can accept any default controller. In addition, we addressed the problem of compounding the error by using inverse RL to learn the expert's intention instead of mimicking the behaviour.

Collision avoidance also is widely studied in the RL community. In [29], the author proposed a hybrid policy to address the collision avoidance problem among other decision-makers using LiDAR observation. The proposed approach is not the least restrictive policy, and there is no control over the agents' maneuvers. In [11] and [10] author proposed an RL-based approach for socially aware motion planning and collision avoidance. The input of their algorithm is a compound state of the agent and other moving decision-makers. Similar to ORCA, providing input to these algorithms might not be practical in most realistic scenarios.

In safety-critical systems, Hamilton-Jacobi (HJ) reachability theory [13] is used to build up a least-restrictive collision avoidance [75, 36, 12]. However, due to the HJ computation complexity, these works are limited to a small number of players. In [4], we proposed a learning-based (called LR-CAM) approach to address least restrictive policies challenges using reinforcement learning. The proposed policy takes the relative position of all other agents as an input, maps them to the latent space, and decides whether the current situation is safe for the default policy, or it should be interrupted by two discrete actions, namely turning-right and turning left. We showed our proposed approach achieves better performance than the existing least-restrictive approaches; however, our method didn't consider static obstacles and demands for the relative position of other agents. In a follow-up work [3], we proposed the observation-based version of our algorithm (OLR-CAM) using a novel reward function based on a safety value function derived from HJ reachability theory and local cost-map data. We also proposed a meta-learning framework to train our algorithm in different tasks. The OLR-CAM achieves better results than the baseline in all defined criteria; however, the action space is still limited to two discrete actions for evasive actions, and agents were always moving forward with maximum velocity. We observed the limited action space and always moving forward feature caused non-smooth movements for agents. It also prevented the policy from achieving high performance in complicated scenarios and environments with many moving agents because agents might initiate danger zone in each other, and collision be inevitable.

5.2 Problem Definition

We define the multi-agent collision avoidance problem as a Partially Observable Markov Decision Process (POMDP). Each agent i has a state space $s^i \in S$, observation space $o^i \in O$ continuous action space $a^i \in A$ and initial distribution $p(s^i_0)$, transition distribution $p(s^i_{t+1}|s^i_t, a^i_t)$, reward function $r^i_t(s_t, a_t)$ and discount factor $\gamma \in [0, 1]$. We assume the transition and reward functions are unknown, and the learning agent can sample the former while it should learn the latter through expert demonstration trajectories. Moreover, We assume each agent has its default controller, which carries the main navigation responsibility. We add a learning-based collision avoidance module (CLR-CAM) to add an additional safety layer for agent movements among static and dynamic obstacles. The CLR-CAM of each agent parameterized by θ ($\pi^i_\theta(a^i|o^i)$) acts independently and the objective is to maximize the sum of expected returns $R = \mathbf{E}_\pi[\sum_{i=1}^{N} \sum_{t=t'}^{T} (\gamma^{T-t} r^i_t)]$ for all agents working in a shared environment where N is the total number of agents and T is the maximum episode length. Note the θ is shared among all agents. We consider agents are cooperative with each other, which means they are not acting aggressively to cause a collision, and each agent tries to avoid collision with other agents.

5.3 Generative Adversarial Inverse Reinforcement Learning (GAIL)

The inverse reinforcement learning problem attempts to learn a reward function based on an expert behaviour and then deploys the reward function to imitate expert behaviour. In GAIL, a "discriminator" neural network is trained to distinguish between expert demonstrations of a task and actions generated by a "generator" network. The generator network tries to imitate the expert's behaviour and produces actions that are meant to be indistinguishable from those of the expert. The discriminator provides feedback to the generator by classifying the actions as expert or generated. The generator is then updated to produce better imitations, and the process is repeated until the generator is able to produce actions that are almost indistinguishable from those of the expert. The training process for GAIL involves alternating between updating the discriminator and the generator. At each iteration, a batch of expert demonstrations and a batch of generated actions are sampled, and the discriminator is updated to maximize its objective with respect to these two batches. Then, the generator is updated to minimize the discriminator's objective with respect to the generated actions.

We train the discriminator network $D_\psi(o, a)$, parameterized with ψ using the following loss:

$$L_{\text{discriminator}} = \mathbf{E}_{\tau_\pi}[log(D_\psi(o, a)] + \mathbf{E}_{\tau_{\pi_\theta}}[log(1 - D_\psi(o, a)] \tag{5.1}$$

Where τ is the sampled trajectory, the generator's (policy network) responsibility is to generate actions as close as possible to expert behaviour such that the discriminator network couldn't classify the actual expert actions. We train the policy network using the reward provided by the discriminator network defines as:

$$r(o_t, a_t) = -log(1 - D_\omega(o_t, a_t)) \tag{5.2}$$

5.4 Optimal Reciprocal Collision Avoidance

ORCA is a collision avoidance algorithm that is commonly used in robotics and autonomous systems, especially in multi-agent environments. The algorithm works by generating a set of velocity obstacles for each agent in the world that characterize the set of velocities that would result in a collision with another agent. Then, using a linear algorithm to optimize a global objective function, ORCA chooses a collision-free velocity from among the feasible velocities.

The key insight behind ORCA is that collision avoidance can be viewed as a constraint optimization problem in which the objective is to maximize the agents' individual velocities while keeping the velocity obstacles in mind. ORCA is able to handle complex, dynamic environments with many moving parts by formulating the issue in this manner.

The velocity obstacle for agent i with respect to agent j is defined as:

$$VO_{i,j} = v \in \mathbb{R}^2 : \exists t \geq 0, \exists p_{i,j} \in P_{i,j}, ||v - (v_i - v_j)||t + ||p_{i,j} - (p_i - p_j)|| \leq r \tag{5.3}$$

where v_i and v_j are the velocities of agents i and j, respectively, p_i and p_j are their positions, r is their radius of avoidance, and $P_{i,j}$ is the half-plane perpendicular to the vector $p_{i,j} = p_j - p_i$.

The feasible set of velocities for agent i is the intersection of the velocity obstacles for all other agents j:

$$F_i = \mathbb{R}^2 \setminus \bigcup_{j \neq i} VO_{i,j}, \tag{5.4}$$

The optimal velocity for agent i is then selected from the feasible set F_i by solving a linear program that maximizes a global objective function, such as the sum of velocities for all agents or the minimum deviation from the current velocity:

$$\max_{v_i \in F_i} \sum_{i=1}^{n} w_i \cdot v_i \tag{5.5}$$

where n is the number of agents, and w_i is the weight assigned to agent i in the objective function. Our approach can be implemented on any agent's dynamic that OCRA supports, such as differential drive and Omnidirectional agents. However, we are using ORCA-DD [102] (an extension to ORCA for differential drive systems) for experiment and evaluation. While ORCA is a highly effective algorithm for multi-agent collision avoidance, there are a few limitations and disadvantages to consider:

- Computational complexity: ORCA's computational complexity scales linearly with the number of agents in the environment, making it less efficient for large-scale systems with many agents. This can limit its applicability in real-world scenarios.

- Conservative avoidance: ORCA's avoidance strategy is conservative, which means that agents may slow down or stop even when there is no immediate threat of collision. This can result in suboptimal and inefficient trajectories, especially in highly dynamic environments.

- Lack of anticipation: ORCA is a reactive algorithm that only responds to immediate threats of collision. It does not consider future trajectories or anticipate the behavior of other agents, which can lead to collisions or missed opportunities.
- Limited modelling of agent behavior: ORCA assumes that all agents move in a straight line with constant velocity and avoid collisions by adjusting their velocities. This may not be an accurate representation of real-world agent behavior, which can be more complex and diverse.
- Sensitivity to parameters: ORCA's performance can be sensitive to its parameter settings, such as the radius of avoidance and the weight assigned to each agent in the objective function. Choosing optimal values for these parameters can be challenging and may require extensive tuning.

5.5 Method

The objective is to learn an observation-based least restrictive collision avoidance module with continuous action space that can accept any default policy. The default policy could be goal-reaching, performing a specific mission, or given by a human operator. When an agent is in a potential conflict, the CLR-CAM will take over control and modify the default policy output in the least restrictive fashion to bring the agent back to safety. This section will propose a novel learning-based framework to train distributed agents in a multi-agent system. We built our method on top of GAIL [44] and PPO [96] algorithms and we used a modified version of the implementation proposed in [84].

We are using raw LiDAR observation and the default controller action as input to the CLR-CAM. We assume velocity commands control agents. In our experiments, we implement the method on differential drive robots and define the action space as:

$$a_t = (v_t, \omega_t) \tag{5.6}$$

where $v \in [v_{\min}, v_{\max}]$ and $\omega \in [\omega_{\min}, \omega_{\max}]$ are linear and angular velocities respectively.

As we described earlier, we used PPO as our RL agent which is trained with a combination of task reward and least restrictive reward.

$$r(s_t, a_t) = c_1 \cdot r_t(s_t, a_t) + c_2 \cdot r_l(s_t, a_t) \tag{5.7}$$

where c_1, c_2 are scaling factors, r_t is a task reward and r_l is the least-restrictive reward that comes from the discriminator. To stabilize GAIL training we used Least-Square GAN (LSGAN) [71]. LSGAN is a type of Generative Adversarial Network (GAN) that uses a least-squares loss function instead of the binary cross-entropy loss function used in traditional GANs. In LSGAN, the discriminator model is trained to output a continuous score for each input, rather than a binary classification (real/fake) as in traditional GANs. The generator is then trained to minimize the difference between the discriminator's score for the generated output and the score for the real data. The main advantage of using LSGAN is that it provides more stable training, generates more informative rewards and avoids mode collapse, where the generator produces only a limited variety of outputs. The LSGAN discriminator loss function (L_D) can be expressed as follows:

$$L_{\text{discriminator}} = \mathbf{E}_{\tau_\pi}[(D_\psi(o,a)-1)^2] + \mathbf{E}_{\tau_{\pi_\theta}}[(D_\psi(o,a)+1)^2] \tag{5.8}$$

For the discriminator, we used the same observation as the generator, except we feed two consecutive observations to the discriminator. To further stabilize the training procedure we used Gradient Penalty on the discriminator loss function. Gradient penalty is a regularization technique used in the training of Wasserstein GAN (WGAN) [2] to enforce the Lipschitz constraint, which ensures that the gradient of the discriminator does not become too large. In WGAN, the discriminator is trained to maximize the difference between the average score for real data and the average score for generated data, subject to the constraint that the discriminator's gradient norm is equal to one almost everywhere. This constraint is enforced by adding a gradient penalty term to the loss function of the discriminator.

The gradient penalty term penalizes the discriminator if the norm of the gradient with respect to the discriminator's input is different from one. The penalty term is computed as the difference between the norm of the gradient and one, squared. This term is then multiplied by a regularization parameter lambda and added to the loss function of the discriminator.

The use of gradient penalty helps to stabilize the training of WGAN, and leads to better convergence and generation quality of the generator. The overall objective is written by:

$$\begin{aligned} L_{\text{discriminator}} = \mathbf{E}_{\tau_\pi}[(D_\psi(o,a)-1)^2] + \mathbf{E}_{\tau_{\pi_\theta}}[(D_\psi(o,a)+1)^2] \\ +\gamma \mathbf{E}_{\tau_{\pi_\theta}}[||\nabla_\psi D_\psi(o,a)||^2] \end{aligned} \tag{5.9}$$

where λ is the scaling factor.

