Online Action Selection Methods for Multi-Agent Navigation

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Julio Erasmo Godoy Del Campo

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Prof. Maria Gini and Prof. Stephen J. Guy

June, 2016

Acknowledgements

This dissertation could not have been done if not by the immense support of my family and friends.

First, I want to thank my lovely wife, Tahere, for all your love and support through these years. Since before marrying me, you encouraged and supported me in all conditions. You came to the U.S. leaving everything behind just for love. You helped me through the hardships of graduate school in so many ways that I cannot recount. I could not have found a better partner for eternity. Thank you for believing in me and in us, I love you forever. And I cannot thank my wife without thinking of our little daughter Nilufar, whose beautiful smile gave me the last push that I needed to finish this thesis. Thank you Nilu, for choosing me as your father and giving me the privilege of loving you.

I want to thank my parents for their unconditional love through all my life. I cannot express in words my gratitude towards you. You encouraged me to do my best in everything, and to follow my passion. Thank you for exposing me to technology and to the English language when I was just a boy. It is because of you that I chose Computer Science as my professional field. During this long process, you have always been there when I needed you.

I want to thank my advisers, Maria Gini and Stephen J. Guy. Thanks to you, I was able to work in the exciting field of Artificial Intelligence, and I look forward to continue to collaborate with you from Chile. You had the perfect combination of advising styles that was key in helping me to finish this dissertation in a timely manner. Thank you Maria, for your patience and for encouraging me to find new ways of doing things, for taking time off your busy schedule to advise me and to help me with all my academic duties. Finally, thank you for suggesting me to take the Motion in Games

class, which gave me the opportunity to meet professor Stephen J. Guy and post-doc Ioannis Karamouzas. Thank you Stephen, for all the time you spent discussing ideas for new work with me, for your patience with my stubbornness, and for teaching me to believe in my work. Thank you Ioannis, for all your help in the work presented in this dissertation, your detailed corrections to my paper drafts, and for sharpening the ideas that produced this dissertation.

During my stay in Minnesota, I had the privilege of working with very smart students from the Artificial Intelligence and Applied Motion groups, who became the best friends during my time in grad school. Thank you Ernesto, Bilal, Elizabeth, Marie, James, Nick, Mark, Fenix, Nick J. and Nick S., Bobby, Ran and John, for the insightful discussions that helped me in shaping my work. I hope that our friendship and the academic bonds that we forged these years continue for whatever the future holds.

Thank you all!

Dedication

For all your love and support, I dedicate this dissertation to my family: Gladys, Juan, Paulina, my wife Tahere and our little daughter Nilufar.

Abstract

In multi-agent navigation, agents have to move from their start positions to their goal locations while avoiding collisions with other agents and any static element in the environment. Existing methods either compute the motion of each agent centrally or allow each agent to compute its own motion. Using a central controller limits the number of agents that can be controlled in real time, while using a local method produces motions that are optimal locally but do not account for the motions of the other agents, producing inefficient global motions when many agents move in a crowded space. This dissertation proposes a set of online action selection methods that each agent uses to dynamically adapt its behavior to the local conditions. Specifically, we propose four approaches based on learning, planning, coordination and model inference to improve the global motions of a set of agents. These approaches are highly scalable because each agent makes its own decisions on how to move. We validate the approaches experimentally, with multiple simulations in a variety of environments and with different numbers of agents. When compared to other techniques, the proposed approaches produce motions that are more efficient and make better use of the space, allowing agents to reach their destinations faster.

Contents

Acknowledgements **i**

Dedication **iii**

Abstract **iv**

List of Tables **ix**

List of Figures **x**

1 Introduction **1**

1.1 Contributions 2

2 Literature Review **7**

2.1 Multi-Agent Navigation 7

2.2 Online Motion Planning 8

2.2.1 Experience-based motion evaluation 9

2.2.2 Long-term predicted motion value 11

2.2.3 Implicitly coordinated motion 13

2.2.4 Inferring the types of agents 15

3 Problem definition and background **17**

3.1 Problem formulation 17

3.1.1 Notation 17

3.1.2 Objective function 18

3.1.3 Complexity 18

3.1.4 Assumptions . . . 18
3.1.5 Research Question . . . 19
3.2 Collision Avoidance . . . 19
3.3 ORCA . . . 20
3.4 Limitations of ORCA . . . 22

4 Action-Selection Framework for Multi-Agent Navigation 24
4.1 Motion Diversity . . . 25
4.2 Reward Function . . . 26
4.3 Performance Metric . . . 26
4.4 Scenarios . . . 27
4.5 Experimental Setup . . . 28

5 Learning from the past (ALAN) 29
5.1 Action Evaluation . . . 30
5.2 Action Selection . . . 32
5.2.1 ϵ-greedy . . . 33
5.2.2 wUCB . . . 34
5.3 ALAN . . . 35
5.4 Scenarios . . . 37
5.5 ALAN Results . . . 38
5.6 Analysis . . . 39
5.6.1 Runtime Complexity . . . 41
5.6.2 Effect of the parameter γ . . . 41
5.7 Limitations of ALAN . . . 42

6 Predicting the Future (PHOP) 44
6.1 Motion Simulation . . . 44
6.2 Uncertainty . . . 45
6.3 Hindsight Optimization . . . 46
6.4 Progressive Hindsight Optimization . . . 47
6.4.1 Plan Generation and Execution . . . 48
6.4.2 Algorithmic Description . . . 48

6.5 PHOP Results . . . 49
6.6 Analysis . . . 53
6.6.1 Runtime Complexity . . . 53
6.6.2 Optimizing Action Evaluation . . . 53
6.6.3 Effect of the parameter γ . . . 58
6.6.4 Heterogeneous Agents . . . 59
6.7 Limitations of PHOP . . . 63

7 Implicitly Coordinated Navigation (C-Nav) 65
7.1 The *C-Nav* Approach . . . 66
7.2 Agent neighborhood information . . . 67
7.2.1 Motion similarity . . . 68
7.2.2 Constrained neighborhood motion . . . 71
7.3 Action evaluation and selection . . . 74
7.4 Experiments . . . 77
7.4.1 Results . . . 79
7.5 Analysis . . . 81
7.5.1 Runtime Complexity . . . 83
7.6 Theoretical Analysis . . . 83
7.6.1 Scalability . . . 85
7.6.2 Effect of the coordination-factor (γ) . . . 85
7.6.3 Effect of number of constrained neighbors k . . . 91
7.7 Limitations of C-Nav . . . 91

8 Navigation Among Heterogeneous Agents 93
8.1 Formulation . . . 93
8.1.1 Proposed method . . . 94
8.1.2 Navigation models . . . 95
8.2 Bayesian Model Inference . . . 96
8.2.1 Model Estimation and Selection . . . 97
8.3 Action Planning . . . 99
8.3.1 Planning in the space of preferred velocities . . . 99
8.3.2 Planning in the space of navigation models . . . 101

8.4 Results . . . 102
8.4.1 Model Prediction . . . 103
8.4.2 Time-efficiency . . . 104
8.4.3 Social Planning . . . 106
8.5 Analysis . . . 107
8.5.1 Runtime Complexity . . . 108
8.5.2 Number of collisions . . . 108
8.6 Limitations of the approach . . . 109

9 Conclusions and Future Work **110**
9.1 Limitations . . . 111
9.1.1 Discretized action set . . . 111
9.1.2 Application to multi-robot navigation . . . 111
9.1.3 Application to pedestrian simulation . . . 112
9.2 Future Work . . . 112
9.2.1 Theoretical Analysis . . . 113
9.2.2 Validation in Multi-Robot Systems . . . 113
9.2.3 Validation in Pedestrian Simulations . . . 113

References **114**

List of Tables

8.1 Accuracy (%) of VelPlan in terms of goal and type prediction in four scenarios. 104
8.2 Interaction overhead (s) in Social Planning. 106
8.3 Average collisions per iteration using ORCA, NoInference and VelPlan. 108

List of Figures

1.1 Conflicting constraints between the agents produce congestion and inefficient global motion. (a) Two groups of agents moving in opposite directions must pass through a narrow doorway. (b) Multiple agents must exit a room through a narrow doorway to reach their goals. . . . 2

3.1 (a) Two agents, A_i and A_j, moving towards a potential collision. (b) The set of allowed velocities for agent A_i induced by agent A_j is indicated by the half-plane delimited by the line perpendicular to $\hat{\mathbf{u}}$ through the point $\mathbf{v_i} + \frac{1}{2}\mathbf{u}$, where $\mathbf{u}$ is the vector from $\mathbf{v_i} - \mathbf{v_j}$ to the closest point on the boundary of $VO_{i|j}$ 21

3.2 Three agents cross paths. (a) Initial positions of the agents. (b) Goal positions of the agents. (c) When navigating with ORCA, the agents run into and push each other resulting in inefficient paths. (d) When using the proposed PHOP approach the agents plan around the potential results of the upcoming collision, resulting in more efficient paths. 22

4.1 Actions. The eight available actions correspond to moving at 1.5 m/s with different angles with respect to the goal: $0°$, β, $-\beta$, $90°$, $-90°$, $180°$, $180° + \beta$ and $180° - \beta$. In our implementation, $\beta = 45°$. 25

4.2 Example of simulated scenarios: (a) *Intersection*, (b) *Deadlock*, (c) *Perpendicular Crossing* and (d) *Circle*. 27

5.1 Example of reward values for different actions under clear and congested local conditions. The reward $\mathcal{R}_a$ of each action a is described by a goal-oriented and a politeness component ($\mathcal{R}_a^{goal}$, $\mathcal{R}_a^{polite}$). 31

5.2 Simulated scenarios: (a) *Intersection*, (b) *Deadlock*, (c) *Perpendicular Crossing* and (d) *Circle*. 38

5.3 Performance comparison between ORCA, ϵ-greedy, wUCB and ALAN. In all scenarios with the exception of the Circle, ALAN agents outperform agents using ORCA, ϵ-greedy and wUCB. 40
5.4 Performance of ALAN agents with different values of γ. 41
6.1 Agent A's goal-oriented motion is locally optimal from its initial state (a), but results in local minima which either cannot be resolved (b) or requires backtracking on the agent's path (c). 46
6.2 Example of progressive agent trajectories with 1, 2 and 3 partitions respectively. 49
6.3 Additional simulated scenarios for PHOP. (a) *3-Room*: 32 agents start in the rightmost room, and must cross two sets of exits to reach their goal in the leftmost room. (b) *Square*: 100 agents uniformly distributed must reach their randomly generated goals, while a static obstacle in the middle of the scenario blocks their path. 51
6.4 Performance comparison between ORCA, ALAN and PHOP. 52
6.5 Speedup in planning time using the optimized action evaluation method, with respect to doing a complete search evaluating all actions. 58
6.6 Performance of ALAN and PHOP agents for different values of γ. 59
6.7 *Intersection* scenario with the ID of each agent 60
6.8 The *Deadlock* scenario using different combinations of ORCA and PHOP agents. The top illustration in each subfigure shows the initial positions of the agents, while the bottom one shows the resulting behavior. 61
6.9 Interaction overhead for different percentages of PHOP agents, in the *Circle* scenario. The rest of the agents use only ORCA. 63
6.10 Interaction overhead for different percentages of PHOP agents, in the *Square* scenario. The blue line denotes the number of times, out of 10 iterations, when all agents reached their goals. 64
7.1 Two groups of 9 agents each move to the opposite side of a narrow corridor. (a) ORCA agents get stuck in the middle. (b) Using our *C-Nav* approach, agents create lanes in a decentralized manner. 66
7.2 Neighborhood-based actions: follow a specific neighbor agent or the goal-oriented motion at maximum speed. 67

7.3 Neighbors considered in the computation of constraints. Considering only agents closer to the goal ensures that no two agents with the same goal will simultaneously defer to each other. 73
7.4 Computation of constraints based on the preferred and observed velocities of the neighbors. 73
7.5 Scenarios. *Bidirectional*: two groups of agents move to the opposite side of a narrow corridor. *Crowd*: agents' initial and goal positions are placed randomly in a small area. *Circle*: agents move to their antipodal positions. *Congested*: agents try to exit an area through a small doorway. *3-Exit*: agents have three homotopic paths that they can take to reach their goals, while congestion forms in the closest goal-directed path. *PerpCrossing*: the goal paths of two agents orthogonally interesect a crowd of 24 agents. 78
7.6 Performance comparison between ORCA, ALAN and the *C-Nav* approach. In all scenarios, agents using our coordination approach have the lowest overhead times. The error bars correspond to the standard error of the mean. 80
7.7 Performance comparison between the different variations of *C-Nav*, where agents can: not communicate any motion information ($None$), communicate only their preferred velocities ($Vpref$), only their goal velocities ($Vgoal$) or both ($Vpref + Vgoal$). Results show that, in almost all cases, communicating motion-related information helps to reduce the interaction overhead of the system of agents. The error bars correspond to the standard error of the mean. 82
7.8 Scalability results in the Bidirectional and Circle scenarios, in terms of interaction overhead time. In Bidirectional, the number of agents varied from 50 to 200. In the Circle, the number of agents varies from 64 to 512. 85
7.9 Performance of *C-Nav* agents in the Bidirectional scenario, with different values of the coordination-factor γ. 86
7.10 Performance of *C-Nav* agents in the Congested scenario, with different values of the coordination-factor γ. 88

7.11 Performance of *C-Nav* agents in the Circle scenario, with different values of the coordination-factor γ. 89
7.12 Interaction overhead time as a function of the number k of constrained neighbors. 90
8.1 Example of the Bayesian model inference. (a) α predicts the position that its neighbor would move to in the next timestep, using either m_0 or m_1. (b) After the agents move, α computes the likelihood of each model given its neighbor's observed position. 96
8.2 The 9 actions of VelPlan correspond to moving at 1.5 m/s with different angles with respect to the goal: $0°$, β, $-\beta$, $90°$, $-90°$, $180°$, $180° + \beta$, $180° - \beta$ and complete stop. In our implementation, $\beta = 45°$. (a) Simple plans consist of single actions for the entire time horizon. (b) More complex plans are computed if planning time allows. 100
8.3 Example of different ORCA behaviors. (a) Using a shy model, α decides to move around the group. (b) Using the aggressive model, instead, α makes way between the incoming agents to reach the goal faster. 102
8.4 Scenarios. (a) *Incoming*. (b) *Perp1*. (c) *Perp2*. (d) *Circle*. (e) *TwoPaths*. 103
8.5 Interaction overhead for all evaluated approaches. In all scenarios, VelPlan outperforms ORCA and NoInference. The error bars denote the standard error of the mean. 105
8.6 Interaction overhead for all evaluated approaches with agents of the same model in the *Incoming* scenario. In all cases, VelPlan matches or outperforms ORCA and NoInference. 106
8.7 Social Planning example. (a) Initial position of the agents. (b) The agent uses aggressive behavior until its motion is constrained by the group. (c) The agent switches to a Tense behavior which pushes him away from the group. (d) After the group has passed by, the agent again assumes an aggressive behavior to move fast and unconstrained to its goal. 107

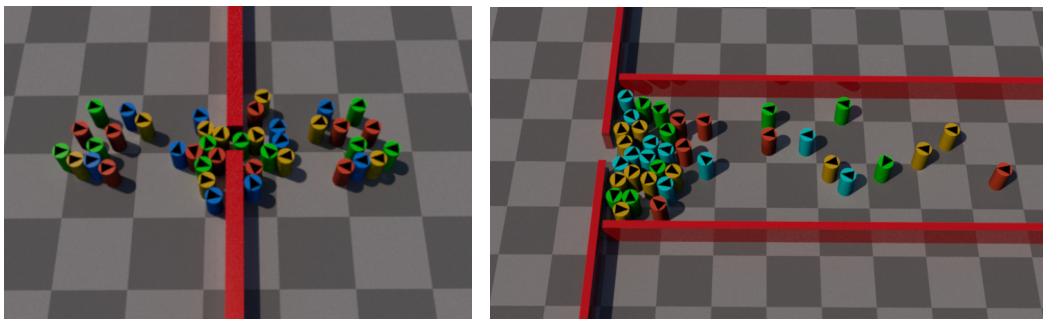
Chapter 1

Introduction

Real-time goal-directed navigation of multiple agents in crowded environments, where each agent can only observe its nearby agents, has important applications in many domains such as swarm robotics, planning for evacuation, crowd simulation and traffic engineering. The problem is challenging because the agents have conflicting constraints. On one hand, they need to reach their goals as soon as possible while avoiding collisions with each other and any static obstacle present in the environment. On the other hand, due to the presence of many agents and the real-time constraints, agents need to compute their motions independently of each other and in a decentralized manner instead of planning in a joint configuration space.

A recently introduced decentralized technique for real-time multi-robot navigation, ORCA [1] guarantees collision-free motion for the agents. Although ORCA generates locally efficient motion for each agent, the overall behavior of the agents can be far from efficient; actions that are locally optimal for one agent are not necessarily optimal for the entire system of agents. Consider, for example, the two possible scenarios in Fig. 1.1 for the agents that try to reach the other side of a narrow doorway. Here, the congestion around the doorway causes long delays and an inefficient crowd motion. If, instead, agents coming from the back were able to choose different motions, such as to stop or temporarily move away from the doorway, the congestion would resolve and the global motion would be much more efficient.

This dissertation focuses on applying *online* methods to select motions that are completely distributed and require limited or no communication among the agents.



(a) (b)

Figure 1.1: Conflicting constraints between the agents produce congestion and inefficient global motion. (a) Two groups of agents moving in opposite directions must pass through a narrow doorway. (b) Multiple agents must exit a room through a narrow doorway to reach their goals.

This work aims at enabling agents to make intelligent motion decisions that will take them to their goals faster, for example, to efficiently perform critical tasks such as search and rescue operations. Current literature on multi-agent learning focuses on scenarios with only a few agents and sparse interactions, where the policies for each agent are computed offline. Offline approaches are not suitable when quick response time and robustness to dynamic environments are desired, as they require a long training phase and the prediction of all potential scenarios the agent might encounter. In addition, the computational complexity of centralized offline learning methods becomes prohibitively high as the number of agents increases. In contrast, online approaches are suitable for dynamic environments, as agents must be able to quickly adapt their behaviors to changes in their surroundings.

1.1 Contributions

The contributions of this dissertation allow individual agents to select – independently and in real time – actions that help all agents reach their goals in a time-efficient manner. Specifically, this dissertation proposes four technical contributions, which correspond to four methods that agents can use to evaluate and select their next action. These methods use either learning, planning, group-based motion optimization, and Bayesian inference

to make action decisions in a navigation task. In all methods, we extend the range of motions of the agents to increase the diversity of behaviors they can exhibit, beyond the myopic goal-oriented motion commonly considered in the literature [1].

In the first part of this dissertation, we study the problem of learning the value of the actions from past experience. We propose a novel and general framework for incorporating learning methods in multi-agent navigation, the ALAN framework (Adaptive Learning for Agent Navigation). In this framework, action selection approaches can be incorporated in multi-agent navigation tasks, allowing agents to exhibit a diversity of behaviors. We formulated the problem of selecting the best motion at each time as an action selection problem in a multi-armed bandit setting. In this formulation, the challenge is to carefully balance action exploration and exploitation. ALAN uses an action selection method inspired by the principles of two well-known action selection techniques, ϵ-greedy and Upper Confidence Bounds (UCB) [2]. Agents using ALAN exploit the best action in a greedy fashion and perform biased exploration using a version of UCB more suited to non-stationary domains. Further, ALAN introduces game-theoretic elements, considering the local context of an agent to strategically adapt the amount of exploration performed [3, 4]. Combined with a reward function that considers goal-oriented motion and neighborhood interactions, ALAN allows agents to adapt their motion to their local conditions (i.e., move back or sideways when goal-oriented motion is constrained). Agents using ALAN take advantage of pure goal-oriented motion when they are able to, and perform biased exploration when that motion is constrained. This indirectly improves the global efficiency of the motions of all agents, allowing them to reach their destinations faster while outperforming other action selection techniques in different environments.

In the second part of this dissertation, we study the problem of reducing the uncertainty of estimating the long-term value of each action. To address this problem, we propose an anytime local approach to plan the motions of the agents in a decentralized manner, by adapting the Hindsight optimization technique [5] in a progressive manner. We call this method Progressive Hindsight Optimization (PHOP) [6]. PHOP reduces the uncertainty in the long-term effects of the motions of the agents by generating 'snapshots' of potential future scenarios. Specifically, each agent simulates possible plans of actions for a given time horizon, and after assessing each one of these simulated plans,

it evaluates in 'hindsight' the quality of the first action of the plan. Each plan consists of a sequence of motion primitives. By performing multiple simulations with each initial action, the agent reduces the uncertainty associated with the long-term consequences of each one of them. With PHOP, agents are able to predict regions in the environment that will introduce motion constraints, allowing them to act accordingly (for example, by completely avoiding paths going through those regions). Results of comparing PHOP with ORCA indicate lower travel time for the agents using PHOP.

Although both PHOP and ALAN produce time-efficient motions for all agents, they do not scale appropriately to highly-dense environments, where the agents' movement is very constrained. Further, in both approaches the agents consider each other only as dynamic obstacles, while agents have intended motions and goal locations that guide their motions. To address this problem, we propose each agent selects its actions in a way that benefits not only itself but also its neighboring agents.

We implemented this idea in the third part of this dissertation by proposing C-Nav (short for Coordinated Navigation), a distributed approach to improve the global motion of a set of agents in crowded environments by implicitly coordinating their local motions [7]. This coordination is achieved by allowing each agent to learn the likely motion of its nearby neighbors. C-Nav requires no bidirectional communication, and as such, it can scale well to hundreds of agents. Agents using C-Nav choose actions that help their nearby agents to move to their goals, mimicking the way humans move in congested environments and effectively improving the time-efficiency of the entire crowd.

A key assumption in the three contributions described is that the agents are homogeneous (i.e., they use the same navigation algorithm), and only differ in their initial and goal positions. However, as robotic agents are deployed in the real world, they will need to interact with humans and other types of agents, whose motion might not be accurately predicted using this homogeneity assumption. This uncertainty limits the ability of the agents to compute safe and efficient paths to their goals.

In the fourth and last part of this dissertation, we address this problem. We use Bayesian inference to estimate the types of other agents in the environment and, based on their estimated type, we plan a path for a single controlled agent that maximizes its time efficiency while computing collision-free paths. We consider a set of possible

navigation models for the agents which are widely used in the multi-agent navigation community, at the same time taking into account agents that might not use any of these models (for example, due to robot malfunction) [8]. Results show that an agent using our approach to estimate the type of each neighbor moves more efficiently and with less collisions to its goal, compared to its motion without the model estimation.

All these contributions improve the quality of the global motion of the agents in a decentralized manner, in real-time, and using only limited communication or no communication at all. Agents adapt online to large-scale complex multi-agent environments using observations of the dynamics of other agents.

Finally, to evaluate the performance of the contributions presented in this dissertation, we propose a new metric called *Interaction Overhead* which measures the time that agents spend in interactions with other agents. An *Interaction Overhead* value of 0 represents the minimum travel time for the agents in each environment, and is also the best possible result that any approach can achieve.

The remaining of this dissertation is organized as follows:

- Chapter 2 presents a literature review of the research in multi-agent navigation, learning and coordination, which are relevant to this thesis.
- Chapter 3 presents the problem addressed in this dissertation, the main definitions and background knowledge on the collision-avoidance approach used as a building block for the proposed contributions.
- Chapter 4 introduces the overall action selection framework for multi-agent navigation that forms the central contribution of this thesis.
- Chapter 5 presents the ALAN approach, which uses a multi-armed bandit formulation and proposes a new action selection technique for multi-agent navigation.
- Chapter 6 presents the PHOP approach, which shows how planning can be used to estimate the value of each action.
- Chapter 7 presents the C-Nav approach, which allows agents to implicitly coordinate their motions to exhibit intelligent behavior.

- In Chapter 8 presents VelPlan and SocialPlan, which allow agents to navigate in environments with different types of agents, by inferring their navigation model and goal.
- Chapter 9 concludes this thesis and presents possible lines of future work.

Chapter 2

Literature Review

The work developed in this dissertation spans over the areas of navigation, on-line learning and planning, coordination and goal inference in multi-agent systems. In this chapter, we present an overview of the relevant prior work that is related to this thesis.

2.1 Multi-Agent Navigation

Extensive research in the area of multi-agent navigation has been conducted over the last decade. Here we review the prior work most closely related to our research. For a more comprehensive discussion of multi-agent navigation, we refer the reader to the survey of Pelechano et al. [9].

Numerous models have been proposed to simulate individuals and groups of interacting agents. The seminal work of Reynolds on *boids* has been influential on this field [10]. Reynolds used simple local rules to create visually compelling flocks of birds and schools of fishes. Later, he extended this model to include autonomous agent behaviors [11]. Since Reynolds's original work, many interesting crowd simulation models have been introduced that account for groups [12], cognitive and behavioral rules [13, 14], biomechanical principles [15] and sociological or psychological factors [16, 17, 18]. Recent work has been done to model the contagion of psychological states in a crowd of agents, for example, in evacuation simulations [19].

An extensive literature also exists on modeling the local dynamics of the agents and computing a collision-free motion among static and/or dynamic obstacles. Over

the past two decades many different agent-based techniques for local collision avoidance have been proposed in control theory, traffic simulation, robotics and animation.

Khatib [20] pioneered the use of artificial potential fields in robotics, where the robot is treated as a particle which is attracted to the goal and repelled by obstacles. Brooks [21] popularized the use of artificial potential fields for robot navigation, using the artificial force for position control and not force control. There is a long history of research approaches to deal with dynamic environments where robots need to adapt to the motions of other robots to avoid collisions. Approaches range from methods to compute the motions in real-time as we do in this dissertation (e.g., [22, 23]) to methods for planning the motions in advance (e.g., [24]).

Force-based approaches which treat agents as particles and model their interactions using simulated physical forces are popular not just for robots but also for agents [11, 25, 26]. However, these methods do not account for velocities of the agents, which leads to issues especially in environments where agents move at high velocities. To address this issue, *geometrically-based* algorithms have been proposed, which compute collision-free velocities for the agents using either sampling [27, 28, 29, 30] or optimization techniques [1, 31]. Recently, the authors in [32] developed a statistical-mechanical approach to derive a power law which efficiently describes the interactions between pedestrians. In this dissertation, we use a geometrically-based navigation method which accounts for agents' velocities and it is robust to different types of environments.

2.2 Online Motion Planning

This dissertation focuses on methods to improve the time-efficiency in the navigation of a set of agents, by allowing each agents to adapt its motion to its navigation conditions. To make these motion decisions appropriately, agents need to estimate their value in real-time, using the limited available information. We propose a set of mechanisms that agents can use to evaluate their motions: learning their value using past experience, predicting their long-term expected value, predicting their effect in the nearby agents, and predicting their value based on interactions with agents of different types. In this section, we present an overview of the work most related to the proposed approaches in each case.

2.2.1 Experience-based motion evaluation

Agents can learn the value of their motions using past experience, by trying them in the environment and obtaining feedback from their execution. Here, we cast the problem of evaluating the motions as a reinforcement learning problem, which has been used extensively in the multi-agent literature [33]. Reinforcement Learning (RL) is a popular machine learning technique that addresses how autonomous agents can learn by interacting with the environment to achieve a desired goal [34].

A RL agent performs actions that affect its state and environment, and receives a reward value indicating the quality of the performed action and state transition. This reward is used as feedback for the agent to make better future decisions. RL is often used to solve optimization problems framed as Markov Decision Processes (MDP), where the objective is to find a mapping of actions to states (a policy) that maximizes a given performance metric. Different approaches have been proposed to incorporate RL when multiple agents share the environment (see [33, 35, 36] for an extensive overview).

In multi-agent RL algorithms, agents typically need to collect information on how all the other agents behave and find a policy that maximizes their reward. This is expensive when the state space is very large and requires a significant degree of exploration to create an accurate model for each agent. Hence, approaches that model the entire environment are focused on small problems and/or a very limited number of agents. To reduce complexity, some approaches focus on the local interaction neighborhood of each agent [37, 38]. By considering a local neighborhood, the state space of each agent is reduced.

Most approaches for multi-agent reinforcement learning address the learning process by first training the agents in the environment [39, 40, 41, 42]. During the training phase, agents learn the MDP model by executing a given task (for ex., reaching a goal state) multiple times, until the learning process converges to a stationary policy. In these cases, learning is typically evaluated by the rate at which agents learn better policies, and the quality of the policy obtained.

In real-time multi-agent navigation, however, agents need to learn the value of their actions on-line while the simulation is running, in a single learning task. Since it is unlikely agents will find themselves again in a similar local situation while moving to

their goals, and since they have only limited information of their surroundings, a state-based MDP-based learning is not ideal. Instead, we adapt a stateless representation for learning the motions' values.

Online learning problems with a stateless representation can be well formulated as a multi-armed bandit problem [34], where the agents use the past reward of each action to make future decisions. In multi-armed bandit problems, the relation between exploiting the current best action and exploring potentially better actions, that is, the exploration/exploitation dilemma, is critical [43, 44]. To address this problem, a variety of action selection techniques has been proposed in the literature.

Action Selection approaches. To address the exploration/exploitation dilemma, agents must carefully balance the exploitation of the currently best known action and the exploration of other actions that might potentially perform better.

One of the most popular action selection approaches, ϵ-greedy [34], chooses among the best action and a random one in a probabilistic manner. Softmax (also called Boltzmann) action selection [34] chooses actions probabilistically, proportionally to their relative observed rewards. Upper Confidence Bounds (UCB) [2] is a deterministic approach that works by sampling the actions proportionally to the upper-bound of the estimated value of their rewards. We take advantage of the properties of some of these approaches in the navigation domain, hence enabling the agents to make intelligent motion decisions.

Learning in Multi-Agent Navigation. Extensive work has also been done on learning and adapting the motion behavior for agents in crowded environments. Depending on the nature of the learning process, this work can be classified in two main categories: offline and online learning. In offline learning, agents repeatedly explore the environment and try to learn the optimal policy given an objective function. Examples of desired learned behaviors include collision avoidance, shortest path to destination and specific group formations. As an example, the work in [45] uses inverse reinforcement learning for agents to learn paths from recorded training data. Similarly, the approach in [46] applies Q-learning to plan paths for agents in crowds. In this approach, agents learn in a series of episodes the best path to their destination. Q-learning with Vector Quantization (VQQL) was used in [47] for simulating the navigation of multiple pedestrians in two constraining scenarios. A SARSA-based [48] learning algorithm has also

been used in [49] for offline learning of behaviors in crowd simulations. The approach in [50] analyzes different strategies for sharing policies between agents to speed up the learning process in crowd simulations. In the area of swarm intelligence, the work in [51] uses evolutionary algorithms for robotics, learning offline the parameters of the fitness function and sharing the learned rules in unknown environments.

Offline learning has important limitations which arise from the need to train the agents before the environment is known. In contrast, this dissertation presents an online learning approach. In online approaches, agents are given only partial knowledge of their environment, and are expected to adapt their strategies as they discover more of the environment. The approaches proposed in this dissertation allow agents to adapt online to unknown environments with no previous training process.

Learning the value of actions in an online manner, based on their previous performance, allows agents to make motion decisions. However, agents can also estimate the values of the actions by planning their execution for some time in the future, evaluating their potential long-term performance.

2.2.2 Long-term predicted motion value

Agents can estimate the value of their motions using predictions of their expected long term behavior. By simulating the evolution of its local environment and planning the execution of the actions for some time in the future, an agent can predict the effect that a motion taken in the present will have in a long-term. With this information, the agent can decide how to move.

