Online Learning and Planning for Crowd-aware Service Robot Navigation

Anoop Aroor
The Graduate Center, City University of New York

How does access to this work benefit you? Let us know!
Follow this and additional works at: https://academicworks.cuny.edu/gc_etds
Part of the Artificial Intelligence and Robotics Commons

Recommended Citation
https://academicworks.cuny.edu/gc_etds/3123

This Dissertation is brought to you by CUNY Academic Works. It has been accepted for inclusion in All Dissertations, Theses, and Capstone Projects by an authorized administrator of CUNY Academic Works. For more information, please contact deposit@gc.cuny.edu.
ONLINE LEARNING AND PLANNING FOR CROWD-AWARE SERVICE ROBOT NAVIGATION

by

Anoop Aroor

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

2019
Online learning and planning for crowd-aware service robot navigation

by

Anoop Aroor

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirements for the degree of Doctor of Philosophy.

Date

Susan L. Epstein
Chair of Examining Committee

Date

Robert Haralick
Executive Officer

Supervisory Committee:

Ioannis Stamos

Scott Dexter

Elizabeth Sklar

THE CITY UNIVERSITY OF NEW YORK
Abstract

Online learning and planning for crowd-aware service robot navigation

by

Anoop Aroor

Advisor: Susan L. Epstein

Mobile service robots are increasingly used in indoor environments (e.g., shopping malls or museums) among large crowds of people. To efficiently navigate in these environments, such a robot should be able to exhibit a variety of behaviors. It should avoid crowded areas, and not oppose the flow of the crowd. It should be able to identify and avoid specific crowds that result in additional delays (e.g., children in a particular area might slow down the robot), and to seek out a crowd if its task requires it to interact with as many people as possible. These behaviors require the ability to learn and model crowd behavior in an environment. Earlier work used a dataset of paths navigated by people to solve this problem. That approach is expensive, risks privacy violations, and can become outdated as the environment evolves. To overcome these drawbacks, this thesis proposes a new approach where the robot learns models of crowd behavior online and relies only on local onboard sensors. This work develops and tests multiple planners that leverage these models in simulated environments and demonstrate statistically significant improvements in performance. The work reported here is applicable not only to navigation to target locations, but also to a variety of other
services.
Acknowledgements

I would like to express my sincere gratitude to my advisor Prof. Susan L. Epstein for her continuous support, mentorship, and advice. I could not have imagined having a better advisor and mentor for my Ph.D study. Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Elizabeth Sklar, Prof. Ioannis Stamos, and Prof. Scott Dexter, for their insightful comments, encouragement, and critiques.

I thank all my fellow labmates specially Raj Korpan and Matthew Evanusa for wide ranging discussions, for listening to my off-the-cuff ideas and for being overall superb colleagues. Last but not least, I would like to thank my family: my parents and my sister for supporting me in every possible way.

Thank you all!
Contents

1 Introduction .............................................................. 1
   1.1 Foundations ..................................................... 3
   1.2 Research questions and principal results ...................... 4

2 Related work .......................................................... 8
   2.1 Local collision avoidance during plan execution ............. 8
      2.1.1 Plan repair .................................................. 9
      2.1.2 Replanning .................................................. 10
   2.2 Planning to avoid obstacles ...................................... 12
   2.3 Online learning .................................................. 13
      2.3.1 Markov decision processes ................................ 13
      2.3.2 Reinforcement learning (RL) ............................... 17
      2.3.3 Multi-arm bandits (MAB) .................................. 20

3 System design and experimental setup .............................. 24
   3.1 SemaFORR ....................................................... 24
   3.2 MengeROS: A robot crowd simulation environment .......... 30
      3.2.1 The Menge crowd simulator ................................ 31
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.2 MengeROS</td>
<td>32</td>
</tr>
<tr>
<td>3.2.3 Performance of MengeROS</td>
<td>34</td>
</tr>
<tr>
<td>3.3 Experimental design</td>
<td>34</td>
</tr>
<tr>
<td>3.3.1 Robot tasks and environments</td>
<td>35</td>
</tr>
<tr>
<td>3.3.2 Crowd simulation scenarios</td>
<td>36</td>
</tr>
<tr>
<td>3.4 Evaluation metrics</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Crowd-aware navigation</td>
<td>44</td>
</tr>
<tr>
<td>4.1 Learning a crowd density map online</td>
<td>44</td>
</tr>
<tr>
<td>4.2 Learning a crowd flow model online</td>
<td>47</td>
</tr>
<tr>
<td>4.3 Online crowd-aware planning</td>
<td>49</td>
</tr>
<tr>
<td>4.4 Evaluation of CSA* and Flow-A*</td>
<td>51</td>
</tr>
<tr>
<td>4.4.1 Fourth-floor experiments</td>
<td>51</td>
</tr>
<tr>
<td>4.4.2 Fifth-floor experiments</td>
<td>53</td>
</tr>
<tr>
<td>4.4.3 Conclusion</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>5 Risk-aware navigation</td>
<td>57</td>
</tr>
<tr>
<td>5.1 Robot-specific behaviors in MengeROS</td>
<td>58</td>
</tr>
<tr>
<td>5.2 Learning a risk map</td>
<td>58</td>
</tr>
<tr>
<td>5.3 Risk-A*</td>
<td>60</td>
</tr>
<tr>
<td>5.4 Evaluation</td>
<td>62</td>
</tr>
<tr>
<td>5.4.1 Fourth-floor experiment</td>
<td>62</td>
</tr>
<tr>
<td>5.4.2 Fifth-floor experiment</td>
<td>65</td>
</tr>
</tbody>
</table>
CONTENTS

6 Change detection for crowd models 68
   6.1 Discounting ........................................................................ 69
   6.2 Background: CUSUM .......................................................... 69
   6.3 CUSUM-A* ....................................................................... 70
   6.4 Evaluation ......................................................................... 72
       6.4.1 Fourth-floor experiment .............................................. 72
       6.4.2 Fifth-floor experiment ................................................ 76

7 Planning under uncertainty 79
   7.1 Explore-A* ......................................................................... 79
   7.2 Evaluation ......................................................................... 82

8 Planning for availability 87
   8.1 Help-A* ............................................................................. 88
   8.2 Evaluation ......................................................................... 89

9 Conclusion 93
   9.1 Limitations ........................................................................ 93
   9.2 Future work ....................................................................... 94
   9.3 Summary .......................................................................... 95
List of Figures

1.1 Example scenario .................................................. 2

2.1 D* updates .......................................................... 11

2.2 Common sequential-decision making strategies for robot navigation ........ 14

2.3 POMDP [Kaelbling et al., 1998], page 106, Figure 2 ......................... 15

3.1 SemaFORR’s decision cycle ......................................... 25

3.2 Spatial models for two simple environments ............................... 28

3.3 (left) Aerial view of a simple world with 1 robot and 14 pedestrians. (right)
Corresponding robot rangefinder readings. ..................................... 33

3.4 A trade show world with 1000 pedestrians and 20 robots in a row at the lower
right. ............................................................................. 33

3.5 The fourth floor with elevators at E and six targets ............................ 35

3.6 The fifth floor with elevators at E and four targets ............................ 36

3.7 Two crowds with spatially distinct patterns of movement on the fourth floor. 37

3.8 Two crowds with spatially distinct patterns of movement on the fifth floor.. 37

3.9 Crowd behavior used to evaluate CUSUM-A* on the fourth floor (left) initially
(right) after 20 tasks. ......................................................... 38

x
LIST OF FIGURES

3.10 Crowd behavior used to evaluate CUSUM-A* on the fifth floor (left) initially
(right) after 20 tasks. ................................................................. 38
3.11 Crowd behavior used to evaluate Explore-A* on the fourth-floor (left) initially
(right) after 20 tasks. ................................................................. 40
3.12 Robot’s initial location and pedestrian movement in the fourth floor (left) and
fifth floor (right) of MOMA ........................................................... 41

4.1 (Repeated from Figure 3.5) The fourth floor with elevators at E and six tar-
ggets. Crowd flow indicated by red arrows ....................................... 51
4.2 Plans in the fourth floor to move to target 4. ................................. 52
4.3 (Repeated from Figure 3.6) The fifth floor with elevators at E and four targets.
Crowd flow indicated by red arrows ................................................. 54
4.4 Plans in the fifth floor to move to target 3. .................................... 55

5.1 (Repeated from Figure 4.1) The fourth floor with elevators at E and six targets 63
5.2 All plans generated by A*, CSA* and Risk-A* in the fourth-floor environment.
Arrows describe crowd behavior. (See text.) ................................. 64
5.3 (Repeated from Figure 4.3) The fifth floor with elevators at E and four targets 65
5.4 All plans generated by A*, CSA* and Risk-A* in the fifth-floor environment.
Arrows describe crowd behavior. (See text.) ................................. 66