Fig. 5.1 visualize the proposed learning framework in multi distributed agent system. The learning algorithm has two parts. First, a discriminator network is trained to discriminate between expert actions and policy actions. It also provided the generator network with a reward estimation. Second, a generator actor-critic network is trained to generate actions similar to expert behaviour.

The complete training algorithm is summarized in Alg. 4.

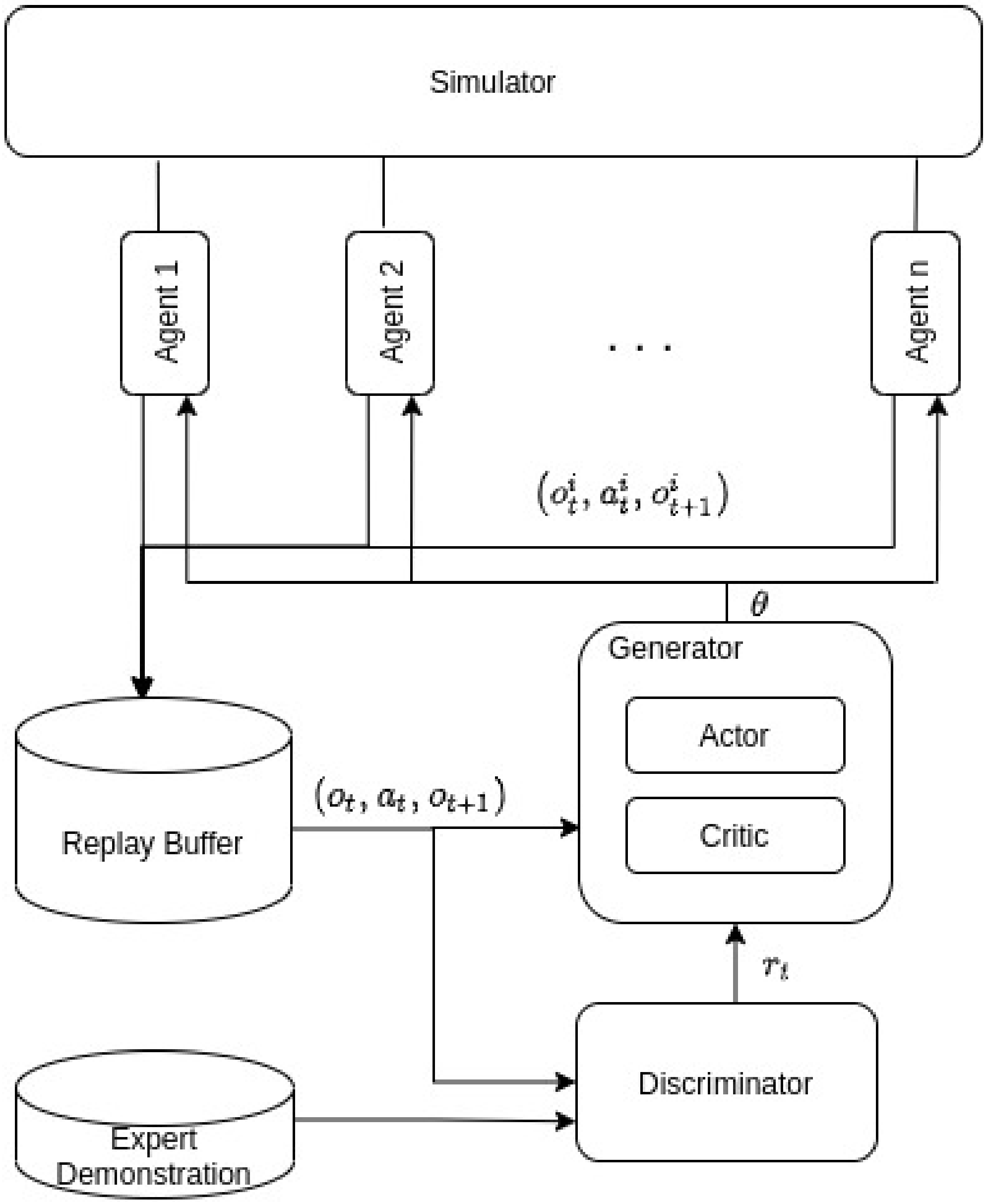


Figure 5.1: The proposed learning framework for distributed multi-agent settings. The IRL algorithm is trained centralized, and the policy is executed decentralized in the environments.

Algorithm 4 proposed meta-training loop

```
Require: initialize parameters $\phi, \theta, \psi$
1: Run expert and add trajectory to reply buffer $\mathcal{B}$
2: while True do
3:    Randomize the episode task, agents and obstacles
4:    while episode is not done do
5:       for agent in task do
6:          Roll out policy $\pi_\theta(a_t|o_t)$ to collect data $\{o_t, a_t, o_{t+1}\}$ and add to reply buffer $\mathcal{B}_p$
7:       end for
8:       if step > heat-up-training-steps then
9:          sample mini batch $b_p \sim \mathcal{B}$
10:         $\psi \leftarrow \psi - lr_1 \nabla_\psi L_{discriminator}$
11:         Calculate $r_l$ using (5.2)
12:         Calculate aggregated reward $r$ using (5.7)
13:         $\phi \leftarrow \phi - lr_2 \nabla_\phi L_{critic}(b_p, r)$
14:         $\theta \leftarrow \theta - lr_3 \nabla_\theta L_{actor}(b_p, r)$
15:      end if
16:   end while
17: end while
```

5.6 Implementation and Training

To train our approach on realistic distributed multi-agent systems with LiDAR observations, we create our simulation environments using Isaac Sim Simulator with a thousand agents. Massive parallel simulations have recently become popular in RL framework [91, 83]. Massive parallel simulation increased sample efficiency in terms of speed, enable more diverse exploration, reduce variance in on-policy learning and expedite training procedures. [91]. Fig. 5.2 Visualizes a snapshot from the simulation environment. The black subfigure in the visualization represents a massively parallel simulation involving over a thousand agents. They are grouped into various tasks, and only one of these tasks is visualized in the top-right figure.

During the training and testing phases of our algorithm, we utilize random scenario tasks. In these scenarios, a varying number of agents and obstacles are initialized within a shared environment. Each agent is assigned the objective of reaching a randomly selected goal by following its default controller. The randomness in the number of agents, obstacles, and goal locations introduces variability and challenges that the algorithm needs to adapt to and navigate successfully. The objective of each agent in this scenario is to use CLR-CAM to provide a safe execution of the default policy and avoid any safety violations. An episode for each agent will be terminated if agents finish their default objective if the agent violates safety, and finally, if the total number of steps reaches the maximum number of steps in an episode. If an agent violates safety or reaches its goal, the agent will be restarted in the environment with a new initial/target location. We also randomly change the obstacle

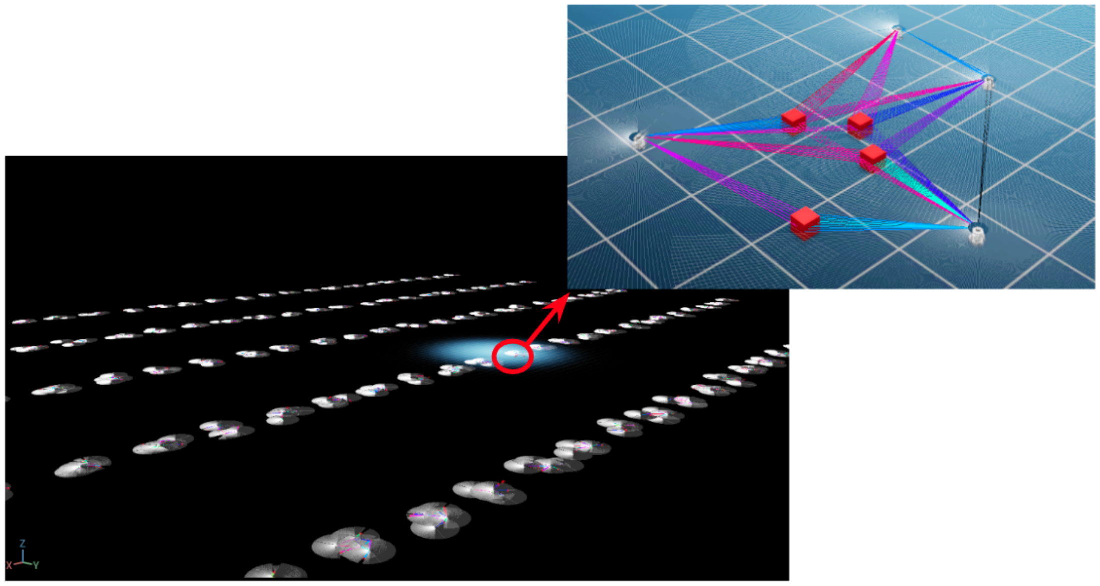


Figure 5.2: Snapshot from the developed simulation environment

position in the task during training. Note that target location and localization data are only provided to the default policy and are unknown to the learning agents.

For training and experiments, we used TurtleBot3 burger robots as our agents. Each robot is equipped with an onboard 360-degree 2D LiDAR and Gyro. We used the same simulation environments to generate expert demonstrations and implemented a modified version of ORCA-DD that can accept any default controller actions. We provided ORCA-DD with all the required and accurate data such as shape, location and velocity of other agents, and we record pairs of (o_t, a_t^{orca}) as expert demonstrations where o_t is raw, noisy observation. For the default controller, we used a PID goal-reaching controller and a globally optimal goal-reaching controller derived from Hamilton-Jacoby (HJ) reachability theory that is taken from [13]. None of the default controllers have any collision avoidance, and they are randomly assigned to agents during training. We implemented the learning algorithm in Pytorch. The training procedure took 30 hours to converge using a computer with Nvidia RTX 3090 GPU and 32 GB RAM. We trained the algorithm in a simulation only.

5.7 Results and Discussion

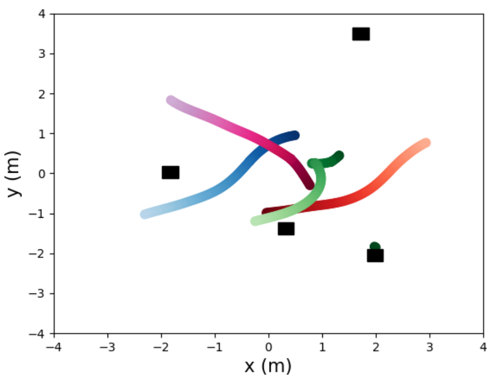
This section will discuss the results of the proposed method. Following Chapter 4, we compare our method with OLR-CAM as a learning-based least-restrictive controller that accepts Lidar Observation and also ORCA which we used as the expert demonstration. The results for the random task scenario are summarized in Table.5.1. Note that we didn't consider the effect of noise in the ORCA controller and provided all detailed information.