Extensive literature has studied the problem of global planning, that is, to compute a predefined route for an agent to move from its start to its goal position. Global planning approaches have been proposed using PRMs [52], RRTs [53] and navigation meshes [54]. Approaches have also been introduced that dynamically update the weights of a roadmap [55] and periodically replan paths in an anytime, deterministic fashion [56, 57, 58]. However, due to the decentralized nature of these planners, an agent cannot typically account for the uncertainty induced by the presence of the other agents and static entities of the environment. In addition, it is not always obvious when an agent needs to replan its path.

In real-time multi-agent navigation, agents typically do not have global knowledge of

the environment beforehand, and this prevents the direct application of the aforementioned techniques. Instead, agents need to rely on their local perception and compute plans with a limited time horizon in an on-line fashion, interleaving planning with execution while navigating the environment. In this line, Receding Horizon Control (RHC) approaches [59] provide a useful framework. Agents using RHC periodically compute plans of fixed time length, using a time horizon that allow an agent to make decisions based on a foreseeable future. Given the domain restrictions, it is natural to tradeoff optimality of the generated plans with the computational efficiency of RHC, which allows plans to be generated and evaluated in real-time, while the agents are navigating.

Hindsight optimization (HOP) [5] is an online action selection technique used to compute an upper bound on the expected reward of an action by allowing the agent to 'peek' on potential future scenarios. Although the expected reward is not a tight upper bound on the real value of the action, it is often indicative enough for action comparison purposes [60]. HOP can be used as a planning technique [61, 62] by simulating a set of non-deterministic future scenarios and estimating "in hindsight" the best action at the present. Besides its application in the planning domain, HOP has also been used to plan moves in the solitaire game [63] and to allow a robotic hand to predict and assist a human in grasping objects [64]. With HOP, agents can reduce the uncertainty associated with the dynamic nature of the environment by considering potential scenarios and acting accordingly.

In this thesis, we adapt HOP to the multi-agent navigation domain, allowing agents to estimate the value of their actions based on potential future scenarios.

Although predicting the evolution of the environment can provide valuable information for action selection in multi-agent navigation tasks, agents could gain a better insight of the state of the environment by modeling each other as BDI agents [65]. A BDI model characterizes rational agents that have mental attitudes of beliefs, desires and intentions. In the multi-agent navigation domain, considering other agents as BDI entities (instead of dynamic obstacles) that have an intended motion and a goal location allows them to coordinate and compute motions that benefit the entire crowd.

2.2.3 Implicitly coordinated motion

Agents can estimate the value of their actions by considering not only their individual motion but also the motions of the nearby agents. Then, agents can select actions that also benefit the progress of their neighbors, resulting in coordinated joint motions.

The coordination of multiple agents sharing a common environment has been widely studied in the Artificial Intelligence community. When communication is not constrained, coordination can be achieved by casting it as a distributed constraint optimization problem (DCOP), where agents send messages back and forth until a solution is found [66, 67]. Guestrin et al. [68] assume that agents can directly communicate with each other by using coordination graphs, which also rely on a message-passing system. Graph-based methods for achieving consensus and cooperation have also been studied from a control-theoretic perspective [69]. A main issue in these graph-based approaches is how to maintain connectivity between the agents [70, 71, 72], which is required for performance guarantees.

Coordination can also be achieved using learning methods, where agents learn the value of the joint actions performed by all interacting agents [73]. These methods do not scale as the number of agents increases (the action space increases exponentially with each added agent). To address this, some methods such as the one proposed by Melo and Veloso in [74] focus the learning of joint actions only when coordination is required, and use Q-learning when it is not. This is useful in cases where the interactions between agents are sparse. Our problem domain, however, features dense agent interactions.

Rule-based approaches have also been proposed to bias the agents' decision making towards actions that promote coordination [75] when it is needed. All these approaches make some assumptions regarding the ability of the agents to communicate and share information. Even when considering only local agent neighborhoods, the complexity of these approaches increases exponentially with the number of neighbors. The authors in [76] argue that requiring agents to share information limits the ability of an approach to generalize the learning of coordinated policies. Instead, they propose an implicit coordination approach by having agents use Q-learning in a box-moving task, where agents need to perform complimentary motions to move a box to a desired position. In their work, agents use the effect of their actions to implicitly communicate and coordinate.

Coordination in Multi-Agent Navigation.

Work has been done to allow agents to coordinate their motions when moving to their goals. When simulating the motion of pedestrians, coordination can be achieved through the emergence of social norms, as in [77], which describe a series of learned behavioral patterns that humans follow in social settings. Specifically, in [78] agents are given a navigation task with two groups having opposite goals, The agents can choose between two actions: pass on right or pass on left. They compute the value of the joint actions and use ensemble learning methods to converge to one of the two actions for each group. The work in [79] use cognitive models of social comparison to determine agents with similar behavior. Similarly, in this dissertation agents compare themselves to their neighbors in terms of motion features, but unlike [79], no socio-psychological theory is used.

Coordination has also been studied to pre-compute collision-free goal paths for a set of agents. Solving this problem in a centralized setting, for multiple agents, is intractable (PSPACE-hard [80]). This has given rise to the development of decentralized approaches, known as multi-agent path finding methods, which decouple the problem by allowing each agent to compute its path independently. However, to ensure that the computed paths are collision-free, agents are required to share their trajectories.

With this assumption, Silver proposed a number of methods that allow agents to plan their trajectories while respecting the constraints induced by the other agents [81]. This way, agents can plan ahead to deviate from their goal-directed motion to prevent deadlocks, implicitly coordinating their motions. Silver's work introduces a set of A* [82] variants which introduce the notion of potential collisions in the computation of costs in each path.

Following a similar principle, Jansen and Sturtevant proposed a method to induce cooperation between agents by using the idea of Direction Maps (DM) [83]. In their approach, DM are an abstraction of a 2D grid where in each individual cell, a Direction Vector (DV) is computed based on the observed velocities of agents that have traveled through that cell. Agents use this information to compute weights in their planned paths, which promote the selection of routes with similar Direction Vectors. Although Direction Maps allow agents to consider crowd information when computing paths, in very chaotic environments this information becomes highly non-stationary, and it

does not consider the crowd density, which might produce overly pessimistic paths for the agents. Another technique, called *congestion maps* [84], allows agents to consider both aggregate velocity and density of the crowd, incorporating this information when computing the path traversal cost for each agent. By doing this, agents are able to take paths that might be longer but that are faster to travel, avoiding congestion at critical points of the environment. An issue with this technique is that it requires fine tuning of the discretization used, which might be different depending on the environment at hand. This technique does not consider the evolution of the system over time, i.e., areas that might drastically change their density in a short time. This produces inconsistent behaviors that might be avoided by taking the temporal dimension into account.

Another approach for achieving coordinated navigation is to explicitly form groups of agents, have the groups follow specific directions while keeping a level of cohesiveness between group members. The work of Reynolds in boids [10], mentioned above, addresses the problem of simulating large cohesive groups such as birds or school of fishes. In terms of pedestrian simulation, most of the proposed approaches focus on steering small groups of a fixed number of agents to their destinations, while avoiding static obstacles [29, 85, 86]. Recently a new approach based on proxemic was proposed, that addresses emergent group formation of different sizes, but cannot simulate merging or splitting behaviors for the groups [87].

All these approaches assume some form of homogeneity in the agents, as they assume all agents execute the same coordination algorithm. Although this assumption is easy to guarantee in controlled environments, it does not necessarily hold in unknown environments where agents might differ, for example, in the navigation method used.

2.2.4 Inferring the types of agents

Robotic agents navigating in real world environments cannot assume that other moving entities will behave in a similar way. When there is uncertainty on the dynamics of the environment, an agent should be extra cautious to reduce the risk of collisions when moving to its goal. However, this behavior could also translate into unnecessary long paths, resulting in inefficient motion. This limits the agent's ability to compute safe and efficient paths. Work has been done to predict the motion of unknown moving entities in real-world environments, to maximize the safety of the robot while moving to its goal.

Knepper and Ros [88] developed a sampling-based collision avoidance method inspired by human behavior. Specifically, the authors use the concept of *civil inattention* to decide between fully avoiding collision or performing only part of the effort. They assume robots can share their plans and also account for human motion, projecting their motion with a constant velocity assumption. They performed experiments with real robots and in ROS simulations.

The approach of Choi et al. [89] uses a reactive Gaussian-based motion controller to address the issue of safe robot navigation in human populated environments. Trautman et al. [90] study the problem of robot navigation in dense pedestrian crowds, proposing a method to predict pedestrian trajectories and achieve cooperative motion. The complexity of these approaches limits their ability to predict trajectories of more than 10 dynamic entities, in order to have real-time operation. We aim at computing both safe and time-efficient trajectories in simulated environments with up to 100 agents in real time.

Finally, prior work has also addressed interactions with heterogeneous agents based on Bayesian learning (see, e.g. [91, 92]). In particular, Barrett et al. [91] used a Bayesian framework to learn the types of other agents in the Robocup domain. In the RoboCup domain, agents can only move according to a given set of rules, and only interact with agents of known types. In contrast, in our multi-agent navigation domain, agents can assume both known and unknown types.

Chapter 3

Problem definition and background

In this chapter, we formally define the problem studied in this dissertation and the assumptions considered. Finally, we describe the state-of-the-art navigation framework underlying the proposed contributions.

3.1 Problem formulation

The work presented in this dissertation focuses on improving the efficiency in the navigation of the agents. More formally, given an environment and a number of agents with specified start and goal positions, the task is for the agents to move as fast as possible to their goals without collisions.

3.1.1 Notation

Given n agents, let agent A_i have a radius r_i, a goal position $\mathbf{g}_i$, and a maximum speed $v_i^{\max}$. Let $\mathbf{p}_i^t$ and $\mathbf{v}_i^t$ denote the agent's position and velocity, respectively, at a given time t. Furthermore, agent A_i has a preferred velocity $\mathbf{v}_i^{\text{pref}}$ at which it prefers to move. Let $\mathbf{v}_i^{\text{goal}}$ be the preferred velocity directed towards the agent's goal $\mathbf{g}_i$ with magnitude equal to $v_i^{\max}$.

3.1.2 Objective function

The main objective of our work is to minimize the arrival time of the last agent or, equivalently, the maximum travel time of the agents, while guaranteeing collision-free motions. Formally:

$$\begin{aligned} \text{minimize} \quad & \max_{i}(TimetoGoal(A_i)) \\ \text{s.t.} \quad & \|\mathbf{p}_i^t - \mathbf{p}_j^t\| > r_i + r_j, \underset{i \neq j}{\forall} i, j \in [1, n] \\ & dist(\mathbf{p}_i^t, O_j) > r_i, \qquad \forall i \in [1, n], j = [1, k] \\ & \|\mathbf{v}_i^t\| \leq v_i^{\max}, \qquad \forall i \in [1, n] \end{aligned} \tag{3.1}$$

where $TimetoGoal(A_i)$ is the travel time of agent A_i from its start position to its goal and $dist(\cdot)$ denotes the shortest distance.

3.1.3 Complexity

Finding a solution to Eq. 3.1 is equivalent to solving the problem of coordinated motion planning for a set of objects, sometimes called the "warehouseman's problem," which has been proved to be $PSPACE$-hard in its general form [80]. Hence, this problem is intractable (assuming $P \neq PSPACE$).

Specifically, the time complexity of a centralized solver for minimizing Eq. 3.1 is exponential in the number of agents and their degrees of freedom [93]. Therefore, Eq. 3.1 cannot be minimized directly and has to be solved in a decentralized manner, where each agent computes its path independently. Although this significantly reduces the complexity of the problem, it introduces new difficulties arising from the lack of knowledge of the other agents and their computed paths. Under these constraints, agents would not be able to effectively navigate an environment unless some assumptions are made. In the next subsection, we establish the assumptions on which are based all contributions presented in this dissertation.

3.1.4 Assumptions

The assumptions considered in this dissertation are not restrictive to a specific application, allowing the proposed contributions to be implemented in a variety of application domains such as multi-robot systems, video games and crowd simulations.

We assume that agents navigate *independently* on a 2D plane that may also contain a set of k static obstacles O. We further assume that each agent, modeled as a disc, can sense the positions of the obstacles, and the position and velocities of other agents inside its sensing range. For a given agent A_i, its *agent neighborhood* is defined as the set of agents that, at any given time, are inside A_i's sensing range. We note A_i's agent neighborhood as NN_i, where $NN_i \subset A$, the set of all agents. For the majority of this dissertation, we do not allow any communication between the agents. The only exception to this is our implicit coordination approach (Chapter 7), where we assume agents have limited one-way communication capabilities. Finally, we assume that each agent A_i knows its goal location $\mathbf{g}_i$, but it is not aware of the goal locations of its neighbors.

As agents do not know in advance the paths planned by the other agents, they need to decide how to move, in *real time*, based on the observed motions of their nearby agents.

3.1.5 Research Question

This dissertation seeks to answer the following overarching research question:

How should each agent move at each time step, given its local perception of the environment, to minimize the total travel time of all agents? (equivalently, to minimize Eq. 3.1 while meeting its constraints)

The answer to this question requires that agents find, at each time instant, a velocity that respects the agent's geometric and kinematic constraints, avoiding collisions with other agents and static obstacles in the environment, while contributing to the goal progress of the entire system of agents in an efficient manner.

The following sections describe different techniques that agents can employ to meet the collision avoidance constraints of Eq. 3.1, before focusing on the collision avoidance technique used throughout this thesis.

3.2 Collision Avoidance

Several methods have been proposed to prevent collisions while agents navigate towards their goals. Broadly, these methods can be classified as *reactive* and *anticipatory*.

In reactive collision avoidance, agents adapt their predefined motion to other agents and obstacles along their paths. Many reactive methods, such as Reynolds' flocking model [10] use repulsive forces to avoid collisions. However, all these techniques do not anticipate. Only when agents are sufficiently close, they react to avoid collisions. This can lead to oscillations between agents and to local minima. Another limitation of these methods is that the forces must be tuned separately for each scenario, limiting their robustness.

In anticipatory collision avoidance, agents detect and avoid potential upcoming collisions by linearly extrapolating their current velocities. An important concept in many anticipatory approaches is the notion of $Velocity\ Obstacle$ (VO) used in robotics [23]. For an agent A_i, the VO represents the set of all the agent's velocities that would eventually result in collision with another agent or obstacle. Van den Berg et al. [27] extended the notion of VO and introduced the concept of *Reciprocal* $Velocity\ Obstacle$ (RVO). RVO depends on random sampling of velocities and provides oscillation-free motion between two moving agents. More recently, the principle of Optimal Reciprocal Collision Avoidance (ORCA) for multi-agent navigation was proposed [1]. ORCA overcomes the limitations of RVO and guarantees collision and oscillation-free motion between multiple moving agents. ORCA accounts for anticipatory avoidance, reciprocity among the agents as well as sensor noise and motion uncertainty [94]. An extension to ORCA that accounts for localization uncertainty applied to differential drive robots is presented in [95].

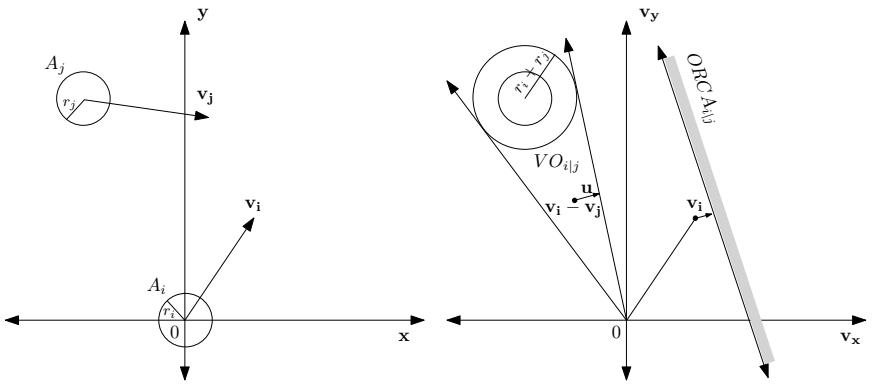
3.3 ORCA

ORCA builds on the concept of Velocity Obstacles. Given two agents, A_i and A_j, the set of velocity obstacles $VO_{A_i|A_j}$ represents the set of all relative velocities between A_i and A_j that will result in a collision at some moment in time. Using the VO formulation, we can guarantee collision avoidance by choosing a relative velocity that lies outside the set $VO_{A_i|A_j}$. Let $\mathbf{u}$ denote the minimum change in the relative velocity of A_i and A_j needed to avert the collision. ORCA assumes that the two agents will *share* the responsibility of avoiding it and requires each agent to change their current velocity by at least $\frac{1}{2}\mathbf{u}$. Then, the set of feasible velocities for A_i induced by A_j is the half-plane of

velocities given by:

$$ORCA_{A_i|A_j} = \{\mathbf{v} \mid (\mathbf{v} - (\mathbf{v}_i + \frac{1}{2}\mathbf{u})) \cdot \hat{\mathbf{u}}\}, \tag{3.2}$$

where $\hat{\mathbf{u}}$ is the normalized vector $\mathbf{u}$ (see Figure 3.1). Similar formulation can be derived for determining A_i's permitted velocities with respect to a static obstacle O_k. This set is denoted as $ORCA_{A_i|O_k}$.



(a) Agents A_i and A_j moving at velocities $\mathbf{v_i}$ and $\mathbf{v_j}$, respectively.

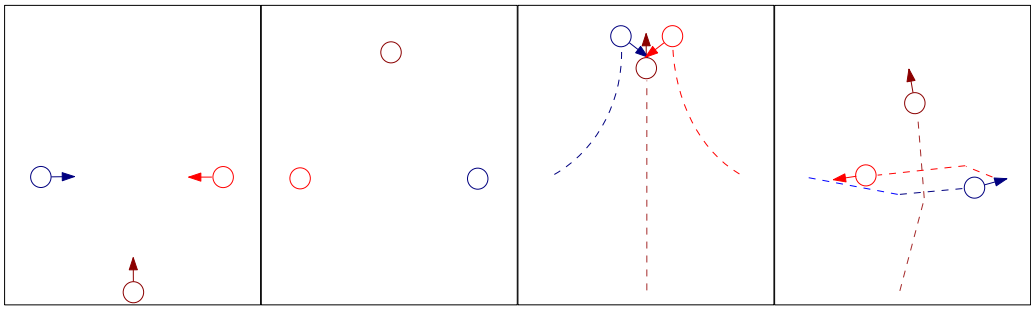
(b) A_i's allowed velocities, in the velocity space

Figure 3.1: (a) Two agents, A_i and A_j, moving towards a potential collision. (b) The set of allowed velocities for agent A_i induced by agent A_j is indicated by the half-plane delimited by the line perpendicular to $\hat{\mathbf{u}}$ through the point $\mathbf{v_i} + \frac{1}{2}\mathbf{u}$, where $\mathbf{u}$ is the vector from $\mathbf{v_i} - \mathbf{v_j}$ to the closest point on the boundary of $VO_{i|j}$

The overall approach works as follows. At each time step of the simulation, every agent A_i takes into account its neighboring agents and static obstacles to compute a new collision-free velocity. First, A_i infers its set of *feasible* velocities, FV_{A_i}, from the intersection of all permitted half-planes $ORCA_{A_i|A_j}$ and $ORCA_{A_i|O_k}$ induced by each neighboring agent A_j and obstacle O_k, respectively. Having computed FV_{A_i}, the agent selects a new velocity $\mathbf{v}_i^{\text{new}}$ for itself that is closest to its preferred velocity $\mathbf{v}_i^{\text{pref}}$ and lies inside the region of feasible velocities:

$$\mathbf{v}_i^{\text{new}} = \underset{\mathbf{v} \in FV_{A_i}}{\arg\min} \|\mathbf{v} - \mathbf{v}_i^{\text{pref}}\|. \tag{3.3}$$

The optimization problem in Eq. (3.3) can be efficiently solved using linear programming, since FV_{A_i} is a convex region bounded by linear constraints. Finally, agent



(a) Start Positions (b) Goal Positions (c) ORCA (d) PHOP

Figure 3.2: Three agents cross paths. (a) Initial positions of the agents. (b) Goal positions of the agents. (c) When navigating with ORCA, the agents run into and push each other resulting in inefficient paths. (d) When using the proposed PHOP approach the agents plan around the potential results of the upcoming collision, resulting in more efficient paths.

A_i updates its position based on the newly computed velocity.

As ORCA is a decentralized approach, each agent computes its velocity independently. The notation we use is the same for all the agents. Therefore, to simplify the notation in the remaining of this document, we do not specify the agent being referred.

The notation we use is the same for all the agents. Unless otherwise specified and to simplify the notation, in the rest of the thesis we omit the index of the specific agent being referred.

3.4 Limitations of ORCA

Although ORCA guarantees collision-free motions and provides a locally optimal behavior for each agent, the lack of coordination between agents can lead to globally inefficient motions. See, for example, Fig. 3.2. Here, three agents start from the initial positions as illustrated in Fig. 3.2a and must reach the final positions shown in Fig. 3.2b. Because the agents do not coordinate their motions and follow only their goal-oriented preferred velocity, they will get stuck in local minima which generates the trajectories shown in Fig. 3.2c. Therefore, if agents were able to behave differently in those situations, for example, by selecting a $\mathbf{v}^{\text{pref}}$ in a different direction for a short period of time, they

might be able to find a larger region of feasible velocities. This might indirectly help avoid/solve the overall congestion, benefiting all agents. The algorithms proposed in this dissertation directly address this limitation, allowing agents to intelligently adapt their preferred velocity in an online fashion, improving their efficiency in challenging and partially known environments. In Chapter 4 we present a general framework that agents can use to select from a set of preferred velocities in an online fashion and, in Chapters 5, 6, 7 and 8, we present a set of methods that agents can use to evaluate and select a preferred velocity at each time. An example trajectory generated by one of our approaches can be seen in Fig. 3.2d.

Chapter 4

Action-Selection Framework for Multi-Agent Navigation

This chapter presents a general action-selection framework to improve the time-efficiency of the agents, within the context of the multi-agent navigation problem, to address the problem defined in Chapter 3. Using this framework, along with the techniques presented in Chapter 5 to Chapter-8, an agent can choose how to move at each timestep given its perception of the local environment.

In this framework, each agent runs a continuous cycle of sensing and acting until it reaches its goal. At each cycle, the radii, positions and the velocities of nearby agents and obstacles are obtained by sensing. The goal is to compute for each agent a new velocity that can be adopted until the next cycle. In this dissertation, unless otherwise specified, we consider that each cycle (simulation timestep) lasts for 25 ms. This represents a hard constraint on the time that each agent has to compute its new velocity, which guarantees real-time behavior. At each cycle, we seek to find a velocity which respects the agent's geometric and kinematics constraints while ensuring its progress towards its goal in a timely-efficient manner. To achieve this, we follow a two-step process. At each simulation cycle, for each agent, we first compute a preferred velocity $\mathbf{v}^{\text{pref}}$ indicating the agent's desired direction of motion and speed. As such velocity may be colliding, we then project $\mathbf{v}^{\text{pref}}$ to a collision-free velocity $\mathbf{v}^{\text{new}}$ before it can be executed. For the mapping of $\mathbf{v}^{\text{pref}}$ to $\mathbf{v}^{\text{new}}$ we use the ORCA navigation framework.

In Chapters 5-8 we explore action selection techniques to determine a new $\mathbf{v}^{\text{pref}}$ among a set of given preferred velocities, allowing us to find in real-time near-optimal solutions to the optimization problem in Eq. 3.1.

4.1 Motion Diversity

Typically, at each simulation cycle, agents aim at selecting preferred velocities that drive them closer to their goals. Additional velocities, however, can increase the set of potential behaviors that the agents can exhibit, and may also help them in the longer run to reach their destinations faster (consider, for example, an agent moving backwards to alleviate congestion). Therefore, we allow the agents to use different motion primitives, also referred to as *actions*, which correspond to different preferred velocities (throughout this dissertation, we use the terms preferred velocities and actions interchangeably). This requires each agent to make a choice at every simulation step in a continuous 2D space, the space of all possible speeds and directions. In real-time domains, though, the agents have limited computational resources at their disposal to make decisions. To address this issue, agents plan their motions over a discretized space of a small number of preferred velocities as shown in Fig. 4.1. In all the contributions presented in this dissertation, this set of velocities provided the proper balance between having enough actions to demonstrate a variety of behaviors and keeping the size of the action space small enough to enable real-time computations.

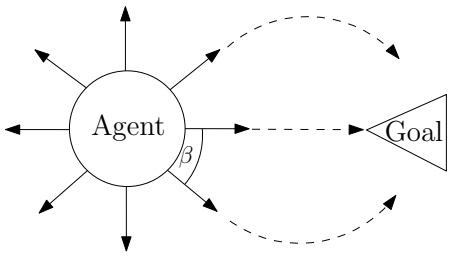


Figure 4.1: Actions. The eight available actions correspond to moving at 1.5 m/s with different angles with respect to the goal: $0°$, β, $-\beta$, $90°$, $-90°$, $180°$, $180° + \beta$ and $180° - \beta$. In our implementation, $\beta = 45°$.

4.2 Reward Function

We hypothesize that each agent can improve the time-efficiency of all agents by selecting, at each time, an action that considers both the agent's own progress as well as the progress of its neighbors. To this end, we propose a reward function that evaluates the quality of an agent's selected action, based on how goal-oriented the agent's behavior is and the effect that such an action has on the motion of the agent's neighbors. Specifically, the reward $\mathcal{R}_a$ for an agent performing action a is a convex combination of a *goal-oriented* component and a *politeness* component:

$$\mathcal{R}_a = (1 - \gamma) \cdot \mathcal{R}_a^{goal} + \gamma \cdot \mathcal{R}_a^{polite}, \tag{4.1}$$

where the parameter γ controls the influence of each component in the total reward ($0 \leq \gamma \leq 1$).

The *goal-oriented* component $\mathcal{R}_a^{goal}$ measures how much a given action a moves an agent towards its goal, while the *politeness* component $\mathcal{R}_a^{polite}$ represents the social aspect of action a, by measuring how it affects the motion of an agent's neighbors. Each component has an upper bound of 1 and a lower bound of -1. There is no strict relation between the values of these two components, i.e., an action with a value of 1 for $\mathcal{R}_a^{goal}$ can also have a value of 1 for $\mathcal{R}_a^{polite}$. The details of how each component is computed are specified in each respective chapter.

Equation 4.1 provides the agents with real-time feedback on the value of their actions. To evaluate how each approach minimizes Eq. 3.1, however, we need to quantify the total travel time of the agents, after a simulation finishes.

4.3 Performance Metric

We can evaluate the performance of the different approaches proposed in this dissertation by measuring, in hindsight, the time that agents spend in interactions with other agents before reaching their goals. We call this metric *interaction overhead*.

Definition 1 (Interaction Overhead).

The interaction overhead is the difference between the maximum travel time of the agents and the theoretical maximum travel time if agents were able to follow their

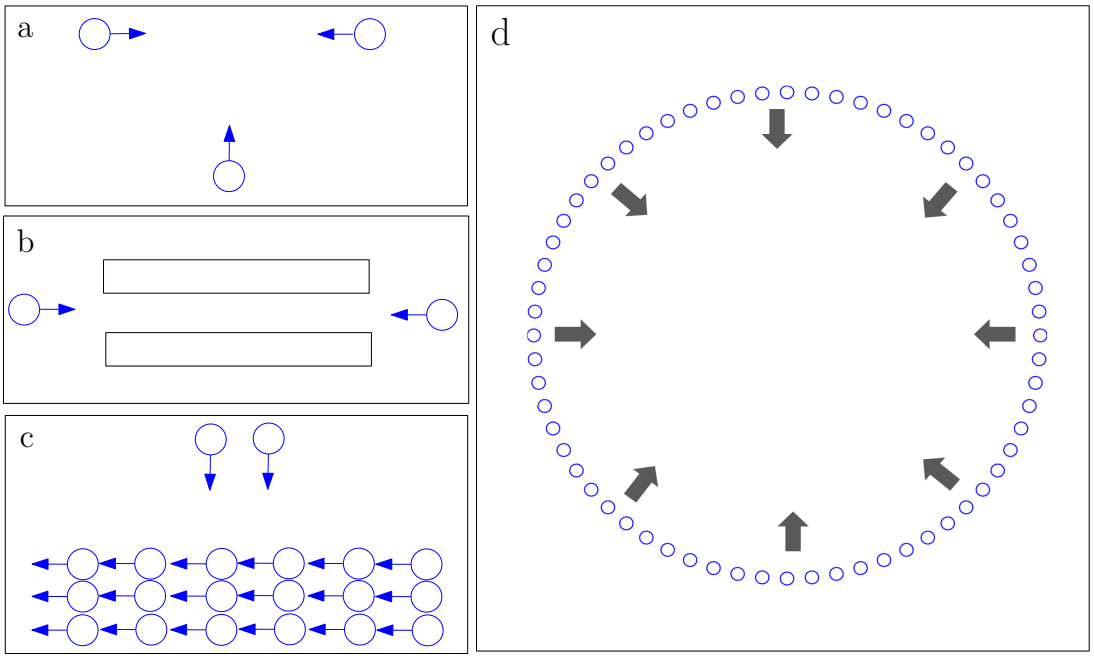


Figure 4.2: Example of simulated scenarios: (a) *Intersection*, (b) *Deadlock*, (c) *Perpendicular Crossing* and (d) *Circle*.

shortest paths to their goals at maximum speed without interacting with each other, i.e.:

$$Interaction\ Overhead = \max_i(TimetoGoal(A_i)) - \max_i\left(\frac{shortestPath(A_i)}{v_i^{\max}}\right),$$

where $shortestPath(A_i)$ is the length of the shortest path that agent A_i would need to traverse to reach its goal, assuming no other agent in the environment.

A theoretical property of this metric is that an interaction overhead of zero represents a lower bound on the optimal travel time for the agents, and it is the best result that an optimal centralized approach could potentially achieve.

4.4 Scenarios

We evaluate the interaction overhead of our proposed approaches in a variety of scenarios, with different number of agents and, in some cases, with static obstacles. In terms

of structure, we consider open areas as well as indoor environments with up to 3 rooms, where in many cases the agents have to select between multiple homotopic paths to reach their goals.

Figure 4.2 shows a representative sample of the scenarios considered in this dissertation. These include environments with many agents and conflicting goals (*Circle*, Fig 4.2d), where agents can find more efficient motions by moving sideways in an open area, as well as scenarios where agents are constrained by static obstacles such as walls (*Deadlock*, Fig. 4.2b). In this case, agents need to defer to their neighbors which might involve backtracking in their goal-oriented motions, to improve the time-efficiency of all agents. We also consider scenarios where agents are initially distributed in groups (*Perpendicular Crossing*, Fig. 4.2c), to analyze how this structure is maintained (or modified) in each of the approaches. Finally, we consider scenarios with few agents (*Intersection*, Fig. 4.2a) to exemplify more clearly the interactions between the agents in the different contributions.