6.1 (Repeated from Figure 4.1) The fourth floor with elevators at E and six targets 73
6.2 (left) On task 10, plans generated by A*, CSA* and CUSUM-A* in the fourth floor. (right) On task 33, plans after the crowd changes at task 20. Arrows describe crowd behavior. 74

6.3 (Repeated from Figure 4.3) The fifth floor with elevators at E and four targets 76

6.4 (left) On task 13, plans generated by A*, CSA* and CUSUM-A* in the fifth floor. (right) On task 32, plans after the crowd changes on task 20. Arrows describe crowd behavior. 77

7.1 fourth-floor environment with elevators at E, and two patterns for pedestrian 82

7.2 All plans generated by CSA*(0.33) (left) and CSA*(0.66) (right) in the fourth floor. 83

7.3 All plans generated by CSA* (left), and CUSUM* (right) in the fourth floor. 84

7.4 All plans generated by Explore-A* in the fourth floor. 84

8.1 (Repeated from Figure 3.7) Robot’s initial location and pedestrian movement in the fourth floor (left) and fifth floor (right) of MOMA 90
## List of Tables

2.1 Classification of related work ........................................... 23

3.1 SemaFORR’s Advisors and their rationales .............................. 26

4.1 CSA* and Flow-A* improve navigation performance in the fourth-floor map 53

4.2 CSA* and Flow-A* improve navigation performance in the fifth-floor map 56

5.1 Risk-A* improves navigation performance in the fourth floor ........ 63

5.2 Risk-A* improves navigation performance in the fifth floor .......... 67

6.1 CUSUM-A* improves navigation performance in the fourth floor .... 75

6.2 CUSUM-A* improves navigation performance in the fifth floor ....... 78

7.1 Explore-A* improves navigation performance in the fourth floor compared to CUSUM-A* and to CSA* with/without discounting ............. 85

8.1 Help-A* receives more commands on MOMA’s fourth floor ........ 91

8.2 Help-A* receives more commands on MOMA’s fifth floor ............ 91
# List of Algorithms

1. Learn crowd density map for CSA* ........................................... 47
2. Learn crowd flow map for Flow-A* ........................................... 49
3. Learn risk map for Risk-A* ...................................................... 61
4. Learn crowd density map with CUSUM ..................................... 72
5. Planning with Explore-A* ....................................................... 82
Chapter 1

Introduction

Mobile robots are increasingly deployed in indoor environments that are not specifically designed for them, including hospitals, shopping malls, and homes. The International Organization for Standardization defines a service robot as a robot that performs useful tasks for humans or equipment, excluding industrial automation applications (ISO 8373). Such service robots are expected to help non-expert users automate manual tasks, such as vacuuming or delivery. This work focuses on the unique challenges of service robot navigation in large, crowded indoor environments.

The thesis of this work is that the ability to learn a global crowd behavior model online is essential to the wide deployment of service robots. An accurate global crowd model can reduce human-robot collisions, encourage conformation to social norms of navigation, and support robots as they solve navigation tasks that involve crowds.

Navigation can be effective if the robot knows the crowd behavior both locally (in the vicinity of the robot) and globally (throughout the entire environment). With that knowledge, a robot can choose travel routes that reduce human-robot collisions and/or improve the robot’s travel efficiency. Crowd knowledge also provides the ability to travel though routes that conform to social norms. For example, a robot can choose routes that allow it to move
with the flow of the crowd as opposed to against it, and thereby cause less inconvenience
to the people in its environment. Moreover, many new navigation problems can be defined
based on crowd behavior. For example, it would be useful for a robot museum guide to move
to an area of the environment where it is most visible to people, or where most people are
nearby. As another example, a telepresence robot in a conference might be instructed to
move through the environment to maximize the likelihood of meeting a particular person.

Figure 1.1, is an example of a robot that could reach its target along a path through a less
crowded region. If the robot uses only its sensor information, it might choose an uninformed
plan because it lacks information about the crowd outside its limited sensor range. A robot
that predicts the density of people in any contiguous part of the environment, however, can
estimate that travel under the uninformed plan could be shorter but increase the likelihood
of human robot collision. Instead, the robot could choose the crowd-aware route to the
target and thus be more likely to reach the target with fewer collisions.

Previous approaches used crowd trajectory datasets. Overhead cameras recorded pedes-
trians’ continuous movement in a given environment, and individual trajectories were ex-
tracted from video recording. The collection of such trajectories in a given environment can
be used to learn a model of crowd movement before the deployment of the robot in the envi-
ronment [Alahi et al., 2016, Kim and Pineau, 2016, Kretzschmar et al., 2016]. This approach
has many limitations, however. Such datasets are not available in every environment, either because it is too expensive to install additional sensors and monitor the entire environment, or because privacy and data ownership issues prevent data access. Even if a dataset were available, crowd behavior in an indoor environment evolves with changes in the setting itself. For example in a museum, creation of a new exhibit might completely alter the flow of the crowd. In this thesis, I present algorithms that learn models of crowd behavior online, and test their effectiveness in a simulated environment.

The following section briefly describes the basic terminology in robot navigation. The final section summarizes the research questions and principal results of this thesis.

1.1 Foundations

*Navigation* is the ability to know one’s location in an environment and move to a specified \((x, y)\) coordinate location (a *target*). Increasingly, mobile ground-based robots navigate *autonomously*, that is, without human intervention. Examples include vacuum cleaners [Elara et al., 2014] and office assistants [Veloso et al., 2015]. *Robot sensors* are devices that measure quantities in the real world and convert them into digital information. Existing state-of-the-art robots, when told to go from one location to another, use sensors to detect the movement of people in their vicinity and use that data to avoid collisions.

A *map* is a representation of the static spatial features in the environment that are useful for navigation. *Mapping* uses sensor information to generate a map. The \((x, y)\) coordinates for the position of the robot along with its orientation is called its *pose*. *Localization* is the ability to detect one’s pose in the environment. Simultaneous localization and mapping (*SLAM*) maps an environment as it continuously detects the robot’s pose in the environment. Advances in SLAM now allow a robot to build an accurate map of an unknown environment using its sensors [Montemerlo and Thrun, 2007]. This work therefore assumes that the robot
has access to the map of the environment, and that the robot can localize using SLAM.

The standard navigation task for a robot is to reach a target on the map. Before it begins to move, the robot can use a map to generate a plan, a sequence of coordinate points \((x, y)\) on the map that it could move through to reach a target. Each \((x, y)\) in a plan is called a waypoint. Given a plan, the robot issues one or more commands to its actuators (motors that cause physical motion) intended to move the robot to the first waypoint in its plan.

### 1.2 Research questions and principal results

This section lists the research questions (RQs) addressed in this thesis, the principal results associated with them, and the papers that published those results.

**RQ 1. How to design a simulation test bed for crowd-aware navigation algorithms?**

I have implemented a version of SemaFORR [Epstein et al., 2012, Epstein et al., 2015a], a cognitively-inspired robot navigation controller in ROS [Quigley et al., 2009]. SemaFORR uses common sense heuristics and spatial models to generate effective decision making that is both satisficing and interpretable. SemaFORR is the platform for this thesis, and is described in Section 3.1.

I also designed and implemented in ROS a novel robot-crowd simulation environment called MengeROS [Aroor et al., 2017] that allows simultaneous simulation of crowds and robots in large 2-D environments. This work is described in Section 3.2. This work is also described in the following papers.

CHAPTER 1. INTRODUCTION


RQ 2. How can a robot navigate with online learning to avoid partially observable, predictable crowds?

I developed CSA*, an algorithm that learns crowd patterns online from laser scanner data and then avoids crowded areas that are likely to cause collision. I developed Flow-A*, an algorithm that learns and incorporates the direction of crowd flow in the environment to allow the robot to follow social norms (avoid going against the flow of the crowd). Chapter 4 describes this work [Aroor and Epstein, 2017a].

People are known to change their behavior in the presence of robots [Nomura et al., 2015, Trautman and Krause, 2010]. In such scenarios, the learned model should account for such interactions. In my algorithm, Risk-A*, the robot combines the crowd data from its laser scanner with its actual travel experience through a crowded area to build a hybrid model [Aroor et al., 2018]. Risk-A* improves safety. Chapter 5 describes this work. The results are also published in the following papers.


CHAPTER 1. INTRODUCTION

RQ 3. How can a robot navigate with online learning to avoid dynamic, partially observable crowds?