As it can be inferred from the results, the CLR-CAM success rate outperforms our previous method because OLR-CAM only acts at agents that move forward with maximum velocity. As a result, agents might initiate into each other or obstacles reachable tubes and the collision might be inevitable. On the other hand, CLR-CAM is capable of stopping or backing up which increases its safe navigation in those scenarios. However, it is observed that the CLR-CAM restrictiveness factor is higher compared to the other methods, indicating that CLR-CAM tends to take control more frequently. This could be attributed to the fact that the restrictiveness behavior of CLR-CAM is inferred from ORCA demonstrations, which tend to be more conservative in nature. Additionally, the combined task reward may also play a role in influencing the behavior of CLR-CAM. Next, we visualize trajectories for

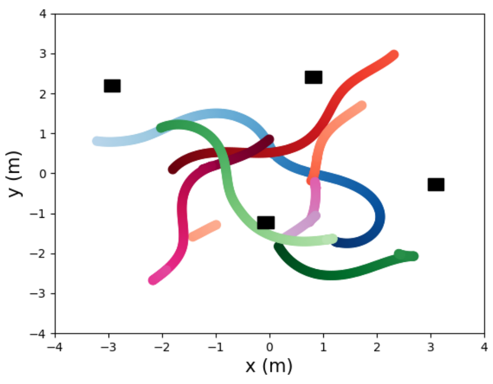
Table 5.1: Comparison of success rate and restrictiveness factor

Controller	Success Rate	Restrictiveness Factor
ORCA	0.75	0.44
OLR-CAM	0.79	0.35
CLR-CAM	0.95	0.71

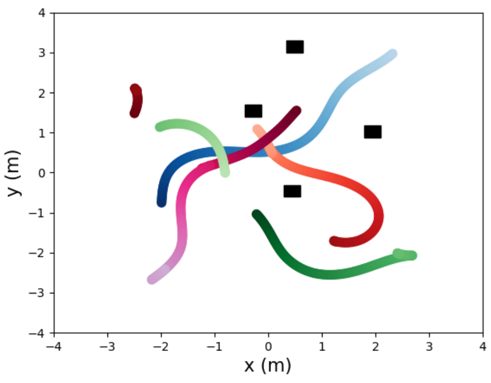
agents. Fig 5.3 visualize some random trajectories from the experimented tasks. Note that agents are colour coded and the level of opacity varies with time. Moreover, in each graph, only the trajectory of the blue agent is plotted in full from the initial point to the target points. Other agents' trajectories might not be completed because the agent's episodes are not synced. Please refer to the implementation and training section for more details on episodes/terminal definitions.



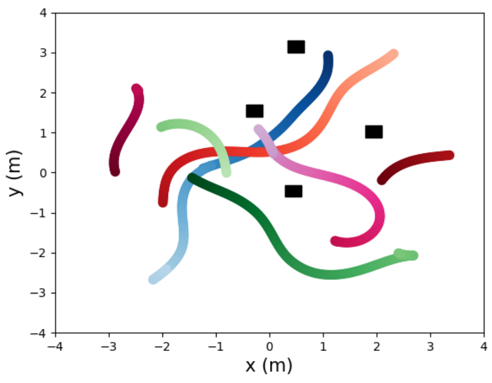
(a)



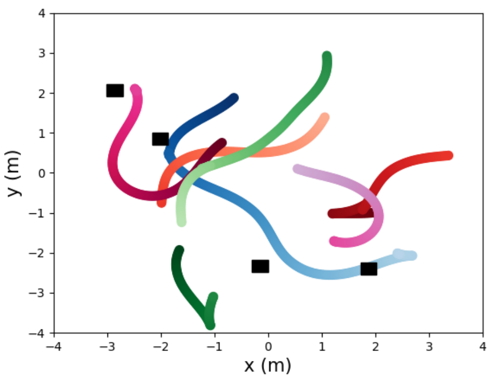
(b)



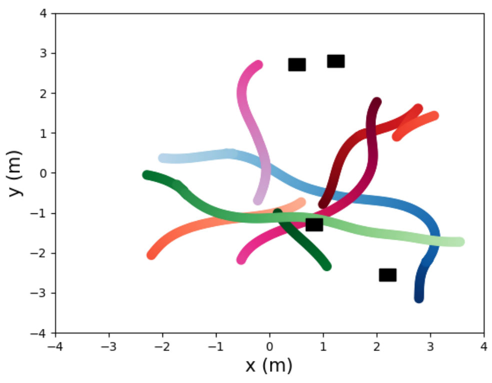
(c)



(d)



(e)



(f)

Figure 5.3: Generated Trajectories for multi-agent navigation using CLR-CAM.

Chapter 6

Visual Servoing

Visual Servoing (VS) is a classical control problem that has been studied for decades and refers to controlling the robot's motion through visual feedback [46]. Perception and action are the two key capabilities that make intelligent agents capable of performing complex tasks such as navigating to a goal or picking up an object. Such scenarios require the agent to have high-level reasoning and a good representation of its surrounding environment to take actions efficiently. In robotics, VS in fact encompasses a variety of tasks that lie at the intersection of perception and control, from robotic manipulation to vision-based navigation of mobile robots.

Classical approaches to VS decouple the problem into image processing and control [9, 46, 8], and often require considerable application-specific feature and model engineering. These methods heavily rely on extracting, tracking, and matching a group of 2D or 3D visual features, which limits their capability. For example, their performance can drastically deteriorate in the presence of input noise, illumination changes, and camera occlusion. In addition, the aforementioned methods assume that the features can be detected seamlessly throughout the entire servoing process; however, in practice, due to the limited field of view of the camera, feature-loss is inevitable. A relatively new approach called Direct Visual Servoing (DVS) [19] eliminates the need for image processing for feature extraction and matching, but in comparison with classical techniques, DVS has a limited convergence area.

State-of-the-art (SOTA) VS systems employ Deep Neural Networks (DNNs) either in the feedback control loop of the system [1, 53, 30] or on an end-to-end basis [6, 123, 31, 61, 54, 64]. These methods alleviate the aforementioned disadvantages of classical methods because DNNs are known to be highly precise feature detectors and extractors, which are robust to changes in scale, position, illumination, and occlusion albeit with slightly higher end effector pose error [123, 30]. The first group of methods [53, 1, 30] uses DNNs to close the feedback loop for the explicit control module. In the latter group [6, 123, 31, 61, 54], DNNs are trained to learn control policies using supervised, unsupervised, or reinforcement learning (RL) methods. This group of approaches views the entire problem of mapping raw input images to control commands as a unified task that can be acquired end-to-end.

Despite this, most of these methods lack the scalability to handle a large number of objects during training, and they are usually overfitted to a single or limited set of objects.

In this paper, we propose an RL-based method for visual servoing that decouples the representation learning from policy learning in a way that the control actions also affect the representation learning. In this framework, a goal-conditioned stochastic sequential latent representation is trained in an unsupervised manner. The latent space intuitively represents the essential features that are extracted from the raw input images. Using this latent space, an actor-critic agent is trained to control the camera movement. Additionally, we demonstrate that separating these parts creates an attractive specification for our method that facilitates domain adaptation to real-world situations by allowing fine-tuning of the representation learning part alone. To facilitate the sim2real transfer, we used a hybrid approach to decouple the camera motion control from the manipulator dynamics.The hybrid method consists of an RL-based high-level controller that predicts the movement of the camera in Cartesian space and a low-level joint controller that regulates the manipulator's movement in Joint-space by using the Inverse Kinematics (IK) solution of the manipulator.

Our experiments, both in simulation and real-world implementation, indicate that visual servoing in stochastic sequential latent space (VSLS) can effectively address the aforementioned limitations in classical VS methods such as partial camera occlusion, and lack of image-feature bands. To the best of our knowledge, our method is the first scalable modern VS method that is capable of training on a large variety of objects from multiple classes, and a stepping stone towards the development of a scalable robust solution for general VS problems. The contributions of this paper are as follows:

- We propose a novel goal-conditioned RL-based algorithm for visual servoing using stochastic latent variables that is scalable to a large number of scenes and objects.
- We show our method is robust to occlusion and environmental uncertainties and achieve a high convergence rate.
- We show that our method can easily adapt to real-world by single-shot domain transfer on the representation learning part only and validate the effectiveness of our method by implementing it on a 7-DoF manipulator.

6.1 Related Work

6.1.1 Modern Visual Servoing

SOTA visual servoing systems employ DNNs either in the feedback control loop of the system [1, 53, 30] or on an end-to-end basis [6, 123, 31, 61, 54].

In [1], a Convolutional Neural Network (CNN) is trained to detect leaves for a visual servoing task. The CNN was combined with another visual servoing method known as monoscopic depth analysis, which involves comparing two images to determine the location of some feature points in the image relative to the camera in Cartesian space. [53] presented a robotic grasp detection system. The robot is capable of predicting the best grasping pose of a robotic gripper using an RGB-D image of the scene. The method uses two parallel ResNet-50 CNNs, to extract features from RGB and depth in parallel to produce grasp configurations for the objects in a planar scene. [30] recently proposed a VS framework in latent space, where the error in latent space is used to close the feedback loop for an analytical control module. An auto-encoder based on ResNet-18 is trained on different views of 2D images in simulation during the training phase. The trained auto-encoder maps the raw input images into a latent space with a size of 32. During the inference time, the auto-encoder weights are frozen, and the error for the analytic control module is computed as the difference between the latent space vector for the current and desired images. Their experiments demonstrate that servoing in the latent space is an effective method, providing precise positioning and a superior convergence domain than other DVS methods. Our method has the advantage of not requiring any supervised data set and, instead of training on different scenes from one object, is capable of training on various objects and scenes.

[6] and [123] train a CNN to estimate the relative position error between the current image and the goal image through supervised learning. A major limitation of [6] is that the CNN must be retrained for each reference pose. In [123], current and goal images are fed into a siamese network. The extracted features are then compared at successive layers in order to achieve precise positioning. Siame-se(3) [31], achieves high precision positioning in spite of large initial errors. In Simae-se(3), rather than estimating the positioning error, the camera velocity is directly regressed and learned end-to-end. One of the key aspects of this research is the choice of a loss function that is less sensitive to non-homogeneous scaling between the regressed components (translation and orientation). As a disadvantage of this method, the network needs to be trained in a supervised manner, which requires complex synthetic labelled data.

In addition, recently several Deep RL (DRL) based approaches are proposed for manipulation [61, 54, 93, 34, 63, 49] and navigation via a goal image [92, 126]. As one of the early researchers in RL-based visual servoing, Levine et al. [61] introduced a guided policy search for learning policies that directly map raw image observations to torques at the robot's motors and showed that training the perception and control systems jointly end-to-end provide better performance than training each component separately. While this method achieves impressive results on real-world manipulation tasks, significant human involvement is required for the data collection process. Zhu et al. [126] presented a deep siamese actor-critic method for the mobile robot target-driven navigation. In their work the inputs to the network are two images that represent the agent's current observation and the target to transform the current state and the target into the same embedding space. The joint embedded features are passed to the scene-specific layers and action are chosen from a set of predefined discrete action. [93] trains a convolutional recurrent neural network that can control a robot manipulator to approach the user-specified objects. A query image and observed image form the input to the network. Separate convolutional layers are applied to each image, and their features are concatenated. Concatenated feature vectors are fed into an LSTM layer and the result is an end-effector movement command in Cartesian space. Policy is trained using a combination of action supervision gathered from demonstrated trajectories in simulation and reinforcement learning-based value function prediction (Monte Carlo). A recent study reported in [49] is similar to our work in that an actor-critic is trained in the latent space of an auto-encoder (although is not goal-conditioned). Their investigation suggests that explicit representation learning is not suitable for extracting pertinent information applicable to the subsequent acquisition of grasping skills. As a result, preprocessing of the input images is necessary. However, vsls proposes learning the policy based on the latent space of a variational autoencoder that efficiently learns the relevant information necessary for visual servoing end-to-end without any pre-processing.