4.5 Experimental Setup

In all the contributions of this dissertation, we performed experiments using programs written in C++. Results were gathered on an Intel Core i7 at 3.5 GHz. Unless noted otherwise, each experimental result is the average over 30 simulations. In all our runs, we updated the positions of the agents every $\Delta t = 25\,\text{ms}$ and set the maximum speed $v^{\max}$ of each agent to $1.5\,\text{m/s}$ and its radius r to $0.5\,\text{m}$. Agents could sense other agents within a $15\,\text{m}$ radius, and obstacles within $1\,\text{m}$. To avoid synchronization artifacts, agents are given a small random delay in how frequently they can update their $\mathbf{v}^{\text{pref}}$ (with new $\mathbf{v}^{\text{pref}}$ decisions computed every $0.1\,\text{s}$ on average). This delay also gives ORCA a few timesteps to incorporate sudden velocity changes before the actions are evaluated. Small random perturbations were added to the preferred velocities of the agents to prevent symmetry problems. We refer the reader to Chapters 5.5, 6.5, 7.4.1 and 8.4 for the corresponding results.

Chapter 5

Learning from the past (ALAN)

In a partially observable and highly dynamic environment, the action for each agent that maximizes the time-efficiency for all agents can change rapidly, based on their individual local contexts. Therefore, it is critical that the agents can quickly evaluate their navigation contexts and use this evaluation to guide the action selection.

In this chapter we present the ALAN approach, which allows agents to learn the value of their actions to dynamically adapt their navigation based on their local conditions. Agents using ALAN formulate the navigation problem as a multi-armed bandit problem, where they must balance between selecting the action with the highest known value and selecting other actions whose value might potentially be better.

Algorithm 1 shows an overview of ALAN in the context of the traditional sensing-acting cycle of a navigation task. In each cycle, each agent acquires information about the positions and velocities of nearby agents and obstacles, and then decides how to move by selecting an appropriate action. ALAN's action selection method outputs a $\mathbf{v}^{\text{pref}}$ (line 5) which is given as input to the ORCA framework. After determining potential collisions, ORCA outputs a new collision-free velocity $\mathbf{v}^{\text{new}}$ (line 7) which is used to update the position of the agent for the next simulation timestep (line 8). The agent also determines the quality of its previously taken action by computing its reward value (lines 9-11). This value becomes available to the action selection mechanism, which selects a new $\mathbf{v}^{\text{pref}}$ and the cycle repeats until the agent reaches its goal.

Algorithm 1: The ALAN learning approach for an agent

initialize simulation
initialize $\mathcal{R}_a$ for all the actions to their maximum values
while not at the goal **do**
  if $UpdateAction(t)$ **then**
    $\mathbf{v}^{\text{pref}} \leftarrow ActionSelection$
  end if
  $\mathbf{v}^{\text{new}} \leftarrow ORCA(\mathbf{v}^{\text{pref}})$
  $\mathbf{p}^t \leftarrow \mathbf{p}^{t+1} + \mathbf{v}^{\text{new}} \cdot \Delta t$
  $\mathcal{R}_a^{goal} \leftarrow GoalReward(a^{t-1})$ (cf. Eq. 5.2)
  $\mathcal{R}_a^{polite} \leftarrow PoliteReward(a^{t-1})$ (cf. Eq. 5.3)
  $\mathcal{R}_a \leftarrow (1-\gamma) \cdot \mathcal{R}_a^{goal} + \gamma \cdot \mathcal{R}_a^{polite}$
end while

5.1 Action Evaluation

Based on the reward function model proposed in Chapter 4.2, each agent evaluates the quality of each action, after it has been executed, based on how goal-oriented ($\mathcal{R}_a^{goal}$) it is and on how it affects the motion of its nearby agents ($\mathcal{R}_a^{polite}$). Although the general expression of the reward function was outlined in Chapter 4.2, we include it here as well for completeness and convenience.

The reward $\mathcal{R}_a$ for an agent performing action a is a convex combination of a *goal-oriented* component and a *politeness* component:

$$\mathcal{R}_a = (1-\gamma) \cdot \mathcal{R}_a^{goal} + \gamma \cdot \mathcal{R}_a^{polite}, \tag{5.1}$$

where the parameter γ controls the influence of each component in the total reward ($0 \leq \gamma \leq 1$).

The *goal-oriented* component $\mathcal{R}_a^{goal}$ computes the scalar product of the collision-free velocity $\mathbf{v}_i^{\text{new}}$ of the agent with the normalized vector pointing from the position $\mathbf{p}_i$ of the agent to its goal $\mathbf{g}_i$. This component promotes preferred velocities that lead an agent as quickly as possible to its goal. More formally:

$$\mathcal{R}_a^{goal} = \mathbf{v}^{\text{new}} \cdot \frac{\mathbf{g} - \mathbf{p}}{\|\mathbf{g} - \mathbf{p}\|} \tag{5.2}$$

The *politeness* component $\mathcal{R}_a^{polite}$ compares the previously executed preferred velocity with the resulting collision-free velocity. If the chosen preferred velocity leads to potential collisions, a different velocity is returned by ORCA for both the agent and its potentially colliding neighbor (due to ORCA's reciprocity property) . This means that the chosen velocity negatively affects one of the agent's neighbors. If, instead, the preferred velocity does not conflict with other agents' motions, a similar collision-free velocity is computed for the agent. This means that the action executed did not produce significant changes in a neighbors' velocity (again, due to ORCA's reciprocity). Hence, the similarity between $\mathbf{v}^{\text{new}}$ and $\mathbf{v}^{\text{pref}}$ indicates how polite is the corresponding action, with respect to the motion of the other agents. Polite actions reduce the constraints on other agents' motions, allowing them to move and therefore advancing the global simulation state. Formally:

$$\mathcal{R}_a^{polite} = \mathbf{v}^{\text{new}} \cdot \mathbf{v}^{\text{pref}} \tag{5.3}$$

If an agent only aims at maximizing its goal progress ($\mathcal{R}_a^{goal}$), its behavior would be myopic and it would not consider the effects of its actions on the other agents. On the other hand, if the agent only maximizes its politeness behavior ($\mathcal{R}_a^{polite}$), it has no incentive to move towards its goal, which means it might never reach it. Therefore, an agent should aim at maximizing a combination of both components. Different behaviors may be obtained with different values of γ.

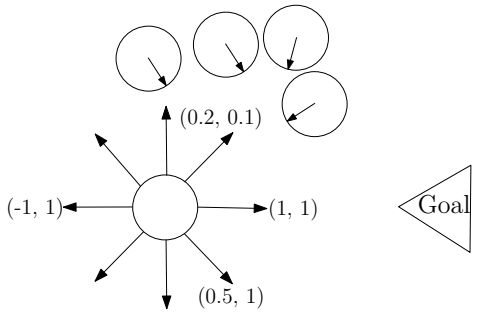


Figure 5.1: Example of reward values for different actions under clear and congested local conditions. The reward $\mathcal{R}_a$ of each action a is described by a goal-oriented and a politeness component ($\mathcal{R}_a^{goal}$, $\mathcal{R}_a^{polite}$).

Figure 8.2 shows an example of different local conditions an agent may encounter.

The reward values shown correspond to ($\mathcal{R}^{goal}$, $\mathcal{R}^{polite}$) tuples of the action set. Here, congestion has formed on one side of the agent, which results in low reward values for the left angled motion. All of the other actions are not constrained, and consequently their reward value is higher. In this case, the agent will choose the straight goal-oriented action, as it maximizes $\mathcal{R}_a$.

5.2 Action Selection

In ALAN, agents learn the reward values of their actions in an online fashion, as they move towards their goals, and select at each simulation cycle one of these actions based only on their past experience. As agents can encounter so many different states, we adapt a stateless representation for our approach. Online learning problems with a discretized set of actions and stateless representation can be well formulated as multi-armed bandit problems.

In a multi-armed bandit problem, an agent makes sequential decisions on a set of actions to maximize its expected reward. This formulation is well-suited for stationary problems, as existing algorithms guarantee a logarithmic bound on the regret. Although our problem is non-stationary in a global sense, as the joint local conditions of all agents are highly dynamic, individual agents can often undergo long periods of stationary reward distributions. Therefore, by learning the action that maximizes Eq. 5.1 in each of these stationary periods, we allow agents to adapt to different local conditions. To learn this optimal action in each of these periods of stationary rewards, the agents must balance the exploitation of the current best action with the exploration of potentially better ones.

The exploration vs exploitation dilemma is a common problem in sequential decision making under uncertainty, and is particularly important in online learning; with excessive exploration the agent might take suboptimal actions too many times, while too little exploration might prevent quick adaptation to local changes. ALAN achieves this balance by using ideas from two existing action selection techniques, ϵ-greedy and upper confidence bounds (UCB), and a strategy from game theory that dynamically adapts the amount of exploration based on the agent's local conditions. Below, we elaborate on the different components of ALAN.

5.2.1 ϵ-greedy

The ϵ-greedy approach works by selecting the highest valued action with probability 1-ϵ, and a random action with probability ϵ, $0 \leq \epsilon \leq 1$. The value of ϵ indicates the degree of exploration that the agent performs [48]. Although this value can be fixed, it typically decreases over time since the need to explore decreases as the reward estimates become more accurate. This ensures a logarithmic bound on the regret [96].

A key property of ϵ-greedy is that there is no distinction between actions other than the highest-valued action. This can be beneficial if the optimal action is unique and does not change frequently. However, such an assumption does not hold for problems where the distribution of rewards changes frequently, as in multi-agent navigation. In this case, random exploration alone cannot take advantage of the underlying reward structure.

Theorem 1. *A memory-less ϵ-greedy is expected to find and exploit a new optimal action a^* if, on average, the reward distribution changes, at least, every $\frac{|Actions|}{\epsilon}$ timesteps, where Actions denote the set of discretized $\mathbf{v}^{\text{pref}}$.*

Proof: Assume that the optimal action $a^* \neq max(\mathcal{R}_a)$, $\forall a \in Actions$. ϵ-greedy is expected to choose a random action, on average, every ϵ^{-1} timesteps. Since an agent has to choose between $|Actions|$ actions, each one is expected to be selected every $\frac{|Actions|}{\epsilon}$ timesteps. Since a memory-less ϵ-greedy replaces the previous reward estimate of an action with each new sample, once a^* is sampled it will be exploited, on average, every $(1-\epsilon)^{-1}$ timesteps.

■

Therefore, as long as the reward distribution remains stationary for a period longer than $\frac{|Actions|}{\epsilon}$ timesteps, ϵ-greedy is expected to learn the new reward distribution and optimize accordingly. However, the performance of ϵ-greedy depends heavily on the value of ϵ used and must be manually tuned for different navigation tasks. Furthermore, as mentioned above, the random exploration of ϵ-greedy can only take advantage of the reward distribution of the optimal action, which can lead to inefficient motions as discussed in Section 5.5.

5.2.2 wUCB

Another widely used action-selection technique is the upper confidence bounds (UCB) algorithm. UCB samples the actions proportionally to the upper-bound of the estimated value of their rewards [2]. However, UCB assumes that the distributions of the rewards do not change over time, an assumption that does not apply to our domain. To address this issue and allow UCB to adapt to changing local conditions, similar to [97], we consider a moving time window (of T timesteps) instead of the entire history of rewards. We call our modified implementation wUCB.

Formally, given an agent, the wUCB estimate of an action a is a sum of two terms:

$$w\text{UCB}(a) = \overline{\mathcal{R}_a} + c\sqrt{\frac{\ln(\nu)}{\nu_a}}. \tag{5.4}$$

The first term, $\overline{\mathcal{R}_a}$, denotes the average reward of action a over the specified time window. The second term corresponds to the size of the one-sided confidence interval for the average reward. This confidence interval bounds the true expected reward with very high probability. It is computed using the number of times action a was selected (ν_a) and the total number of action decisions made so far by the agent (ν), weighted by an exploration parameter c (typically set to $\sqrt{2}$).

At each timestep, wUCB selects the action a with the maximum value of wUCB(a) across all actions. Using the time window, wUCB(a) is able to adapt to changes in the distribution of the rewards and exploit the optimal action faster.

Theorem 2. *wUCB is guaranteed to identify and exploit the optimal action a^* of any reward distribution in at most T timesteps, where T is the size of the time window and $T > |Actions|$.*

Proof: Assume a worst case scenario where the agent sampled action a at time t and obtained the lowest reward value of all actions denoted as $\mathcal{R}_{min}$. Assume further that at time $t+1$ the distribution of rewards changes so that action a becomes a^*, that is, the optimal action. a^* will be sampled again when $wUCB(a^*) = max(wUCB(a)), \forall a \in Actions$. As the value of $\overline{\mathcal{R}_a}$ (cf. Eq. 5.4) does not change while action a is not selected, this will only occur when at two different times, t and t', the following equation is satisfied:

$$c\sqrt{\frac{\ln(\nu)}{\nu_{a^*}(t^{'})}} - c\sqrt{\frac{\ln(\nu)}{\nu_{a^*}(t)}} \geq max(wUCB(a)) - wUCB(a^*) \tag{5.5}$$

As the rewards of the actions are bounded by Eq. 5.1, the right side of the inequality in Eq. 5.5 will always have a finite value. However, ν is bounded by T, the size of the time window. If $\mathrm{T} < (t^{'} - t)$, then $wUCB(a^*)$ will become ∞ when $\nu_{a^*} = 0$, and hence it will be selected, as long as $T > |Actions|$. In this case, $wUCB(a^*) = \mathcal{R}_{a^*}$, and hence a^* will be regarded as the optimal action and exploited accordingly. ■

5.3 ALAN

Although the exploration performed by ϵ-greedy and wUCB would help agents to keep learning the new values of the actions when the environment changes, the constant exploration mechanism (fixed ϵ or fixed T) represents the assumption that all stationary periods have at least a fixed length (T for wUCB or $\frac{|Actions|}{\epsilon}$ for ϵ-greedy). However, there is no guarantee for such assumption to hold in all navigation tasks. An agent can go through longer periods of time where no exploration is required (e.g., an unconstrained path to goal), followed by shorter periods of highly constrained motion, where exploration must be done to find an efficient path. Although choosing large values of ϵ or small values of T may help identify more changes in the reward distributions, it comes with the price of excessive trashing due to unnecessary exploration. If, instead, the value of ϵ were to be adjusted online based on local conditions, a more refined exploration-exploitation process could produce more efficient motions. To address this issue, ALAN dynamically adapts the exploration mechanism to the local conditions of the agent by using the '*win-stay, lose-shift*' learning strategy from game theory.

The '*win-stay, lose-shift*' strategy was originally proposed as a strategy for improving the action selection performance in multi-armed bandit problems [98]. It has been shown to promote cooperation in multi-agent systems and other domains [99, 100]. It can be simply described as the strategy of maintaining one's course of action if such an action is 'winning', and modifying it otherwise. The interpretation of a 'winning' situation varies depending on the application domain, though it refers in general to the interaction between agents. We propose to use this strategy in our problem domain

by assuming that an agent is 'winning' if it is allowed to move freely towards its goal without any constraint from its environment ($\mathbf{v}^{\text{new}} = \mathbf{v}^{\text{goal}}$). Our course of action in this case is to keep the goal-oriented velocity $\mathbf{v}^{\text{goal}}$, without performing unnecessary exploration. Consequently, the agent is in a 'losing' situation if its goal-oriented motion is constrained, i.e., the agent has to slow down or deviate from its direct path to the goal. In this case, the agent should explore for potentially better actions.

Algorithm 2: ALAN's action selection

1: **Input:** $\epsilon \in [0, 1]$
2: **Output:** a
3: $a \leftarrow action(t-1)$
4: **if** $a =$ goal-oriented motion **then**
5: **if** $\mathbf{v}^{\text{new}} = \mathbf{v}^{\text{goal}}$ **then**
6: $\epsilon \leftarrow 0$ (the agent is 'winning')
7: **else**
8: $\epsilon \leftarrow \epsilon + \beta$ (the agent is 'losing')
9: **end if**
10: **end if**
11: randomly generate variable $p \in [0, 1]$
12: **if** $p \leq$ (1-ϵ) **then**
13: $a \leftarrow \arg\max_a(\mathcal{R}_a)$
14: **else**
15: $a \leftarrow \arg\max_a(w\text{UCB(a)})$
16: **end if**

Algorithm 2 summarizes the action selection mechanism employed in our ALAN learning approach (Alg. 1, line 5). Similar to ϵ-greedy, the parameter ϵ controls the probability of selecting the best current action or exploring a different one. However, unlike ϵ-greedy, the exploration is performed by choosing the wUCB-suggested action. In our implementation, the value of ϵ is controlled by the '*win-stay lose-shift*' strategy. The agent uses its previously chosen action to determine whether it is in a 'winning' situation or not. When the agent is 'winning', it sets ϵ to 0 (line 6) to fully exploit the goal-oriented action. However, when this motion is constrained, the agent is encouraged

to explore different actions by gradually incrementing ϵ using steps of size β (line 8). This increases the probability of selecting the wUCB-suggested actions that have high wUCB estimates. When the agent finds a new best action (other than the goal-oriented one), it will exploit it with (1-ϵ) probability, while keeping ϵ fixed. This prevents the algorithm from increasing the exploration rate when it is not necessary. Finally, the algorithm outputs a new action a, that is, a new $\mathbf{v}^{\text{pref}}$, which is passed to the collision-avoidance component of our ALAN framework (Alg. 1, line 7).

Theorem 3. *The ALAN action selection is expected to find and exploit the optimal solution in at most* $T \cdot \epsilon^{-1}$ *timesteps.*

Proof: We can divide the analysis based on the two cases that an agent can encounter: 'winning' or 'losing'. If the agent is 'winning' ($\epsilon = 0$), it fully exploits the goal-oriented motion. In this case, the goal-oriented motion is the optimal action a^*, hence there is no need to explore for potentially better ones. If instead, the agent is 'losing' ($\epsilon > 0$), we know from Theorem 1 that it exploits the estimated optimal action on average every $(1 - \epsilon)^{-1}$ timesteps, while exploring on average every ϵ^{-1} timesteps. From the wUCB convergence proof of Theorem 2, we derived an upper bound of T timesteps for exploiting the optimal action. However, actions suggested by wUCB are selected with probability ϵ. Therefore, our ALAN approach is expected to identify and exploit the optimal action a^* at most in T $\cdot \epsilon^{-1}$ timesteps. ■

5.4 Scenarios

Figure 5.2 summarizes the different simulation scenarios where ALAN was evaluated. These include:

- *Intersection*: Three agents cross paths while moving towards their goals (Fig. 5.2a).
- *Deadlock*: Ten agents start at opposite sides of a long, narrow corridor. Only one agent can fit in the narrow space (Fig. 5.2b).
- *Perpendicular Crossing*: Two agents have orthogonally intersecting paths with a crowd of 24 agents (Fig. 5.2c).
- *Circle*: 64 agents walk to antipodal points on a circle. (Fig 5.2d).

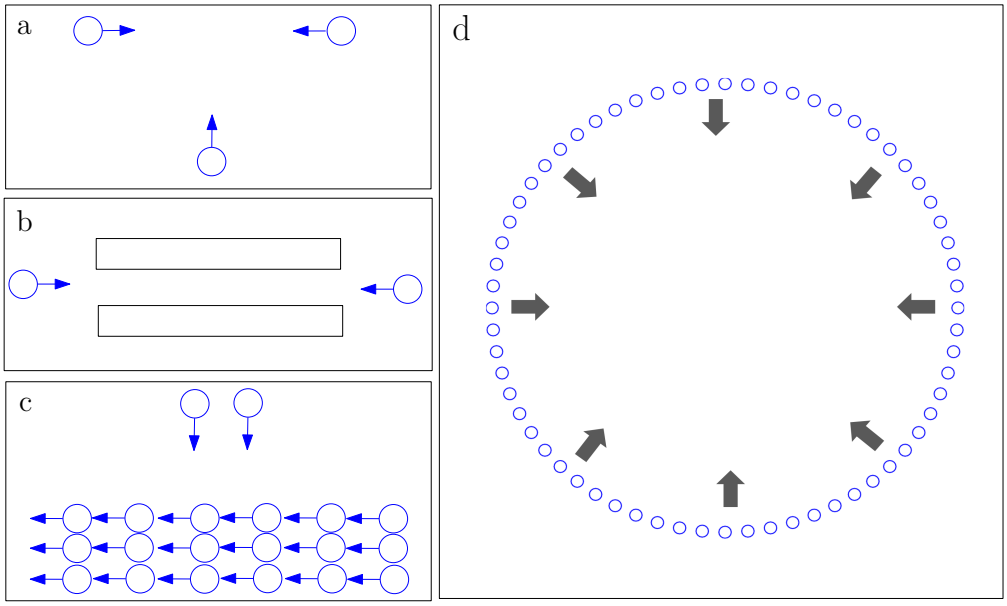


Figure 5.2: Simulated scenarios: (a) *Intersection*, (b) *Deadlock*, (c) *Perpendicular Crossing* and (d) *Circle*.

5.5 ALAN Results

We evaluated the interaction overhead of ALAN across the scenarios described in Fig. 5.2 and compared it to ORCA, a memory-less implementation of ϵ-greedy and wUCB. In the ALAN implementation, the values of γ (Eq. 5.1), c and T were empirically determined to 0.6, $\sqrt{2}$ and 50, respectively, whereas a fixed value of 0.1 was used for β (Alg. 2). Whenever ϵ-greedy and wUCB were used, we also experimentally determined the best ϵ, c and T values respectively (0.1 for ϵ, 0.4 for c and 50 for T).

As can be inferred from the figure, ALAN outperforms ORCA in all but the *Circle* scenario. As ORCA agents use a single goal-oriented action, they can easily get stuck in local minima, especially in scenarios where they have conflicting goals. This is particularly evident in the *Deadlock* scenario, where the two groups of agents end up getting trapped in the middle of the corridor, unable to reach their goals. Increasing the number of actions that the agents can choose and selecting a new one using wUCB or ϵ-greedy can resolve such conflicting issues enabling the agents to find their goals.

However, agents using ϵ-greedy and wUCB tend to exhibit oscillatory behavior due the frequent changes in their preferred velocities. In addition, both of these approaches exhibit excessive exploration that diminishes any advantage from learning the best action, and are unable to outperform ALAN in both the *Circle* and *Perpendicular Crossing* scenarios.

In contrast, ALAN, by using the principles behind ϵ-greedy and wUCB, and incorporating the concept of '*win-stay-lose shift*', is able to consistently improve the overall time-efficiency of the agents producing in many cases a travel time that is close to the theoretically optimal (interaction overhead ≈ 0). The only exception is the Circle scenario where ALAN agents are unable to predict the congestion that will develop when all agents reach the middle of the scenario, and when they reach this area none of the actions can move the agents closer to their goals. Hence, using the '*win-stay-lose shift*' strategy, they find themselves exploring actions for a long time (while in a 'losing' situation), until eventually the congestion resolves, and the agents can resume their goal-oriented motions. This behavior results in worse interaction overhead than ORCA agents, as their pushing behavior helps them to make their way through the congestion, and reach their goals faster.

5.6 Analysis

In this section, we analyze the contribution of ALAN in the problem of minimizing the travel time of all agents, its computational complexity and its sensitivity to the parameter γ of the reward function.

ALAN is a computationally fast method to find the optimal action based on Eq. 4.1, where agents use knowledge of the past (specifically, the average reward value for each action in the time window) to estimate the value of each action. By using a small time window over which the average reward is computed, ALAN allows agents to quickly adapt to changing navigation conditions (for example, moving from a clear to a congested area or vice versa). This, together with the use of the 'win-stay, lose-shift' strategy, an ϵ exploration parameter and wUCB for choosing exploratory actions, allows agents to maximize their time-efficiency both in clear environments (when it is 'winning') as well as in congested conditions (when it is 'losing').

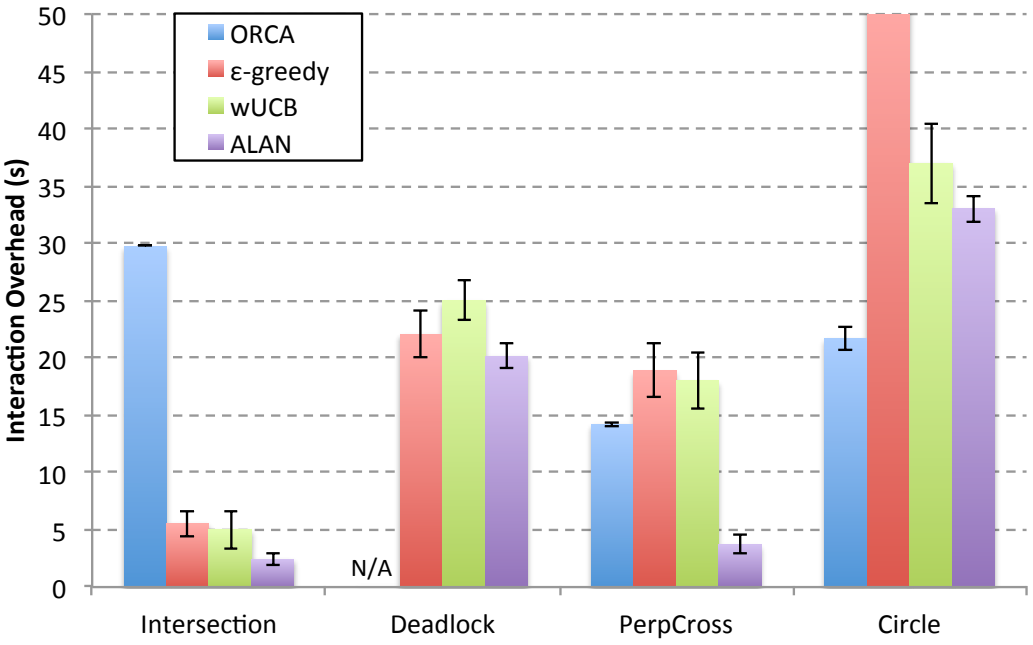


Figure 5.3: Performance comparison between ORCA, ϵ-greedy, wUCB and ALAN. In all scenarios with the exception of the Circle, ALAN agents outperform agents using ORCA, ϵ-greedy and wUCB.

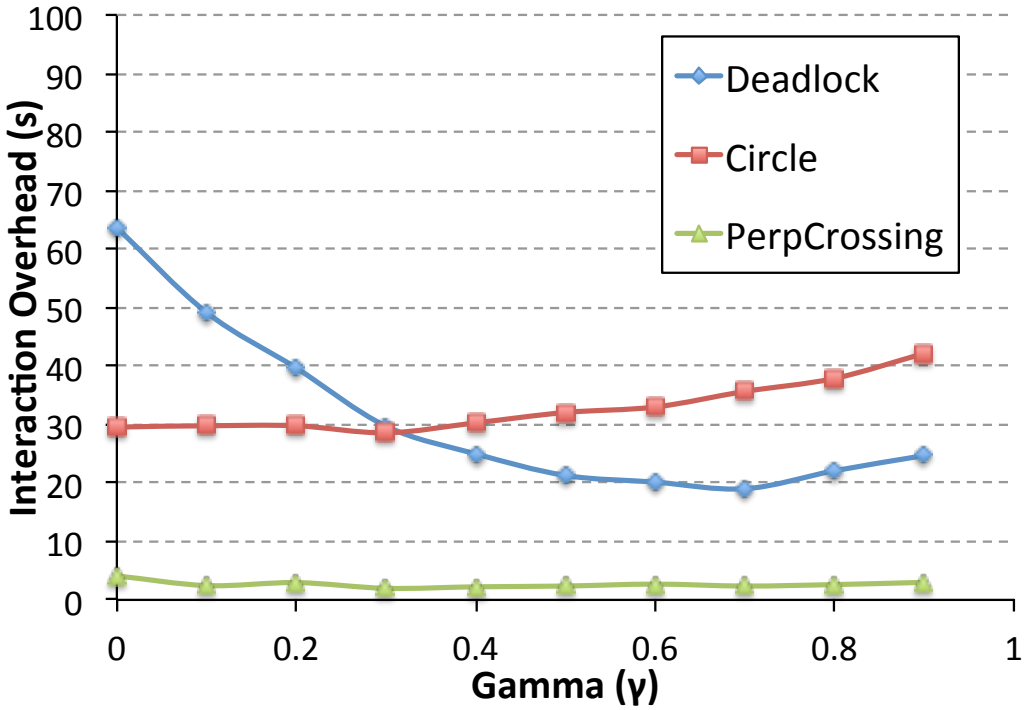


Figure 5.4: Performance of ALAN agents with different values of γ.

5.6.1 Runtime Complexity

In ALAN, during each simulation cycle, each agent performs two main operations: it first chooses a new preferred velocity using its online action-selection algorithm (cf. Algorithm 2) and then maps this velocity to a collision-free one using ORCA. In practice, since the number of actions that need to be evaluated is fixed, the step of selecting a new action is negligible in runtime, and the ORCA step dominates the overall runtime performance. Consequently, similar to ORCA, ALAN runs in $\mathcal{O}(n)$ time per agent, where n is the number of neighboring agents and obstacles used to compute the non-colliding constraints of the agent.

5.6.2 Effect of the parameter γ

Figure 5.4 shows how the balance between the goal-oriented and the politeness components of our reward function (Eq. 5.1), controlled by the parameter γ, affects the

performance of ALAN. As can be observed from the figure, ALAN performs well with γ values between 0.5 and 0.8, which indicates that considering both components of Eq. 5.1 consistently improves the overall time-efficiency of the agents, as compared to agents that are only goal-oriented ($\gamma = 0$) or only polite ($\gamma = 1$). In particular, pure goal-based evaluation forces agents to choose conflicting actions which can lead to high interaction overheads. On the other hand, just using the politeness component prevents the agents from reaching their goals, since all unconstrained actions have the same reward value and an agent is equally likely to select one of them.

Looking at the individual scenarios, we can observe that, in the *Circle* scenario, ALAN is much more robust to lower values of γ than in the *Deadlock* scenario. The reason for this is that in the *Circle* scenario, the absence of obstacles allows agents to find a path to their goals avoiding the congestion that develops in the middle. However, in the *Deadlock* scenario, agents are constrained by the walls on the side, which means that giving more weight to the goal progress does not allow them to find a clear path to their goals. In this scenario, showing a high level of politeness, which allows the other group to move to their goals, pays off in the long term.

Finally, in the *Perpendicular Crossing* scenario, the performance of ALAN is invariant to the specific choice of γ.

Given the sensitivity analysis in Fig. 5.4, for all of the experiments presented in this dissertation, we used $\gamma = 0.6$ for ALAN agents.

5.7 Limitations of ALAN

Although ALAN helps agents to reach their goals faster, requiring significantly less exploration than ϵ-greedy and wUCB, the agents can still choose suboptimal actions in order to address the exploration/exploitation dilemma. This suboptimal behavior introduces a noisy signal to the input of the other agents, which can have a negative impact on their motions. Also, agents evaluate their actions based only on their past observations without considering their long-term consequences. This prevents a robust evaluation of the actions, resulting, at times, in inefficient behaviors, such as in the *Circle* scenario mentioned above.

If an agent, instead, had more computational resources available, it could avoid

exploration by generating 'snapshots' of possible future situations that it may encounter using each one of its actions for a longer period of time, and making a decision in hindsight. In Chapter 6, we present a progressive planning approach, PHOP, that performs this computation. With PHOP, each agent simulates its own motion and the potential motions of the other nearby agents, and selects, at each cycle, the action that directly maximizes Eq. 5.1. This enables a higher level of interaction between the agents, at the cost of extra computation time, allowing them to follow near time-optimal paths and reach their goals in environments where ORCA or even ALAN agents are unable to.

Chapter 6

Predicting the Future (PHOP)

In Chapter 5 we presented ALAN, a method for agents to learn the values of their actions by executing them and evaluating their feedback. Although agents using ALAN are able to find the optimal action in a dynamic and rapidly evolving environment, its uncertainty about the future effect of the selected actions prevents the agents from committing to long-term plans. However, agents could simulate the potential future state of the environment based on their individual actions, and use this information to reduce the uncertainty and choose an action based on its potential long term value.