To detect and adjust for temporal changes in crowd patterns, I use discounting and CUSUM [Aroor et al., 2018], a statistical change detection technique. These methods allow my learning algorithms to forget stale models and learn new ones that reflect the current state of the world, as described in Chapter 6. The results are also published in the following paper.


In the methods noted above, a generated route does not deliberately gather useful information about the crowd. In Chapter 7, Explore-A* models this as a multi-arm bandit problem and shows that the application of Thompson sampling improves performance. So supported, the robot chooses routes that not only reduce travel time and distance but also afford opportunities to improve learning.

RQ 4. How can a robot navigate with online learning to move toward partially observable crowds?

Chapter 8 describes Help-A*, which demonstrates the application of online learning of crowd patterns in a different task, where the robot, while it visits a sequence of targets, must also place itself close enough to receive voice commands from people.

This chapter has described the motivations behind this work, listed the research questions addressed and summarized the results. The next chapter reviews the related literature on
crowd-aware navigation.
Chapter 2

Related work

This chapter reviews collision avoidance and planning for robot navigation in environments with dynamic obstacles. There is a large body of work on collision avoidance and navigation for multi-robot teams that require coordination or communication. This work focuses on single-robot systems where the robot has no access to the decision making behavior of other agents in the environment. Collision avoidance for a single robot in dynamic environments can be addressed in either the planning phase or in the plan execution phase.

2.1 Local collision avoidance during plan execution

Many existing approaches treat pedestrians as dynamic obstacles [Sun et al., 2014, Hennes et al., 2012, Savkin and Wang, 2014]. It is typically assumed that the robot can sense the position and the velocity of any obstacle in its vicinity. In such settings, there are three broad approaches to avoid dynamic obstacles: plan repair, replanning, and local collision avoidance.
2.1.1 Plan repair

A* is the most common approach to search for a least-cost path from one location to another [Nilsson, 2008]. A* represents the start node as the initial state and the goal node as the goal state. It also assumes that the robot is a single point in space and hence ignores problems of motion planning for solid-bodied robots in constrained spaces. A* estimates the total cost from the initial state to the goal state as the actual cost from the initial state to the current state plus a heuristic estimate of the cost from the current state to the goal state. The algorithm returns an optimal path from the start node to the end node as long as the heuristic always underestimates the true distance to the goal state. The optimal path is then treated as a plan that a robot can follow to reach its target. A* search keeps a queue of states to be explored, and explores the states with the lowest total estimated cost first.

On large graphs, A* consumes considerable memory, because the number of states in its queue grows exponentially. In contrast, Iterative Deepening A* (IDA*) only requires memory linear in the length of the solution that it constructs [Korf, 1985]. IDA* is an iterative algorithm that begins with an initial cutoff for the total estimated cost of the solution. Each iteration begins with a depth-first search for a solution whose estimated total cost is less than the cutoff. If a solution is not found, the cutoff is incremented and depth-first search is repeated.

One way to repair a plan is to use local information about dynamic obstacles to diverge minimally from it. How to minimize the cost of divergence is treated as an optimization problem [Guy et al., 2009]. Other approaches include the use of reactive rules [Sun et al., 2014] or potential fields [Khatib, 1986]. Although such methods are computationally fast, they suffer from local minima [Koren and Borenstein, 1991].

A common approach to crowds initially generates a global plan that ignores them, and then adjusts that path locally with human-aware planners [Khambhaita and Alami, 2017].
CHAPTER 2. RELATED WORK

Given a graph of nodes that represent locations and edge weights that represent distances, the A* algorithm finds a shortest path [Hart et al., 1968]. Because it ignores the costs of navigation through a crowd, however, such a plan may prove globally inefficient.

Many recent methods make local, short-term predictions about the future trajectories of moving obstacles and use them to improve collision avoidance. These approaches have predicted human trajectories and planned a path around them with a Gaussian process [Trautman and Krause, 2010], with neural networks [Alahi et al., 2016], or with bio-mechanical turn indicators [Unhelkar et al., 2015]. One local path planner learned reward functions on data from human experts who controlled the robot [Kim and Pineau, 2016]. Another approach used pedestrian trajectory datasets to learn a model that jointly predicted the trajectories of both a robot and nearby pedestrians, and then generated socially compliant paths [Kretzschmar et al., 2016].

2.1.2 Replanning

In replanning, dynamic obstacles are treated as changes in the map. When dynamic obstacles block a robot’s path, the plan becomes infeasible. The system incorporates this information about the dynamic obstacles into the map and replans to generate a new feasible plan. The use of A* to replan every time a plan fails is computationally expensive in a large map. Many incremental search algorithms that address the need for fast replanning have been proposed, including D* [Stentz, 1994], D* Lite [Koenig and Likhachev, 2002], focused D* [Stentz et al., 1995], LPA* [Koenig et al., 2004], and MPGAA* [Hernández et al., 2015].

An incremental search algorithm assumes that the unknown parts of the map have no obstacles, and finds a shortest path to the goal node under this assumption. The robot then follows that path. When the robot observes new parts of the map or a new obstacle, it adds this information to the map and, if necessary, generates a new shortest path to the goal node from the current position. Generation of the new path is more efficient because it uses
information from the earlier search, as described next.

![Diagram of D* updates](image)

Figure 2.1: D* updates

D* search starts from the goal node and explores until the robot’s start node is reached. D* search maintains a list of nodes to be evaluated, along with the shortest distance from each node to the goal node. The list begins with the goal node. In every iteration, D* chooses the node closest to the goal node from the list and adds its neighbors to the list. When the start node is added to the list, the search stops, and the path to the goal can be found by backtracking. The robot then follows that path to the goal. When the robot encounters a new obstacle, cost updates are propagated only from the position of the obstacle to the robot. D* updates, as shown in Figure 2.1, are efficient because most changes to the map are detected near the robot, and the updates move backward from the place where the obstacle was detected to the robot’s current position. This requires fewer cost updates. Focused D* improves upon D* with a heuristic that focuses the propagation of cost changes toward the robot.

LPA* returns the shortest path in a graph, from a fixed start node to a fixed end node while the edge weights of the graph are allowed to change. LPA* is similar to A* search. Each node in LPA* uses heuristics to maintain an estimate of the path length through that node. As the edge costs change, however, these estimates are updated and the updated estimates are used to re-compute the plan. D* Lite is a modified version of LPA* where the robot’s position is represented by the start node, the target is represented by the goal node,
and as the robot moves, the edge weights of the graph change and the start node changes to represent the current position of the robot.

Multipath generalized adaptive A* (MPGAA*) is an empirically faster yet simpler algorithm than D* Lite. MPGAA* initially finds an optimal path from the robot to the goal with A*. When the robot’s environment changes, MPGAA* efficiently replans in two ways. First, it uses the information from the initial A* search procedure to improve the heuristic estimate to the goal. This makes subsequent searches faster. Second, it saves all previous optimal paths and tries to reuse them instead of repeatedly replanning.

Both plan repair and replanning improve collision avoidance and navigation, but they are restricted to local detection of obstacles to trigger collision avoidance behavior. In contrast, the next section reviews a complementary problem: how to learn global navigation costs in a given crowded environment and then use them to improve global path planning, so that instead of triggering collision avoidance behavior during plan execution, the robot can generate path plans that avoid crowded areas.

### 2.2 Planning to avoid obstacles

Other work has made global, long-term predictions about the behavior of a crowd, and adapted navigation behavior accordingly, typically with end-to-end pedestrian trajectories. One approach treated a single trajectory as a Markov decision process, learned a distribution over trajectories, applied inverse reinforcement learning to find the reward function that best fit those trajectories, and used it to predict new ones [Ziebart et al., 2009]. Another approach used an end-to-end simulated pedestrian trajectory dataset to initialize a Gaussian-process-based model, updated it from local sensor observations, and then used inverse reinforcement learning to make the robot’s behavior more human-like [Henry et al., 2010].

This is impractical or infeasible in many environments. Instead, my work considers how
a robot limited to only an onboard 2D range sensor can learn a cost map, a grid-based spatial model of global navigation costs that incorporates the density and flow of crowds in indoor environments. The approaches proposed in this thesis learn online and do not require end-to-end pedestrian trajectory datasets.

2.3 Online learning

This section reviews different classes of sequential decision-making methods for robot navigation in changing environments. Figure 2.2 diagrammatically summarizes the scenarios under which these classes of sequential decision-making methods are most applicable. The focus is on Markov decision processes, reinforcement learning and multi-arm bandits.