6.1.2 Reinforcement Learning from Pixels

DRL algorithms can in theory employ large-scale deep networks to directly learn policies from pixel inputs [76, 39, 122]. In practice, learning directly from high-dimensional images with a standard end-to-end DRL algorithm can be slow, sensitive to changes in hyperparameters, and data inefficient, since it must address two distinct problems: representation- and policy-learning. The state-of-the-art attempts to overcome these limitations by leveraging different representation learning methods [104, 58, 78, 69, 121, 97, 56]. CURL [104] proposed a framework to extract high-level features in model-free and model-based RL using contrastive learning and performing off-policy control on top of these features. Lee et al. [58] showed that capturing the predictive information (mutual information between the past and the future) can be beneficial for RL agents. Their work trains a Soft Actor-Critic [39] agent from pixels with an auxiliary task that learns a compressed representation of the predictive information of the agent's environment's dynamics using a contrastive loss. [78] learns a visual representation by training a generative model that gives agent the opportunity to perform self-supervised practice by imagining goals and attempts to achieve them during training, since goals at the test time may not be known in advance. SLAC [56] proposed an efficient RL algorithm that combined an off-policy model-free RL with representation learning via a stochastic sequential model that models the high-dimensional observations as the consequence of a latent process, with a Gaussian prior and latent dynamics.

6.2 Perliminaries

Based on [57], our method involves learning the VS task through representation learning. However, unlike [57] our approach employs goal-conditioned multi-task RL to solve the VS problem. Our method also relates to [30] in the sense that we also learn a latent representation. However, since the VS is a sequential decision making task, a single observation is not sufficient to estimate the latent space, and previous decisions should also be taken into account [57].

We define a goal-conditioned Partially Observable Markov Decision Process (POMDP) with goal distribution $p(g)$ as a tuple $(\mathcal{X}, \mathcal{Z}, \mathcal{X}^g, \mathcal{Z}^g, \mathcal{A}, \rho, r)$ where $x \in \mathcal{X}$ denotes the observation (i.e. images in our problem) and $z \in \mathcal{Z}$ is its corresponding latent variables, $x^g \in \mathcal{X}^g$ the goal observation and $z^g \in \mathcal{Z}^g$ is corresponding goal latent variables, $a \in \mathcal{A}$ the action, ρ the initial observation distribution, r the reward function. In addition, we interpret z as the unobservable part of POMDP and $p(z_{t+1}|z_t, a_t)$ is the stochastic transition dynamics. We assume transition dynamic and reward functions to be unknown to the learning agent, and that they can be sampled through interaction with the environment.

To learn the appropriate representation for VS task, we used Variational Autoencoders (VAEs) [51]. VAEs is an unsupervised machine learning approach that processes the input image in pixel space and teases out a compact and meaningful independent representation called the latent space. VAEs architecture consists of an encoder network $q(z|x)$ that maps the pixel space into the latent space followed by a decoder network $p(x|z)$ responsible to reconstruct the input image x from the latent representation z. Compared to regular auto-encoders (AE), latent space parameters in VAEs are stochastic variables conditioned on isotropic Gaussian priors. The prior gives control over the distribution of latent variables and makes the VAEs more practical and efficient to large-scale dataset [73]. The objective of VAEs are maximizing the likelihood of observation marginal $p(x)$. Due to the computational intractability of this objective, the encoder/decoder are jointly trained to maximize the evidence lower bound (ELBO) for the log-likelihood of observation marginal $\log p(x)$ [51],

$$\mathop{\mathbb{E}}_{z\sim q}[\log p(x|z)] - D_{KL}(q(z|x)||p(z)], \tag{6.1}$$

where D_{KL} is the Kullback–Leibler divergence between two distributions. We note that using separate representation learning doesn't provide the required information for RL policy to perform the task. Therefore, to incorporate the latent variable representation learning under a sequential decision making framework, we define z_t as the corresponding sequential latent representation of observation x_t with a transition distribution $p(z_{t+1}|z_t, a_t)$. The prior observation and action provides richer information to infer the latent variable z_t [57]. Based on the sequential latent representation definition, the distribution of interest to maximize is $\log p(x_{1:\tau+1}|a_{1:t})$ and similar to (6.1) can be bound by:

$$\mathop{\mathbb{E}}_{z_{1:\tau+1}\sim q}[\sum_{t=0}^{\tau} \log p(x_{t+1}|z_{t+1} - D_{KL}(q(z_{t+1}|x_{t+1}, z_t, a_t) \mid\mid p(z_{t+1}|z_t, a_t)] \tag{6.2}$$

where $p(x_t|z_t)$ is the decoder model, $p(z_{t+1}|z_t, a_t)$ is the prior model, $p(z_1)$ is the initial prior, $q(z_{t+1}|x_{t+1}, z_t, a_t)$ is the variational posterior.

Following the work on variational inference [60], and SLAC [57], we incorporate the control into the inference problem and define the problem as maximizing the following goal-conditioned marginal likelihood:

$$p(x_{1:\tau+1}, \mathcal{O}_{\tau+1:T} | a_{1:\tau}, x^g) \tag{6.3}$$

where $\mathcal{O}$ is a binary optimality variable, with

$$p(\mathcal{O}_t = 1) = \exp(r_t). \tag{6.4}$$

Eq. (6.3) represents both representation learning and policy learning by maximizing the likelihood of observed data and maximizing the policy behaviour over future steps, respectively in a single objective. Similar to [57], we factorized the latent variable z_t into two stochastic latent variables (z_t^1, z_t^2) to make the learning process more expressive. Similar to (6.2), we maximize ELBO over (6.3) instead of directly maximizing the (6.3),

$$\begin{aligned} &\mathop{\mathbb{E}}_{(z^1_{1:T}, z^2_{1:T}, a_{\tau+1:T}, z^g) \sim q} \Bigg[\sum_{t=0}^{T} (\log p(x_{t+1} | z^1_{t+1}, z^2_{t+1}) \\ &- D_{KL}(\log q(z^1_{t+1} | x_{t+1}, z^2_t, a_t) || \log p(z^1_{t+1} | z^1_{t+1}, a_t)) \\ &+ \sum_{t=\tau+1}^{t} \Big(r_t - \log p(a_t) - \log \pi(a_t | x^1_{1:t}, a_{1:t-1}, x^g) \Big) \Bigg], \end{aligned} \tag{6.5}$$

The first summation term in (6.5) is similar to (6.2) trying to maximize the likelihood of $x_{1:\tau+1}$ in (6.3), however, with two sequential latent variable (z_1, z_2). The second summation term corresponds to maximizing the likelihood of the optimality variable in (6.3). For more details please refer to [57] and [60].

Eq. (6.5) has an excellent interpretation that the first part corresponds to representation learning and the second part corresponds to VS task objective part which is similar to the maximum entropy RL objective [39]:

$$J(\pi) = \mathop{\mathbb{E}}_{p(g)} \mathop{\mathbb{E}}_{p(\tau|\pi,g)} [\sum_{t=0}^{T_{\max}} \gamma^t (r_t + \alpha \mathcal{H}(\pi_\theta)], \tag{6.6}$$

where $p(\tau|\pi, g)$ is the probability distribution over trajectory τ,

$$p(\tau|\pi, g) = p(z_0) \prod_{t=0}^{T_{\max}} p(z_{t+1}|z_t, a_t)\pi(a_t), \tag{6.7}$$

$\gamma \in [0, 1)$ denotes the discount factor, α is a temporal variable that controls the trade-off between expected return and expected entropy and $\mathcal{H}$ is the entropy of policy π_θ. Note that the latent variables are not incorporated in (6.5) for the policy learning part and we will explain later in 6.3.4 how to use latent variables in the learning process.

6.3 Method

6.3.1 Observation and Action space

The observation here is defined a history of images-actions $x_t = (o_t, o_{t-1}, ...o_{t-T_L}, a_{t-1}, a_{t-2}, ...a_{t-T_L})$ where o_t denotes the image captured by the camera at time step t. The defined observation space will be compacted and disentangled into latent space components and fed to the control policy (see 6.3.4 for more details).

We define the action space to be the camera displacement within the Cartesian space. The predicted output of the policy network is a short-term navigational goal. We assume that the policy is of stochastic nature and it is taken from a normal distribution, whose mean and variance are predicted through the agent as a tuple:

$$a_t = (\delta_x, \delta_y, \delta_z, \delta_{\text{roll}}, \delta_{\text{pitch}}, \delta_{\text{yaw}}) \tag{6.8}$$

where $\delta_w \sim \mathcal{N}(\mu_w, \sigma_w)$ is assumed to have a normal distribution with mean of μ_w and variance of σ_w and $w \in \{x, y, z, \text{roll}, \text{pitch}, \text{yaw}\}$. Note that the predictions of the policy network are bounded such that the low-level controller can reach the requested goal location within a one-time step if the goal is possible. If not possible the low-level controller simply rejects the goal.

6.3.2 Reward Function

One can see the VS task alternatively as reaching a specific 6D pose in the manipulator's work-space such that the target image is observed. We assume the target and current poses of the end-effector/camera are known during the training steps. We define the camera pose as follows:

$$p_t = (p_t^T, p_t^R) \tag{6.9}$$

where p_t^T denotes the camera translation in Cartesian coordinate and p_t^R denotes the camera rotation in quaternion format at time t. Furthermore, the reward function is formulated as:

$$r(x_t, a_t) = \begin{cases} r_g & \text{if } d^t_{\text{goal}} < \text{c} \\ d^{t-1}_{\text{goal}} - d^t_{\text{goal}} & \text{otherwise} \end{cases} \tag{6.10}$$

where c denotes a threshold value for reaching goal and d^t_{goal} is a defined distance function from current to goal configuration given by:

$$d^t_{\text{goal}} = c_1 ||p_t^T - p^T_{\text{goal}}|| + c_2 |(p^R_{\text{goal}})^\top [p_t^R]^{-1}| \tag{6.11}$$

where c_1, c_2 denote constant scaling factors. The first term in (6.11) indicates the Euclidean distance between the current and goal position in Cartesian space, and the second term calculates the magnitude of rotation from current to goal pose. Considering the work-space boundaries of the manipulator, (c_1, c_2) scaled to [0, 1].

The proposed reward function gives a positive reward r_g for reaching the target and appropriate reward/punishment for moving toward/away from the goal position.

6.3.3 Curriculum Learning

To improve and direct the learning procedure, we take inspiration from [30] to develop our curriculum algorithm and employ a reverse expansion on goal distribution. Fig. 6.1 depicts a schematic overview of the curriculum used in this method. We assumed the object is planar and unlike [30, 31], it can have a bounded perturbation in all 6 DOFs.

Pseudo code for the entire approach is provided in algorithm 5. For training, we considered two scenarios: look-at and screw motion. In first, the goal positions P are sampled from a box of $[d_1, d_2, d_3]$ that its dimensions are bounded by the manipulator's workspace. The pitch and yaw angles are set such that the camera looks at a random point within the r radius of the center of the object while the roll angle is sampled randomly from a normal distribution $\mathcal{N}(0, \sigma)$. In the latter, we assume the end-effector only has a movement and rotation along the x-axis (camera focal/principal axis) and the movement and rotations are randomly sampled from $(d1, \theta_{max})$. The goal x and the initial image will overlap enough this way. Next, we determine the reachability/feasibility of the goal for the manipulator using the IK solver. Once a reachable goal is generated, we first evaluate the policy on the selected goal and only add the goals to the goal list if the policy couldn't solve the task. Note that, we observed preventing training on a solvable goal has a huge improvement on the overall convergence rate. We consider N stages in the curriculum. Each stage includes generating K goals. Following the completion of K goals, all curriculum variables (sample box dimensions (d_1, d_2, d_3), scene perturbation, look-at radius r and max pitch angle θ_{max}) are increased incrementally. To prevent catastrophic forgetting, we randomly sampled from the solved goal list in a non-uniform manner.