In this chapter, we propose the progressive hindsight optimization (PHOP) approach, an anytime approach which further improves the quality of the interactions between the agents, as compared to ALAN. Using PHOP, agents simulate potential trajectories for many timesteps into the future, at the cost of increased computational complexity.

PHOP uses the same reward function as in ALAN (Eqs 5.1, 5.2 and 5.3 for the general, *goal-oriented* and *politeness* components respectively). We refer the reader to Chapter 5.1 for a detailed description.

6.1 Motion Simulation

In ALAN, agents need to perform suboptimal actions in order to address the exploration/exploitation dilemma that arises from the multi-armed bandit formulation of the multi-agent navigation problem. The exploration process can be seen as erratic motion

that introduces noise in the sensing input of the other agents. Instead, if more computational resources are available, agents could avoid this exploration by simulating the outcome of Eq. 5.1 for each action, before deciding on how to move.

Algorithm 3 shows how to estimate the reward of a given action a for the next timestep. The algorithm projects the collision-free velocity that would result if the agent executed a (line 6). When simulating a, the agent also projects the velocity of its neighbors into the next timestep, assuming that they will continue along their current trajectories and use ORCA to avoid colliding with each other (lines 3-5) . After simulating the action a, the agent computes its expected reward value using Eq. 5.1 (lines 7-9).

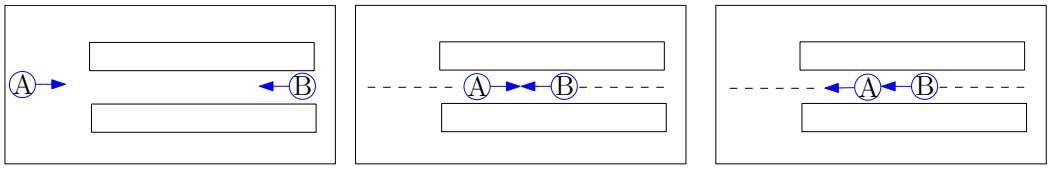
Algorithm 3: ActionSim(a) for an agent

1: **Input:** integer $a \in Actions$
2: **Output:** $\mathcal{R}_a$, estimated value of action a
3: **for all** $j \in NN_i$ **do**
4: project neighbor j velocity for next timestep
5: **end for**
6: project action a for next timestep
7: $\mathcal{R}_a^{goal} \leftarrow GoalReward(a)$ (cf. Eq. 5.2)
8: $\mathcal{R}_a^{polite} \leftarrow PoliteReward(a)$ (cf. Eq. 5.3)
9: $\mathcal{R}_a \leftarrow (1-\gamma) \cdot \mathcal{R}_a^{goal} + \gamma \cdot \mathcal{R}_a^{polite}$

By using Algorithm 3 to simulate and evaluate the estimated reward for all of its actions, the agent can directly maximize Eq. 5.1 and choose an optimal action at any time. This eliminates the need for performing exploratory actions, generating a smoother trajectory and giving the other agents a clear signal of its intended motion.

6.2 Uncertainty

Simulating all actions before choosing one helps agents optimize their local motions. However, the reward value computed for each action is only a short term estimate of the real value of an action. An action that appears to be of high quality in the short term could be detrimental for the agent's progress in the long term. Consider, for example,



(a) Initial State (b) ORCA (c) ALAN

Figure 6.1: Agent A's goal-oriented motion is locally optimal from its initial state (a), but results in local minima which either cannot be resolved (b) or requires backtracking on the agent's path (c).

the scenario in Fig. 6.1. Here, using ALAN and computing the reward for each action might indicate that following a goal-directed motion would be optimal, as any other motion would deviate from it. However, the agent is not taking into account the fact that in the long term, its local decision will significantly impact its travel time, as it will either get stuck in the middle of the corridor with the other agent (using ORCA, Fig. 6.1b), or it will be forced to backtrack (using ALAN, Fig. 6.1c).

This exemplifies a fundamental problem for an agent navigating in an environment shared with other agents, which is the uncertainty in the future motion of the other agents, and the uncertainty in how these agents will respond to any new agent approaching them. One way to account for this issue is for an agent to consider a variety of potential trajectories and the future scenarios that it might encounter, and use that information to perform short term action selection. In this way, an agent can, in 'hindsight', evaluate and select an action that in the long term seems more efficient, even if it may be less efficient right now. We adapt a planning approach, called *hindsight optimization*, to address this issue.

6.3 Hindsight Optimization

Hindsight optimization (HOP) is a technique for online action selection used to compute an upper bound on the expected reward of an action by allowing the agent to 'peek' on potential future scenarios and plan accordingly. Although the expected reward is not a tight upper bound on the real value of the action, it is often indicative enough for action comparison purposes [60]. HOP has been applied to many domains [5, 61]. An agent

using HOP formulates a series of *T-Horizon* futures of length T, computes a policy for each timestep $t \in T$ and obtains rewards for executing that policy from $t = 0$ until $t = T$, which is associated with the policy at time $t = 0$ (i.e. the first action executed). By generating multiple deterministic future scenarios, and averaging the results under the same initial action, the agent can estimate, in hindsight, the best action to perform in the present. For a more detailed theoretical description of Hindsight optimization, we refer the reader to [61].

Although using HOP can result in large computational gains [61], the planning time must be balanced with the real-time constraint. This constraint represents an upper bound on the time that the agent has at its disposal to plan its next action. We adapt the HOP approach to progressively generate more complex *T-Horizon* plans, using the available computational resources while guaranteeing its real-time applicability. We call our approach *progressive hindsight optimization* (PHOP).

6.4 Progressive Hindsight Optimization

The progressive hindsight optimization approach uses HOP to compute paths in a progressive manner to ensure that when the planning process must be stopped, the current best plan can be executed. PHOP works by allowing each agent to simulate progressively complex plans of actions for a given time horizon, and to analyze in 'hindsight' the consequences of each plan. It accounts for the uncertainty in the environment by creating potential future scenarios, in order to obtain an estimate of the value of the agent's immediate actions. This approach can be seen as a form of Receding Horizon Control [101], although it does not require communication between the agents. To account for unforeseen changes in the environment (for example, other agents changing their motions in an unexpected manner), it becomes necessary to replan at every timestep, allowing the agent to adapt to the new positions and velocities of the other agents. Hence, the proposed planner is executed at every simulation cycle to increase the agent's adaptability. At each timestep, the agent follows the sense-act cycle.

6.4.1 Plan Generation and Execution

The agent performs the plan generation process based on Hindsight optimization. This process consists of simulating multiple plans of actions in the future. For each plan, the agent internally simulates the execution of a sequence of actions for a specified number of timesteps (*T-Horizon*) in the future. During each timestep, the agent uses Alg. 3 to evaluate its reward and (virtually) update its position and the positions of its neighbors.

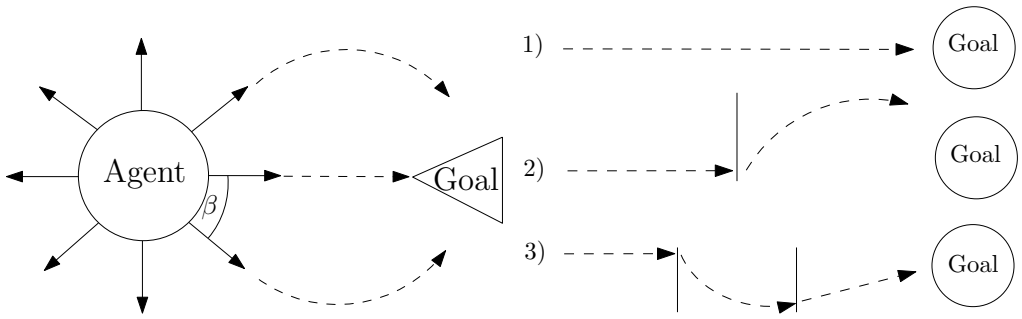
After the *T-Horizon* of the projected plan is reached, the agent averages its reward across the T simulated timesteps, and associates the averaged reward with the initial action of the plan. The planning continues until a specified time limit is reached. This allows the agent to have a broad estimate of the value of each of its actions when time is very limited, and to get a more accurate estimate of the expected reward for each action when more time is available. Finally, the agent performs the action that maximizes the expected reward based on the plans generated.

Compared to the original HOP technique, our approach has two main differences. First, instead of generating different futures independent of the agent's actions, we consider the different futures that could occur as a consequence of the agent taking a particular sequence of actions. This is because the agent and its neighbors influence each other with their actions. Second, to make the approach anytime, it needs to be ensured that the quality of the obtained solution increases as more time is given to the algorithm. To do this, we increasingly partition the *T-Horizon* across the different available actions, computing a more detailed plan as the number of partitions increases. Figure 6.2 shows three example action plans with 1, 2 and 3 partitions respectively.

6.4.2 Algorithmic Description

Algorithm 4 presents the pseudocode for a simulation loop using PHOP. It takes as input the *T-Horizon* desired for the plan ($timeHorizon$) and a time limit for the planning process ($timeLimit$), which in our implementation is fixed at 25ms.

The function $generatePlans(partitions)$ returns a set of plans ($PlanSet$) for the agent to simulate. We adopted a top-down approach for plan generation: we first simulate the agent following each of its individual actions for the entire *T-Horizon*, and then incrementally partition the *T-Horizon* (according to the value of $partitions$) to



(a) Action space (b) Example Trajectories

Figure 6.2: Example of progressive agent trajectories with 1, 2 and 3 partitions respectively.

combine the simulation of multiple actions. Instead of populating vectors for each plan, we adopted a memory efficient approach that dynamically generates each action, based on the current number of partitions desired. Each action a of each generated plan is evaluated using the *ActionSim*() function from Algorithm 3, for the entire *T-Horizon* (lines 11-14). After each plan, its value is computed using the average of the estimated rewards for each of its actions (line 15). The function *intialAction*(*plan*) returns the first action in the sequence specified by *plan*, and it is used to relate the reward of the plan to that initial action.

The planning process finishes when the time limit has been reached, as checked both in line 7 and the *checkTime*(*timeLimit*) function. If this is the case, the algorithm returns the currently best evaluated action ($\arg\max_{a \in Actions} \mathcal{R}_a$). This action serves as an input preferred velocity to the $ORCA(\mathbf{v}^{\text{pref}})$ function, which in turn computes a new collision-free velocity for the the agent and the sense-act cycle is repeated.

6.5 PHOP Results

We evaluated the interaction overhead of PHOP in the scenarios described in Fig. 5.2. Given that PHOP agents use a longer time horizon than ALAN to plan their motions, we also evaluated it in scenarios where ORCA and ALAN agents are not always able to reach their goals, that is, when the direct path to the goals for some of the agents is blocked with an obstacle. Figure 6.3 summarizes the new simulation scenarios. These

Algorithm 4: PHOP step for an agent

Input: $timeHorizon \in \mathbb{Z}$, $timeLimit \in \mathbb{R}^+$
initialize simulation, $\mathbf{p} = (x_{start}, y_{start})$
while not at the goal **do**
  if $UpdateAction(t)$ **then**
    $partitions \leftarrow 1$
    $time \leftarrow 0$
    while $time \leq timeLimit$ **do**
      $PlanSet \leftarrow generatePlans(partitions, timeHorizon)$
      for all $plan \in PlanSet$ **do**
        $PlanR \leftarrow 0$
        for $t = 0, \ldots, (timeHorizon - 1)$ **do**
          $a^t \leftarrow$ action of plan at time t
          $PlanR \leftarrow PlanR + ActionSim(a^t)$
        end for
        $PlanR \leftarrow \frac{PlanR}{timeHorizon}$
        $a \leftarrow initialAction(plan)$
        $Count(a) \leftarrow Count(a) + 1$
        $\mathcal{R}_a \leftarrow \frac{\mathcal{R}_a \cdot (Count(a)-1) + PlanR}{Count(a)}$
        $\mathbf{v}^{\text{pref}} \leftarrow \arg\max_{a \in Actions} \mathcal{R}_a$
        $checkTime(timeLimit)$
      end for
      $partitions \leftarrow partitions + 1$
    end while
  end if
  $\mathbf{v}^{\text{new}} \leftarrow ORCA(\mathbf{v}^{\text{pref}})$
  $\mathbf{p}^t \leftarrow \mathbf{p}^{t+1} + \mathbf{v}^{\text{new}} \cdot \Delta t$
end while

include:

- *3-Room*: 32 agents start in the rightmost room, and must cross two sets of exits

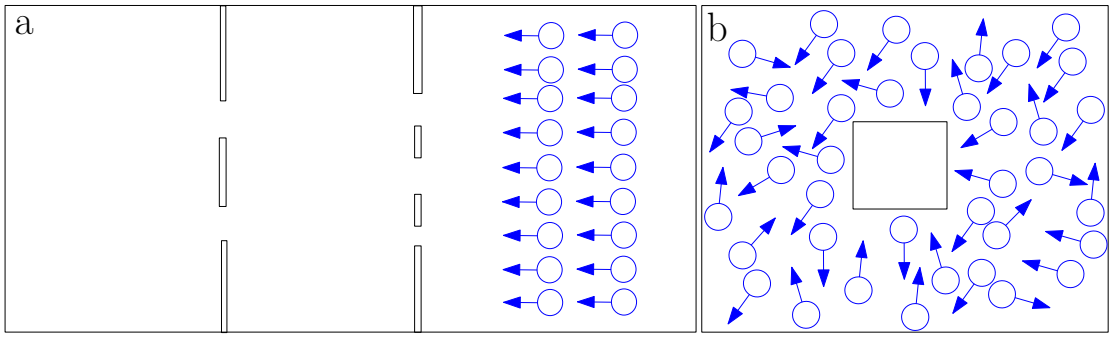


Figure 6.3: Additional simulated scenarios for PHOP. (a) *3-Room*: 32 agents start in the rightmost room, and must cross two sets of exits to reach their goal in the leftmost room. (b) *Square*: 100 agents uniformly distributed must reach their randomly generated goals, while a static obstacle in the middle of the scenario blocks their path.

to reach their goals in the leftmost room (Fig. 6.3a).

- *Square*: 100 agents uniformly distributed must reach their randomly generated goal, while a static obstacle in the middle of the scenario blocks their path (Fig. 6.3b).

Figure 6.4 shows the different values for interaction overhead for each of the described approaches. We can observe from the figure that PHOP agents clearly outperform ORCA and ALAN agents in all scenarios (or at least are not worse than ALAN in the Perpendicular Crossing scenario), and are even able to reach to their goals when there are obstacles in their goal-directed path. This performance improvement is due to several factors, which we explain in the context of the scenarios that highlights them.

First, PHOP agents do not need to perform exploration, which prevents them from choosing suboptimal actions (which ALAN needs to do). Using PHOP, agents can directly execute the estimated action that maximizes Eq. 5.1. This explains why PHOP is able to outperform ALAN in the *Intersection* and *Perpendicular Crossing* scenarios, where ALAN's interaction overhead values were already close to the optimal.

Second, PHOP agents can predict the formation of congestion, and can take actions to avoid it. This ability translates into lower interaction overhead than ORCA in the *Circle*, a scenario where ALAN could not outperform ORCA. Here, PHOP agents take sideways motions when congestion is predicted in the center of the environment, which

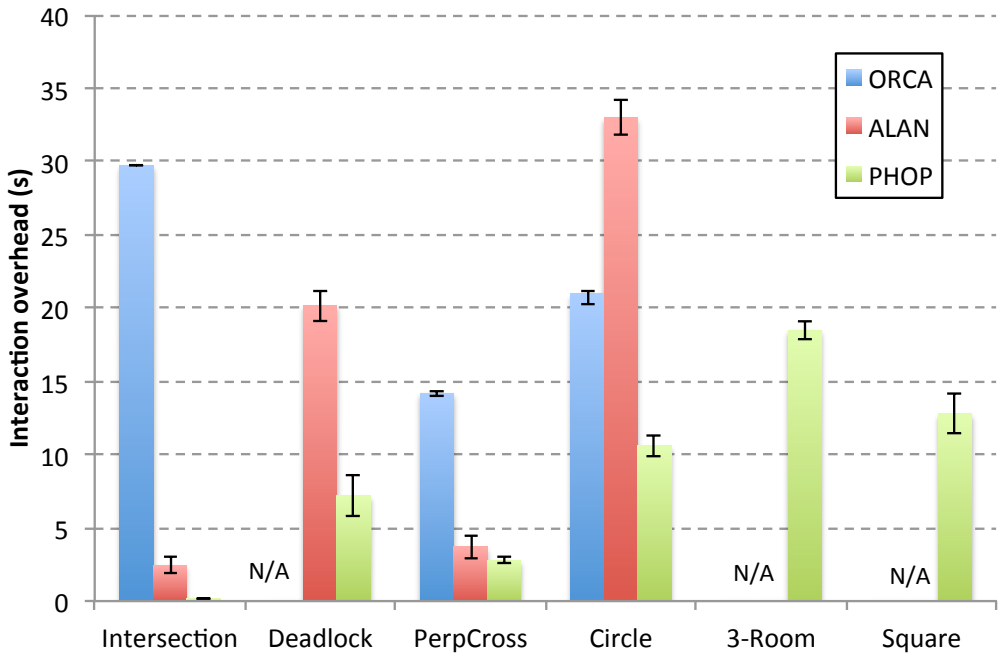


Figure 6.4: Performance comparison between ORCA, ALAN and PHOP.

results in a 50% improvement over ORCA's interaction overhead, and over 65% when compared to ALAN's performance. This prediction ability also helps PHOP to solve the *Deadlock* scenario significantly faster than ALAN and ORCA. Specifically, PHOP agents predict the local minima that will form in the middle of the corridor, and take actions to avoid the corridor. This translates into lower interaction overhead than ALAN, as PHOP agents do not need to perform the backtracking behavior that allowed ALAN to solve this scenario.

Finally, the longer *T-Horizon* that PHOP agents use to evaluate their actions allows them to reach their goals in scenarios where ORCA and ALAN cannot reach them, such as the *3-Room* and *Square* scenarios. Here, unlike the previously evaluated scenarios, the goal directed path of many agents is blocked by large obstacles. ORCA agents are unable to select velocities other than goal-oriented one, which prevents them from finding non-direct paths to their goals. Even though the ALAN agents have a variety of velocities to choose from, they use only the past observations in their action evaluation, and do not have the capability to determine the long term effect of each of their actions.

6.6 Analysis

In this section, we first analyze the role of PHOP in minimizing the total travel time of the agents. Then, we analyze the computational complexity of PHOP (compared to ALAN's complexity), before discussing a method to optimize the evaluation of actions to significantly reduce PHOP's planning time. We then study the sensitivity of both ALAN and PHOP to the parameter γ of the reward function, and finally elaborate on scenarios where different types of agents (ORCA, ALAN, and PHOP) share the same environment, investigating how such heterogeneity affects the performance.

PHOP agents estimate the value of the actions by predicting their potential long-term value, and greedily chooses the action with the largest estimated value for its execution. By choosing actions that maximize the local reward function for a time horizon, and re-evaluating the actions each timestep (to account for prediction errors and new information) PHOP empirically reduces the travel time of the agents (as measured by the interaction overhead values in Figure 6.4), by predicting, for example, the development of congestion in their goal paths.

Both PHOP and ALAN (see Chapter 5) select actions that move the agents closer to their goals while also being 'polite' to other agents motions. The combination of these behaviors, in the long term, minimizes the travel time of the set of agents.

6.6.1 Runtime Complexity

As compared to ALAN, the runtime performance of PHOP is dominated by its action-selection routine (Alg. 4, lines 7-23). In particular, at each simulation cycle, each agent needs to simulate its neighborhood dynamics for each of the T-steps of its *T-Horizon* and for each one of its actions. Given an action, this involves computing for each of the agent's neighbors a new collision-free velocity using ORCA, which takes $\mathcal{O}(n)$ time per neighbor per *T-Horizon* step. Therefore, assuming there are n neighbors and k actions to be evaluated, the runtime complexity of PHOP is $\mathcal{O}(k \cdot T \cdot n^2)$ per agent.

6.6.2 Optimizing Action Evaluation

Although PHOP provides lower travel time for the agents compared to both ALAN and ORCA, it does so at the expense of a larger computation time. This would indicate

that PHOP is more suitable for multi-robot navigation (where each agent only needs to compute its own optimal action) than for real time simulation of a large scale multi-agent system moving through an environment. This is the by-product of the complete search that PHOP performs in the tree formed by the trajectories generated while evaluating the actions. Although this comprehensive search is what allows PHOP to compute the optimal action, there are tools that we can use to speed-up this computation. In this section, we present a branch-and-bound ([102]) type of method that allows us to considerably reduce the number of trajectories that PHOP needs to evaluate, optimizing the planning process before selecting an action. Branch-and-bound is an algorithmic paradigm used to restrict the space to be searched based on its solution's upper bound and on the current solution. In our domain, a 'solution' corresponds to the optimal action found by PHOP.

The method proposed optimizes the evaluation of actions based on the pre-computation of the upper bound of their reward values. These upper bounds need to be computed for the different number of sub-trajectories that compose each PHOP trajectory (see Fig. 6.2b), based on the number of partitions used in that stage of the planning process (Algorithm 4). The agent can use these upper bounds while evaluating the actions to decide which actions it needs to evaluate, as well as which actions it does not need to evaluate. These evaluation decisions have a significant impact in the planning time of the agent, as the evaluation of each action a using p partitions of each trajectory involve the generation and evaluation of $|Actions|^{p-1}$ sub-trajectories, where $|Actions|$ correspond to the total number of actions. In what follows, we describe the two stages of this method.

Upper bound action value computation

First, PHOP agents perform a one-time computation of the upper bound of the reward value of each action, based on trajectories divided in sub-trajectories according to the number of partitions considered. The evaluation begins with trajectories with a single partition, and progressively generate more partitions until the time horizon limit is met. As the actions have a different value based on the number of partitions considered, the upper bounds are computed for each number of partitions. This is done in three steps:

1. Compute the upper bound for actions based on a single trajectory: Agents compute the potential reward for each action a (using Eq. 4.1), $SingleRw(a)$, assuming that no other agent or static obstacles are present. In this case, the collision-free velocity ($\mathbf{v}^{\text{new}}$) for each action is assumed to be equivalent to the preferred velocity ($\mathbf{v}^{\text{pref}}$) from which it is computed. These reward values correspond to the maximum possible values if each action was executed once, and are also equivalent to their upper bound in case the PHOP trajectories are not partitioned (where each action is repeatedly executed for the entire time horizon).

2. Compute average of individual reward of all actions: Agents compute the average reward values of the actions, $AvgRw$, based on the upper bound of their individual reward, $SingleRw(a)$, as follows:

$$AvgRw \leftarrow \frac{\sum_{a=1}^{|Actions|} SingleRw(a)}{|Actions|} \tag{6.1}$$

   This average represents the value that is added, in the best case scenario, to the reward of each action when increasing the number of partitions in the action trajectories.

3. Compute the upper bound for actions based on multiple sub-trajectories: Using the individual ($SingleRw(a)$ for action a) and average ($AvgRw$) reward values, we analytically derived a formula that computes the upper bound of the reward value for each action a, at any given partition level p:

$$UppBound(a,p) \leftarrow SingleRw(a) + (\frac{p-1}{p}) \cdot (AvgRw - SingleRw(a)) \tag{6.2}$$

Agents store the $UppBound(a,p)$ values in a table where each value can be easily accessed in constant $\mathcal{O}(1)$ time, or all values in $\mathcal{O}(|Actions| \cdot |P|)$ time (where $|P|$ is the maximum number of partitions). Algorithm 5 shows the one-time procedure to generate these upper bounds for each action, at each partition level.

This process needs to be done only once per simulation task by each agent, therefore it does not introduce any overhead to the on-line planning process. Once all

Algorithm 5: Upper bound action value generation

1: **for all** $a \in Actions$ **do**
2: $\quad SingleRw(a) \leftarrow (1-\gamma) \cdot \mathcal{R}_a^{goal} + \gamma \cdot \mathcal{R}_a^{polite}$
3: **end for**
4: $AvgRw \leftarrow \frac{\sum_{a=1}^{|Actions|} SingleRw(a)}{|Actions|}$
5: **for** $p = 1, \ldots, timeHorizon$ **do**
6: $\quad$ **for all** $a \in Actions$ **do**
7: $\quad\quad UppBound(a,p) \leftarrow SingleRw(a) + (\frac{p-1}{p}) \cdot (AvgRw - SingleRw(a))$
8: $\quad$ **end for**
9: **end for**

$UppBound(a,p)$ values are computed, the agent uses this information to decide, at each timestep, which actions does it need to evaluate and at which partition level.

PHOP evaluation optimization

At each timestep, an agent evaluates each action in the order of the upper bound of their values. Therefore, it first evaluates the action of moving directly towards the goal (as it has the highest upper bound), followed by the actions whose preferred velocity directions have the closest angle with respect to the direction of the goal. Hence, the last action evaluated corresponds to the action of moving in the opposite direction of the agent's goal.

After evaluating each action, the agent keeps track of the current highest evaluated action and uses this value, together with the table of $UppBound(a,p)$ values, to decide which actions it does not need to evaluate, for each number of partitions. This decision is made based on a comparison between the current highest action value, and the upper bound of each value on the table. If the upper bound of the value of action a using p partitions is lower than the currently highest computed value, then there is no need to perform that evaluation as, even in the best case, action a would not be chosen over the current best action. On the other hand, if the upper bound of the value of the pair (a,p) is larger, it means that it might have a better evaluation than the current best action, therefore it needs to be evaluated.

By going through this process, the agent only evaluates those actions that are candidates to being the optimal action, while it skips the evaluation of actions that, even in the best case, will not be chosen. This can significantly reduce the time that agents spend evaluating their actions, while guaranteeing the same level of performance, in terms of interaction overhead, compared to doing the complete search in the action space.

The degree to which this process reduces the computation time of PHOP depends on the surrounding conditions of each agent: when the agent is in a clear area, the goal-oriented action will have the largest reward possible at any partition level (i.e., going to the goal is always the best action, if the motion of the agent is unconstrained), and hence no other action will need to be evaluated. This is similar to the effect of the 'Win-Stay, Lose-Shift' strategy used in ALAN (see Section 5.3), where no exploration is done if the goal-oriented action is allowed. Instead, if the agent is in a congested situation, then the goal-oriented action will not likely have a large reward, requiring the evaluation of other actions at different partition levels, before finding the optimal one. Because in the worst case the agent might need to evaluate all of its actions, no theoretical guarantee can be made with respect to how much this method reduces the planning time. In practice, however, we observed speed ups ranging from 65% (in very congested scenarios) to 93% (in scenarios with few agents) of the action evaluation times when doing a complete search (see Figure 6.5).

We can see from Figure 6.5 that the largest speedup is achieved in the Intersection scenario. This scenario is characterized by sparse interactions between the agents, which means that during the majority of the simulation the agents are free to move towards their goals and do not need to evaluate the other actions at any partition level. On the other hand, the speedup is less aggressive in the Circle and Deadlock scenarios. Agents in the Circle scenario, although initially able to move to their goals unconstrained, need to evaluate more actions when congestion is predicted to develop in the middle of the environment. In the Deadlock scenario, agents need to deviate from their direct goal-oriented path to avoid getting trapped inside the narrow corridor, which involves the need to evaluate multiple actions.

The method presented in this section can be useful not only in reducing the time that agents spend finding the optimal action, but also to evaluate more fine-grained

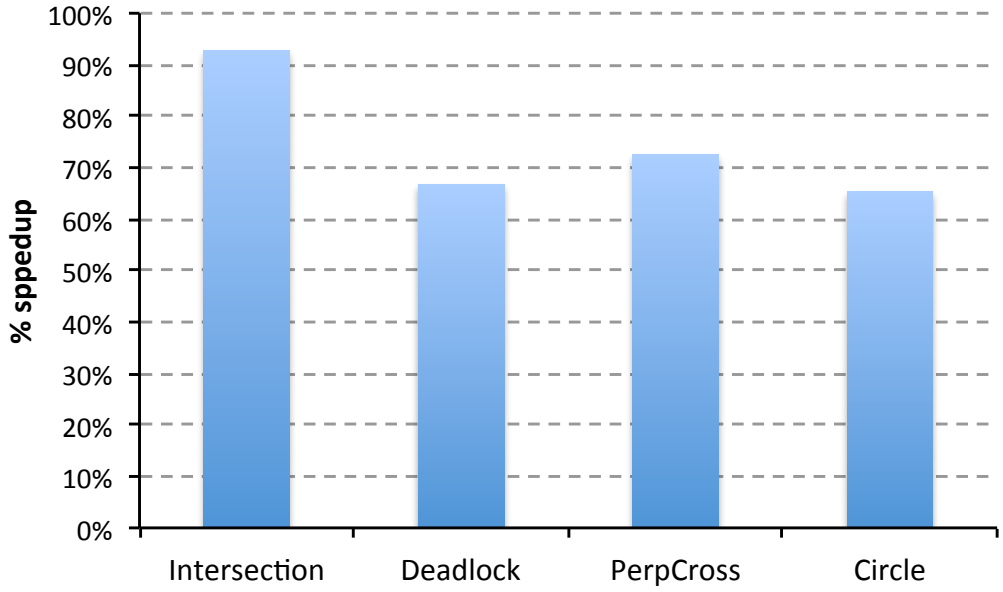


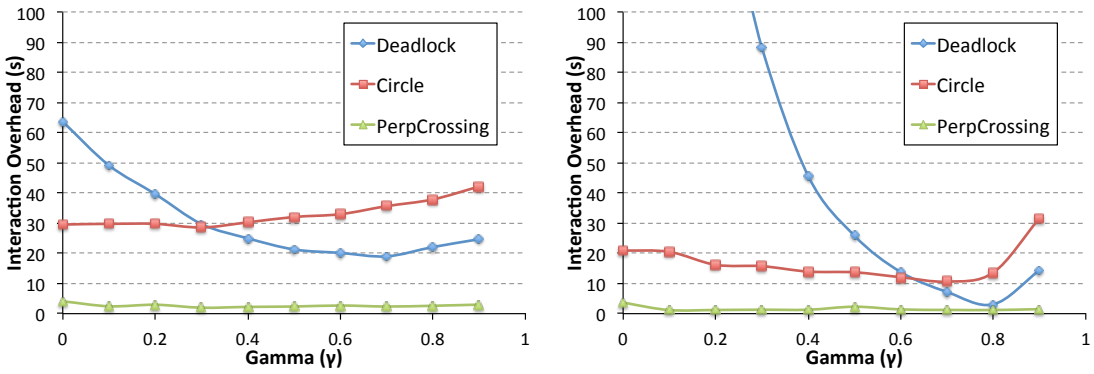
Figure 6.5: Speedup in planning time using the optimized action evaluation method, with respect to doing a complete search evaluating all actions.

trajectories for a fixed planning time, compared to performing a complete search in the space of actions and partitions.

6.6.3 Effect of the parameter γ

Figure 6.6 shows how the balance between the goal-oriented and the politeness components of our reward function (Eq. 5.1), controlled by the parameter γ, affects the performance of ALAN and PHOP. As can be observed from the figure, both approaches perform well with γ values between 0.5 and 0.8, which indicates that considering both components of Eq. 5.1 consistently improves the overall time-efficiency of the agents, as compared to agents that are only goal-oriented ($\gamma = 0$) or only polite ($\gamma = 1$). In particular, pure goal-based evaluation forces agents to choose conflicting actions which can lead to high interaction overheads. On the other hand, just using the politeness component prevents the agents from reaching their goals, since all unconstrained actions have the same reward value and an agent is equally likely to select one of them.