2.3.1 Markov decision processes

The uncertainties in many environments can be modeled. For example, when a robot’s actuators are imperfect, the outcome of its chosen action is uncertain. One can model this uncertainty, however, as a Gaussian distribution. In such scenarios the state of a robot after an action can be represented as a probability distribution over the possible states. A Markov decision process (MDP) can be used to model such domains [Bellman, 1957]. An MDP can be described as a tuple $\langle S, A, T, R, \pi \rangle$, where:

- $S$ is a finite set of states of the world. A continuous world can be represented as a finite set of states by discretization.
- $A$ is a finite set of actions.
- $T : S \times A \to [0, 1]$ is a state-transition function, which for every state $s \in S$ and action $a \in A$ returns a probability distribution over the world states.
Figure 2.2: Common sequential-decision making strategies for robot navigation

- $R : S \times A \rightarrow R$ is the reward function that returns a reward when a robot chooses an action in a specific state.

- $\pi : S \rightarrow A$ is policy function that specifies the action to be chosen in any given state.

The objective of an MDP framework $\langle S, A, T, R, \pi \rangle$ is to find an optimal policy $\pi$ that results in the maximum total reward under $R$. An MDP framework, however, assumes that the world is completely observable (i.e., that the robot can accurately detect its current state). In many domains, a robot with noisy sensors may not able to determine its current state with complete reliability. A partially observable MDP (POMDP) provides a framework to find optimal policies in such domains [Kaelbling et al., 1998]. A POMDP can be described...
as a tuple \( \langle S, A, T, R, \pi, O, Of \rangle \) where

- \( S, A, T, R \) and \( \pi \) describe an MDP.
- \( O \) is a finite set of observations the robot can experience through its sensors. A continuous-valued sensor can be discretized to generate a finite observation set.
- \( Of : S \times A \to p(O) \) is the observation function which returns for each action and resulting state a probability distribution over possible observations.

![Figure 2.3: POMDP [Kaelbling et al., 1998], page 106, Figure 2](image)

Because a POMDP does not know the current state accurately, it maintains a probability distribution over the possible current states of the robot. This probability distribution is called the belief state. In Figure 2.3, the belief state \( b \) is estimated by a separate component called the State Estimator (SE) [Kaelbling et al., 1998]. SE generates \( b \) based on the previous action of the agent, the current observation, and the previous belief state, as shown in Figure 2.3. The objective of the POMDP framework is to generate a policy \( \pi \) that suggests actions expected to be optimal from any belief state.

In a POMDP, at (discrete) time step \( t \) the environment is assumed to be in some state \( s_t \). The agent then performs an action \( a_t \), and at the same time the environment (stochastically) changes to a new state \( s_{t+1} \). The agent does not know the true environment state, but instead
receives an observation $o_t$, which is some (stochastic) function of $s_t$. In addition, the agent receives a special observation signal called the reward, $r_{t+1}$.

The goal of the agent is to learn a policy $\pi$ that maps the observation history (trajectory) into an action $a_t$ to maximize the agent’s utility. The utility of a policy can be defined in several ways, one of which is as the expected discounted infinite sum of rewards. For this model, there exists an optimal deterministic stationary policy. The optimal value $V^*(s)$ of a state $s$ is defined as the maximum expected sum of the discounted rewards that the agent can receive if its initial state is $s$ and it executes the optimal policy. Future rewards are discounted by a factor of $\gamma$:

$$V^*(s) = \max_{\pi} E \left( \sum_{t=0}^{\infty} \gamma^t r_t \right)$$ (2.1)

$V^*(s)$ can also be represented recursively as a set of equations where the value of a particular state $s$ is the maximum sum of the immediate reward $R(s, a)$ the agent receives for action $a$ in state $s$ plus the discounted expected value of the next state $s'$ [Bellman and Dreyfus, 2015]:

$$V^*(s) = \max_a \left( R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') \right), \forall s \in S$$ (2.2)

An optimal policy function $\pi^*(s)$ can then be defined in terms of this optimal value function. An optimal policy $\pi^*$ is one whose recommended action $a$ at state $s$ is expected to generate the optimal value for that state. This is expressed mathematically as:

$$\pi^*(s) = \arg \max_a \left( R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') \right), \forall s \in S$$ (2.3)

There are two broad approaches to find the optimal policy: value iteration and policy iteration. Value iteration initializes $V(s)$ arbitrarily and then iteratively updates it with

$$V^{\text{new}}(s) := \max_a \left( R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^{\text{old}}(s') \right), \forall s \in S$$ (2.4)
These updates converge to the correct $V^*(s)$ values [Bellman and Dreyfus, 2015]. The optimal policy is then simply to move to the state with the maximum value function.

While value iteration finds the optimal policy indirectly with the optimal value function, policy iteration manipulates the policy directly. It begins by choosing an arbitrary policy function $\pi$. The value function $V^\pi(s)$ of this arbitrary policy $\pi$ is defined as the expected infinite discounted reward that will be gained if the agent starts from state $s$, and executes $\pi$. The first step of the algorithm attempts to find this value function as:

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s'), \forall s \in S \quad (2.5)$$

This is formulated as a set of linear equations whose solution is the value function for this policy. The new updated policy $\pi'$ is then obtained from the value function for the previous policy as:

$$\pi'(s) := \arg \max_a \left( R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^\pi(s') \right) \quad (2.6)$$

These updates continue until the value function no longer increases.

Both value iteration and policy iteration assume that the transition function $T(s, a, s')$ and the reward function $R(s, a)$ are known. The next section describes reinforcement learning, which learns a policy function without knowledge of the transition function.

### 2.3.2 Reinforcement learning (RL)

When the underlying transition model of the environment is unknown, the robot must either learn a policy without the use of a transition function (model-free) or learn both a policy and a transition function simultaneously (model-based). Model-free approaches learn a policy without explicitly learning the transition function. One class of such algorithms is known as temporal difference methods [Sutton, 1988]. Another class of algorithms is based on Monte-
Carlo sampling. Both are discussed briefly below.

**Q-learning** is a popular temporal difference method. Let \( Q^*(s, a) \) be the expected discounted reward if the agent takes action \( a \) in state \( s \), and then chooses all subsequent actions optimally. Let \( V^*(s) \) be maximum discounted reward if the robot begins from state \( s \). Then \( V^*(s) = \max_a Q^*(s, a) \). Also

\[
Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \max_{a'} Q^*(s', a') \tag{2.7}
\]

The main idea in Q-learning is that maintenance of an estimate for \( Q^*(s, a) \) permits a direct estimate of the optimal policy.

\[
\pi^*(S) = \arg \max_a Q^*(s, a) \tag{2.8}
\]

The Q-learning rule updates the Q-values for an experience \( \langle s, a, s', r \rangle \) with

\[
Q^{\text{updated}}(s, a) := Q(s, a) + \alpha (r + \gamma \max_{a'} Q(s', a') - Q(s, a)) \tag{2.9}
\]

Monte Carlo methods [Singh and Sutton, 1996] learn the value function \( V^\pi(s) \) for a given policy \( \pi \). Monte Carlo methods assume *episodic* tasks, that is, experience is divided into episodes, and all episodes eventually terminate no matter what actions are selected. Value estimates and policies are changed only upon the completion of an episode. Each occurrence of a state in an episode is a *visit* to \( s \). Given a set of episodes obtained by following \( \pi \) and passing through \( s \), the value of a state under policy \( V^\pi(s) \) is the average of the reward after all visits to \( s \) in the set of episodes. Thus, instead of an explicit transition function, the average reward forms a sampled estimate of the value.

To approximate an optimal policy, Monte Carlo methods maintain both an approximate policy and an approximate value function. The initial policy function is evaluated to learn
the value function using the sampled average method. This value function is then used to
generate a new policy which is in turn evaluated. Iteration halts when the value of the policy
no longer improves.

Model-free approaches like Q-learning and Monte Carlo methods learn an optimal policy
without knowledge of the models $T(s, a, s')$ or $R(s, a)$ and without any attempt to learn them
by the agent. They may use available data inefficiently, and so require much data to achieve
good performance. In contrast, model-based methods learn a model of the MDP online while
they interact with the environment, and then use their approximate model to calculate a
policy. If such an algorithm can learn an accurate model quickly enough, a model-based
method can be more sample efficient than model-free methods.

Dyna, a modified version of Q-learning, uses experience to improve the current model ($T$
and $R$), uses the updated model to improve the policy, and then uses the updated policy to
execute an action and generate more experience [Sutton, 1991]. Given an experience tuple,
an initial version of a model, and an intial version of the Q-value function, it behaves as
follows for each tuple:

- Use the information from the tuple about states $s$ and $s'$, action $a$ and the reward $r$ to
  update the model.