6.3.4 Training Algorithm

The use of latent variables for training purposes is one of the design decisions. As noted in [57], we utilized latent variables to train the critic network and used the feature vector directly as the input to the actor network. This choice of design not only makes the algorithm more robust and generalizable to environment noises, but it also reduces the computational complexity during the deployment of the algorithm as discussed in [57] and validated in our experiments.

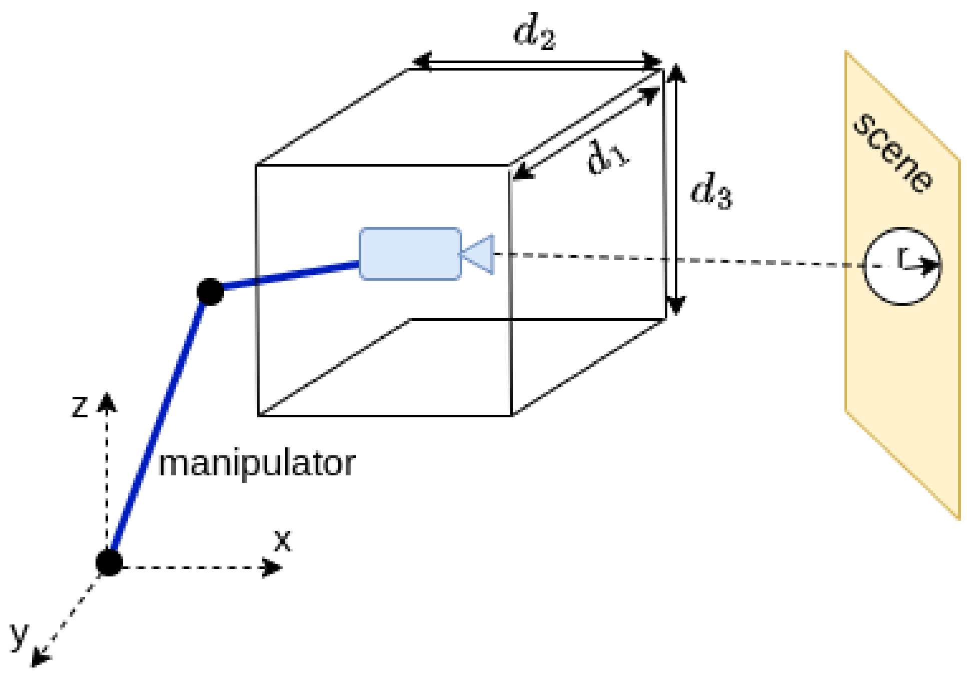


Figure 6.1: An overview of Curriculum learning procedure

Algorithm 5 Curriculum

```
Require: initialize parameters $d_1, d_2, d_3, \theta_{max}, r, N, K, \sigma, \epsilon$
    initialize goal list $G$
 1: while True do
 2:     Randomize scenario from look-at and screw motion
 3:     select a random goal $x$ at stage $N$
 4:     if isReachable($x$) then
 5:         $R \leftarrow$ EvaluatePolicy($\pi_\theta$, x)
 6:         if $R \leq R_t$ then
 7:             $G$.append($x$)
 8:             break
 9:         end if
10:     end if
11: end while
12: if $len(G) \,\%\, K = 0$ then
13:     increase stage $N$
14:     increase $[d_1, d_2, d_3]$ and $\theta_{max}$
15: end if
16: if $\mathcal{N}(0,1) \leq \epsilon$ then
17:     $x^g$ sampled from $G$
18: else
19:     $x^g = x$
20: end if
21: $GD$.append($x^g$)
22: return $x^g$
```

Fig. 6.2 shows the proposed architecture. The architecture consists of representation and manipulation components. The manipulation components are an actor-network depicted by the purple box, parameterized by θ, and a critic network depicted by the yellow box, parameterized by ϕ. The representation components consist of an encoder network, a latent model, and a decoder network, depicted in blue and parameterized by ψ. Note that for simplicity, we didn't explicitly include the decoder network in Fig. 6.2. The inputs that consist of a history of image observation ($o_{t:t-t_L}$) and goal image (x_g) observation are first fed to the Encoder network to produce the feature vector for each image observation ($f_{t:t-t_L}$) and the goal image (f^g). The feature vectors and the history of past actions then are used to estimate the latent variables ($z^1_{t:t-t_L}, z^2_{t:t-t_L}, z^1_g, z^2_g$). In Fig. 6.2, the encoder and latent model networks are the same for both the history of images and the goal image; however, for better visualization, we showed them in separate paths. Note that since the goal observation is sampled from goal distribution, the goal latent variables (z^1_g, z^2_g) are not sequential and simply calculated using the initial posteriors estimation of goal observation as:

$$\begin{aligned} z^1_g &\sim q^1_\psi(z^1_g | x^g), \\ z^2_g &\sim q^2_\psi(z^2_g | z^1_g) \end{aligned} \tag{6.12}$$

where q^1_ψ, q^2_ψ are the initial posteriors networks of latent variable z^1 and z^2, respectively. Then we calculate the history of image observation error between the target image and the given image in latent space,

$$e^z_{1:t} = [e^{z^1}_1, ..., e^{z^1}_t, e^{z^2}_1, ..., e^{z^2}_t], \tag{6.13}$$

where,

$$e^{z^1}_t = z^1_t - z^1_g, e^{z^2}_t = z^2_t - z^2_g, \tag{6.14}$$

and finally fed to multiple fully connected layers. In contrast, for the actor network, we only used the encoder to calculate the feature vector error ($e^f t : t - t_L$) and fed it to multiple MLP layers to predict the actions. The difference between the latent vector and feature vector is that the latter does not need to satisfy the Markov property. We observed, that this choice of design not only makes the algorithm more robust in domain change to real-robot but also makes the algorithm simpler in execution time.

We train the critic network (parameterized by ϕ) using the following loss:

$$J_\phi = \mathop{\mathbb{E}}_{(e_{1:t+1}) \sim (q^1_\psi, q^2_\psi)} \left[\frac{1}{2} \Big(Q_\phi(e_t) - (r_t + \gamma V_{\bar{\phi}}(e_t, o_t, x^g) \Big)^2 \right] \tag{6.15}$$

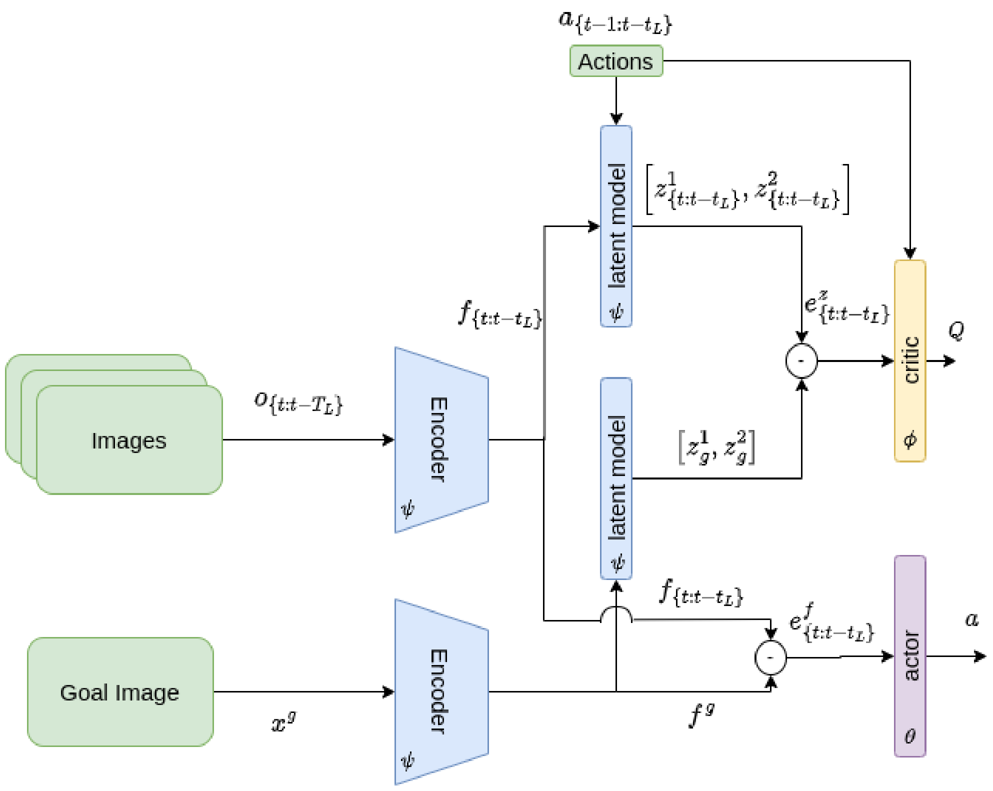


Figure 6.2: The VSLS proposed architecture

where $\bar{\phi}$ is the delayed target network that is given by

$$V_{\bar{\phi}}(e_t, o_t, x^g) = \mathop{\mathbb{E}}_{a_{t+1}\sim\pi_\theta}\Big[Q_{\bar{\phi}}(e_t) - \alpha \log \pi_\theta(a_{t+1}|o_t, x^g)\Big]. \tag{6.16}$$

We train the policy (actor) network (parameterized by θ) by maximizing the following objective [39]:

$$J_\theta = \mathop{\mathbb{E}}_{e_{1:t+1}\sim(q^1_\psi, q^2_\psi)}\left[\mathop{\mathbb{E}}_{a_{t+1}\sim\pi_\theta}\Big[\alpha \log \pi_\theta(a_{t+1}|o_{t+1}, x^g) - Q_\phi(e_{t+1})\Big]\right] \tag{6.17}$$

Note that in (6.17), the J_θ does not depend on the representation model parameters ψ. Based on (6.5), the latent space parameters (ψ) are optimized to minimize the ELBO loss using the reparameterization trick and

$$\begin{aligned} J_\psi = &\mathop{\mathbb{E}}_{(z^1_{1:t+1}, z^2_{1:t+1})\sim(q^1_\psi, q^2_\psi)}\Bigg[\sum_{t=0}^{\tau} - \log p_\psi(x_{t+1}|z^1_{t+1}, z^2_{t+1}) \\ &+ D_{KL}(\log q^1_\psi(z^1_{t+1}|x_{t+1}, z^2_t, a_t)|| \log q^2_\psi(z^2_{t+1}|z^1_{t+1}, a_t))\Bigg]. \end{aligned} \tag{6.18}$$

Alg. 6 describes the training algorithm. We first initialize network parameters and select a random goal in first curriculum stage. After M training steps the policy is evaluated on the current goal and if the evaluation return exceeds R_t the goal is considered solved. Next we repeat this process until the algorithm succeeds in solving K goals in stage N. Each time we solve the K goals in a curriculum stage, we increase N and repeat the process.

6.3.5 Data Augmentation

To address VS challenges and have a robust domain transfer we train our algorithm with a wide range of realistic images and conduct several data augmentations. First, we used a pretrained ResNet-18 [40] as our encoder. Second, we augmented the training data with CutOut data augmentation. CutOut data augmentation makes a random rectangle part of the image black to resemble the partial occlusion problem. Finally, we randomly added noises, change lighting color and lighting direction, posterize the image by reducing the number of bits in color channel and convert some of the images to gray.