Looking at the individual scenarios, we can observe that, in the *Deadlock* scenario,



(a) ALAN (b) PHOP

Figure 6.6: Performance of ALAN and PHOP agents for different values of γ.

ALAN is much more robust than PHOP to lower values of γ. The reason for this is that PHOP agents do not explore suboptimal actions and hence, due to the nature of this scenario, the more goal-oriented the agents are, the more likely it is to end up inside the corridor and get trapped in local minima. However, as the politeness of PHOP agents increases ($\gamma \geq 0.6$), they are able to predict such time-consuming situations and avoid them by going around the corridor, outperforming the ALAN agents. In the *Circle* scenario, PHOP leads to much more time-efficient behavior than ALAN for all values of γ, as PHOP agents anticipate that congestion will evolve in the center of the environment and resolve it by taking sideways motions. In contrast, ALAN agents perform too much unnecessary exploration which leads to an increase in the interaction overhead as the value of γ increases. Finally, in the *Perpendicular Crossing* scenario, both PHOP and ALAN are invariant to the specific choice of γ.

Given the sensitivity analysis in Fig. 6.6, for all of the experiments presented in this Chapter, we used $\gamma = 0.6$ for ALAN agents and $\gamma = 0.7$ for PHOP agents.

6.6.4 Heterogeneous Agents

In all of the experiments discussed so far, all agents were of a single type (i.e., either ORCA agents, ALAN agents, or PHOP agents). Using this assumption, we empirically show that PHOP and ALAN improve the time-efficiency of the agents without the need for explicit communication and prior coordination. However, this assumption may not

always hold in real-life, as different types of agent can simultaneously share the same environment (e.g. humans). Here, we investigate how such heterogeneity in the types of agents affects the interaction overhead of the entire system.

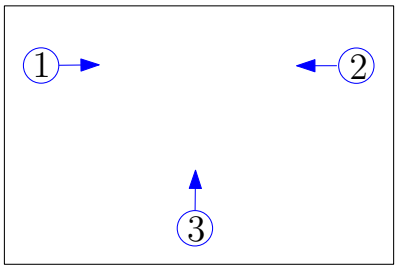
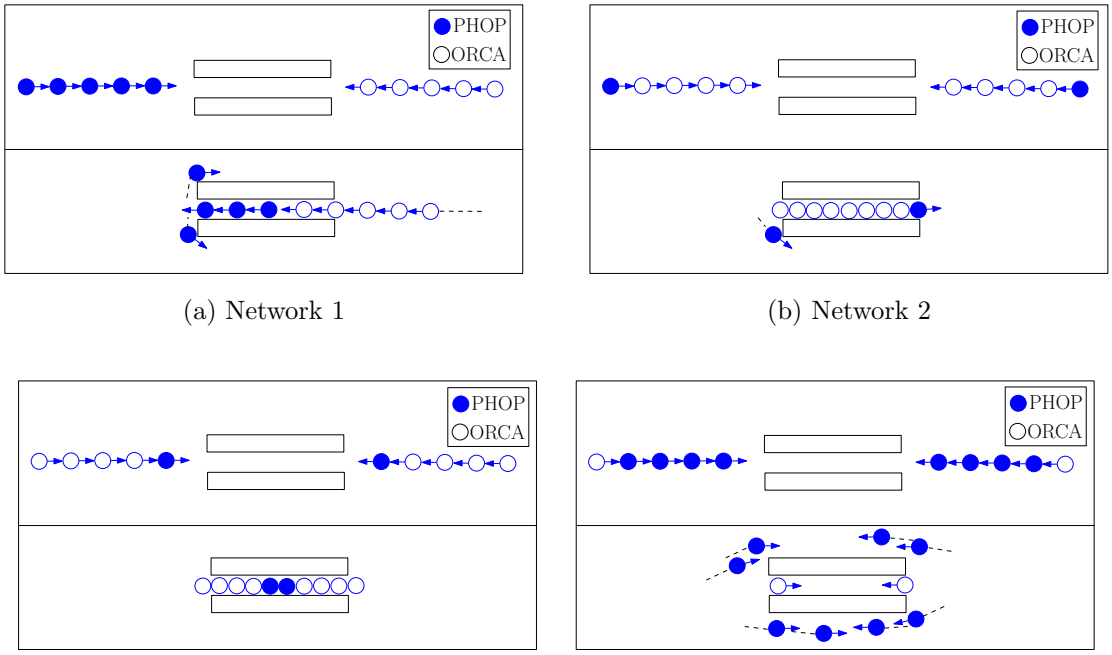


Figure 6.7: *Intersection* scenario with the ID of each agent

Consider, for example, the *Intersection* scenario where three agents cross paths while moving to their goals (Fig. 6.7). In this scenario, agents 1 and 2 have to swap positions, while agent 3 moves upwards having to traverse a longer distance than the other two agents. If all agents employ ORCA, agents 1 and 2 are pushed by agent 3 as they are unable to find goal-oriented velocities that are collision-free, resulting in a large interaction overhead (30 s). If, instead, at least one of the agents uses ALAN or PHOP, the interaction overhead is considerably reduced (< 3 s). In this case, the type of agent 3 is critical for the performance of the entire system. If this agent uses ALAN, it performs some exploration when its goal-directed motion is constrained, leading to an interaction overhead of 2.9 s. When agent 3 uses PHOP, the interaction overhead is reduced to less than 1 s, regardless of the types of the other two agents. This is because the PHOP agent does not explore suboptimal actions and quickly finds an unconstrained goal-oriented velocity. However, an optimal performance (0 s) is obtained when agent 3 uses ORCA, and the other two agents are PHOP agents. PHOP exploits the symmetry of agents 1 and 2, guiding one of them slightly upwards and the other slightly downwards, while allowing the ORCA agent to take a straight path to its goal.

This simple scenario showcases that even with 3 agents, different combinations of agent types can lead to significantly different performance. Besides their types, the relative position of the agents can also play an important role in the interaction overhead of the system. As an example, we tested four cases of different agent configurations in



(a) Network 1

(b) Network 2

(c) Network 3

(d) Network 4

Figure 6.8: The *Deadlock* scenario using different combinations of ORCA and PHOP agents. The top illustration in each subfigure shows the initial positions of the agents, while the bottom one shows the resulting behavior.

the *Deadlock* scenario using a mix of PHOP and ORCA agents, as shown in Fig. 6.8. For each case, we ran 10 simulations and measured the percentage of time that all agents were able to reach their goals.

The best performance (100% success rate) was obtained when one group was controlled by PHOP and the other group by ORCA (Fig. 6.8(a)). In this case, PHOP agents are able to predict that the ORCA agents will try to move inside the corridor, and resolve the congestion by going around the constriction. The worst performance (0% sucess rate) was obtained when the last member of each group used PHOP while the other agents were ORCA agents (Fig. 6.8(b)). Here, the PHOP agents had no influence in the motion of the ORCA agents, with the latter getting stuck in the middle of the corridor unable to reach their goals in any of the 10 simulation runs. In contrast, PHOP agents can have a positive impact on the performance when they are leading the two groups (Fig. 6.8(c)). In these simulations, if one of the leaders decides not to enter the corridor, the size-balance between the two groups breaks and the bigger group can push the smaller one outside of the corridor (20% success rate). Finally, when all but the last members of each group used PHOP, 50% of the time the agents were able to reach their goals (Fig. 6.8(d)). Similar to Fig. 6.8(b), the PHOP agents cannot prevent the two ORCA agents from continuing their goal-directed motion, which leads to local minima issues inside the corridor.

Similar types of analysis can be very useful in scenarios where the number of agents is small, as it allows us to strategically place different types of agents and influence the behavior of the system as a whole. In large-scale environments, however, the relative number of each type of agents is more important than their initial placement.

Figure 6.9 evaluates how different percentages of PHOP and ORCA agents affect the interaction overhead in the *Circle* scenario. The results indicate that even having 25% of PHOP agents can significantly improve the performance, as compared to just using ORCA agents. As more PHOP agents are added into the system, the interaction overhead continues to decrease. This shows the positive influence that PHOP agents have, not only when interacting with other PHOP agents, but with other types of agents as well.

We also evaluated the influence of PHOP agents in the *Square* scenario. Here, due to the large obstacle in the middle of the environment, ORCA agents can get

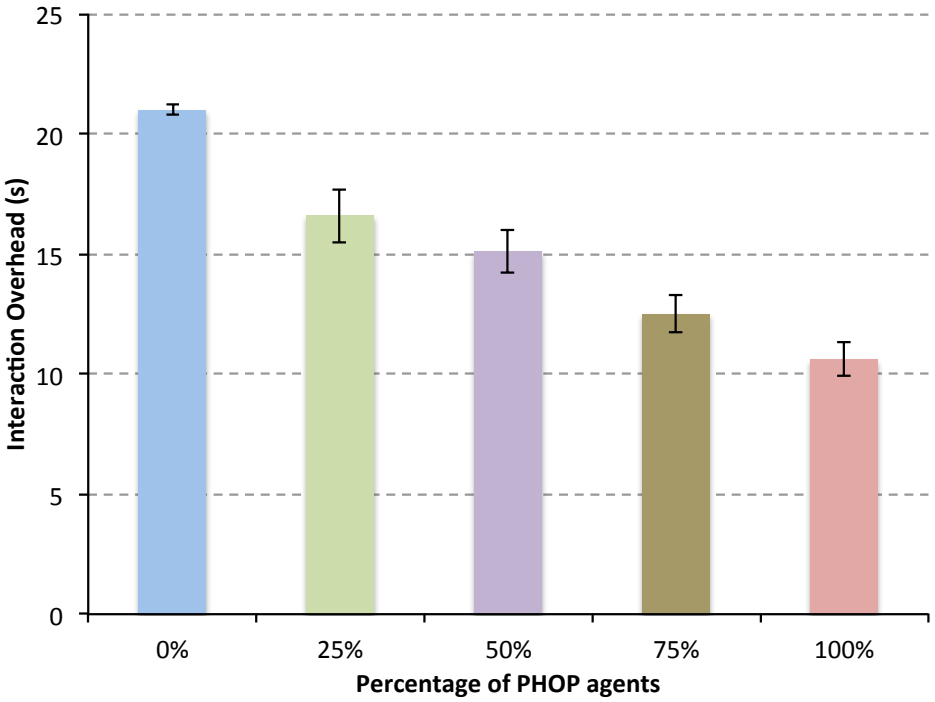


Figure 6.9: Interaction overhead for different percentages of PHOP agents, in the *Circle* scenario. The rest of the agents use only ORCA.

stuck and never reach their goals. The results in Fig. 6.10 show that having PHOP agents, once again, helps the overall performance, as such agents reduce the interaction overhead and increase the goal reachability. This confirms the benefits of using PHOP in heterogeneous environments where the assumption of all agents being of the same type does not always hold.

6.7 Limitations of PHOP

Because of the extended *T-Horizon* used by PHOP to plan their motions, they are able to find paths that move them closer to their goals, even when there are obstacles that block their goal-oriented motion. This helps PHOP agents reach their goals in situations where ORCA and ALAN agents cannot do so. In other environments, however, PHOP agents might be unable to achieve this. For PHOP agents to be able to reach their

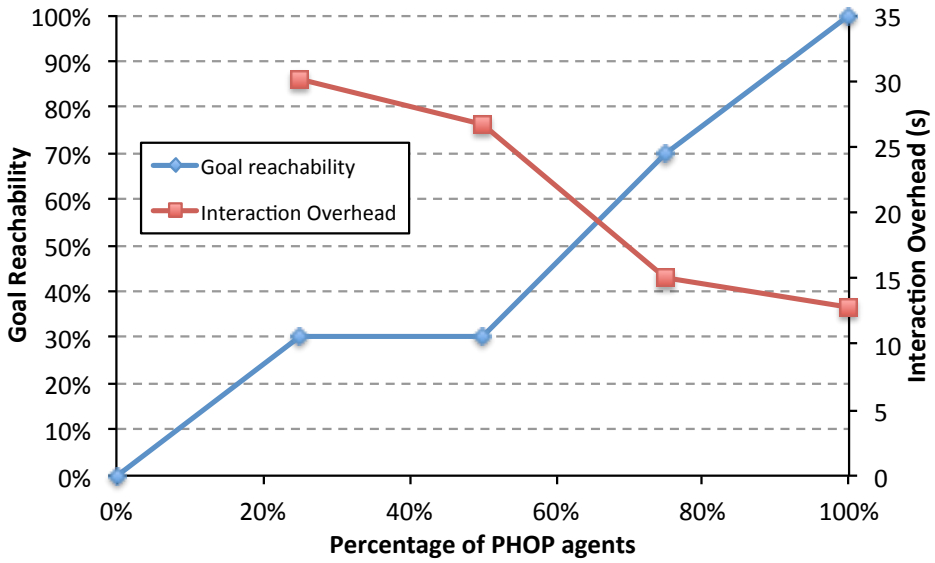


Figure 6.10: Interaction overhead for different percentages of PHOP agents, in the *Square* scenario. The blue line denotes the number of times, out of 10 iterations, when all agents reached their goals.

goals, that they can move around the obstacles in their paths. For this to occur, a necessary condition is that at least one of the trajectories generated by PHOP should be able to partially circumvent the obstacle. This means that the *T-Horizon* used by PHOP should be at least as large as this trajectory. If this is not the case, for example, if the side of the obstacle that faces the agent is too large with respect to the *T-Horizon* used, the agent will not be able to find an action that moves it closer to its goal and will get stuck in a local minima. A possible way to address this is to allow the agent to increase its *T-Horizon* based on its goal progress. However, as agents have a limited sensing range, this would only help until the *T-Horizon* intersects with the distance that the agents can sense.

A limitation of the both ALAN and PHOP is that each agent considers its nearby agents only as reactive dynamic obstacles. This fails to model a more cognitive aspect of the agents, such as their intended motions and their goals. In Chapter 7 we present a method that accounts for the cognitive aspect of the agents, allowing them to implicitly coordinate their motions and ultimately reduce their travel time.

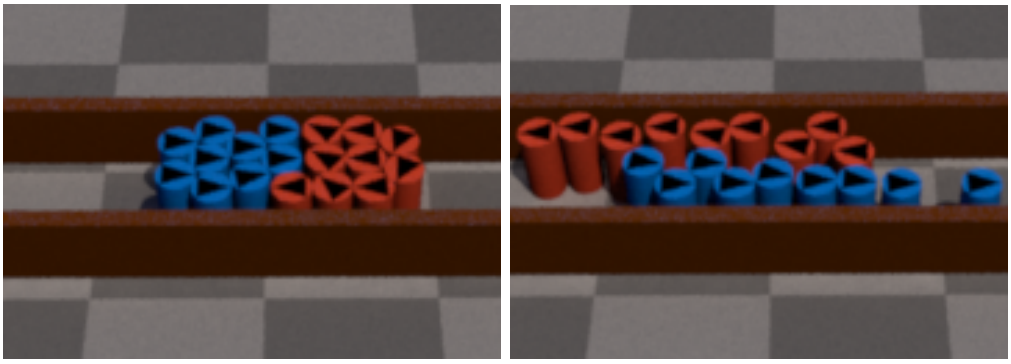
Chapter 7

Implicitly Coordinated Navigation (C-Nav)

In the approaches proposed in Chapters 5 and 6, each agent considers only the position and velocities of its neighbors when making motion decisions. However, in multi-agent navigation, the agents are not only dynamic obstacles in each other's paths, but can also be modeled as BDI agents [65]. A BDI model characterizes rational agents that have mental attitudes of beliefs, desires and intentions. Specifically, each navigating agent has an intended motion that it wants to keep throughout the simulation and a goal location. If an agent could account for the intended motion of its nearby agents, then it could use this information to choose a velocity in a way that benefits not only itself but also these neighboring agents.

In this chapter, we present *C-Nav* (short for Coordinated Navigation), a distributed approach to improve the global motion of agents in crowded environments by implicitly coordinating their local motions. This coordination is achieved using observations of the nearby agents' motion patterns and a limited one-way communication, allowing *C-Nav* to scale to hundreds of agents. With this approach, agents choose velocities that help their nearby agents to move to their goals, effectively improving the time-efficiency of the entire crowd.

As a motivating example, consider the navigation deadlock shown in Figure 7.1a. Here, two groups of agents try to move past each other in a narrow hallway. The agents



(a) ORCA (b) C-Nav

Figure 7.1: Two groups of 9 agents each move to the opposite side of a narrow corridor. (a) ORCA agents get stuck in the middle. (b) Using our *C-Nav* approach, agents create lanes in a decentralized manner.

navigate using a predictive collision-avoidance technique, but still end up getting stuck in congestion. *C-Nav* encourages coordination to emerge through agents' interactions by allowing agents to account for their neighbors' intended velocities during their planning. Figure 7.1b shows an example of such coordinated motion.

7.1 The *C-Nav* Approach

Agents using *C-Nav* broadcast information related to their intended motion to their nearby agents. An agent uses this information as well as the observed velocities and positions of its nearby agents to evaluate its motions and guide the action selection. Specifically, the agent can compare the observed velocities of the neighbors with their intended velocities, to determine how constrained is their individual motions, which allows the agent to take actions as to reduce these constraints. Further, the agent can determine which neighbors have a similar goal to its own and use this information to enable following behaviors which, while moving the agent to its goal, reduce the constraints imposed to other agents.

Algorithm 6 outlines *C-Nav*. For each agent that has not reached its goal, a new action (i.e., preferred velocity) is computed on average every 0.1 seconds (line 3). In each new update, the agent computes which of its neighbors move in a similar manner

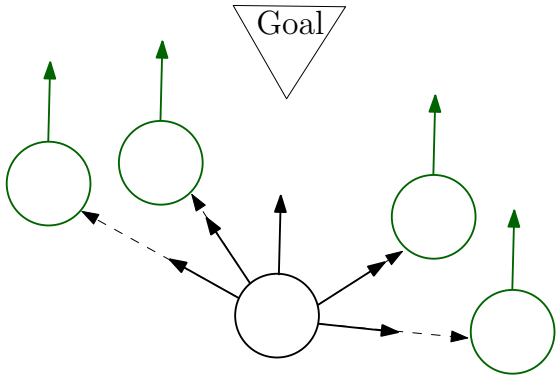


Figure 7.2: Neighborhood-based actions: follow a specific neighbor agent or the goal-oriented motion at maximum speed.

as itself and which neighbors are most constrained in their motions (line 4 and line 5, respectively), and uses this information to evaluate all of its actions (line 7). After this evaluation, the best action is selected (line 9). Finally, its intended motion is broadcasted to the agent's neighbors (line 10) and mapped to a collision-free velocity $\mathbf{v}^{\text{new}}$ via the ORCA framework (line 12) which is used to update the agent's position (line 13) and the cycle repeats ($\Delta t = 25\,\text{ms}$).

7.2 Agent neighborhood information

We consider three variants of *C-Nav*, which differ in what type of information is broadcasted as the agents' intended motion:

- Preferred Velocity: In this variant, agents broadcast only their current preferred velocities ($\mathbf{v}^{\text{pref}}$), which are subject to change based on the action selected, from the set of actions defined in Chapter 4.1.
- Goal Velocity: Agents broadcast only the velocities at which they intend to move toward their goals ($\mathbf{v}^{\text{goal}}$). These velocities do not change through the entire simulation.
- Preferred Velocity and Goal Velocity: Agents broadcast both their current preferred velocity and the goal velocity. This allows each agent to determine if a

Algorithm 6: The *C-Nav* framework for an agent

```
1: initialize simulation
2: while not at the goal do
3:   if UpdateAction(t) then
4:     compute most similar agents
5:     compute most constrained agents
6:     for all a ∈ Actions do
7:       R_a ← SimMotion(a)
8:     end for
9:     v^pref ← arg max_{a∈Actions} R_a
10:    broadcast ID and intended velocity to nearby agents
11:  end if
12:  v^new ← CollisionAvoidance(v^pref)
13:  p^t ← p^{t+1} + v^new · Δt
14: end while
```

neighbor's preferred velocity is aligned with its goal-oriented motion.

With information obtained by sensing (radii, positions and velocities) and via one-way communication (IDs and intended velocity) from all the neighbors within the sensing range, each agent evaluates the quality of the motion of its nearby agents. Specifically, the agent can determine which neighbors have a similar motion to its own and which ones are most constrained with respect to their intended motions.

7.2.1 Motion similarity

Each agent evaluates the similarity between the motion of its neighbors and its own (see Algorithm 7), identifying those neighbors that are moving faster than itself and in a similar direction. By following such neighbors, the time-efficiency of the set of agent can be increased.

To evaluate this similarity, the agents follows a two-step process. In the first step, each agent compares the intended velocity of a neighbor to its own goal oriented motion. Based on this comparison, the agent can filter those neighbors whose intended velocity

is not in the direction of the agent's goal. In the second step, the agent compares a neighbor's current velocity with its own velocity, which allows it to quantify the similarity of their motions.

First step: Intended motion similarity

Based on the type of information that the agents communicate, an agent can determine, for each neighbor, a *temporary* similarity (based on their preferred velocity), a *long-term* similarity (based on their goal velocities), or a combination of both (using the preferred and goal velocities). In all cases, this similarity is evaluated by computing the dot product between the neighbor's intended motion and the agent's goal-oriented vector. If this dot product is positive, then the neighbor is classified as having a similar intended motion with the agent (regardless of the actual value of the dot product).

Temporary similarity. In case agents communicate their preferred velocities, it is possible to compute a *temporary* similarity between the agents. Specifically, each agent determines which neighbors want to move in a similar direction by comparing their current preferred velocity to its own goal direction . This type of similarity is subject to change as the neighbor can select a different preferred velocity at any time, and it is not necessarily an indication of a long-term goal similarity.

Long-term similarity. In case agents communicate their goal velocities, it is possible to compute a *long-term* similarity between the agents. Here, each agent compares a goal-oriented vector with its neighbors' goal velocities. This type of similarity only needs to be computed once per each neighbor, as its goal velocity is not subject to change (unlike the preferred velocity).

Simultaneous temporary and long term similarity. In case agents communicate both their preferred and goal velocities, it is possible to compute a combination of *temporary* and *long term* similarity between the agents. To do this, each agent first compares the two vectors broadcasted by each neighbor (its goal velocity with its preferred velocity) to determine if its preferred velocity moves the neighbor to its goal. If this is the case (i.e., their dot product is > 0), then the preferred velocity is further analyzed to determined its similarity with the agent' goal vector, as it is done in the *temporary* similarity case.

Second step: Observed motion similarity

After considering each neighbor's intended velocity, the agent quantifies the similarity by computing the dot product between the observed velocity of each of its neighbors and a unitary vector to its own goal. The value of this dot product corresponds to the final similarity value, $SimVal$, which is computed for each neighbor and is used to create a raking of those neighbors that have the most similar motion with respect to the agent.

Algorithm 7 shows how each agent determines its similar neighbors and computes their similarity values, when agents only broadcast their preferred velocities as intended motions. Here, the preferred velocities of the nearby agents are used to first select neighbors that have intended motions in the same direction as the agent i (line 4). The actual velocity of each of these neighbors is then compared to i's goal-oriented vector to quantify the similarity between the agents (line 5). The similarity value for each neighbor j, $SimVal_j$, measures how closely related are the agent's goal vector and the neighbor's current observed velocity. Algorithm 7 sorts these similarity values in a descending order and returns the corresponding list of the neighbors' indices.

If agents instead communicate their goal velocities, then line 4 of Algorithm 7 is replaced by the following line:

$$\text{if}(\mathbf{v}_j^{\text{goal}} \cdot \frac{\mathbf{g}_i - \mathbf{p}_i}{\|\mathbf{g}_i - \mathbf{p}_i\|} > 0) \tag{7.1}$$

This reflects the comparison with the neighbor's goal velocity instead of its preferred velocity. If agents communicate both their preferred and goal velocities, Algorithm 7 is modified to include, between lines 3 and 4, the following condition statement:

$$\text{if}(\mathbf{v}_j^{\text{pref}} \cdot \mathbf{v}_j^{\text{goal}} > 0) \tag{7.2}$$

The extra information communicated in this case allows agents to first filter the preferred velocities of their neighbors based on their goal velocities. If the dot product between both vectors is positive, its preferred velocity is considered an indication of its intended motion.

Algorithm 7: Compute most similar neighbors of i

1: **Input:** list of neighbors NN_i
2: **Output:** Sim_{rank}, list of indices of the most similar neighbors
3: **for all** $j \in NN_i$ **do**
4: **if** $\mathbf{v}_j^{\text{pref}} \cdot \frac{\mathbf{g}_i - \mathbf{p}_i}{\|\mathbf{g}_i - \mathbf{p}_i\|} > 0$ **then**
5: $SimVal_j \leftarrow \mathbf{v}_j^{\text{new}} \cdot \frac{\mathbf{g}_i - \mathbf{p}_i}{\|\mathbf{g}_i - \mathbf{p}_i\|}$
6: **end if**
7: **end for**
8: $Sim_{rank} \leftarrow Sort(SimVal)$

Neighborhood-based actions

Once an agent knows how similar is the motion of each of its neighbors, it can choose a velocity with maximum speed towards one of these neighbors (Figure 7.2), in addition to the goal-oriented motion. These following actions, unlike the ones in Figure 4.1, do not have a fixed angle with respect to the agent's goal, as they depend on the position of individual neighbors.

When using one of these following behaviors, agents show emergent lane formations which efficiently move them to their goals (Figure 7.1b), while minimizing the constraints that they impose into other agents.

7.2.2 Constrained neighborhood motion

Agents can also use the information broadcasted by their neighbors to evaluate how constrained is their motion and, thus, determine agents that are more likely to slow down the overall progress of the crowd. By explicitly aiming at reducing the constraints of these neighbors, the global time-efficiency of the system increases.

To evaluate how constrained is the motion of each neighbor, each agent compares a neighbor's intended motion with its observed velocity. The larger the difference, the more likely it is that its motion is impeded. To avoid circular dependencies which can give rise to deadlocks, each agent only considers neighbors that are closer than itself to its goal. This ensures that no two agents with the same goal will simultaneously defer to each other. Figure 7.12 shows, for a given agent i, which neighbors are considered

when evaluating motion constraints.

Similar to the case of agent similarity, we can determine a neighbor's constraints with respect to either its current preferred velocity, its goal velocity, or with respect to its preferred velocity as long as it is similar to its goal velocity.

Constraints with respect to the preferred velocity. In case agents broadcast only their preferred velocities, each agent can estimate a neighbor's constraints based on the difference between this velocity and its observed velocity (see Figure 7.4).

Constraints with respect to the goal velocity. In case agents broadcast only their goal velocities, an agent can estimate a neighbor's constraints based on the difference between its goal-oriented and observed velocities. This gives the agent an indication of how constrained is the goal-directed motion of the neighbor, regardless of the preferred velocity that it might be currently performing.

Constraints with respect to the preferred velocity, as long as it is similar to the goal velocity. In case agents communicate both their preferred and goal velocities, the agent can compute simultaneously the *preferred velocity* constraints as well as the *goal velocity* constraints for each neighbor. As in the similarity case, the idea is to consider the constraints on the neighbor's preferred velocity as long as this is similar to the goal velocity of the neighbor. Specifically, the agent compares the neighbor's goal velocity with its preferred velocity. If these velocities are similar (i.e., their dot product is > 0), the preferred velocity is further analyzed to determine the preferred velocity constraint, as done in the first case.

Algorithm 8 shows how each agent evaluates the constraints on the motion of its neighbors, when agents broadcast their preferred velocities. For each neighbor, the agent first determines its distance to the agent's goal (line 4). If the neighbor is closer than the agent to its goal, then it computes its constraints based on its intended motion (line 5). The agent keeps a list C which quantifies the constraints of each neighbor. After all neighbors have been evaluated, Algorithm 8 sorts C in descending order, and returns a list C_{rank} of the indices of the sorted neighbors (line 8). The agent uses this information when evaluating each of its available actions.

If agents instead communicate their goal velocities, then the constraints of a neighbor are evaluated with respect to this velocity instead of its preferred velocity. Hence, line

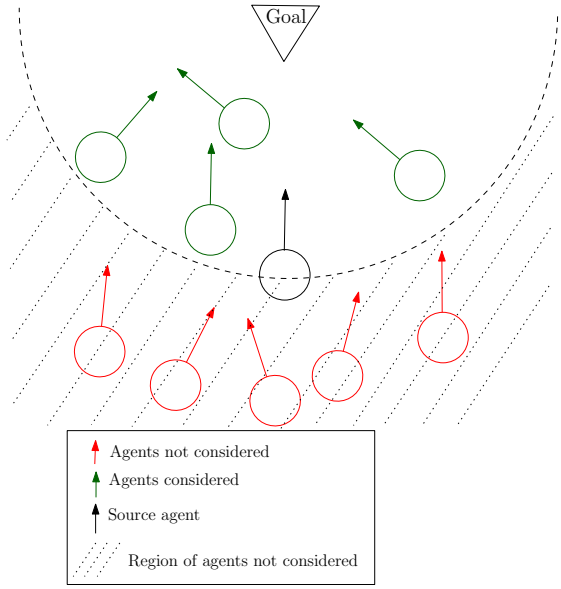


Figure 7.3: Neighbors considered in the computation of constraints. Considering only agents closer to the goal ensures that no two agents with the same goal will simultaneously defer to each other.

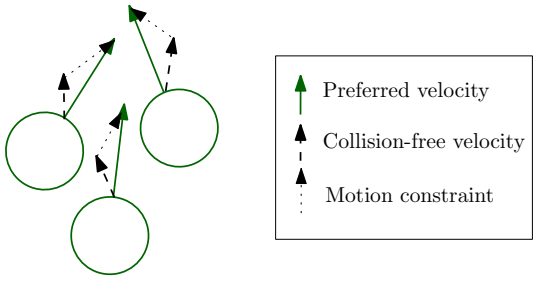


Figure 7.4: Computation of constraints based on the preferred and observed velocities of the neighbors.

5 of Algorithm 7 is replaced by the following line:

$$C_j \leftarrow \| \mathbf{v}_j^{\text{goal}} - \mathbf{v}_j^{\text{new}} \| \tag{7.3}$$

If agents communicate both their preferred velocities and their goal velocities, Algorithm 8 is modified to include, in line 4, an extra condition for evaluating the constraints of a neighbor j:

$$\text{if } ((\mathbf{v}_j^{\text{pref}} \cdot \mathbf{v}_j^{\text{goal}} > 0) \text{ and } (\|\mathbf{g}_i - \mathbf{p}_j\| < \|\mathbf{g}_i - \mathbf{p}_i\|)) \tag{7.4}$$

The extra information communicated in this case allows agents to consider only the constraints of neighbors that are closer than itself to its goal, and whose preferred velocity moves the neighbor to its goal. Hence, if the dot product between both vectors is positive, its preferred velocity is considered indication of its intended motion.

Algorithm 8: Compute most constrained neighbors of i

1: **Input:** list of neighbors NN_i
2: **Output:** C_{rank}, list of indices of the most constrained neighbors
3: **for all** $j \in NN_i$ **do**
4: **if** $\|\mathbf{g}_i - \mathbf{p}_j\| < \|\mathbf{g}_i - \mathbf{p}_i\|$ **then**
5: $C_j \leftarrow \| \mathbf{v}_j^{\text{pref}} - \mathbf{v}_j^{\text{new}} \|$
6: **end if**
7: **end for**
8: $C_{rank} \leftarrow Sort(C)$

Once the agent computes a ranking of the most similar and most constrained neighbors, it can use this information to bias the action selection towards velocities that, on one hand, move the agent closer to its goal in an efficient manner while, on the other, help nearby agents to move according to their intended motions.