- Use the updated model ($T'$ and $R'$) to update the Q-value for state $s$

\[
Q(s, a) := R'(s, a) + \gamma \sum_{s' \in S} T'(s, a, s') \max_{a'} Q(s', a')
\]  

(2.10)

- Choose $k$ state-action pairs at random and update them with

\[
Q(s_k, a_k) := R'(s_k, a_k) + \gamma \sum_{s' \in S} T'(s_k, a_k, s') \max_{a'} Q(s', a')
\]  

(2.11)
• Choose an action $a'$ to perform in state $s'$, based on the updated Q-values and an exploration strategy.

Dyna requires about $k$ times the computation of Q-learning per tuple, but this is typically vastly less than for the naive, model-based method. A reasonable value of $k$ can be determined based on the relative speeds of computation and of taking action.

This method, however defines the model $T$ as a transition function that predicts the next state $s'$ given the current state $s$ and action $a$ as a simple function without any explicit representation. Instead of this simple representation one can think of the transition function as $T(s'|s,a)$, a function that returns the likelihood of $s'$ given $s$ and $a$.

### 2.3.3 Multi-arm bandits (MAB)

In stochastic environments where the underlying model of the environment can be learned but the actions do not influence the future state of the environment, a sequential decision problem can be formulated as a multi-arm bandit problem (MAB). The MAB problem is a classic problem that demonstrates the exploration-exploitation dilemma. Given multiple slot machines where each machine has an unknown probability of a reward on one play, the multi-arm bandit problem is to find best strategy, that is, the sequence of actions (arm pulls) that achieves the greatest long-term reward.

Given $K$ slot machines with reward probabilities, $\theta_1, ..., \theta_K$ and the ability to take an action $a$ at each time step $t$ on one slot machine, to receive reward $r$, a Bernoulli multi-arm bandit problem is described as a tuple $\langle A, R \rangle$ where $R$ is a stochastic reward function and $A$ is a set of actions, each referring to interaction with one slot machine. The value of an action (action value) $a \in A$ is the expected reward, $Q(a) = E[r|a] = \theta$. If action $a$ at time $t$ is on the $i$-th machine, then $Q(a_t) = \theta_i$. At time $t$, $r_t = R(a_t)$ returns reward 1 with a probability $Q(a_t)$ or 0 otherwise.
The goal here is to maximize the cumulative reward $\sum_{t=1}^{T} r_t$. Three common heuristic strategies to maximize the cumulative reward are $\varepsilon$-greedy, upper confidence bound, and Thompson sampling [Thompson, 1933]. The $\varepsilon$-greedy heuristic estimates the action value as the average rewards associated with the target action $a$ that have been observed so far (up to the current time step $t$). A random action is selected with a small probability $\varepsilon$, otherwise the action with the highest action value is chosen. Instead of random exploration, the upper confidence bound (UCB) heuristic prefers actions which do not yet have confident value estimations. For each action, along with the averaged action value, the confidence interval of the action value is estimated. The action with the highest upper confidence value is chosen. This ensures that given two actions with equal averaged action values, UCB prefers action with higher variance in the action value estimate. The Thompson-sampling heuristic models each action-value as a random variable whose distribution is learned over multiple rewards. A sample from each action-value distribution is drawn, and the action with the maximum value in its sample is chosen. This ensures that an action with higher variance is explored more often. Although both UCB and Thompson sampling are known to perform well [Chapelle and Li, 2011], an important difference between them is that Thompson sampling takes the entire distribution of each arm into consideration instead of only the upper confidence bound. Thompson sampling is also more flexible, because it can incorporate prior knowledge about the arms, unlike UCB which begins by treating all arms equally.

In summary, this section has described the related work and motivated the assumptions made in this thesis. Table 2.3.3 classifies the related work along the following dimensions.

- **Robot team**: A multi-robot team whose robots can communicate and coordinate with each other or a single robot system that can only observe dynamic obstacles

- **Collision avoidance**: The robot avoids collision locally after plan generation or plans to avoid the crowd globally
• **Observability**: The robot has either full or partial observability of the environment

• **Learning**: The robot does not learn about the behavior of the obstacles in the environment, learns offline, or learns online

• **Datasets**: The robot does or does not use end-to-end pedestrian datasets

To the best of my knowledge there is no work in the current literature that addresses the problem of online learning and planning to avoid crowds for a single robot at a global level without making the assumption of full observability or assuming access to an end-to-end pedestrian dataset. These requirements are particularly useful in service-robot settings. Learning a global model of obstacle behavior is useful when the obstacles have predictable patterns, such as crowding in indoor environments. Assuming partial observability makes the service robot less expensive because it avoids additional sensor costs. Learning from end-to-end pedestrian datasets is problematic because of privacy concerns and because changes in crowd behavior cannot be effectively captured. The next chapter describes the experimental setup, the SemaFORR controller, and the MengeROS simulation environment.
<table>
<thead>
<tr>
<th>Related work</th>
<th>Robot team</th>
<th>Collision avoidance</th>
<th>Observability</th>
<th>Learning</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Guy et al., 2009]</td>
<td>Multi</td>
<td>Local</td>
<td>Partial</td>
<td>No learning</td>
<td>No</td>
</tr>
<tr>
<td>Sun et al., 2014</td>
<td>Multi</td>
<td>Local</td>
<td>Partial</td>
<td>No learning</td>
<td>No</td>
</tr>
<tr>
<td>Sklar et al., 2012</td>
<td>Multi</td>
<td>Local</td>
<td>Partial</td>
<td>Offline</td>
<td>No</td>
</tr>
<tr>
<td>Khatib, 1986</td>
<td>Single</td>
<td>Local</td>
<td>Full</td>
<td>No learning</td>
<td>No</td>
</tr>
<tr>
<td>Koren and Borenstein, 1991</td>
<td>Single</td>
<td>Both</td>
<td>Full</td>
<td>No learning</td>
<td>No</td>
</tr>
<tr>
<td>Khambhaita and Alami, 2017</td>
<td>Single</td>
<td>Local</td>
<td>Partial</td>
<td>Offline</td>
<td>Yes</td>
</tr>
<tr>
<td>Ziebart et al., 2009</td>
<td>Single</td>
<td>Both</td>
<td>Full</td>
<td>Online</td>
<td>No</td>
</tr>
<tr>
<td>Fox et al., 1997</td>
<td>Single</td>
<td>Local</td>
<td>Partial</td>
<td>No learning</td>
<td>No</td>
</tr>
<tr>
<td>Trautman and Krause, 2010</td>
<td>Single</td>
<td>Local</td>
<td>Partial</td>
<td>Offline</td>
<td>Yes</td>
</tr>
<tr>
<td>Alahi et al., 2016</td>
<td>Single</td>
<td>Local</td>
<td>Partial</td>
<td>Offline</td>
<td>Yes</td>
</tr>
<tr>
<td>Unhelkar et al., 2015</td>
<td>Single</td>
<td>Local</td>
<td>Partial</td>
<td>Offline</td>
<td>Yes</td>
</tr>
<tr>
<td>Kim and Pineau, 2016</td>
<td>Single</td>
<td>Local</td>
<td>Partial</td>
<td>Offline</td>
<td>Yes</td>
</tr>
<tr>
<td>Kretzschmar et al., 2016</td>
<td>Single</td>
<td>Local</td>
<td>Partial</td>
<td>Offline</td>
<td>Yes</td>
</tr>
<tr>
<td>Henry et al., 2010</td>
<td>Single</td>
<td>Both</td>
<td>Partial</td>
<td>Both</td>
<td>Yes</td>
</tr>
<tr>
<td>Chen et al., 2019</td>
<td>Single</td>
<td>Local</td>
<td>Partial</td>
<td>Offline</td>
<td>No</td>
</tr>
<tr>
<td>Rashed et al., 2017</td>
<td>Single</td>
<td>Local</td>
<td>Full</td>
<td>Offline</td>
<td>No</td>
</tr>
<tr>
<td>Thrun, 1999</td>
<td>Single</td>
<td>Local</td>
<td>Partial</td>
<td>No learning</td>
<td>No</td>
</tr>
<tr>
<td>Thrun et al., 2000</td>
<td>Single</td>
<td>Local</td>
<td>Partial</td>
<td>No learning</td>
<td>No</td>
</tr>
<tr>
<td>Trautman et al., 2015</td>
<td>Single</td>
<td>Local</td>
<td>Partial</td>
<td>Offline</td>
<td>Yes</td>
</tr>
<tr>
<td>Trautman, 2017</td>
<td>Single</td>
<td>Local</td>
<td>Partial</td>
<td>Offline</td>
<td>Yes</td>
</tr>
<tr>
<td>Vemula et al., 2017</td>
<td>Single</td>
<td>Local</td>
<td>Partial</td>
<td>Offline</td>
<td>Yes</td>
</tr>
<tr>
<td>Our approach</td>
<td>Single</td>
<td>Both</td>
<td>Partial</td>
<td>Online</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 2.1: Classification of related work
Chapter 3

System design and experimental setup

This chapter describes the experimental framework used throughout this thesis including the SemaFORR robot controller, and the MengeROS simulator. It also describes the crowd simulation scenarios and the metrics used to measure the performance of the algorithms.