6.3.6 Single-Shot Domain Transfer

As we will show in sec. 6.5, the method's performance degraded when we changed the domain of the trained policy from simulation to the real robot. We observed that with all of the considerations we applied in data augmentation and simulation design, there are some differences between the source (simulation) and the target (real-robot) domains that are not measurable and taken into account. To remedy this, we propose the single-shot transfer algorithm. In this method, we first run the algorithm for one episode (24 steps) and record the observed data on the real robot. Then we performed a few fine-tuning steps on the representation learning part, according to Line 11 in Algo. 6. Since the detail states, including the target pose and the reward function, may not be available in real scenarios, the proposed algorithm has an appealing specification that we can fine-tune the representation learning part without changing the manipulation learning part. Otherwise, the fine tunning procedure may not be possible in a real setting.

6.4 Training and Experiments

We implemented our algorithms using Pytorch and conducted all training on the Isaac Gym simulator[70]. Isaac Gym is a high-performance GPU-based learning platform for training policies. In this environment, physics simulation and neural network policy training communicate by passing data directly from physics buffers to PyTorch tensors on GPUs. To expedite the domain adaptation to real-robot by exposing the algorithm to large photo-realistic images, we used ImageNet [23], and Microsoft COCO [66]. We add images as a texture of an object to the Issac GYM simulation environment. For encoder architecture, we used the pre-trained ResNet-18 [40] model, and for the decoder, we used a custom

Algorithm 6 VSLS algorithm training loop

```
Require: initialize parameters $\phi, \theta, \psi, lr_\phi, lr_\theta, lr_\psi$
    initialize buffer $D$ and goal $x^g$
 1: while Not converge do
 2:     $R \leftarrow$ EvaluatePolicy($\pi_\theta$, goal)
 3:     if $R \geq R_t$ then
 4:         $x^g \leftarrow$ Curriculum()
 5:     end if
 6:     for step $s$ in {1, ... , M} do
 7:         $a_t \sim \pi_\theta(a_t|o_{1:t}, x^g)$
 8:         $r_t, o_{t+1} \leftarrow$ rollout action
 9:         $D \leftarrow D \cup (a_t, r_t, o_{t+1}, x^g)$
10:         sample mini batch $b^i \sim D$
11:         $\psi \leftarrow \psi - lr_\psi \nabla_\psi J_\psi(b^i)$ Optimize (6.15)
12:         $\theta \leftarrow \theta - lr_\theta \nabla_\theta J_\theta(b^i)$ Optimize (6.17)
13:         $\phi \leftarrow \phi - lr_\phi \nabla_\phi J_\phi(b^i)$ Optimize (6.18)
14:     end for
15: end while
```

decoder architecture. Fig. 6.3 depicts a view from developed simulation environments with a random scene from ImageNet [23]. As we discussed in sec. 6.2, we defined our problem as a finite-time horizon and solved the task in the finite-episodic method. As a result, we only consider 24 time-step episode lengths for our algorithm. Once the average translation error is less than 4 cm and the average orientation error is less than 5°, we consider the task completed. We chose these numbers because our method can bring the end-effector within the convergence area of Direct Visual Servoing (DVS) [19, 31], so then DVS may be applied to achieve precise end-point positioning It takes 4-5 days to train the algorithm through two million iterations on an NVIDIA RTX-3090 GPU, Intel® Core™ i7-9700K processor, and 64 GB of RAM.

For real robot experiments, we used a Kinova Gen-3 robot with 7 degrees of freedom and a RealSense RGBD camera[1] mounted in an eye-in-hand design. In the simulation, to solve the robot's inverse kinematic, a damped-least-square method was utilized [98]. This method provides robust and stable solutions that prevent bringing the robot to a singular pose. However, for the real robot, we used the built-in IK solver of the Kinova manipulator.

6.5 Results and Discussions

In this section, we will discuss the results of our proposed algorithm in simulation and real-world and aim to answer the following questions:

[1]Note that the depth data is not used in this study.

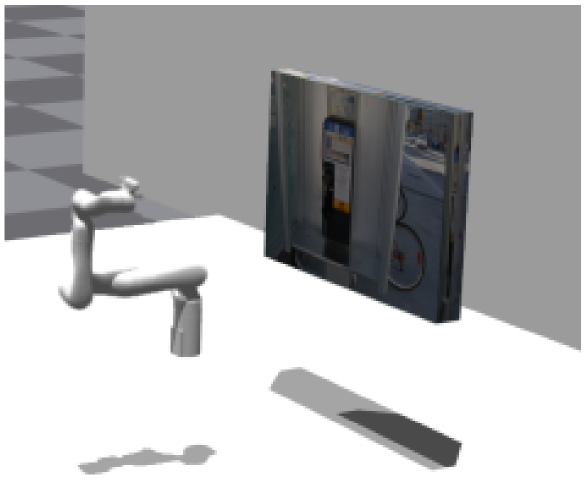


Figure 6.3: A snapshot from simulation environment.

- How scalable is our method?
- What is our proposed method's convergence rate compared to SOTA and baseline?
- How robust is our method in domain change to a real robot?

Scalability: To study the scalability of our method, we performed training on three cases. First, We used a dataset of 1000 images from ImageNet [23] by choosing one sample from each class. In the second, we used 1000 images from a single dog class in the Microsoft Coco dataset[66]. Third, we used a smaller version of ImageNet and custom images with only 104 scenes. Fig. 6.4 shows the average test return during training for three different datasets over five runs with different random seeds for each graph. In tests during training, the policy is acting deterministic, and we only used the mean prediction values of eq. (6.8) to the learning agent. As inferred from the graph, it took longer steps for a larger dataset to converge; however, the overall method is scalable to a large dataset. Moreover, unlike [30, 31], the convergence is not sensitive to a single class and is scalable to different large classes.

Convergence Rate comparison: To compare our method's convergence rate, we first train our method with a dataset of 1000 different scenes and objects from ImageNet[23] by sampling one image from each class of ImageNet. Adapting the test scenarios proposed in [30], we considered two cases: Look-at and Screw-Motion. In the first test case, the end-effector carrying the camera randomly moves to a point sampled from the robot's workspace and attempts to look at a random point within in the r-radius neighbourhood of the image center. The roll angle of the end-effector is randomly selected. Due to physical robot and experiment limitations, we sampled pose from a box [0.9 m, 0.9 m, 0.5 m] expanded from the home position of the robot equally in all directions. We calculate the pitch and yaw angles such that the camera look-at the 10 cm neighbourhood of the center of the image, and the roll angle is kept small. In the latter, we assume the end-effector has a displacement

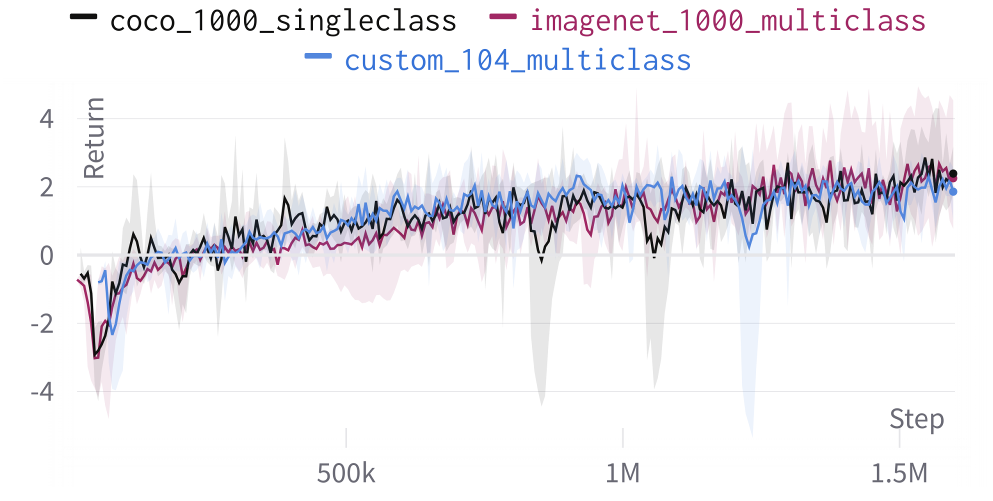


Figure 6.4: Average test return during training over five run with different random seed

in the range of [−0.25 m, 0.25 m] in the x-axis and rotation in the range of $[-70°, 70°]$. For each scenario, we randomly generate 500 samples and ensure that the sampled points are feasible for the manipulator by simply replacing any that are not feasible. The starting average errors are 21.12 cm ± 4 cm and 17.7° ± 4°.

We evaluate and compare our method with other direct VS approaches, namely DVS [7] as the baseline classical approach, AEVS [30] and Siamese-se(3) [31] as SOTA methods. Since the AEVS and Siamese-se(3) implementations are not open-source, we tried to make the experiments similar and used their reported result [30]. Note that AEVS and Siamese-se(3) results are not based on a camera attached to a robot with physic-based simulation, and are generated only for training with a one-single scene. In addition, we implemented DVS in our physics-based simulator using ViSP [72]. Table 6.1 compares our method with other direct VS methods quantitatively and qualitatively. As a measure of scalability, we consider the number of different images used in training, and as a measure of robustness, we consider whether the method includes data augmentation. According to Table 6.1, the success rate of our method for all scenarios is higher than the existing methods, while we tried to avoid overfitting in a single scene and did test on 1000 random scenes from the training dataset. Since the task is defined episodic with a finite time horizon, the RL task will be done once the end-effector is within the defined threshold in sec.6.4. Therefore, we didn't include the end-point accuracy as all successful tasks achieved the defined limit.
Real Robot Experiment: We finally deployed our method to a real 7 DOF Kinova Gen3 robot to test in real scenes. We used some of the scenes[2] from our custom-104 dataset as test objects. To show the importance of single-shot domain transfer, we used the decoder

[2]Images taken from Simon Fraser University library with permission

	Convergence Rate %		Scalability	Robustness
	Look-at	Screw motion		
DVS[19]	72	64	✓	✗
S-se(3)[31]	100	87	1 object	N/A
AE VS[30]	93.9	87	1 object	N/A
VSLS (ours)	96.8	88.6	1000 objects	✓

Table 6.1: comparison with other direct visual servoing methods

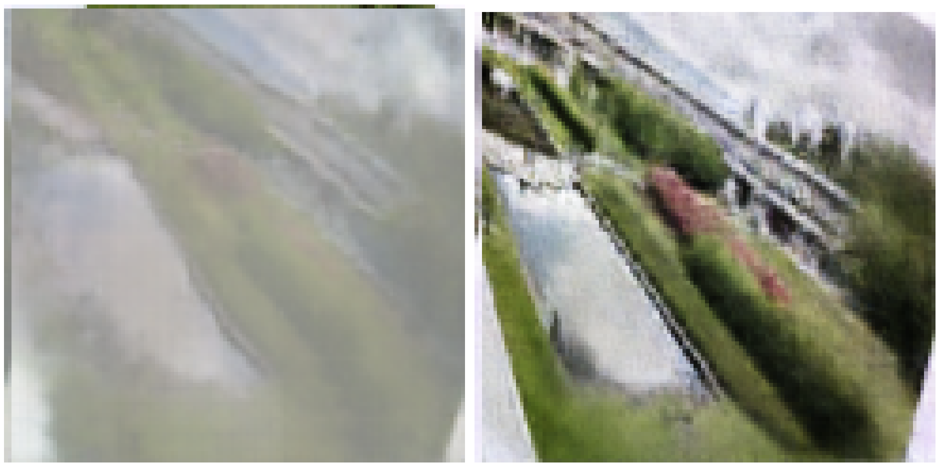
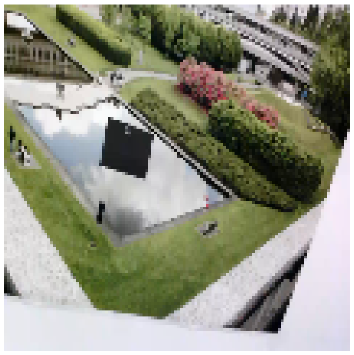