7.3 Action evaluation and selection

Agents can choose a preferred velocity from two sets of actions, an agent-based (Figure 4.1, Section 4.1) and a neighborhood-based (Figure 7.2) action set. In the latter,

agents consider up to s neighbors ($0 \leq s \leq |NN|$). To estimate the fitness of the actions, the agent simulates each action for a number of timesteps and evaluates two metrics: its potential progress towards its goal, and its effect in the motion of its k most constrained neighbors ($0 \leq k \leq |NN|$). Algorithm 9 outlines this procedure.

Motion simulation. As a first step towards the selection of an action, an agent simulates the evolution of its neighborhood dynamics (line 4), that is, it updates the velocities and positions of itself and its neighbors for each timestep within a given time horizon T (line 3), for each possible action. Two timesteps is the minimum time horizon required to observe the effect of the agent's chosen motion.

Algorithm 9: SimMotion(a) for agent i

1: **Input:** integer $a \in Actions$, list of neighbors NN_i
2: **Output:** R_a, estimated value of action a
3: **for** $t = 0, \ldots, T-1$ **do**
4: simulate evolution of neighborhood dynamics
5: **if** $t > 0$ **then**
6: **for all** $j \in NN_i$ **do**
7: **if** $rank(j \in C_{rank}) < k$ **then**
8: $\mathcal{R}_a^{polite} \leftarrow \mathcal{R}_a^{polite} + v_i^{\max} - \| \mathbf{v}_j^{\text{pref}} - \mathbf{v}_j^{\text{new}} \|$
9: **end if**
10: **end for**
11: **end if**
12: $\mathcal{R}_a^{goal} \leftarrow \mathcal{R}_a^{goal} + \mathbf{v}_i^{\text{new}} \cdot \frac{\mathbf{g}_i - \mathbf{p}_i}{\|\mathbf{g}_i - \mathbf{p}_i\|}$
13: **end for**
14: $\mathcal{R}_a^{goal} \leftarrow \frac{\mathcal{R}_a^{goal}}{T \cdot v_i^{\max}}, \mathcal{R}_a^{polite} \leftarrow \frac{\mathcal{R}_a^{polite}}{(T-1) \cdot k \cdot v_i^{\max}}$
15: $\mathcal{R}_a \leftarrow (1-\gamma) \cdot \mathcal{R}_a^{goal} + \gamma \cdot \mathcal{R}_a^{polite}$

It should be noted here that in very crowded areas, agents often have no control over their motions, as they are being pushed by other agents in order to avoid collisions. Hence, simulating the dynamics of all the agent's neighbors often results in the same velocity for all simulated actions. This does not help, because the agent would not be able to select a velocity that improves the motion of its most constrained neighbors.

Therefore, the agent considers in its simulation only the neighbors that are closer to its goal than itself, 'ignoring' the agents that are behind it. Even if the best valued action is not currently allowed, we expect that the neighboring agents will eventually try to relax the constraints that they impose on the agent.

Neighborhood influence. After simulating a specific action for the given time horizon, the agent can estimate how this action affects each of its k most constrained neighbors. It computes this based on the difference between j's predicted collision-free velocity $\mathbf{v}_j^{\text{new}}$ and its communicated intended motion which, in this case, corresponds to its preferred velocity $\mathbf{v}_j^{\text{pref}}$ (line 8). If agents instead communicate only their goal velocities, line 8 would be modified accordingly.

Motion evaluation. To decide what motion to perform next, the agent aims at minimizing the amount of constraints imposed by its neighbors, while also ensuring progress towards its goal.

Following the reward function template proposed in Chapter 4.2, we use a function that linearly combines a *goal-oriented* component, and a *constrained-reduction* component (Eq. 4.1). Each component has an upper bound of 1 and a lower bound of -1 and is weighted by the *coordination-factor* γ. We include the reward function here for convenience:

$$\mathcal{R}_a = (1-\gamma)\cdot\mathcal{R}_a^{goal} + \gamma\cdot\mathcal{R}_a^{polite} \tag{7.5}$$

As in the previous chapters, the *goal-oriented* component $\mathcal{R}_a^{goal}$ computes, for each timestep in the time horizon, the scalar product of the collision-free velocity $\mathbf{v}^{\text{new}}$ of the agent with the normalized vector which points from the position $\mathbf{p}$ of the agent to its goal $\mathbf{g}$. This component encourages preferred velocities that lead the agent as quickly as possible to its goal. Formally:

$$\mathcal{R}_a^{goal} = \frac{\sum_{t=0}^{T-1}\left(\mathbf{v}_i^{\text{new}}\cdot\frac{\mathbf{g}_i-\mathbf{p}_i}{\|\mathbf{g}_i-\mathbf{p}_i\|}\right)}{T\cdot v_i^{\max}} \tag{7.6}$$

The *politeness* component $\mathcal{R}_a^{polite}$ is different from its previous definition in Chapters 5 and 6, given that *C-Nav* agents are able to determine more accurately the effect that each action has in their neighbors. Here, the *politeness* component averages the

amount of constraints that action a would introduce in an agent's k most constrained neighbors. This component promotes preferred velocities that do not introduce constraints into these k agents. More formally:

$$\mathcal{R}_a^{polite} = \frac{\sum_{t=1}^{T-1} \sum_{j \in C_{rank}} \left(\upsilon_i^{\max} - \| \mathbf{v}_j^{\text{pref}} - \mathbf{v}_j^{\text{new}} \| \right)}{(T-1) \cdot k \cdot \upsilon_i^{\max}} \tag{7.7}$$

If an agent only aims at maximizing $\mathcal{R}_a^{goal}$, its behavior would be selfish and it would not consider the constraints that its actions impose on its most constrained neighbors. On the other hand, if the agent only tries to maximize $\mathcal{R}_a^{polite}$, it might have no incentive to move towards its goal, which means it might never reach it. Therefore, by maximizing a combination of both components, the agent implicitly coordinates its goal-oriented motion with that of its neighbors, resulting in lower travel times for all agents.

7.4 Experiments

We evaluated *C-Nav* on three different scenarios using a varying number of agents (see Figure 7.5). Each result corresponds to the average over 30 simulations (see `http://motion.cs.umn.edu/r/CNAV/` for videos). The scenarios are as follows:

- *Bidirectional:* 18 agents are clustered in two groups that move to the opposite side of a narrow corridor formed by two walls.
- *Circle:* 128 agents are placed along the circumference of a circle and must reach their antipodal positions.
- *Crowd:* 300 agents are randomly placed in a densely populated area and are given random goals.
- *Congested:* 32 agents are placed very close to the narrow exit of an open hallway and must escape the hallway through this exit.
- *3-Exit:* Ten agents exit a room with three exits. The central exit is closest to most agents, but the congestion will slow down the agents if all of them go through the same exit.

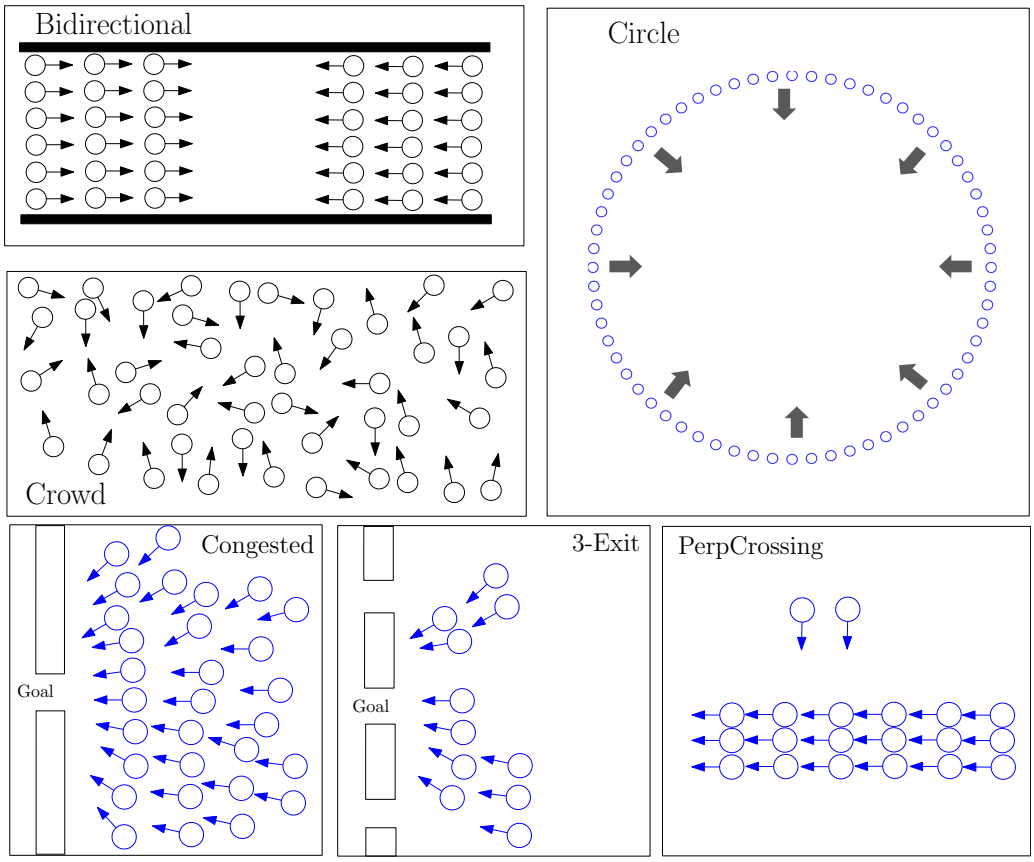


Figure 7.5: Scenarios. *Bidirectional*: two groups of agents move to the opposite side of a narrow corridor. *Crowd*: agents' initial and goal positions are placed randomly in a small area. *Circle*: agents move to their antipodal positions. *Congested*: agents try to exit an area through a small doorway. *3-Exit*: agents have three homotopic paths that they can take to reach their goals, while congestion forms in the closest goal-directed path. *PerpCrossing*: the goal paths of two agents orthogonally interesect a crowd of 24 agents.

- *PerpCrossing:* Two agents have orthogonally intersecting paths with a crowd of 24 agents.

Using the interaction overhead metric (see Section 4.3), we compared *C-Nav* to vanilla ORCA (greedy goal-oriented motion) and the ALAN approach (see Chapter 5). Although these main results are with respect to the *C-Nav* variant where agents communicate only their preferred velocities, we also evaluated the interaction overhead of the other variants (see Section 7.2). In all our experiments, we used ORCA's default settings for the agent's radii (0.5 m), sensing range (15 m) and maximum number of agents sensed ($|NN|$=10). We set T=2 timesteps, $v^{\max}$=1.5 m/s, γ=0.8, k=3 and s=3. The timestep duration, Δt, is set to 25 ms. The number of timesteps T was chosen because it is the minimum needed to observe the effects of the motion and it produces the best results, though even with larger values our approach still outperforms both ALAN and ORCA. All simulations ran in real time for all evaluated methods.

7.4.1 Results

We evaluated the interaction overhead time in all scenarios in Figure 7.5. Results can be seen in Figure 7.6. The interaction overhead of *C-Nav* is significantly lower than ORCA and ALAN in all cases, which indicates that by considering information about their neighborhood, agents can coordinate their motion and improve their time-efficiency. Even in scenarios where agents are constrained by other agents and static obstacles (such as in the Bidirectional scenario), *C-Nav* is able to significantly improve the time-efficiency of the agents. In terms of qualitative results, we observe emergent behavior in the Bidirectional and Circle scenario, where agents going in the same direction form lanes. Such lanes reduce the constraints in other agents leading to more efficient simulations.

On the other hand, agents using ALAN outperform ORCA agents in the Bidirectional, Congested, 3-Exit and PerpCrossing scenarios. However, in scenarios with more than 100 agents (as in the Circle and Crowd scenarios), ALAN cannot outperform ORCA. In general, the unnecessary exploration performed by ALAN agents introduces noise in the navigation of the crowd, which impairs their time-efficiency.

We also evaluated the interaction overhead of agents using the different variants of

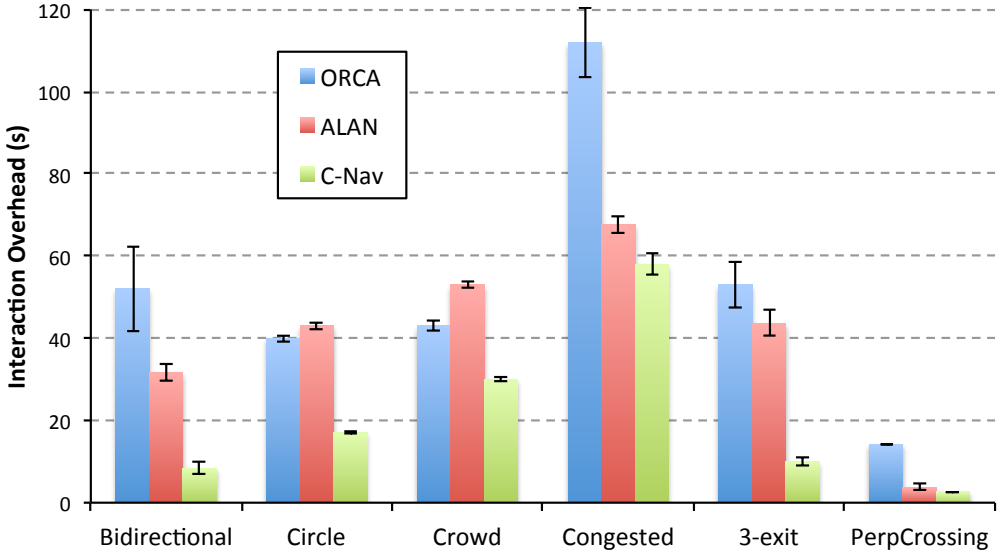


Figure 7.6: Performance comparison between ORCA, ALAN and the *C-Nav* approach. In all scenarios, agents using our coordination approach have the lowest overhead times. The error bars correspond to the standard error of the mean.

C-Nav that differ in the type of intended motion communicated (see Section 7.2): the preferred velocity ($Vpref$), the goal velocity ($Vgoal$), or both the preferred and goal velocities ($Vpref + Vgoal$). We also compared these variations of *C-Nav* with the case where no information is communicated ($None$), in which case the agents can only use the observed velocities of their neighbors as their intended motions. This last case is similar to the PHOP approach of Chapter 6.

The results (Figure 7.7) show that, in almost all cases, communicating at least some information related to their intended motions help agents improve their time-efficiency as compared to the case with no communication. The only exception can be seen in the 3-Exit scenario where agents communicate their goal velocities. In this case, the direct path to goal of the agents is blocked by obstacles, which means that most of the time, agents will not be able to follow their goal-oriented velocities, but instead must find a way around through one of the three exits. This means that, once agents reach the walls, the goal velocity is not a good indicative of their intended motion compared to, for example, the preferred velocity or even the observed velocities (which are usually different from their goal velocities until the agents move through the exits).

7.5 Analysis

In this section, we analyze various aspects of the *C-Nav* approach. We begin by analyzing the role of *C-Nav* in minimizing the total travel time of the agents. Then, we analyze its runtime complexity as well as its robustness to the number of agents in the environment in the *Circle* and *Bidirectional* scenarios. We then show the conditions under which agents using *C-Nav* agents are less likely to found themselves in livelocks. We also perform sensitivity analysis of *C-Nav* with respect to the value of the coordination factor (γ) used, in all evaluated scenarios. Finally, we evaluate the sensibility of the results with respect to the number k of neighbors considered by each agent to reduce their constraints.

C-Nav agents redefine the concept of 'polite' behavior, as compared to ALAN (Chapter 5) and PHOP (Chapter 6), to explicitly consider their nearby agents' intended motions. With this extra piece of information, each agent determines the optimal action that maximizes both its own goal progress as well as its neighbors'. Hence, C-Nav agents

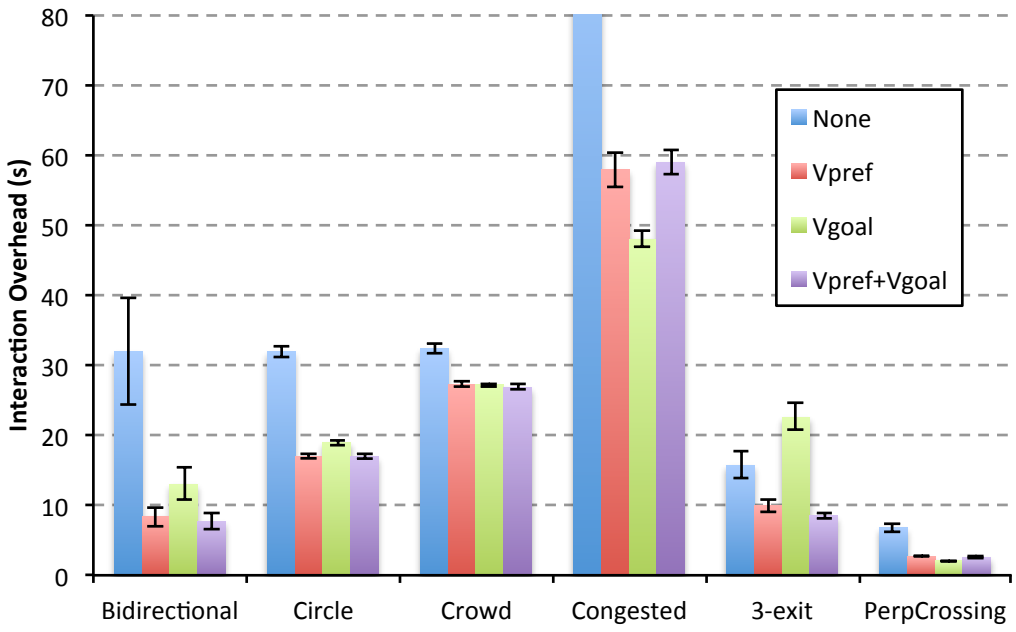


Figure 7.7: Performance comparison between the different variations of *C-Nav*, where agents can: not communicate any motion information (*None*), communicate only their preferred velocities ($Vpref$), only their goal velocities ($Vgoal$) or both ($Vpref+Vgoal$). Results show that, in almost all cases, communicating motion-related information helps to reduce the interaction overhead of the system of agents. The error bars correspond to the standard error of the mean.

take actions that benefit their entire neighborhood. This 'polite' behavior minimizes the occurrence of congestion which, in long term, minimizes the travel time of all agents.

7.5.1 Runtime Complexity

Similar to PHOP, the runtime performance of *C-Nav* is also dominated by its action-selection routine (Alg. 9, lines 3-13), as it is necessary to simulate the motion of the agent and its neighbors for each possible action. The main difference is that the use of information of the motion of the neighbors allows *C-Nav* to achieve good performance with a small time horizon ($T = 2$ in all experiments), which means that the time horizon does not play a major role in the computational complexity of *C-Nav*. Therefore, assuming there are n neighbors and q actions to be evaluated, the runtime complexity of *C-Nav* is $\mathcal{O}(q \cdot n^2)$ per agent.

The complexity added by the one-way communication in *C-Nav* is negligible because a single message is communicated regardless of the number of neighbors. Therefore, this only requires a constant and small amount of resources (each message is formed by either one or two 2D vectors of real numbers).

7.6 Theoretical Analysis

We focus our theoretical analysis on showing the conditions under which *C-Nav* agents can avoid livelocks.

A livelock corresponds to executing a series of repeated motions that do not move the agent to its goal. In *C-Nav*, a livelock would occur if two or more agents repeatedly switch from goal-oriented motions to deferent motions that move the agents away from their goals, which would prevent the progress of the agents. We show that the probability of livelocks occurring approaches 0 as the value of γ (Eq. 4.1) asymptotically approaches 1, in scenarios where all agents share the same goal (e.g., the Congested scenario in Figure 7.5).

For the purpose of this analysis, assume that after an agent reaches the goal, it is removed from the environment. Assume also that A_α corresponds to the agent that, at any given time, is closest to the goal.

Lemma 1. *At any time, A_α is able to choose an action that maximizes its progress to the goal, without deferring to the motion of other agents.*

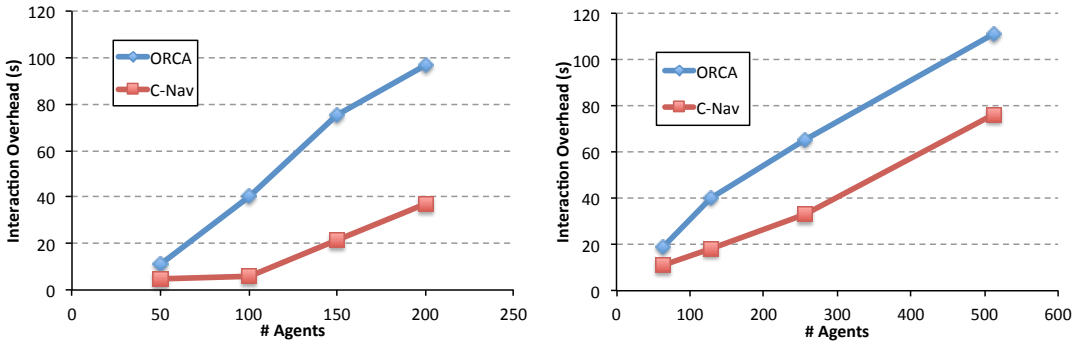
Proof: The proof follows from the fact that an agent only accounts for neighbors that are closer than itself to the goal for the evaluation of constraints and for motion simulation purposes (Section 7.3). As A_α 'ignores' agents coming from behind, and there are no agents closer than A_α to the goal, then for all of its actions a, $\mathcal{R}_a^c = 0$, which means that A_α will optimize only on the value of the action's goal progress $\mathcal{R}_a^g$. To ensure that A_α has incentive to reach the goal, $\mathcal{R}_a^g$ must be greater than zero for at least one of the actions. Hence, as long as $\gamma < 1$, A_α will choose the action with the collision-free velocity that maximizes its progress to the goal. ■

Lemma 2. *Any agent A_i (where $A_i \neq A_\alpha$ and $A_\alpha \in \mathcal{N}(i)$) can choose an action a' that moves it backwards from its goal, that does not introduce constraints into A_α.*

Proof: To allow A_α to maximize its progress to the goal, A_i should always choose an action a' that does not introduce constraints into A_α ($a' = \arg\max_{a \in Actions} \mathcal{R}_a^c$). Such an action a' always exists; in the worst case, a' corresponds to the preferred velocity backwards from the goal. As A_i 'ignores' agents coming from behind (Section 7.3), it assumes it can freely move in this direction. This backwards velocity moves A_i away from A_α, minimizing the difference between A_α's intended velocity and its collision-free velocity. For A_i to choose this action, it must hold that $a' = \arg\max_{a \in Actions} \mathcal{R}$. Although there is no single value of γ that guarantees this condition for all possible values of $\mathcal{R}_a^c$ and $\mathcal{R}_a^g$, the probability that A_i will choose action a' approaches 1 as $\gamma \to 1$ (see Eq.4.1). As each agent A_i chooses a', it will not introduce constraints into A_α's motion, which will be able to move as if it was the only agent in the system, and it will be able to reach the goal. Once this occurs, another agent takes the role of A_α, until all agents reach the goal. ■

From Lemmas 1 and 2, the following Theorem holds:

Theorem 4. *The probability of livelocks occurring in C-Nav, in environments with a single goal, approaches 0 as γ asymptotically approaches 1. Under these conditions, the probability that all agents reach the goal approaches 1.*



(a) Bidirectional (b) Circle

Figure 7.8: Scalability results in the Bidirectional and Circle scenarios, in terms of interaction overhead time. In Bidirectional, the number of agents varied from 50 to 200. In the Circle, the number of agents varies from 64 to 512.

7.6.1 Scalability

We analyzed the scalability of our approach in the Bidirectional and Circle scenarios by varying the number of simulated agents. The results, depicted in Figure 7.8 show that, in both scenarios, the overhead times *C-Nav* increase linearly as more agents are added. At the same time, the overhead time introduced by each added agent in the system is not larger in our approach than in ORCA.

7.6.2 Effect of the coordination-factor (γ)

We evaluated how the balance between the *goal-oriented* and the *politeness* components of our reward function (Eq. 7.5) controlled by the coordination-factor γ, affects the performance of our *C-Nav* approach in the Bidirectional, Congested and Circle scenarios. Note that we do not evaluate the interaction overhead while using a γ value of 1, as with this value the agents have no motivation to move towards their goal.

Bidirectional. The first thing we note from the results in the Bidirectional scenario (Figure 7.9) is the noisy interaction overhead times obtained for γ values smaller than 0.7, reflected by a large error of the means. This is a specially complex scenario where agents are constrained by neighbors coming in the opposite direction, as well as the side walls which limit the ability of the agents to find alternative goal-oriented motions. As

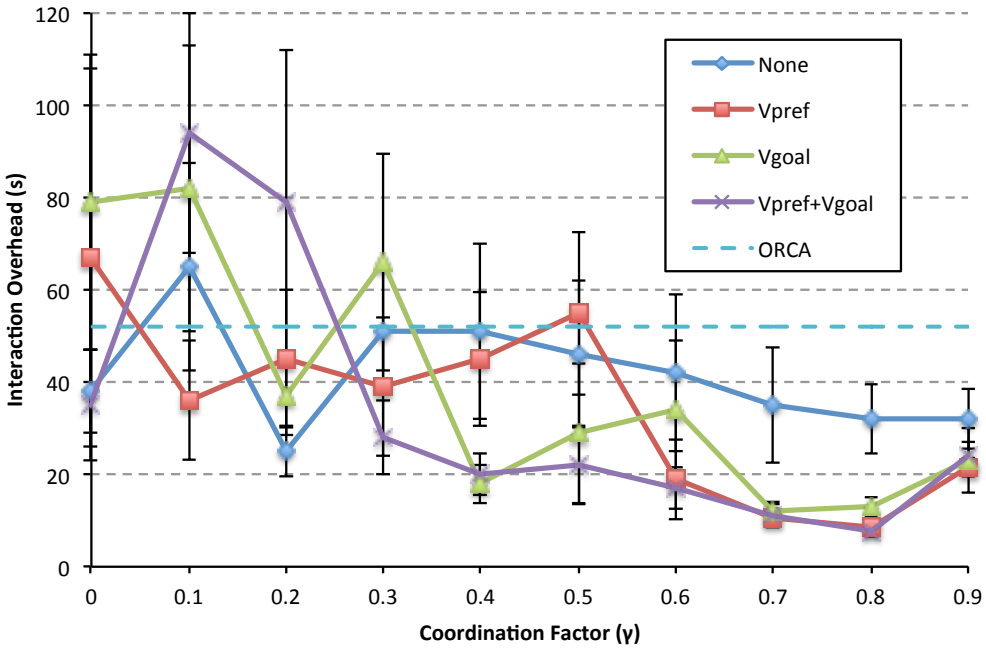


Figure 7.9: Performance of *C-Nav* agents in the Bidirectional scenario, with different values of the coordination-factor γ.

agents give more weight to their own goal progress (lower values of γ), they are more likely to get stuck in contention with agents coming from the other group. This situation forces one group of agents to push the other group until their reach their goals, clearing the path for the pushed agents.

Agents in this scenario benefit from a careful balance between goal-oriented and politeness behaviors. They need to be polite enough to make space for agents coming in the opposite direction, while also trying to find a motion that moves them to their goals. The behavior shown is Figure 7.1b is an example of this careful balance. The best results are obtained with γ values between 0.7 and 0.9, which represent a large weight in the politeness component of the reward function (Eq. 7.5). However, more politeness does not always translate to better performance in this scenario. Too much politeness (γ=0.9) means that the agents' action selection is almost entirely biased to favor their neighbor's progress. In this case, the agents' lack of 'aggressiveness' translates into longer times to coordinate their motions, as compared, for example, to the faster coordination observed with γ=0.8.

Congested. The Congested scenario has walls around the only goal, represented by a small opening between the walls. The results, shown in Figure 7.10, indicate that using values of γ smaller than 0.5 produces more contention close to the goal, as agents prioritize their own goal progress to their neighbors'. On the other hand, values of γ larger than 0.5 significantly reduce the interaction overhead times for all *C-Nav* variants where agents communicate their intended motions. In fact, results indicate that, up to γ=0.9, more politeness is always preferable (unlike in the Bidirectional case). As the value of γ approaches 1, agents are more likely to always defer to the motion of agents closer to the only goal. In this scenario, this creates a priority for reaching the goal, where no two agents will defer to each other at the same time. Ultimately, this translates into better overall performance.

We can also observe that the results obtained in the case of no communication are not strongly dependent of the value of γ used. This is because agents cannot predict large deviations from the 'intended' velocities (which, in this case, are the observed velocities) in a small number of timesteps used in the time horizon. In other words, agents are not likely to go through abrupt changes in velocity for the predicted time horizon, which minimizes the effect of the politeness component in the action choice.

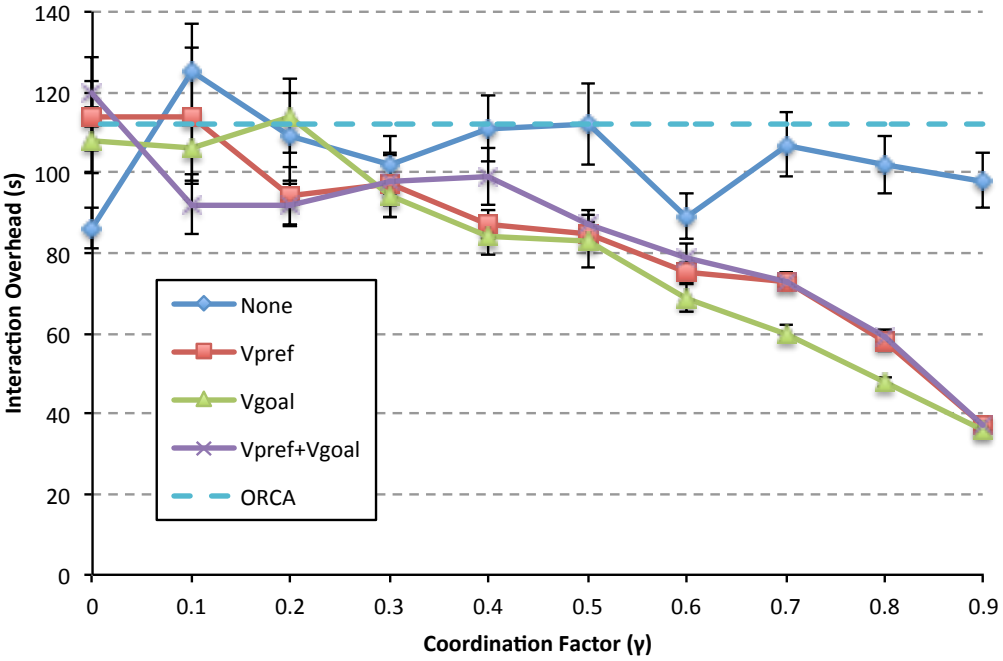


Figure 7.10: Performance of *C-Nav* agents in the Congested scenario, with different values of the coordination-factor γ.