3.1 SemaFORR

This section describes the background work on SemaFORR that supports this thesis. Two important features of SemaFORR in the context of service robots are its ability to avoid dynamic obstacles and the interpretability of its decision making process. SemaFORR is based on a general decision-making framework called FORR (FOr the Right Reasons) [Epstein, 1994]. FORR’s decision making is based on the idea that efficient decisions in computationally complex domains require judicious integration of multiple simple decision-making procedures. These procedures can range from a reactive rule-based procedure to a deliberative planner. In SemaFORR, these procedures are called Advisors. As in Figure 3.1, SemaFORR’s input includes the actions available to the robot, its pose and current target, and the current laser rangefinder data.
As shown in Figure 3.1 and Table 3.1, Advisors are partitioned into three tiers. This work uses only a subset of the Advisors in SemaFORR. Tier-1 Advisors are reactive and rule-based. Decisions made by tier-1 Advisors are executed immediately. If the state does not match the rules of tier-1 Advisors, control is passed to tier-2. Tier-2 Advisors are planners that recommend a sequence of waypoints. Once a plan is generated it is not discarded until the end of the task, and its waypoints are kept in a plan store. To move to the next waypoint, the robot passes control to tier-3. Tier-3 Advisors are heuristics. Each tier-3 Advisor scores all possible actions according to its heuristic. Voting then combines the scores from all tier-3 Advisors and chooses the action with the highest aggregate score. Thus, once a plan is in
Tier 1, in order

<table>
<thead>
<tr>
<th>Advisor</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Victory</td>
<td>Go toward an unobstructed target</td>
</tr>
<tr>
<td>Enforcer</td>
<td>Go toward an unobstructed waypoint</td>
</tr>
<tr>
<td>AvoidObstacles</td>
<td>Do not go within $\varepsilon$ of an obstacle</td>
</tr>
<tr>
<td>NotOpposite</td>
<td>Do not return to the last orientation</td>
</tr>
</tbody>
</table>

Tier 2 planners

<table>
<thead>
<tr>
<th>Planner</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*</td>
<td>Minimize distance traveled</td>
</tr>
<tr>
<td>CSA*</td>
<td>Avoid crowds</td>
</tr>
<tr>
<td>Flow-A*</td>
<td>Avoid movement against the flow of the crowd</td>
</tr>
<tr>
<td>CUSUM-A*</td>
<td>Avoid dynamic crowds</td>
</tr>
<tr>
<td>Risk-A*</td>
<td>Consider human-robot interaction effects</td>
</tr>
<tr>
<td>Explore-A*</td>
<td>Consider future learning opportunities</td>
</tr>
<tr>
<td>Help-A*</td>
<td>Seek out crowds</td>
</tr>
</tbody>
</table>

Tier 3 heuristics

**Based on commonsense reasoning**

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigStep</td>
<td>Take a long step</td>
</tr>
<tr>
<td>ElbowRoom</td>
<td>Get far away from obstacles</td>
</tr>
<tr>
<td>Explorer</td>
<td>Go to unfamiliar locations</td>
</tr>
<tr>
<td>GoAround</td>
<td>Turn away from nearby obstacles</td>
</tr>
<tr>
<td>Greedy</td>
<td>Get close to the target</td>
</tr>
</tbody>
</table>

**Based on the spatial model**

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convey</td>
<td>Go to frequent, distant conveyors</td>
</tr>
<tr>
<td>Enter</td>
<td>Go into the target’s region via an exit</td>
</tr>
<tr>
<td>Exit</td>
<td>Leave a region without the target via an exit</td>
</tr>
<tr>
<td>Unlikely</td>
<td>Avoid leaf regions that do not contain the target</td>
</tr>
<tr>
<td>Trailer</td>
<td>Use a trail segment to approach the target</td>
</tr>
</tbody>
</table>

Table 3.1: SemaFORR’s Advisors and their rationales.

place, for each execution cycle, either tier-1 (based on rules) or tier-3 (based on multiple heuristics that attempt to reach the next waypoint) makes a decision that is forwarded for execution.

Tier-1 Advisors are pre-sequenced and assume perfect knowledge. They are intended to be fast and correct. Each Advisor can either choose an action to execute or eliminate actions from further consideration. **Victory** is a tier-1 Advisor; it chooses the action that gets the robot closest to a target within sensory range when no obstacles block the robot’s path. If the robot has a plan to reach the target, at least one of its unvisited waypoints is
within sensory range, and no obstacles block the robot’s path there, Enforcer chooses the action that best approaches the waypoint that most furthers the robot’s progress along its plan. Otherwise, AvoidObstacles, another tier-1 Advisor, eliminates actions that would cause a collision. It uses the laser rangefinder data to remove actions that would bring it too close to static or dynamic obstacles. If only one action remains, it is returned. Otherwise, SemaFORR forwards the remaining actions to the next tiers.

Tier-2 Advisors are deliberative planners. When control reaches tier-2, if there is no current plan, the user-specified planner generates one and forwards it to the plan store and the cycle ends. Otherwise, if a plan is already in place, SemaFORR forwards control to tier 3, which selects an action from the remaining, collision-free, plan-compliant actions.

Each tier-3 Advisor makes heuristic recommendations based on its own rationale. Given the current plan, tier-3 Advisors treat the next waypoint as if it were the target. SemaFORR has five tier-3 Advisors based on commonsense reasoning. Greedy prefers actions that move the robot closer to that waypoint. Explorer prefers actions that keep the robot away from previously visited areas. Bigstep prefers actions that make the robot take long steps. GoAround prefers actions that make the robot turn away from nearby obstacles. ElbowRoom prefers actions that move the robot away from obstacles. To express its preferences, a tier-3 Advisor assigns a numeric value to each possible action that survived tier 1. A voting mechanism aggregates the preferences of all tier-3 Advisors and returns the most preferred action.

SemaFORR also learns features of the environment that facilitate decision making. For this thesis, SemaFORR learns spatial features of an indoor environment, such as regions, conveyors and trails [Epstein et al., 2015c]. This spatial model is learned from the sensor input and travel history of the robot. Each Advisor can use any number of spatial features to make a decision.

SemaFORR’s spatial model approximately describes a robot’s navigational experience in
its environment, as shown in Figure 3.2. The robot learned each model there over 40 tasks, where a task required the robot to move to a pre-defined target in the map. If the robot never enters a particular area, no spatial model for that area will be included. A region is an area without permanent obstructions. Simulated laser range sensors provide the robot with distances to nearby obstacles in different directions. Wherever the robot senses, it learns a region, represented as the circle centered at the robot’s location with radius equal to the shortest sensed distance. In Figure 3.2, the pink circles denote the regions detected in two simple maps. Where the robot crosses a region’s perimeter, that point becomes an exit from the region. Exits are represented as black dots in Figure 3.2. A region is a leaf region if all its exits lie on a single 180° arc.

A trail is a revision of a path taken by the robot to a target. A path might contain loops or digressions caused by the robot’s decisions on its way to its target. Such unnecessary sections of the path are removed to form the trail. A trail is an approximation of the optimal route between two points. In Figure 3.2, trails are represented as blue lines. Finally, conveyors are small areas regularly used in trails. A conveyor represents a useful, target-independent transit point. Each conveyor has a counter for the number of times a robot used that area.
to reach its target. In Figure 3.2, conveyors are represented as green grid cells, where darker cells represent more frequently visited conveyors.

SemaFORR has multiple tier-3 Advisors that exploit its learned spatial model. The Advisor CONVEY supports moves to conveyors that have a high counter value, with preference for those further from the robot. CONVEY thus advises the robot to move into areas of the map that have been visited frequently in previous successful travel. Preference for more distant conveyors indirectly promotes travel through the conveyors with high counter values rather than merely to them. If the robot is in region $R$, and if the target is in region $T$, the Advisor UNLIKELY opposes actions into a leaf region other than $T$. Another Advisor, EXIT, supports actions toward exits from $R$ unless $R$ is the same as $T$. ENTER supports actions toward $T$. For further details on additional Advisors and features of the spatial model that are not used in this work, see [Korpan and Epstein, 2019].