(a) (b)

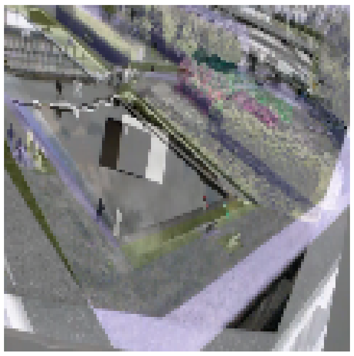
Figure 6.5: Reconstructed Image from Latent Space; (a) Before Single-Shot transfer; (b) After Single-Shot transfer

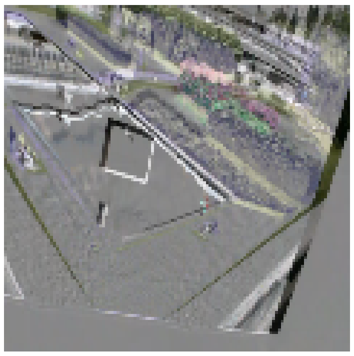
network $p(x_t|z_t^1, z_t^2)$ defined in eq. (6.5) to reconstruct a real image of the experiments from the encoded latent space. The reconstructed image is visualized in Fig. 6.5a and Fig. 6.5b depicts the reconstructed image after applying the single-shot domain transfer algorithm. It illustrates how single-shot domain transfer results in a significant reduction in reconstruction error. Note that the fine-tuning steps in single-shot transfer take less than two minutes on a computer with NVIDIA RTX-2080 GPU (including generating the goal). Fig. 6.6 depicts a whole experiment for a task, including partial occlusion of the test image by adding a black patch. The improvement achieved using the single-shot domain transfer also can be seen by comparing Fig. 6.6f and 6.6g. Based on Fig. 6.6d without the single-shot transfer, the trajectory errors are not converging to the defined threshold. After performing the single-shot transfer, the algorithm successfully converges in the defined task, as reflected in Fig. 6.6c.

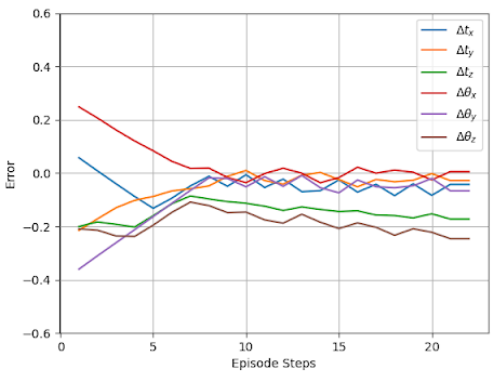
(a)


(b)

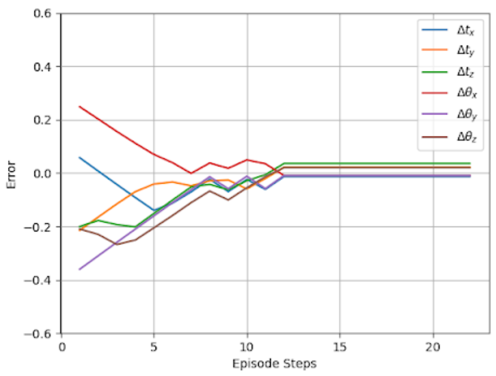
(c)


(d)


(e)



(f)



(g)

Figure 6.6: Real robot experiment with occlusion; (a) initial Image; (b) target Image; (c) initial Image difference; (d) final Image difference before single-shot transfer; (e) final image difference after single-shot transfer; (f) pose difference before single-shot transfer (g) pose difference after single shot transfer

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we propose some practical methods that can be used in robotic applications via deep reinforcement learning. We cover a wide range of robotic applications, including mobile robots, multi-agent navigation and robotic manipulators. From a sensor perspective, we also used Lidar and Image observation, two vital sensors in practical applications.

In Chapter 3, we presented the least-restrictive collision avoidance module (LR-CAM) that can be added on top of autonomous agents to intervene and avoid collisions based on the joint configuration of multiple agents. The LR-CAM used an LSTM-Based Variational Auto Encoder to mitigate varying size problems as the number of agent growth. While the LR-CAM directly used other robot positions that potentially need a detection algorithm for other agents, we showed it requires much less information than classical algorithms such as ORCA. In our study, we demonstrated that by utilizing computed relative information about the pose and orientation of other agents and applying an appropriate noise model to the localization data, the LR-CAM approach achieved zero-shot sim2real transfer.

Chapter 4 introduced a decentralized observation-based least-restrictive collision module (OLR-CAM). OLR-CAM acts as a high-level safety controller that detects potential conflict with dynamic agents and static obstacles and takes action to avoid them. In case of impending danger, it interrupts the default controller. We also proposed a practical reward function using Hamilton-Jacobi (HJ) reachability theory to extend to any unknown environment. In addition, we demonstrated that by employing a meta-training approach, utilizing direct LiDAR observations with an appropriate noise model, and incorporating sufficient domain randomization during the training process, our algorithm achieved zero-shot sim2real transfer. This means that the policy learned in the simulation could generalize effectively to real-world scenarios without requiring additional training or fine-tuning in the real environment. It is worth noting that the OLR-CAM agents utilized the same system dynamic models during the training process. However, the reward function employed is designed to accommodate different dynamics. This allows the same algorithm to be deployed in training in a multi-agent setting with varying system dynamics. The reward function can capture and adapt to these different dynamics, enabling effective training and coordination among agents with diverse behaviours. This flexibility and scalability make the OLR-CAM approach well-suited for various multi-agent applications.

In Chapter 5, we addressed some of the remaining limitations in the LR-CAM family of controllers by proposing the CLR-CAM. CLR-CAM didn't use the complicated reward function proposed in previous chapters and instead used expert demonstrations to accomplish continuous control for least-restrictive tasks. Compared to previous methods that used fix-wing aircraft movement (using constant linear velocity), CLR-CAM provides more robust manoeuvers by stopping, backing and having non-constant linear velocity.

Chapter 6 presents an RL-based goal-conditioned visual servoing algorithm using sequential latent space representation. The proposed approach is a stepping stone to solving the classic visual servoing problem robustly and scalable. It decomposes the problem into representation and manipulation learning. We showed that the trained algorithm could be robustly transferred to a real robot using only single shot-fine tunning on the representation learning part.

7.2 Future Work

Reinforcement Learning is a promising direction that probably will form the future of control and decision-making in robotics. Some ample directions and limitations need to be solved to be able to use RL algorithms in practical robotic applications robustly. In the first section, we discuss the future work on the presented case studies,

- We assumed the acting agent in our multi-agent system is Turtlebot Burger with a differential drive controller. While the system dynamics are known to the learning agent, the RL algorithm will implicitly learn the system dynamics. The learning agent cannot be easily transferred to another robot with the same control type but a different size and specification. One can also randomize the system dynamic and a meta-learning framework to relax this assumption and feed the specification as an observation.

- Due to the difficulties of LR as described in Chapter 3, the size of our multi-agent systems was limited; however, using CLR-CAM gives the promise to increase the number of agents with more data demonstration and more complicated task reward design.

- In the proposed VS approach, one of the limitations of our method is an end-positioning error. While in practice, we showed our method is able to bring the end-effector pose to the convergence area of classical VS approaches, one research direction is to make the algorithm end-to-end. One possible method is to change the action space of the algorithm and directly control the velocity of joints.

- In Chapter 6, we assumed the robot could easily move in the workspace without considering the possible collisions. One possible future work is to also consider collision avoidance in reward function.

Bibliography

[1] Konrad Ahlin, Benjamin Joffe, Ai-Ping Hu, Gary McMurray, and Nader Sadegh. Autonomous leaf picking using deep learning and visual-servoing. *IFAC-PapersOnLine*, 49(16):177–183, 2016. 5th IFAC Conference on Sensing, Control and Automation Technologies for Agriculture AGRICONTROL 2016.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[3] Salar Asayesh, Mo Chen, Mehran Mehrandezh, and Kamal Gupta. Toward observation based least restrictive collision avoidance using deep meta reinforcement learning. *IEEE Robotics and Automation Letters*, 6(4):7445–7452, 2021.

[4] Salar Asayesh, Mo Chen, Mehran Mehrandezh, and Kamal Gupta. Least-restrictive multi-agent collision avoidance via deep meta reinforcement learning and optimal control. In *Robot Intelligence Technology and Applications 7*, pages 213–225, Cham, 2023. Springer International Publishing.

[5] Daman Bareiss and Jur Van den Berg. Generalized reciprocal collision avoidance. *The International Journal of Robotics Research*, 34(12):1501–1514, 2015.

[6] Quentin Bateux, Eric Marchand, Jürgen Leitner, François Chaumette, and Peter Corke. Training deep neural networks for visual servoing. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3307–3314, 2018.

[7] Guillaume Caron, Eric Marchand, and El Mustapha Mouaddib. Photometric visual servoing for omnidirectional cameras. *Autonomous Robots*, 35(2):177–193, 2013.

[8] F. Chaumette. Image moments: a general and useful set of features for visual servoing. *IEEE Transactions on Robotics*, 20(4):713–723, 2004.

[9] Francois Chaumette and Seth Hutchinson. Visual servo control. i. basic approaches. *IEEE Robotics Automation Magazine*, 13(4):82–90, 2006.

[10] Changan Chen, Sha Hu, Payam Nikdel, Greg Mori, and Manolis Savva. Relational graph learning for crowd navigation. In *IROS*, 2020.

[11] Changan Chen, Yuejiang Liu, Sven Kreiss, and Alexandre Alahi. Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6015–6022. IEEE, 2019.

[12] Mo Chen, Jennifer C Shih, and Claire J Tomlin. Multi-vehicle collision avoidance via hamilton-jacobi reachability and mixed integer programming. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 1695–1700. IEEE, 2016.

[13] Mo Chen and Claire J. Tomlin. Hamilton–jacobi reachability: Some recent theoretical advances and applications in unmanned airspace management. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):333–358, 2018.

[14] Mo Chen and Claire J. Tomlin. Hamilton–jacobi reachability: Some recent theoretical advances and applications in unmanned airspace management. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):333–358, 2018.

[15] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 285–292. IEEE, 2017.

[16] Jason J Choi, Donggun Lee, Koushil Sreenath, Claire J Tomlin, and Sylvia L Herbert. Robust control barrier-value functions for safety-critical control. *arXiv preprint arXiv:2104.02808*, 2021.

[17] Yao-Li Chuang, Yuan R Huang, Maria R D'Orsogna, and Andrea L Bertozzi. Multi-vehicle flocking: scalability of cooperative control algorithms using pairwise potentials. In *Proceedings 2007 IEEE international conference on robotics and automation*, pages 2292–2299. IEEE, 2007.

[18] Earl A Coddington and Robert Carlson. *Linear ordinary differential equations*. SIAM, 1997.

[19] Christophe Collewet and Eric Marchand. Photometric visual servoing. *IEEE Transactions on Robotics*, 27(4):828–834, 2011.

[20] Alessandro Colombo and Domitilla Del Vecchio. Least restrictive supervisors for intersection collision avoidance: A scheduling approach. *IEEE Transactions on Automatic Control*, 60(6):1515–1527, 2015.

[21] Michael G Crandall and Pierre-Louis Lions. Viscosity solutions of hamilton-jacobi equations. *Transactions of the American mathematical society*, 277(1):1–42, 1983.