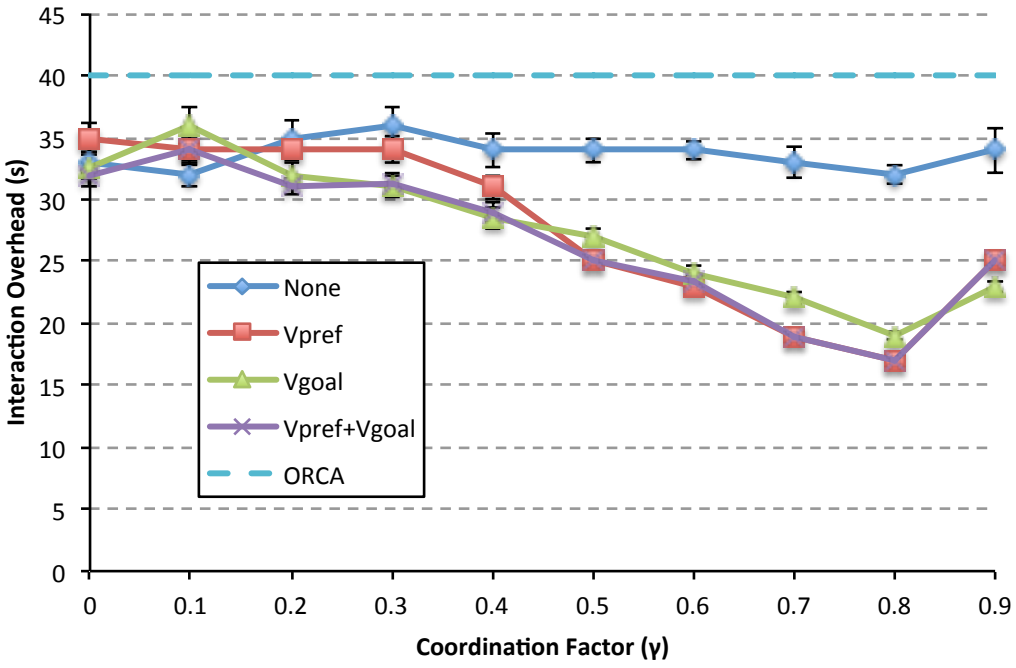


Figure 7.11: Performance of *C-Nav* agents in the Circle scenario, with different values of the coordination-factor γ.

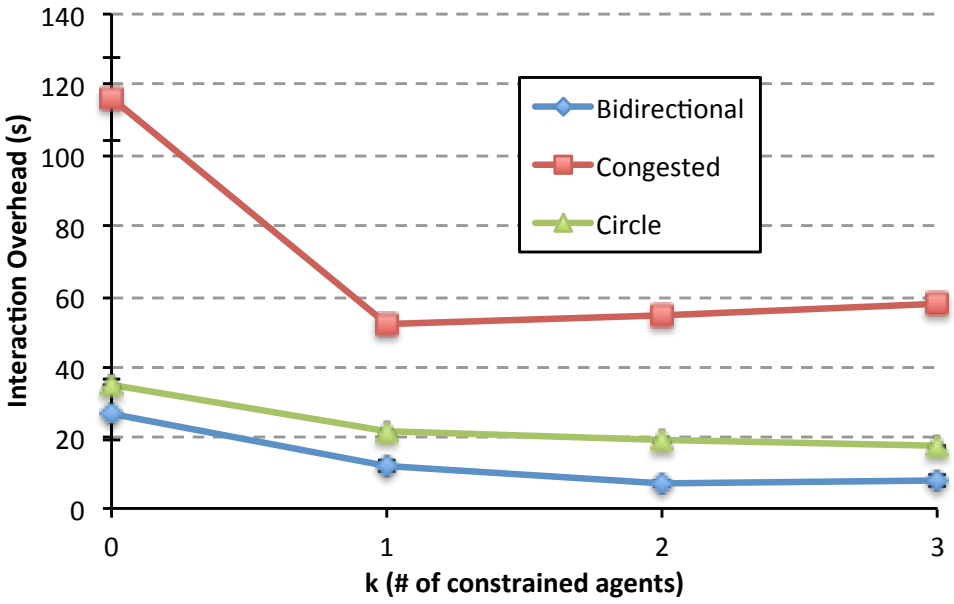


Figure 7.12: Interaction overhead time as a function of the number k of constrained neighbors.

Circle. Results from the Circle scenario (see Figure 7.11) show that there is much less noise in the interaction overhead, compared to the Bidirectional and Congested scenarios, for all evaluated values of γ. This is due to the absence of static obstacles that limited the agents' goal path choices in the previous cases. Because of this, agents are able to take advantage of the open space to move to their goals. Similar to the Bidirectional and Congested scenarios, the best performance in terms of interaction overhead is obtained with large values of γ (between 0.5 and 0.8). Note, however, that very large values of γ (0.9) break this performance trend and result in larger interaction overhead than with lower values of γ. This case is similar to the one analyzed in the Bidirectional scenario, where being too polite negatively affects the travel time of all agents. In this case, given that agents can mutually defer to one another, their extremely polite behavior translates into longer deviations from the straight goal-oriented path for each agent.

7.6.3 Effect of number of constrained neighbors k

We also evaluated how the number of constrained neighbors (k) in the politeness component of the reward function (Eq. 7.7) affects the performance of our approach. In general, the interaction overhead time decreases while k increases (Figure 7.12).

In the Bidirectional and Circle scenarios, the performance improves as each agent evaluates the constraints of more neighbors. We note, though, that in all of our experiments, considering more than 3 neighbors does not lead to any significant performance improvement. As agents account for more neighbors upon computing a new velocity, their motion becomes more coordinated and the travel time of the entire system of agents is reduced. An exception from this can seen in the Congested scenario, when increasing k from 1 to 3 slightly increases the interaction overhead. Although this difference is barely quantifiable, it is still statistically significant. This implies that considering the constraints of only the single most constrained neighbor produces the best performance. In this scenario, better performance is achieved when agents defer to the motion of neighbors closer to the goal. Considering more than one neighbor ($k > 1$), however, reduces the influence of a specific neighbor's motion constraints in the reward function of the agent, as it is averaged with the other $k - 1$ neighbors (Eq. 7.7). This creates situations in which the agent prioritizes its own goal-progress versus the average progress of its k constrained neighbors, where considering only one neighbor avoids this issue.

7.7 Limitations of C-Nav

C-Nav has some limitations that arise from the nature of the interactions in crowded environments. As noted, agents are not always able to take actions that reduce the constraints of their neighbors (given that the agent itself might be constrained). A *C-Nav* agent makes optimistic action decisions by assuming that agents that are farther from its goal than itself will defer to its motion. As this is not always the case, and agents are often pushed against their intended motions, this limits the positive effect of our approach.

C-Nav assumes that agents can broadcast their intended velocities. If this is not the case (i.e. non-communicative agents), our approach would still work, though agents

would only optimize their motions based on their own goal progress and the performance gain might not be as significant (as shown in Section 7.5). To address this limitation, an idea is to explore methods to predict the agents' preferred velocities from a sequence of observed velocities, using, e.g., a hidden Markov model. Adapting *C-Nav* to physical robots moving in human populated environments is another exciting avenue for future work. The recent work of [89] and [90] can provide some interesting ideas in this direction.

In general, all approaches presented at this point (ALAN, PHOP and *C-Nav*) assume that other agents in the environment use the same navigation algorithm and collision avoidance technique. This is a reasonable assumption in controlled settings, but it does not hold in real-world environments. In the latter, agents might need to interact with dynamic entities which might use different navigation models. To ensure collision-free and efficient motion, the agents would need to determine the types of the other agents and act accordingly. In Chapter 8, we address this problem by proposing a Bayesian inference approach that allows an agent to estimate a navigation model and a goal for each neighbor, and to compute a path that minimizes the risk of collision while maximizing its time-efficiency.

Chapter 8

Navigation Among Heterogeneous Agents

In the methods proposed in Chapters 5, 6 and 7, agents assume that all other agents are homogeneous, that is, they all use the same navigation algorithm (ORCA), and only differ in their initial and goal positions. When navigating in real world environments, however, robotic agents are likely to interact with heterogeneous dynamic entities such as humans or robots using different navigation models. In these cases, the uncertainty about the future motions of the other agents means that collision-free navigation might no longer be guaranteed, and it prevents the utility of pre-computing a time-efficient path.

This chapter presents a method that allows an agent to safely navigate to its goal position in a time-efficient manner, in environments populated by agents using an (initially) unknown navigation model. Specifically, the agent uses Bayesian inference to determine the specific model and goal direction of each neighbor, and uses this information to plan a path that avoids collisions and moves the agent closer to its goal.

8.1 Formulation

Unlike the formulation used in the previous chapters, here we are given a single agent α modeled as a disk with radius r that has to reach a goal $\mathbf{g}$. The environment for agent α is defined as a 2D virtual or physical space that includes other n (initially unknown)

heterogeneous agents $A_1 \dots A_n$. Similar to α, each agent A_i has a radius r_i, a position $\mathbf{p}_i$, a velocity $\mathbf{v}_i$ and a goal position $\mathbf{g}_i$.

The task is twofold: first, determine how the neighbors of α navigate in the environment, and second, use this information to steer α to its goal, avoiding collisions with the n heterogeneous agents while minimizing its travel time.

Assumptions. We assume that a fixed number of goal locations $\mathcal{G}$ exist that each agent A_i can choose from. Furthermore, a set $\mathcal{M}$ of different navigation models is given and each agent A_i is only allowed to use one of these models throughout a simulation. Finally, we assume that α has a partial knowledge about the environment in which it navigates. Specifically, α can sense the positions and velocities of its nearby agents, $NN_\alpha \subset A$, located inside a limited sensing range, but it is not aware of the goals of its neighbors and the navigation models that they employ.

8.1.1 Proposed method

We propose a Bayesian inference approach to estimate the probability that an agent in A uses one of the $|\mathcal{M}|$ specified navigation models while moving towards one of the $|\mathcal{G}|$ known goals in the environment. With an estimate computed at each timestep of the model and goal of each agent A_i, α can plan over a set of possible actions to minimize the risk of collisions and generate time-efficient motions.

The agent α can consider two sets of actions. It can either optimize over a set of different preferred velocities while keeping its navigation parameters fixed, or plan over a set of different navigation models while having a fixed goal-oriented preferred velocity with a magnitude equal to $v^{\max}$. See Section 8.3 for more details.

As soon as an optimal action a^* is chosen, it is mapped to a new velocity $\mathbf{v}^{\text{new}}$ using an anticipatory collision avoidance method based on the ORCA navigation framework [1]. While in ORCA each pair of agents involved in a potential collision shares the effort of averting the collision, in our approach, the agent α takes full responsibility of resolving a collision due to the presence of heterogeneous neighbors.

A pseudocode of the approach can be seen in Algorithm 10. While agent α has not reached its goal, it estimates the model and goal for each sensed neighbor (line 5), using Bayesian inference. It then uses this information to determine a new action which takes into account the predicted motion of each neighbor (line 8), according to

Algorithm 10: General approach

1: initialize simulation
2: **while** not at the goal **do**
3: **for all** $i \in NN_\alpha$ **do**
4: observe i velocity and position
5: $Pred_i \leftarrow NeighborInference(i)$
6: $PredSet \leftarrow PredSet \cup Pred_i$
7: **end for**
8: $a^* \leftarrow ActionSelection(PredSet)$
9: $\mathbf{v}^{\text{new}} \leftarrow CollisionAvoidance(a^*)$
10: $\mathbf{p}^{t+1} \leftarrow \mathbf{p}^t + \mathbf{v}^{\text{new}} \cdot \Delta t$
11: **end while**

its estimated model. The action is then given as an input to the collision avoidance routine, to compute a new velocity that is collision-free (line 9). After moving using its computed velocity, the cycle repeats until α reaches its goal.

8.1.2 Navigation models

We consider the following navigation models for the agents in A:

- Personality models: Three of the models are based on the ORCA navigation framework [1]. These models have different sets of parameter values representing social behaviors or personalities, as proposed in [103]. We consider three types of agents: shy, aggressive and tense.
- Social force model: In the work of Helbing et al. [25], the behavior of each agent is modeled as a collection of forces. Each agent has an attractive force that steers it to its goal. In addition, distance-dependent repulsive forces are exerted on the agent from its nearby neighbors and static obstacles. Thus, the social force model is a purely *reactive* model as opposed to the *anticipatory* nature of ORCA and its variants.
- Non-reactive: In this model, agents do not attempt to avoid or resolve collisions, i.e. they are non-reactive and just follow their goal oriented trajectories.

Lastly, we also consider agents that might not use any of the specified navigation models, for example, in case of malfunction, and can have random goals in the environment and random navigation parameters.

8.2 Bayesian Model Inference

We use a Bayesian approach for α to estimate the navigation model and goal position of each of its neighbors. Given the current positions of its neighbors, and the knowledge about their possible models and goals, α computes, at each timestep t the potential position that each neighbor would move to in $t+1$ for each of the model/goal pairs (Fig. 8.1a).

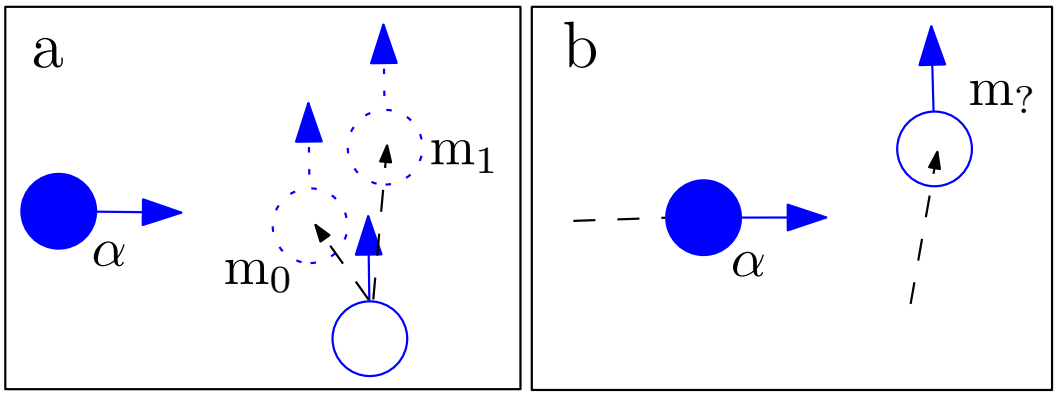


Figure 8.1: Example of the Bayesian model inference. (a) α predicts the position that its neighbor would move to in the next timestep, using either m_0 or m_1. (b) After the agents move, α computes the likelihood of each model given its neighbor's observed position.

At time $t+1$, α observes the new position of each neighbor (Fig. 8.1b), and computes the likelihood of the observed position given each of the possible model and goal combinations (Eq. 8.1), assuming Gaussian noise in their computed velocities.

$$P(\mathbf{p}_i^t | m_j \wedge g_k) \sim \mathcal{N}(\mathbf{p}_i^t; \mu, \sigma) \tag{8.1}$$

To compute the likelihood that the neighbor's position was not generated by any model in $\mathcal{M}$ (for example, due to agent malfunction), that is, the neighbor uses an

'unknown' model, we consider the conjunction of the complements of all the model likelihoods (Eq. 8.2), which is equivalent to their multiplication (Eq. 8.3). Although this formulation does not allow α to learn the navigation parameters of the neighbor using an 'unknown' model, it prevents α from incorrectly classifying such neighbor into one of the known models, which might translate into incorrect motion predictions.

$$P(\mathbf{p}_i^t|m_u \wedge g_u) = \neg P(\mathbf{p}_i^t|m_0 \wedge g_0) \wedge \ldots \wedge \neg P(\mathbf{p}_i^t|m_{|\mathcal{M}|-1} \wedge g_{|\mathcal{G}|-1}) \tag{8.2}$$

$$P(\mathbf{p}_i^t|m_u \wedge g_u) = \prod_{\substack{j=0 \\ k=0}}^{|\mathcal{M}|\cdot|\mathcal{G}|-1} (1 - P(\mathbf{p}_i^t|m_j \wedge g_k)) \tag{8.3}$$

Once the likelihood values have been computed for all model/goal pairs, and for a possible unknown model, the posterior probabilities are computed using a uniform prior in the first observation and the previous posterior as the prior in the consequent observations (Eq. 8.4). This represents the initial uncertainty of the agent with respect to its neighbors models and goals, and as more evidence is incorporated, the true model becomes more likely.

$$P(m_j \wedge g_k|\mathbf{p}_i^t) = \frac{P(\mathbf{p}_i^t|m_j \wedge g_k) \cdot P(m_j \wedge g_k|\mathbf{p}_i^{t-1})}{\sum_{j'=0}^{|M|} \sum_{k'=0}^{|G|} P(\mathbf{p}_i^t|m_{j'} \wedge g_{k'}) \cdot P(m_{j'} \wedge g_{k'}|\mathbf{p}_i^{t-1})} \tag{8.4}$$

8.2.1 Model Estimation and Selection

After computing the posterior probabilities for each model at each timestep, α can estimate the true model of each neighbor. While choosing the model with the maximum posterior would be reasonable, in practice there might be many models that move the agent to the same or a very similar position. This translates into similarly high values for the posteriors. For example, a neighbor agent being pushed by a crowd might be forced to escape from the crowd to avoid collisions, regardless of the model being used.

To address this, we consider a set, C_{mod}, of candidate models for a neighbor that contains all models whose posterior probabilities are within a ratio, λ, of the model with the maximum posterior. C_{mod} also contains the model with the maximum posterior. As

Algorithm 11: Model Selection for neighbor i

for $k = 0, \ldots, |\mathcal{G}| - 1$ **do**
  for $j = 0, \ldots, |\mathcal{M}| - 1$ **do**
    $ComputeLikelihood(i, m_j, g_k)$ (Eq. 8.1)
    $ComputePosterior(i, m_j, g_k)$ (Eq. 8.4)
  end for
end for
$ComputeLikelihood(i, m_u, g_u)$ (Eq. 8.3)
$ComputePosterior(i, m_u, g_u)$ (Eq. 8.4)
for $k = 0, \ldots, |\mathcal{G}| - 1$ **do**
  for $j = 0, \ldots, |\mathcal{M}| - 1$ **do**
    if $\frac{P(m_j \wedge g_k | \mathbf{p}_i^t)}{\max P(m_j \wedge g_k | \mathbf{p}_i^t)} > \lambda$ **then**
      $C_{mod} \leftarrow C_{mod} \cup (m_j, g_k)$
    end if
  end for
end for
$Pred_i \leftarrow$ *randomly chosen* $(m_j, g_k) \in C_{mod}$

the simulation progresses and α obtains more observations of how each neighbor move, its true model should be more easily differentiated from the other candidate models, and the size of C_{mod} should decrease and finally converge to the true model. In all of our experiments, we set $\lambda = 0.999$. Algorithm 11 shows the model selection procedure.

Motion Prediction

Once a model and goal have been estimated for each neighbor, α can use this information to predict their future motions along with their interactions with the environment. We predict the future motion of a neighbor by selecting a single model, denoted $Pred_i$, randomly from C_{mod}. The likelihood of each candidate model being chosen is proportional to their respective normalized posterior values.

8.3 Action Planning

The prediction of future trajectories for each neighbor (given their estimated model/goal pairs) allows α to predict how each one will react to potential collisions and can reduce the uncertainty of the future state of the environment. This enables α to plan a safe and efficient path.

As the motion of the neighbors might be influenced by agents not visible to α, and other neighbors can appear at a later stage, α needs to continuously replan its path to account for this new information. Therefore, we compute a new plan at every timestep, where each plan is computed by simulating the execution of a sequence of actions for a time horizon of T timesteps, similar to the work in [6]. Algorithm 12 shows the pseudocode of the planning process using an action space *Actions*, for the entire time horizon T.

Action Evaluation. To compare the actions available for α, each action is evaluated based on how much it helps the agent to progress towards its goal, that is, using only the *goal-oriented* component of the reward function template in Eq. 4.1. We estimate such a progress in the same manner as in the previous chapters (Eq. 5.2), by projecting the collision-free velocity computed as a result of choosing this action onto the normalized goal-oriented vector of α:

$$\mathcal{R}_a = \mathbf{v}^{\text{new}} \cdot \frac{\mathbf{g} - \mathbf{p}}{\|\mathbf{g} - \mathbf{p}\|} \tag{8.5}$$

As a result, α promotes goal oriented behavior that is collision-free.

We consider two types of plans for α, with action spaces that allow different granularities of control for α's motion: a set of preferred velocities, and a set of navigation models.

8.3.1 Planning in the space of preferred velocities

For a fine-grained motion control, we allow α to select among a set of 9 preferred velocities (Fig. 8.2). This set of velocities, defined in Chapter 4.1, is uniformly distributed in the space of possible directions of motions that α can take, enabling α to adapt its motion to different local conditions by attempting to move sideways or backwards from the goal when needed.

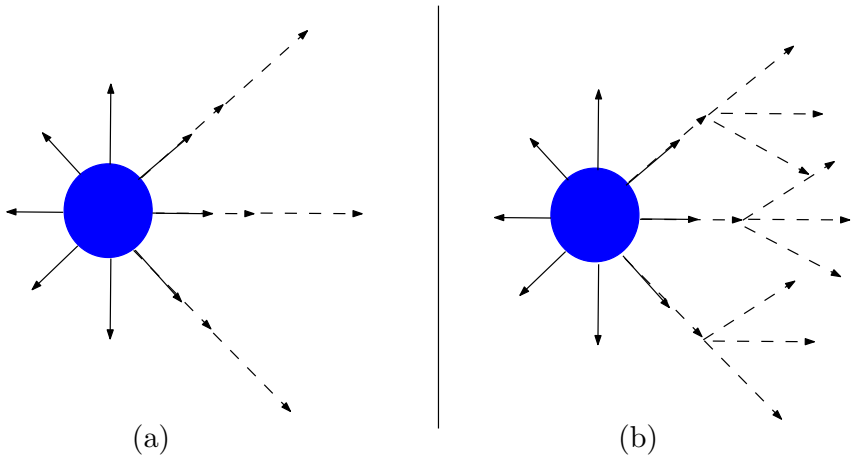


Figure 8.2: The 9 actions of VelPlan correspond to moving at 1.5 m/s with different angles with respect to the goal: $0°$, β, $-\beta$, $90°$, $-90°$, $180°$, $180° + \beta$, $180° - \beta$ and complete stop. In our implementation, $\beta = 45°$. (a) Simple plans consist of single actions for the entire time horizon. (b) More complex plans are computed if planning time allows.

In the planning process, α simulates multiple plans of actions, where each plan involves the execution of a set of preferred velocities for T timesteps in the future. At each timestep, α updates its position along with the position of its neighbors, assuming that they try to reach their inferred goals using the inferred navigation models.

To determine which actions to include in a plan, we follow the approach in PHOP (Chapter 6). Figures 8.2a and 8.2b show an example of the plan generation process. We begin computing simple plans after taking the same action for each timestep (Fig. 8.2a), and progressively (if planning time allows) generate more complex plans composed by multiple actions (Fig. 8.2b). We compute the value of each simulated plan by computing $\mathcal{R}_a$ (Eq. 8.5) for each timestep in the time horizon. Finally, we average the reward value for all the plans that begin with the same initial action, and associate the averaged reward with the initial preferred velocity of the plan. The planning continues, generating more complex plans until a specified planning time limit is reached.

Using the space of preferred velocities, α has a more fine-grained control over its motion which allows it to adapt to a wide range of environments, at higher computational

Algorithm 12: Action Selection

1: **for all** $a \in Actions$ **do**
2: **for** $t = 0, \ldots, T-1$ **do**
3: simulate evolution of neighborhood dynamics
4: $\mathcal{R}_a \leftarrow \mathcal{R}_a + \mathbf{v}^{\text{new}} \cdot \frac{\mathbf{g}-\mathbf{p}}{\|\mathbf{g}-\mathbf{p}\|}$
5: **end for**
6: $\mathcal{R}_a \leftarrow \frac{\mathcal{R}_a}{T}$
7: **end for**
8: $a^* \leftarrow \arg\max_{a \in Actions} \mathcal{R}_a$

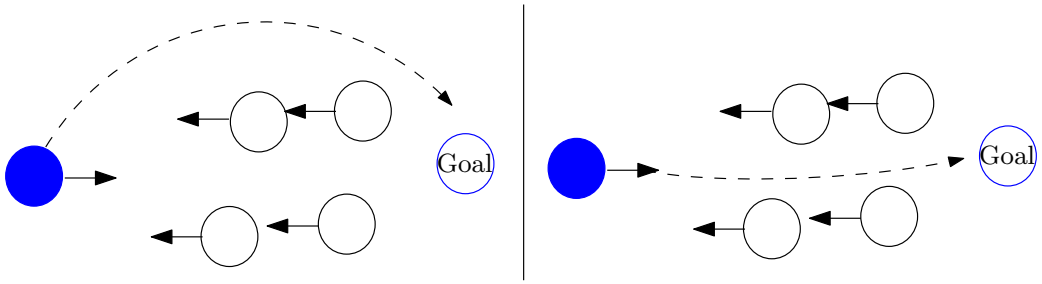
cost (as it simulates at least 9 plans of T timesteps). We call this approach VelPlan.

8.3.2 Planning in the space of navigation models

Instead of making decisions in the space of preferred velocities, the agent could also evaluate the space of navigation models. In this chapter, we assume that the agent can be either shy, aggressive, or tense (see Section 8.1.2). At each timestep, α can choose one of these navigation models by adapting its corresponding ORCA parameters such as time horizon, sensing range, and assumption of reciprocity in avoiding collisions with the other agents.

As opposed to the space of preferred velocities, here α simulates each one of the three models for the entire time horizon, computing at each timestep $\mathcal{R}_a$ (using Eq. 8.5). This type of planning is computationally more efficient than using preferred velocities (it requires only three simulations of T timesteps), and results in behaviors that can be explained from a social point of view. For example, α chooses a more conservative model (e.g. shy or tense) when congestion is predicted which allows the agent to avoid it, and a more aggressive model in scenarios void of static and dynamic obstacles, where α should move as fast as possible to its goal. Figure 8.3 shows different behavior that α can assume using this type of planning.

This 'social' planning approach allows α to determine which high level behavior is best for interacting with neighbors in a given environment, whereas planning over the space of different preferred velocities provides a more fine-grained control of the agent's



(a) Shy (b) Aggressive

Figure 8.3: Example of different ORCA behaviors. (a) Using a shy model, α decides to move around the group. (b) Using the aggressive model, instead, α makes way between the incoming agents to reach the goal faster.

motion. We call this planning method SocialPlan.

8.4 Results

We evaluated our inference approach and our two planning methods, VelPlan and SocialPlan, across different simulation scenarios (see Figure 8.4). In all scenarios, unless otherwise specified, we uniformly distributed all, but the unknown, navigation models defined in Section 8.1.2 over the set of agents A, while the goal for each agent was assigned based on its initial position. To emulate agents that do not follow any model, we randomized the navigation parameters and goals of 5% of the agents after the initial goal and model allocation. For our approach, we empirically set the values of the time horizon T to 50 timesteps, and used a σ value of 10^{-4}, as they provided the best performance compared to larger values. Unlike the approaches proposed in the previous Chapters, here we updated the position of α every $\Delta t = 0.1s$, which was also the planning time limit. We use a larger value of Δ as it allows for a more clear differentiation between the positions predicted for the agents with the different models.

Scenarios. We considered five scenarios: In the *Incoming* scenario, 30 agents move in the opposite direction of α, each having chosen a goal from 8 user-defined goals. In the *Perp1* scenario, a crowd of 100 agents moves in a perpendicular direction to α's motion, towards one goal. In the *Perp2* scenario, two groups of 50 agents each move in

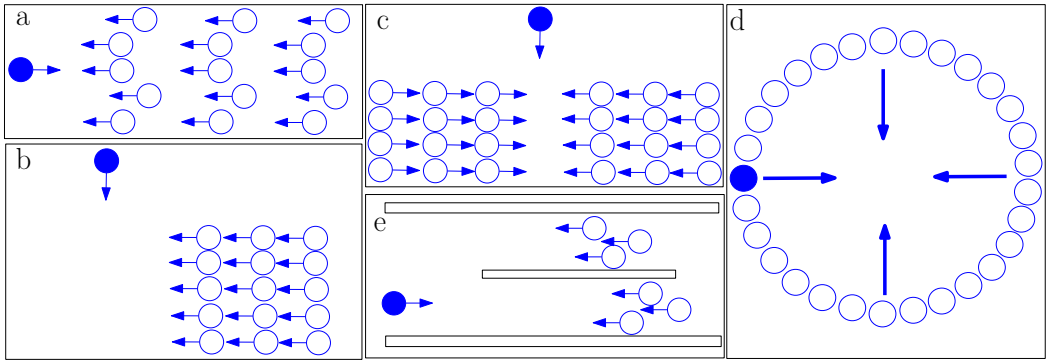


Figure 8.4: Scenarios. (a) *Incoming*. (b) *Perp1*. (c) *Perp2*. (d) *Circle*. (e) *TwoPaths*.

a perpendicular direction to α's motion, towards one of 2 goals. In the *Circle* scenario, α and 63 neighbors are placed along the circumference of a circle and must reach their antipodal positions, towards one of 64 goals. Finally, in the *TwoPaths* scenario, six agents have to navigate through two narrow corridors towards six goals placed behind α's starting position. Here, α must choose between two homotopic paths based on the models of the agents perceived.

We highlight below the performance of our framework as measured by the prediction accuracy of our Bayesian approach and the time-efficiency of the α agent. We also provide comparisons to the ORCA framework and to a receding time-horizon approach that is similar in nature to VelPlan but without an inference component (NoInference).

8.4.1 Model Prediction

We evaluate the accuracy of the Bayesian inference based on two criteria: Candidate accuracy, where a prediction is correct if the true model/goal of the neighbor is in the set C_{mod} (candidate models/goals) and Predicted accuracy, where a prediction is correct if the true mode/goal is the one used for prediction of the neighbor's future motion.

Table 8.1 shows in percentage the accuracy of our VelPlan approach in all scenarios. In general, α can predict with high accuracy the models of its neighbors because of how they interact with other agents. For example, shy and tense agents try to maintain a larger distance with other agents than, for example, non-reactive or aggressive agents. In particular, the lowest accuracy is observed in the *TwoPaths* scenario. Here, the

Accuracy	Incoming	Perp1	Perp2	Circle	TwoPaths
Candidate Model	94.4	95.5	95.1	94.5	86.4
Predicted Model	91.4	93.5	92.9	90.7	79.2
Candidate Goal	94.7	99.5	98.5	94.8	89.9
Predicted Goal	88.5	98.9	95.4	88.2	81.6

Table 8.1: Accuracy (%) of VelPlan in terms of goal and type prediction in four scenarios.

agents are constrained by the walls of the environment, which constrains their motion and makes it more difficult to differentiate between similar models.

In general, α can predict with high accuracy the models of its neighbors because of how they interact with other agents. For example, shy and tense agents try to maintain a larger distance with other agents than, for example, non-reactive or aggressive agents. On the other hand, non-reactive agents are easily differentiated in crowded environments as they do not correct their path in case of collisions. This interactions reveal the the true model of the neighbors. If a single neighbor was evaluated in isolation, the accuracy of its predicted type would be lower given the lack of interactions.