My specific contributions to the work described thus far are as follows.

- Migrated SemaFORR to ROS
- Design and implementation of the tier-1 Advisors VICTORY, AVOIDOBSTACLES, and ENFORCER.
- Design and implementation of all tier-2 planners
- Revision and implementation of the basic tier-3 commonsense Advisors BIGSTEP, EXPLORER, and GREEDY
- Design and implementation of modules that learn regions and exits from the paths taken by the robot
- Design and implementation of tier-3 Advisors that make decisions based on regions and exits.
SemaFORR performs local collision avoidance. Once a tier-2 Advisor generates a plan for a target, the plan remains unchanged. If a waypoint is blocked by a moving obstacle and Enforcer is not able to make a decision, the tier-3 Advisors together generate heuristic decisions, one at a time, to steer the robot around the obstacle. This interaction between different tiers of Advisors generates SemaFORR’s plan repair behavior. Such collision avoidance, however, is local in nature and does not consider the global crowd behavior in the environment. As Figure 1.1 shows, this is not always efficient. The next section describes MengeROS, a crowd and robot simulator that supports the testing of various crowd scenarios in real-world maps.

3.2 MengeROS: A robot crowd simulation environment

Before deployment in a crowded environment, a robot’s navigation controller must be tested extensively, particularly to prevent collision. It is challenging, however, to design and execute appropriate, large-scale, real-world testing for robots across the broad range of crowd conditions that arise in service areas. A shopping-mall crowd, for example, varies with the time, the day of the week, and irregularly scheduled events, and each permutation defines a different test. A flexible, accurate crowd simulator is thus essential before deployment. Such a simulator can also support the evaluation of different navigation algorithms under comparable crowd conditions. This section introduces a novel tool, MengeROS, that integrates a flexible, open-source crowd simulator called Menge [Curtis et al., 2016] with ROS, the standard operating system for robots that navigate.

Specialized robot simulators (e.g., Gazebo [Koenig and Howard, 2004] or Stage [Gerkey et al., 2003]) do not simulate realistic crowds. Moreover, most crowd simulators (e.g., PedSim [Helbing, 2012], OpenSteer [Reynolds, 1999], Menge, and Continuum [Treuille et al., 2006]) do not simulate robots. Only two known crowd simulators include robots: PedSim_ros
[Helbing, 2016], which is restricted to a single collision-avoidance model, and a ROS version of Continuum, which is not freely available for research.

MengeROS is not meant to simulate sophisticated full-body human simulation with gestures and expressions. Instead, it is designed to facilitate research in multi-robot path planning problems in large crowded environments. To the best of my knowledge, MengeROS is the only open-source simulator that supports the movement of multiple robots through a broad range of crowd scenarios.

3.2.1 The Menge crowd simulator

Menge’s crowd scenario includes a map of the environment’s static elements (e.g., walls, furniture), the number of simulated pedestrians (pedestrians) in the crowd, their initial locations, along with one decision-making strategy and one collision-avoidance strategy applied to all pedestrians. Menge describes how each pedestrian determines its goals, and how all of them select their next moves and avoid collisions with one another. A location in Menge is either a pair of coordinates \((x,y)\) or a delineated area in the environment (e.g., the kitchen). Goal selection specifies a target sequence (locations to visit) for each pedestrian. Plan computation specifies how Menge calculates the next location for all pedestrians as they move toward their targets. Both of Menge’s implemented methods, A* and potential fields, generate and assign a velocity vector (direction and distance) to each pedestrian. A* pursues an optimal shortest path; potential fields use an attractor mechanism.

Menge’s collision avoidance adjusts each pedestrian’s intended velocity vector to prevent collisions with the other pedestrians. Menge currently has six collision avoidance models. Four of them are based on the social force model [Helbing and Molnar, 1995], where nearby objects or pedestrians attract or repel a pedestrian, whose revised velocity vector reflects the result of all those forces. These models can cause deadlock in large crowd simulations and so were not used in this work. The other two models, ORCA and PedVO, are based
on velocity obstacles. The velocity obstacle (\(\text{VO}\)) of a pedestrian is the set of all velocity vectors that will result in collision. Collision-free motion requires that every pedestrian have a velocity vector outside its VO. To prevent livelock and find an optimal solution, ORCA shares this responsibility equally among all pedestrians. PedVO adapts ORCA to behave more similarly to people.

In Menge, users can select among precoded options for goal selection, plan computation, and collision avoidance, or implement their own. As a result, Menge can simulate many different kinds of crowd scenarios. This flexibility is a significant improvement over earlier crowd simulators, which hardcoded a single approach. Nonetheless, Menge is not available through ROS and does not simulate robots.

### 3.2.2 MengeROS

A typical ROS-based robot navigation framework uses a *simulator node*. This node accepts as input a velocity command in a ROS-specified format, and returns simulated sensor readings (e.g., laser rangefinder data) at a specified frequency. To determine the robot’s motion, a controller node generates velocity commands based on the most recent sensor reading it has received from the simulator node.

MengeROS simulates both robots and pedestrians in a single node. It allows multiple robots to be introduced, each with its own external controller. A robot in MengeROS executes the velocity commands received from its external ROS controller. This is similar to the way other robot simulators (e.g., Gazebo or Stage) interface with ROS. MengeROS controls pedestrian behavior just as Menge does. Pedestrians avoid the robot and one another with the collision avoidance option specified in the Menge control files. A robot, however, is completely dependent on the external commands from its own controller for navigation and collision avoidance.

MengeROS can simulate a laser scanner mounted on a robot. Figure 3.3 shows two
Figure 3.3: (left) Aerial view of a simple world with 1 robot and 14 pedestrians. (right) Corresponding robot rangefinder readings.

Figure 3.4: A trade show world with 1000 pedestrians and 20 robots in a row at the lower right.

Aerial views of a simple world. On the left is the ground truth, with 1 pink robot and 14 green pedestrians. On the right is the robot’s view when it is located at the arrow’s tail and oriented toward its head. Distances to obstacles are reported by a simulated laser with a 220° (configurable) field of view with a maximum (configurable) range of 25 meters. MengeROS returns the positions of all pedestrians and robots in ROS-compatible format, for use by all other ROS nodes.
3.2.3 Performance of MengeROS

MengeROS readily simulates large crowds, including 1000 pedestrians that move simultaneously in Menge’s complex trade show environment, shown in Figure 3.4. Each decision cycle computes and assigns a new velocity vector to every pedestrian. On an 8-core, 1.2 GHz workstation, 100 decision cycles without robots average 51ms each. Because each robot’s range sensors must be processed separately, more robots slows performance. This slowdown appears to be linear in the number of robots. Average decision cycle times with 5, 10, 15, and 20 robots were 437ms, 824ms, 1204ms and 1568ms, respectively. Code for MengeROS along with documentation, examples, and demos are freely available and hosted on GitHub [Aroor, 2017].

3.3 Experimental design

This section describes the experimental design used in the remainder of this thesis. MengeROS is used to simulate a robot with radius of 0.25m and a laser scanner with a 220° field of view and a range of 25m. These values are the specifications for a mobile robot called Fetch [Wise et al., 2016].

MengeROS requires that all pedestrians use the same collision avoidance strategy to avoid one another and the robot. In preliminary work [Aroor and Epstein, 2017b], both ORCA [van den Berg et al., 2011] and PedVO [Curtis and Manocha, 2012] were tested. Because we detected no significant difference in the robot’s behavior or performance on our tasks when confronted by PedVO or ORCA crowds, we tested on pedestrians whose collision avoidance strategy was only ORCA throughout this work.

The simulated robot uses a sequence of commands generated by SemaFORR to navigate. The commands include forward linear moves, clockwise and counterclockwise rotations (turns), and a pause (a no-op). Although the robot could theoretically make a move or turn
of any size, SemaFORR restricts that choice to a discrete set of possibilities. The intensity of a command is an integer that represents, qualitatively, how far the robot is intended to travel or turn. A move has intensity only between 1 and 5; a turn has intensity between 1 and 4, either clockwise or counterclockwise. The robot repeatedly requests a command from SemaFORR until it reaches the target or it exceeds the decision limit of 500 decision cycles. In crowded environments, regions were not used because they were not accurate enough. This is because the region-detection algorithm depends on a laser scanner, which does not have the ability to distinguish static obstacles from dynamic ones.