[22] Gabriel Rodrigues de Campos, Fabio Della Rossa, and Alessandro Colombo. Optimal and least restrictive supervisory control: Safety verification methods for human-driven vehicles at traffic intersections. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 1707–1712, 2015.

[23] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[24] Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. Goal-conditioned imitation learning. In *Advances in Neural Information Processing Systems 32*, pages 15324–15335. Curran Associates, Inc., 2019.

[25] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[26] Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.

[27] M. Everett, Y. F. Chen, and J. P. How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3052–3059, 2018.

[28] Tingxiang Fan, Pinxin Long, Wenxi Liu, and Jia Pan. Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios. *The International Journal of Robotics Research*, 39(7):856–892, 2020.

[29] Tingxiang Fan, Pinxin Long, Wenxi Liu, and Jia Pan. Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios. *The International Journal of Robotics Research*, 39(7):856–892, 2020.

[30] Samuel Felton, Pascal Brault, Elisa Fromont, and Eric Marchand. Visual servoing in autoencoder latent space. *IEEE Robotics and Automation Letters*, 7(2):3234–3241, 2022.

[31] Samuel Felton, Elisa Fromont, and Eric Marchand. Siame-se(3): regression in se(3) for end-to-end visual servoing. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14454–14460, 2021.

[32] C. Finn and S. Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793, 2017.

[33] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 1126–1135, Aug 2017.

[34] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519, 2016.

[35] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.

[36] Jaime F Fisac, Mo Chen, Claire J Tomlin, and S Shankar Sastry. Reach-avoid problems with time-varying dynamics, targets and constraints. In *Proceedings of the 18th international conference on hybrid systems: computation and control*, pages 11–20, 2015.

[37] Julio Erasmo Godoy, Ioannis Karamouzas, Stephen J Guy, and Maria Gini. Implicit coordination in crowded multi-agent navigation. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[38] Rodrigo Longhi Guimarães, André Schneider de Oliveira, João Alberto Fabro, Thiago Becker, and Vinícius Amilgar Brenner. Ros navigation: Concepts and tutorial. In *Robot Operating System (ROS)*, pages 121–160. Springer, 2016.

[39] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[41] Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.

[42] Sylvia Herbert. *Safe Real-World Autonomy in Uncertain and Unstructured Environments.* PhD thesis, EECS Department, University of California, Berkeley, Aug 2020.

[43] Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. In *International Conference on Machine Learning*, pages 1480–1490. PMLR, 2017.

[44] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[45] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[46] Seth Hutchinson, Gregory D Hager, and Peter I Corke. A tutorial on visual servo control. *IEEE transactions on robotics and automation*, 12(5):651–670, 1996.

[47] Frank J. Jiang, Yulong Gao, Lihua Xie, and Karl H. Johansson. Ensuring safety for vehicle parking tasks using hamilton-jacobi reachability analysis. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 1416–1421, 2020.

[48] van den Berg Jur, J. Guy Stephen, Snape Jamie, and Dinesh Manocha Ming, C. Lin and. Rvo2 library: Reciprocal collision avoidance for real-time multi-agent simulation. `https://gamma.cs.unc.edu/RVO2/`. Accessed: 2021-09-25.

[49] Taewon Kim, Yeseong Park, Youngbin Park, Sang Hyoung Lee, and Il Hong Suh. Acceleration of actor-critic deep reinforcement learning for visual grasping by state representation learning based on a preprocessed input image. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 198–205, 2021.

[50] Diederik P Kingma and Max Welling. Stochastic gradient vb and the variational autoencoder. In *Second International Conference on Learning Representations, ICLR*, volume 19, 2014.

[51] Diederik P Kingma and Max Welling. Stochastic gradient vb and the variational autoencoder. In *Second International Conference on Learning Representations, ICLR*, volume 19, page 121, 2014.

[52] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.

[53] Sulabh Kumra and Christopher Kanan. Robotic grasp detection using deep convolutional neural networks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 769–776. IEEE, 2017.

[54] Alex X Lee, Sergey Levine, and Pieter Abbeel. Learning visual servoing with deep features and fitted q-iteration. *arXiv preprint arXiv:1703.11000*, 2017.

[55] Alex X. Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 741–752. Curran Associates, Inc., 2020.

[56] Alex X Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *Advances in Neural Information Processing Systems*, 33:741–752, 2020.

[57] Alex X. Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 741–752. Curran Associates, Inc., 2020.

[58] Kuang-Huei Lee, Ian Fischer, Anthony Liu, Yijie Guo, Honglak Lee, John Canny, and Sergio Guadarrama. Predictive information accelerates learning in rl. *Advances in Neural Information Processing Systems*, 33:11890–11901, 2020.

[59] Sergey Levin. Deep reinforcement learning. University Lecture, 2021.

[60] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

[61] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[62] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[63] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International journal of robotics research*, 37(4-5):421–436, 2018.

[64] Anjian Li, Somil Bansal, Georgios Giovanis, Varun Tolani, Claire Tomlin, and Mo Chen. Generating Robust Supervision for Learning-Based Visual Navigation Using Hamilton-Jacobi Reachability. In *Conference on Learning for Dynamics and Control*, 2020.

[65] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[66] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[67] Chunming Liu, Xin Xu, and Dewen Hu. Multiobjective reinforcement learning: A comprehensive overview. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(3):385–398, 2015.

[68] Pinxin Long, Wenxi Liu, and Jia Pan. Deep-learned collision avoidance policy for distributed multiagent navigation. *IEEE Robotics and Automation Letters*, 2(2):656–663, 2017.

[69] Clare Lyle, Mark Rowland, Georg Ostrovski, and Will Dabney. On the effect of auxiliary tasks on representation dynamics. In *International Conference on Artificial Intelligence and Statistics*, pages 1–9. PMLR, 2021.

[70] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[71] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2017.

[72] E. Marchand, F. Spindler, and F. Chaumette. Visp for visual servoing: a generic software platform with a wide class of robot control skills. *IEEE Robotics and Automation Magazine*, 12(4):40–52, December 2005.

[73] Qinxue Meng, Daniel Catchpoole, David Skillicom, and Paul J Kennedy. Relational autoencoder for feature extraction. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 364–371. IEEE, 2017.

[74] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*, 2017.

[75] Ian M Mitchell, Alexandre M Bayen, and Claire J Tomlin. A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on automatic control*, 50(7):947–957, 2005.

[76] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[77] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[78] Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. *Advances in neural information processing systems*, 31, 2018.

[79] Sriraam Natarajan and Prasad Tadepalli. Dynamic preferences in multi-criteria reinforcement learning. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, page 601–608, New York, NY, USA, 2005. Association for Computing Machinery.

[80] Reza Olfati-Saber and Richard M Murray. Distributed cooperative control of multiple vehicle formations using structural potential functions. In *IFAC world congress*, volume 15, pages 242–248. Barcelona, Spain, 2002.

[81] Stanley Osher and Ronald Fedkiw. *Level set methods and dynamic implicit surfaces*, volume 153. Springer Science & Business Media, 2006.

[82] Gang Peng, Wei Zheng, Zezao Lu, Jinhu Liao, Lu Hu, Gongyue Zhang, and Dingxin He. An improved amcl algorithm based on laser scanning match in a complex and unstructured environment. *Complexity*, 2018, 2018.

[83] Xue Bin Peng, Yunrong Guo, Lina Halper, Sergey Levine, and Sanja Fidler. Ase: Large-scale reusable adversarial skill embeddings for physically simulated characters. *ACM Transactions On Graphics (TOG)*, 41(4):1–17, 2022.

[84] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics (TOG)*, 40(4):1–20, 2021.

[85] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, 2009.

[86] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5331–5340, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[87] Spyros A. Reveliotis and Elzbieta Roszkowska. On the complexity of maximally permissive deadlock avoidance in multi-vehicle traffic systems. *IEEE Transactions on Automatic Control*, 55(7):1646–1651, 2010.

[88] Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

[89] Diederik M Roijers, Shimon Whiteson, Frans A Oliehoek, et al. Linear support for multi-objective coordination graphs. In *AAMAS'14: PROCEEDINGS OF THE 2014 INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS & MULTIAGENT SYSTEMS*, pages 1297–1304, 2014.

[90] Stephane Ross and Drew Bagnell. Efficient reductions for imitation learning. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 661–668, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[91] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning, 2021.

[92] Fereshteh Sadeghi. Divis: Domain invariant visual servoing for collision-free goal reaching. *arXiv preprint arXiv:1902.05947*, 2019.

[93] Fereshteh Sadeghi, Alexander Toshev, Eric Jang, and Sergey Levine. Sim2real viewpoint invariant visual servoing by recurrent control. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4691–4699, 2018.

[94] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[95] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[96] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[97] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pages 8583–8592. PMLR, 2020.

[98] Masanori Sekiguchi and Naoyuki Takesue. Fast and robust numerical method for inverse kinematics with prioritized multiple targets for redundant robots. *Advanced Robotics*, 34(16):1068–1078, 2020.

[99] Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*, 2016.

[100] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[101] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[102] J. Snape, J. van den Berg, S. J. Guy, and D. Manocha. Smooth and collision-free navigation for multiple robots under differential-drive constraints. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4584–4589, 2010.

[103] Jiaming Song, Hongyu Ren, Dorsa Sadigh, and Stefano Ermon. Multi-agent generative adversarial imitation learning. *Advances in neural information processing systems*, 31, 2018.

[104] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*, 2020.

[105] Krishnan Srinivasan, Benjamin Eysenbach, Sehoon Ha, Jie Tan, and Chelsea Finn. Learning to be safe: Deep rl with a safety critic. *arXiv preprint arXiv:2010.14603*, 2020.

[106] Flood Sung, Li Zhang, Tao Xiang, Timothy Hospedales, and Yongxin Yang. Learning to learn: Meta-critic networks for sample efficient learning. *arXiv preprint arXiv:1706.09529*, 2017.

[107] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[108] Lei Tai, Jingwei Zhang, Ming Liu, and Wolfram Burgard. Socially compliant navigation through raw depth inputs with generative adversarial imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1111–1117, 2018.

[109] Sarah Tang, Justin Thomas, and Vijay Kumar. Hold or take optimal plan (hoop): A quadratic programming approach to multi-robot trajectory generation. *The International Journal of Robotics Research*, 37(9):1062–1084, 2018.

[110] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[111] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[112] Faraz Torabi, Garrett Warnell, and Peter Stone. Generative adversarial imitation from observation. *arXiv preprint arXiv:1807.06158*, 2018.

[113] A. M. TURING. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460, 10 1950.

[114] Peter Vamplew, Richard Dazeley, Adam Berry, Rustam Issabekov, and Evan Dekker. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine learning*, 84(1):51–80, 2011.

[115] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[116] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[117] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 1928–1935. IEEE, 2008.

[118] Hado van Hasselt, Diana Borsa, and Matteo Hessel. Reinforcement learning. University Lecture, 2021.

[119] Pravin P Varaiya. On the existence of solutions to a differential game. *SIAM Journal on Control*, 5(1):153–162, 1967.

[120] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[121] Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack Rae, Piotr Mirowski, Joel Z Leibo, Adam Santoro, et al. Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760*, 2018.

[122] Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021.

[123] Cunjun Yu, Zhongang Cai, Hung Pham, and Quang-Cuong Pham. Siamese convolutional neural network for sub-millimeter-accurate camera pose estimation and visual servoing. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 935–941. IEEE, 2019.

[124] J. Yu and S. M. LaValle. Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, 32(5):1163–1177, 2016.

[125] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager. Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells. *IEEE Robotics and Automation Letters*, 2(2):1047–1054, 2017.

[126] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.