8.4.2 Time-efficiency

To measure the time-efficiency of the path that α takes, we measured the *interaction overhead*, i.e. the time that it takes for the agent to resolve interactions with the other agents and the environment. The overhead is zero if α is able to move at full speed in a straight line to its goal (i.e. without deviating from its goal-oriented path). Figure 8.5 compares the interaction overhead in all of our scenarios obtained using different simulation methods. As can be seen from the figure, predicting the behavior of other agents clearly improves the time-efficiency of α's navigation as well. We can observe that VelPlan significantly outperforms both NoInference and ORCA in all scenarios. This performance difference is the largest in the *Incoming* scenario, where α has enough room to maneuver to avoid non-reactive agents, while also being able to move through agents that will defer to its motion, such as the shy agents. It is worth noting the difference in performance between *Perp1* and *Perp2*. The latter scenario is particularly complex even for our approach, as it was also previously demonstrated with the lower accuracy

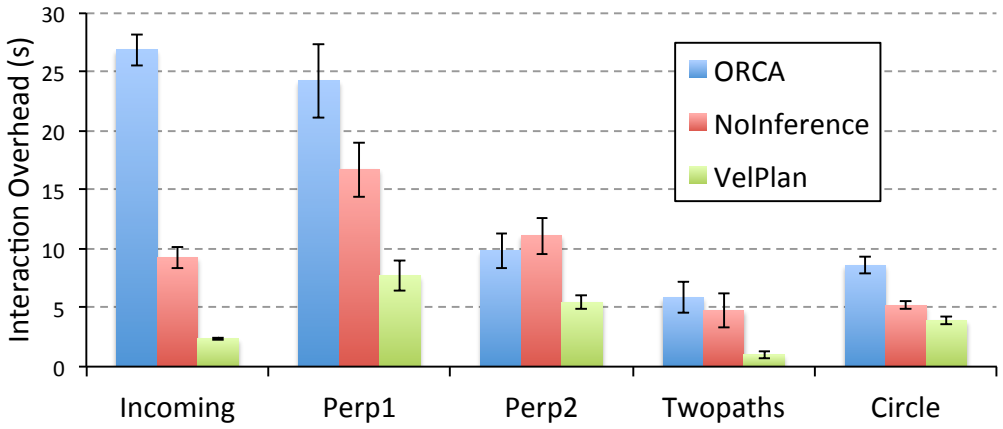


Figure 8.5: Interaction overhead for all evaluated approaches. In all scenarios, VelPlan outperforms ORCA and NoInference. The error bars denote the standard error of the mean.

values and the large number of collisions. In general, these results, together with Table 8.3, demonstrate the utility of the model and goal estimation in the computation of a safer and more efficient path for α.

Single model. We also evaluated the interaction overhead of α using VelPlan, NoInference and ORCA in the *Incoming* scenario, when all other agents in the environment use either the shy model, the social force model, or are non-reactive.

Results shown in Figure 8.6 indicate that, against shy agents, all approaches are able to find paths with low interaction overhead. Here, the shy agents make space for α, hence the deviation from the goal path is minimal. When the other agents are using social forces, however, only VelPlan realizes that it is best to move around the group in order to reach the goal faster. On the other hand, α gets trapped with ORCA and NoInference, as it incorrectly assumes that the other agents will anticipate potential collisions. As a result, α gets caught up in the opposite moving group and gets pushed back by its agents. Similar behavior occurs with non-reactive agents, where only VelPlan is able to successfully interact with them, while in both ORCA and NoInference, α gets pushed until the non-reactive agents reach their goals.

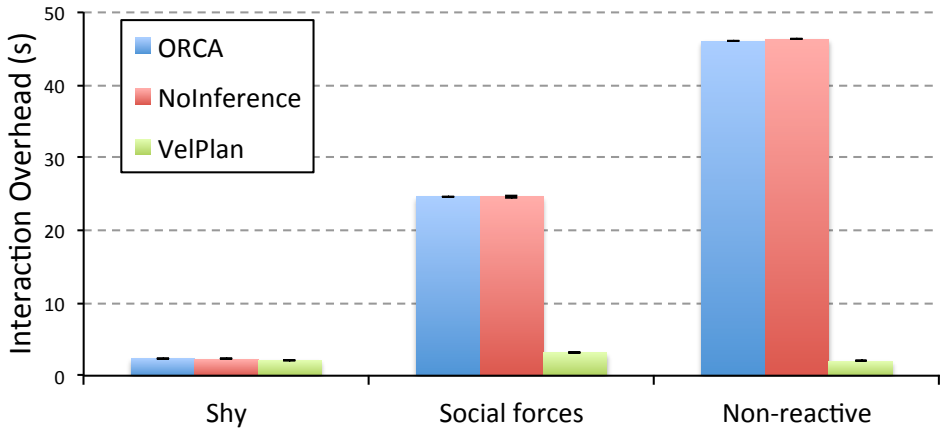


Figure 8.6: Interaction overhead for all evaluated approaches with agents of the same model in the *Incoming* scenario. In all cases, VelPlan matches or outperforms ORCA and NoInference.

Behavior	Shy	Social Forces	Mixed
Shy	2	3.7	2.6
Aggressive	**0**	31.5	2.7
Tense	36.8	27.6	4.94
SocialPlan	**0**	**1.5**	**0.9**

Table 8.2: Interaction overhead (s) in Social Planning.

8.4.3 Social Planning

To evaluate our SocialPlan approach, we tested the behavior and the interaction overhead in a small scenario (see Figure 8.7), where α crosses path with 4 other agents that employ social forces. Here, using SocialPlan means that α had to choose between being shy, aggressive or tense. We observe that, using our approach, α chooses an aggressive personality in the beginning as it gets close to the group (Fig. 8.7b). Once it realizes it will be pushed away from its goal, α switches to a tense personality (Fig. 8.7c), which allows it to move back for a while and let the group pass. Once this happens, it adopts an aggressive personality which enables α to quickly move to its goal (Fig. 8.7d).

We also compared the interaction overhead of α using SocialPlan versus using a fixed

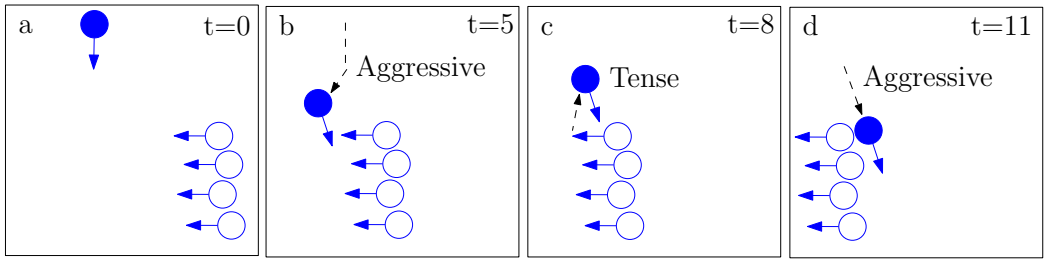


Figure 8.7: Social Planning example. (a) Initial position of the agents. (b) The agent uses aggressive behavior until its motion is constrained by the group. (c) The agent switches to a Tense behavior which pushes him away from the group. (d) After the group has passed by, the agent again assumes an aggressive behavior to move fast and unconstrained to its goal.

personality mode (shy, tense or aggressive) during the entire simulation (see Table 8.2). We also varied the model used by the group of 4 agents (only shy, only social forces or a mix of agent models). We can see that, due to the ability of SocialPlan to determine the types of the other agents and to switch between behaviors in real time, α finds velocities that increase its time-efficiency.

8.5 Analysis

In this section, we first discuss how our approach fits in this dissertation before analyzing its runtime complexity, as well as other metrics such as the average number of collision for α in each evaluated scenario.

The approach proposed in this chapter relaxes some assumptions used in ALAN (Chapter 5), PHOP (Chapter 6) and C-Nav (Chapter 6) such as the homogeneity of the navigation models and action learning method. To evaluate and choose the optimal action (which maximizes the local reward) in this new scenario, the agent α first learns the model and goal of the other agents in order to predict their future trajectories. With these predictions, α follows an approach similar to PHOP to choose the optimal action at each timestep, in order to minimize its travel time.

Method	Incoming	Perp1	Perp2	Circle	TwoPaths
ORCA	51.22	83.28	38.08	5	11.7
NoInference	**0**	44.12	66.18	0.1	13.04
VelPlan	**0**	**6.96**	**28.74**	**0.04**	**0.02**

Table 8.3: Average collisions per iteration using ORCA, NoInference and VelPlan.

8.5.1 Runtime Complexity

We can divide the analysis of complexity of our approach it in two stages: model estimation and trajectory prediction.

For the model estimation, the agent α needs to simulate the potential next position for each of its neighbors, with each navigation model while moving to each of the known goals. Assuming n neighbors, the agent needs to perform $n \cdot g \cdot m$ computations to estimate a model for all of its neighbors, in each timestep. Although the estimation should converge to a model early on the simulation, α still performs the inference process through the entire simulation, for each sensed neighbor. The complexity of this stage is then $\mathcal{O}(n \cdot g \cdot m)$ for the single controlled agent.

For the trajectory prediction, the complexity of VelPlan is similar to PHOP's (Chapter 6), as the agent α needs to simulate the motions of its neighbors for the length of the time horizon (T), for each one of its actions. Therefore, assuming there are n neighbors and k actions to be evaluated, the runtime complexity of the trajectory prediction is $\mathcal{O}(k \cdot T \cdot n^2)$ for the single controlled agent.

It should be noted that the complexity of SocialPlan and VelPlan only differ in the number of actions consider, hence is a constant factor which does not alter the overall runtime complexity.

8.5.2 Number of collisions

Table 8.3 shows the average number of collisions that occur between α and its neighbors, using VelPlan, ORCA and the NoInference simulation method. Overall, using VelPlan results in significantly less collisions than the other two methods. However, even when using our VelPlan approach, α may collide with other agents. This is more evident in very crowded environments such as the $Perp2$ scenario, where the number of agents

limits the capacity of α to find safe motion, for example, when facing non-reactive agents coming in opposite directions. Using either NoInference or ORCA, α runs into more collisions given the incorrect assumption of the behavior of other agents; α assumes that agents will collaborate their efforts to resolve collisions, which is not always true.

8.6 Limitations of the approach

Our approach is limited to navigation models used typically in multi-agent navigation. The extend to which these models can also be applicable to simulate human motion, is limited. Another limitation of our approach is that it does not attempt to learn navigation models outside of the specified set of models. To address this, we would like to explore methods to learn new navigation models using only observations of the agents' motions.

Chapter 9

Conclusions and Future Work

This dissertation presented a general framework for action selection in multi-agent navigation, and a set of methods to allow agents to evaluate and select actions independently, in real-time, and with only partial knowledge of the environment. The contributions proposed in this dissertation allow agents to successfully interact in crowded and partially known environments in order to minimize their total travel time.

Specifically, this thesis has proposed the following contributions:

- A framework for action selection in multi-agent navigation, where agents are given a set of motions to select from at each time instant.
- A learning-based approach for agents to learn the value of their actions, by formulating the navigation problem as a multi-armed bandit problem, where agents select their actions based on their local conditions and their goal progress.
- A planning-based approach for predicting the value of each action, where agents simulate a series of potential future scenarios for each action, and evaluate 'in hindsight' the first action of each plan. This reduces the uncertainty of the long-term effects of each action and allow the agents to avoid congestion before it develops.
- An implicit coordination approach where agents use the intended motions of their neighbors to select actions that benefit the goal progress of the entire neighborhood, effectively displaying coordinated motions while using only limited one-way

communication.

- A Bayesian Inference approach to estimate the navigation models used by other agents in environments with heterogeneous agents. With an estimated model and goal for each neighbor, an agent can plan a path that minimizes the risk of collisions while efficiently moving to its goal.

All these contributions have been empirically shown to minimize the total travel time of all agents, in many cases close to their optimal values.

9.1 Limitations

The methods presented allow agents to efficiently interact with each other in crowded and partially known environments, in order to minimize their total travel time. However, the work developed in this dissertation is not free from limitations. In this chapter, we briefly describe the limitations of the proposed work, which relate to the use of a discretized space of actions and to the relaxation of some real-world constraints.

9.1.1 Discretized action set

In all methods proposed in this dissertation, we require agents to select from a pre-defined set of actions, which correspond the the preferred velocities. Therefore, agents have a fix set of options to evaluate $\mathcal{R}$ (Eq. 4.1). In many cases, the optimal preferred velocity might not be represented by any of the actions defined. This means that the computed optimal actions correspond to approximate solutions to the problem of maximizing the agent's reward (Eq. 4.1). The quality of this approximate solution is directly proportional to the number of pre-defined actions, assuming that they are uniformly distributed in the space of possible directions of motions. However, the more actions the agents need to consider, the more computationally expensive is the process of finding the optimal solution.

9.1.2 Application to multi-robot navigation

The approaches presented do not consider some constraints which are typically present in multi-robot navigation tasks.

First, we do not consider kinodynamic constraints that affect the motion of robots. Agents using the proposed approaches are not subject to acceleration constraints: we assume they can shift from being completely stopped to move the maximum velocity $v^{\max}$ instantaneously. Robots, instead, are subject to a maximum acceleration that restricts the speed at which they are able to change their velocity.

Second, we assume that agents in our approaches are holonomic. This holonomic property assumes that agents are able to move in any direction at any given time. Again, this is not the case for many common types of robots, where a change in direction of motion requires the robot to first perform a rotation (for example, in robots such as the Turtlebot [104]) or more complex motions (for example, in car-like robots).

Finally, we assume that agents can perfectly determine how other agents move and their exact positions (i.e. the center of their disk-shaped structures), effectively assuming zero noise sensing capabilities. Although ORCA (and consequently, our proposed approaches) considers a small uniform perturbation in the computation of the preferred velocities, this perturbation represent noise only in the actuators, not in the sensors. Given the imperfect devices used in robots for sensing purposes, the sensed positions and velocities will include some type of noise. Hence, our perfect sensing assumption does not hold in real-world domains, and must be accounted for using methods to address the uncertainty derived from noisy sensors, or global localization tools such as OptiTrack [105] or Vikon motion capture systems.

9.1.3 Application to pedestrian simulation

Our proposed approaches could be used to simulate intelligent and polite behavior in pedestrian simulations. However, in its current state, the lack of kinodynamic constraints and the holonomic assumption of our methods translate into agent motions that are non-smooth and unrealistic for humans, which prevents their direct application to pedestrian simulations.

9.2 Future Work

There are multiple possible lines of future work for this dissertation. In this section we briefly develop some of these lines.

9.2.1 Theoretical Analysis

We intend to complement the empirical results obtained in this dissertation with a theoretical analysis that gives better insight about the experimental results. In particular, we want to analyze under which environmental conditions is each of the proposed approaches more adequate. This would allow for the development of a meta-algorithm which, in real-time, could tune the specific method to use (from the contributions developed) as well as the specific navigation parameters (for example, the coordination factor γ or the time horizon used in the predictions of the environment).

9.2.2 Validation in Multi-Robot Systems

A natural line of future work involves the evaluation of the proposed approaches in real-world environments with multiple robots. Although in a limited scale, this evaluation would provide us with validation with respect to the advantages of each approach, and the challenges that would need to be addressed to enable large scale adoption of the presented approaches.

9.2.3 Validation in Pedestrian Simulations

Another line of future work focuses on the application of the proposed approaches to the simulation of human motion. This can emerge from the use of the human related concepts of politeness and selfish behaviors throughout this dissertation. By potentially incorporating theories and elements from the social psychology domain, our work can be used to simulate the behavior (and the evolution of the behavior) of humans in a variety of conditions. For example, we would like to simulate evacuation scenarios that can help architects and other professionals in their design of human spaces and their interconnections.

References

[1] Jur van den Berg, Stephen J. Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Proc. International Symposium of Robotics Research*, pages 3–19. Springer, 2011.

[2] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

[3] Julio Godoy, Ioannis Karamouzas, Stephen J. Guy, and Maria Gini. ϵ-ucb for action selection in multi agent navigation. In *Machine Learning and Planning for Control of Robot Motion Workshop at IROS*, 2014.

[4] Julio Godoy, Ioannis Karamouzas, Stephen J. Guy, and Maria Gini. Adaptive learning for multi-agent navigation. In *Proc. Int. Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1577–1585, 2015.

[5] E. Chong et al. A framework for simulation-based network control via Hindsight optimization. In *IEEE Conf. on Decision and Control*, volume 2, pages 1433–1438, 2000.

[6] Julio Godoy, Ioannis Karamouzas, Stephen J. Guy, and Maria Gini. Anytime navigation with progressive hindsight optimization. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2014.

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

[8] Julio Godoy, Ioannis Karamouzas, Stephen J. Guy, and Maria Gini. Moving in a crowd: Safe and efficient navigation among heterogeneous agents. In *Proc. Int. Joint Conf. on Artificial Intelligence*, 2016.

[9] Nuria Pelechano, Jan M Allbeck, and Norman I Badler. Virtual crowds: Methods, simulation, and control. *Synthesis Lectures on Computer Graphics and Animation*, 3(1):1–176, 2008.

[10] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):24–34, 1987.

[11] C.W. Reynolds. Steering behaviors for autonomous characters. In *Game Developers Conference*, pages 763–782, 1999.

[12] O.B. Bayazit, J.-M. Lien, and N.M. Amato. Better group behaviors in complex environments using global roadmaps. In *8th International Conference on Artificial life*, pages 362–370, 2003.

[13] J. Funge, X. Tu, and D. Terzopoulos. Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In *26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 29–38, 1999.

[14] W. Shao and D. Terzopoulos. Autonomous pedestrians. *Graphical Models*, 69(5-6):246–274, 2007.

[15] Stephen. J. Guy, J. Chhugani, S. Curtis, Dubey Pradeep, M. Lin, and D. Manocha. PLEdestrians: A least-effort approach to crowd simulation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 119–128, 2010.

[16] N. Pelechano, J.M. Allbeck, and N.I. Badler. Controlling individual agents in high-density crowd simulation. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 99–108, 2007.

[17] S.J. Guy, S. Kim, M.C. Lin, and D. Manocha. Simulating heterogeneous crowd behaviors using personality trait theory. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 43–52, 2011.

[18] Markéta Popelová, Michal Bída, Cyril Brom, Jakub Gemrot, and Jakub Tomek. When a couple goes together: walk along steering. In *Motion in Games*, volume 7060 of *LNCS*, pages 278–289. Springer, 2011.

[19] Jason Tsai, Emma Bowring, Stacy Marsella, and Milind Tambe. Empirical evaluation of computational fear contagion models in crowd dispersions. *Autonomous Agents and Multi-Agent Systems*, pages 1–18, 2013.

[20] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Robotics Research*, 5(1):90–98, 1986.

[21] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE J. of Robotics and Automation*, RA-2(1):14–23, March 1986.

[22] Steve Ratering and Maria Gini. Robot navigation in a known environment with unknown moving obstacles. *Autonomous Robots*, 1(2):149–165, 1995.

[23] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using Velocity Obstacles. *The Int. J. of Robotics Research*, 17:760–772, 1998.

[24] D. Hsu, R. Kindel, J.C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *Int. J. Robotics Research*, 21(3):233–255, 2002.

[25] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, 407(6803):487–490, 2000.

[26] Ioannis Karamouzas, Peter Heil, Pascal van Beek, and Mark Overmars. A predictive collision avoidance model for pedestrian simulation. In *Motion in Games*, volume 5884 of *LNCS*, pages 41–52. Springer, 2009.

[27] Jur van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 1928–1935, 2008.

[28] J. Pettré, J. Ondrej, A.-H. Olivier, A. Crétual, and S. Donikian. Experiment-based modeling, simulation and validation of interactions between virtual walkers. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 189–198, 2009.

[29] Ioannis Karamouzas and Mark Overmars. Simulating and evaluating the local behavior of small pedestrian groups. *IEEE Trans. Vis. Comput. Graphics*, 18(3):394–406, 2012.

[30] Jan Ondřej, Julien Pettré, Anne-Hélène Olivier, and Stéphane Donikian. A synthetic-vision based steering approach for crowd simulation. *ACM Trans. Graphics*, 29(4):123, 2010.

[31] S.J. Guy, J. Chhugani, C. Kim, N. Satish, M.C. Lin, D. Manocha, and P. Dubey. Clearpath: highly parallel collision avoidance for multi-agent simulation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 177–187, 2009.

[32] Ioannis Karamouzas, Brian Skinner, and Stephen J Guy. Universal power law governing pedestrian interactions. *Physical review letters*, 113(23), 2014.

[33] L. Buşoniu, R. Babuška, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev*, 38(2):156–172, March 2008.

[34] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[35] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[36] William Uther and Manuela Veloso. Adversarial reinforcement learning. Technical report, Carnegie Mellon University, 1997.

[37] Chongjie Zhang and Victor Lesser. Coordinated multi-agent learning for decentralized POMDPs. In *7th Annual Workshop on Multiagent Sequential Decision-Making Under Uncertainty (MSDM) at AAMAS*, pages 72–78, 2012.

[38] Chongjie Zhang and Victor Lesser. Coordinating multi-agent reinforcement learning with limited communication. In *Proc. Int. Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1101–1108, 2013.

[39] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proc. Int'l Conf. on Machine Learning (ICML)*, volume 94, pages 157–163, 1994.

[40] Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215–250, 2002.

[41] Duc Thien Nguyen, William Yeoh, Hoong Chuin Lau, Shlomo Zilberstein, and Chongjie Zhang. Decentralized multi-agent reinforcement learning in average-reward dynamic dcops. In *Proc. Int. Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1341–1342, 2014.

[42] Marco Wiering et al. Multi-agent reinforcement learning for traffic light control. In *Proc. Int'l Conf. on Machine Learning (ICML)*, pages 1151–1158, 2000.

[43] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári. Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19):1876–1902, 2009.

[44] William G Macready and David H Wolpert. Bandit problems and the exploration/exploitation tradeoff. *IEEE Trans. Evol. Comput.*, 2(1):2–22, 1998.

[45] Peter Henry, Christian Vollmer, Brian Ferris, and Dieter Fox. Learning to navigate through crowded environments. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 981–986, 2010.

[46] Lisa Torrey. Crowd simulation via multi-agent reinforcement learning. In *Proc. Artificial Intelligence and Interactive Digital Entertainment*, pages 89–94, 2010.

[47] Francisco Martinez-Gil, Miguel Lozano, and Fernando Fernández. Strategies for simulating pedestrian navigation with multiple reinforcement learning agents. *Autonomous Agents and Multi-Agent Systems*, 29(1):98–130, 2015.

[48] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[49] Francisco Martinez-Gil, Miguel Lozano, and Fernando Fernández. Multi-agent reinforcement learning for simulating pedestrian navigation. In *Adaptive and Learning Agents*, pages 54–69. Springer, 2012.

[50] Bryan Cunningham and Yong Cao. Levels of realism for cooperative multi-agent reinforcement learning. In *Advances in Swarm Intelligence*, pages 573–582. Springer, 2012.

[51] Suranga Hettiarachchi. An evolutionary approach to swarm adaptation in dense environments. In *IEEE Int'l Conf. on Control Automation and Systems*, pages 962–966, 2010.

[52] L. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Autom.*, 12(4):566–580, 1996.

[53] J.J. Kuffner and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 995–1001, 2000.

[54] R. Geraerts. Planning short paths with clearance using explicit corridors. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 1997–2004, 2010.

[55] W. van Toll et al. Real-time density-based crowd simulation. *Computer Animation and Virtual Worlds*, 23:59–69, 2012.

[56] A. Stentz. The focussed D* algorithm for real-time replanning. In *Int. Joint Conf. on Artificial Intelligence*, volume 14, pages 1652–1659, 1995.

[57] M. Likhachev et al. Anytime dynamic A*: An anytime, replanning algorithm. In *Proc. Int'l Conf. on Automated Planning and Scheduling*, pages 262–271, 2005.

[58] J. van den Berg et al. Anytime path planning and replanning in dynamic environments. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 2366 – 2371, May 2006.

[59] David Q Mayne and Hannah Michalska. Receding horizon control of nonlinear systems. *IEEE Trans. Autom. Control*, 35(7):814–824, 1990.

[60] Cameron Browne et al. A survey of Monte Carlo tree search methods. *IEEE Trans. Comput. Intell. and AI in Games*, 4(1):1–43, 2012.

[61] S. Wook Yoon et al. Probabilistic planning via determinization in Hindsight. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 1010–1016, 2008.

[62] Ethan Burns, J Benton, Wheeler Ruml, Sung Wook Yoon, and Minh Binh Do. Anticipatory on-line planning. In *Proc. Int'l Conf. on Automated Planning and Scheduling*, 2012.

[63] Ronald Bjarnason, Alan Fern, and Prasad Tadepalli. Lower bounding klondike solitaire with monte-carlo planning. In *Proc. Int'l Conf. on Automated Planning and Scheduling*, 2009.

[64] Shervin Javdani, J Andrew Bagnell, and Siddhartha Srinivasa. Shared autonomy via hindsight optimization. In *Robotics: Science and Systems*, 2015.

[65] Anand S Rao, Michael P Georgeff, et al. BDI agents: From theory to practice. In *Proc. Int'l Conf. on Multi-Agent Systems*, volume 95, pages 312–319, 1995.

[66] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proc. Int. Conf. on Autonomous Agents and Multi-Agent Systems*, volume 3, pages 161–168, 2003.

[67] Brammert Ottens and Boi Faltings. Coordinating agent plans through distributed constraint optimization. In *Proc. of the ICAPS-08 Workshop on Multiagent Planning*, 2008.

[68] Carlos Guestrin, Shobha Venkataraman, and Daphne Koller. Context-specific multiagent coordination and planning with factored MDPs. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 253–259, 2002.

[69] Reza Olfati-Saber, Alex Fax, and Richard M Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.

[70] Reza Olfati-Saber and Richard M Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Trans. Autom. Control*, 49(9):1520–1533, 2004.

[71] Meng Ji and Magnus Egerstedt. Distributed coordination control of multiagent systems while preserving connectedness. *IEEE Trans. Robot.*, 23(4):693–703, 2007.

[72] Petter Ögren, Magnus Egerstedt, and Xiaoming Hu. A control lyapunov function approach to multi-agent coordination. In *Proc. IEEE Conf. on Decision and Control*, volume 2, pages 1150–1155, 2001.

[73] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 746–752, 1998.

[74] Francisco S Melo and Manuela Veloso. Decentralized MDPs with sparse interactions. *Artificial Intelligence*, 175(11):1757–1789, 2011.

[75] Laëtitia Matignon, Laurent Jeanpierre, and Abdel-Illah Mouaddib. Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 2017–2023, 2012.

[76] Sandip Sen, Mahendra Sekaran, John Hale, et al. Learning to coordinate without sharing information. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 426–431, 1994.

[77] Ernst Fehr and Urs Fischbacher. Social norms and human cooperation. *Trends in cognitive sciences*, 8(4):185–190, 2004.

[78] Chao Yu, Minjie Zhang, Fenghui Ren, and Xudong Luo. Emergence of social norms through collective learning in networked agent societies. In *Proc. Int. Conf. on Autonomous Agents and Multi-Agent Systems*, pages 475–482, 2013.

[79] Natalie Fridman and Gal A Kaminka. Modeling pedestrian crowd behavior based on a cognitive model of social comparison theory. *Computational and Mathematical Organization Theory*, 16(4):348–372, 2010.

[80] John E Hopcroft, Jacob Theodore Schwartz, and Micha Sharir. On the complexity of motion planning for multiple independent objects; pspace-hardness of the" warehouseman's problem". *The Int. J. of Robotics Research*, 3(4):76–88, 1984.

[81] David Silver. Cooperative pathfinding. In *Proc. Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 117–122, 2005.

[82] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Syst. Sci. and Cybern.*, 4(2):100–107, 1968.

[83] M.R. Jansen and N.R. Sturtevant. Direction maps for cooperative pathfinding. In *Proc. Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 185–190, 2008.

[84] Graham Pentheny. Advanced techniques for robust, efficient crowds. *Game AI Pro 2: Collected Wisdom of Game AI Professionals*, page 173, 2015.

[85] Vinicius Graciano Santos and Luiz Chaimowicz. Cohesion and segregation in swarm navigation. *Robotica*, 32(02):209–223, 2014.

[86] Ioannis Karamouzas and Stephen J Guy. Prioritized group navigation with formation velocity obstacles. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 5983–5989, 2015.

[87] Liang He, Jia Pan, Wenping Wang, and Dinesh Manocha. Proxemic group behaviors using reciprocal multi-agent navigation. In *Proc. IEEE Int. Conf. on Robotics and Automation*, 2016.

[88] Ross A Knepper and Daniela Rus. Pedestrian-inspired sampling-based multi-robot collision avoidance. In *Proc. IEEE Int. Symp. on Robot and Human Interactive Communication*, pages 94–100, 2012.

[89] Sungjoon Choi, Eunwoo Kim, and Songhwai Oh. Real-time navigation in crowded dynamic environments using gaussian process motion control. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 3221–3226. IEEE, 2014.

[90] Pete Trautman, Jeremy Ma, Richard M Murray, and Andreas Krause. Robot navigation in dense human crowds: Statistical models and experimental studies of human–robot cooperation. *The Int. J. of Robotics Research*, 34(3):335–356, 2015.

[91] Samuel Barrett and Peter Stone. Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork. In *Proc. AAAI Conf. on Artificial Intelligence*, 2015.

[92] Samuel Barrett, Peter Stone, Sarit Kraus, and Avi Rosenfeld. Teamwork with limited knowledge of teammates. In *Proc. AAAI Conf. on Artificial Intelligence*, 2013.

[93] Jacob T Schwartz and Micha Sharir. On the piano movers' problem: Iii. coordinating the motion of several independent bodies: the special case of circular bodies moving amidst polygonal barriers. *The Int. J. of Robotics Research*, 2(3):46–75, 1983.

[94] Sujeong Kim, Stephen J. Guy, Wenxi Liu, Rynson WH Lau, Ming C. Lin, and Dinesh Manocha. Predicting pedestrian trajectories using velocity-space reasoning. In *Algorithmic Foundations of Robotics X*, pages 609–623. Springer, 2013.

[95] Daniel Hennes, Daniel Claes, Wim Meeussen, and Karl Tuyls. Multi-robot collision avoidance with localization uncertainty. In *Proc. Int. Conf. on Autonomous Agents and Multi-Agent Systems*, 2012.

[96] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *The Journal of Machine Learning Research*, 3:397–422, 2003.

[97] Aurélien Garivier and Eric Moulines. On upper-confidence bound policies for switching bandit problems. In *Algorithmic Learning Theory*, pages 174–188. Springer, 2011.

[98] Herbert Robbins. Some aspects of the sequential design of experiments. In *Herbert Robbins Selected Papers*, pages 169–177. Springer, 1985.

[99] Martin Nowak and Karl Sigmund. A strategy of win-stay, lose-shift that outperforms tit-for-tat in the prisoner's dilemma game. *Nature*, 364(6432):56–58, 1993.

[100] Martin A Nowak. Five rules for the evolution of cooperation. *Science*, 314(5805):1560–1563, 2006.

[101] Jacob Mattingley et al. Receding horizon control. *IEEE Control Systems*, 31(3):52–65, 2011.

[102] Jens Clausen. Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen*, pages 1–30, 1999.

[103] Stephen J Guy, Sujeong Kim, Ming C Lin, and Dinesh Manocha. Simulating heterogeneous crowd behaviors using personality trait theory. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 43–52. ACM, 2011.

[104] Willow Garage. Turtlebot. *Website: http://turtlebot. com/last visited*, pages 11–25, 2011.

[105] Natural Point. Optitrack. *Natural Point, Inc.,[Online]. Available: http://www. naturalpoint. com/optitrack/.[Accessed 22 2 2014]*, 2011.