3.3.1 Robot tasks and environments

In every experiment, the robot is required to visit a predefined sequence of 40 targets. The experiments in Chapters 4, 5, 6 and 7 have the robot do so on the fourth and fifth floors of The Graduate Center of City University of New York. This building occupies an entire Manhattan block (100m by 60). The architectural floor plans for these two floors are shown in Figures 3.5 and 3.6). The experiments in Chapter 8, however, add to navigation an additional goal: the robot is expected to seek out the crowd. These experiments use
architectural floor plans for the fourth and fifth floors of The Museum of Modern Art in New York City, shown in Figure 3.12).

3.3.2 Crowd simulation scenarios

In chapter 4, CSA* generates plans that avoid areas with high crowd density. Flow-A* generates plans that avoid both areas with high crowd density and where a crowd flows in direct opposition toward the robot. The crowd-flow patterns as shown in Figure 3.5 and 3.6 are used in Chapter 4 to test the ability of CSA* and Flow-A*. In the fourth floor of The Graduate Center, a crowd of 60 pedestrians gradually emerges from the elevators (E), and moves along the path described by the red arrows in Figure 3.5. To simulate a steady stream of crowd flow, pedestrians return to the elevators area once they reach the end of the flow path. In the fifth floor of The Graduate Center, a crowd of 60 pedestrians gradually emerges from an elevator, and moves along the hallway that follows the red arrows in Figure 3.6. Again, the crowd flow is a steady stream. Both these scenarios were chosen to provide multiple paths to the targets, some of which are less crowded than others. These scenarios allow the robot to choose either a path where the crowd moves along with the robot or a
Figure 3.7: Two crowds with spatially distinct patterns of movement on the fourth floor.

Figure 3.8: Two crowds with spatially distinct patterns of movement on the fifth floor.

In Chapter 5, Risk-A* learns the spatial distribution of risk in an environment and generates plans that seek to minimize it. To test the ability of Risk-A* to generate such risk-aware plans, crowd scenarios in Chapter 5 exhibit spatially distinct behavior. On the fourth floor, two groups of 35 pedestrians move as shown in Figure 3.7, where blue arrows describe the path of the first group and black arrows the path of the second. Both groups begin and end at the elevators (E) but they visit different parts of the space. Pedestrians in both groups always repeat their trip when they return to the elevator area; this ensures a steady stream of crowd flow. The important difference between the two groups is that the pedestrians in the second group (black arrows) deviate from their normal paths; when one
comes within 1m of the robot, it generates a new path that brings it closer to the robot. This makes it more difficult for the robot to navigate through the second group. On the fifth floor, two groups of people begin at the elevators (E) and move as shown in Figure 3.8. The blue arrows describe how 35 pedestrians travel along the inner loop. The black arrows describe how 50 pedestrians travel along the outer loop. The pedestrians in the outer loop come closer to the robot and make it harder for the robot to navigate. As on the fourth floor, pedestrians in both groups always repeat their trip when they return to the elevator area to ensure a steady stream of crowd flow. The sizes of the crowds on both floors were chosen to ensure a minimum crowd density.
In Chapter 6, CUSUM-A* adapts to changing crowds. To test its ability to do so, the crowd-flow patterns in Chapter 6 change midway through the experiment. In the fourth floor (Figure 3.9), 50 pedestrians begin from the elevators (E), move along the black arrows in Figure 3.9, and return to their starting position. Whenever a simulated pedestrian returns to the elevators, however, it repeats the trip with probability 0.98; otherwise, the pedestrian leaves the environment. As a result, the pedestrian population gradually decreases. The probability of 0.98 was chosen so that the initial set of pedestrian leaves the environment when the robot has addressed about half its targets. At this time, a second group of 50 pedestrians entered the environment. They too began and end at the elevators, but cycle instead along the path shown by the blue arrows in Figure 3.9, and always repeat their trip. On the fifth floor, 60 pedestrians begin from the elevators, move along the black arrows in Figure 3.10, and return to their starting position. As on the fourth floor, whenever a simulated pedestrian returns to the elevators, it repeats the trip with probability 0.98; otherwise, the pedestrian leaves the environment. The probability of 0.98 was chosen to ensure that when the robot has addressed half its targets, the first set of pedestrian is substantially reduced. At this time, a second group of 60 pedestrians enters the environment; they too begin and end at the elevators, but they cycle instead through a smaller inner path, shown by the blue arrows in Figure 3.10, and always repeat their trips.

In Chapter 7, Explore-A* generates plans that allow the robot to explore areas where it lacks a good estimate of crowd cost. In the fourth floor, 50 simulated pedestrians represented by black arrows begin from the elevators (E), visit target points 1,2,1,3,1 in Figure 3.11, and return to their starting position. Whenever a pedestrian returns to the elevators, it chooses to repeat its trip with probability 0.95; otherwise, it leaves the environment. As a result, the pedestrian population gradually decreases. The probability of 0.95 was chosen so that first group of pedestrians are substantially reduced when the robot has addressed about half its targets. A second group of 30 pedestrians represented by blue arrows also begins from the
Figure 3.11: Crowd behavior used to evaluate Explore-A* on the fourth-floor (left) initially (right) after 20 tasks.

elevators and they visit target points 5,6,5,4,5 in Figure 3.11, but always repeat their trips.

In Chapter 8, Help-A* generates plans that have the robot move toward the crowd while it visits a sequence of 40 targets. In both the fourth and fifth floors of MOMA, 20 pedestrians begin at the lower right in the environment, travel along the blue arrows in Figure 3.12, and exit at the upper right. This crowd simulates a museum tour. Whenever a simulated pedestrian exits the floor, it immediately restarts the trip; this ensures a constant crowd flow and is similar to the timed tickets that popular museums offer. Some rooms are ignored in the museum tours, to simulate rooms with negligible crowd movement. This avoids scenarios where all areas are equally crowded, because such a scenario is less apt to demonstrate the effectiveness of Help-A*. Maps from MOMA rather than the CUNY Graduate Center are more realistic for the museum robot task. A drawback of this approach, however, is that it limits comparisons with results from previous chapters.

All these crowd scenarios test algorithmic performance. Each scenario was designed to test a problem that its particular algorithm addressed. For example, plans generated by CSA* in an environment where the crowd density is uniform would not show any performance
gains. In this sense, the experiments in the thesis demonstrate the best case performance of the algorithms. The true understanding of average performance of these algorithms in a real world environment depends on a range of environmental features including the environment’s topology, crowd-flow patterns, crowd-flow predictability, and spatial distribution of pedestrian risk behavior. The next section defines the metrics used to evaluate the performance of the algorithm.

3.4 Evaluation metrics

The experiments reported here evaluated algorithmic performance with the following metrics:

- Total time: Total time taken in simulation by the robot to address 40 targets as actually measured (recorded by logs).
• Total computation time (TCT): Total time taken by SemaFORR to select actions, including the time taken by the planner to generate a plan as actually measured (recorded by logs).

• Total plan length: Sum of the lengths of the plans generated by the planner

• Total distance travelled: Total distance travelled by the robot to address 40 targets, as measured by the length of the trajectory traversed by the robot.

• Risky actions: Number of actions that put the robot within 0.5m of a pedestrian. While 0.5m was chosen as a parameter for this metric, however, the true collision risk to a person depends on the size, velocity and hardness of the robot. Risky actions can also be seen as a proxy for the robot’s adherence to human personal space. Any object that is within 1.5 to 4.5 feet of a person is within their personal space [Hall et al., 1968].

• Failures: Number of targets not reached within 500 decision cycles.

In addition, in Chapter 8, performance is also gauged by the number of voice commands received by the robot, instead of risky actions. In those experiments, when a person comes within 0.5m of the robot, there is a 50 percent chance that the robot receives a voice command. To evaluate the difference between the means of the above metrics unpaired t-test was used. The Shapiro-Wilk test was used to confirm the normality of the data [Shapiro and Wilk, 1965].

A* serves as the baseline in all experiments here, for several reasons. As described in Chapter 2, all other approaches that learn a crowd model assume either full observability or assume access to end-to-end pedestrian trajectory datasets. One could replace A* with more recent planners such as D* or MPGAA*, but all those planners simply produce the same global plan, only faster. Finally, standard ROS navigation architecture uses A* as a
global planner. This chapter has described the experimental framework, the SemaFORR controller, and MengeROS. The next chapter describes and evaluates CSA* and Flow-A*.
Chapter 4

Crowd-aware navigation

This chapter describes two algorithms for crowd-aware navigation. Sections 4.1 and 4.2 describe how a robot learns cost maps from local crowd data collected by range sensors as it moves through an environment simulated by MengeROS. Section 4.3, describes how CSA* and Flow-A* uses these learned cost maps to generate crowd-aware plans. Finally, Section 4.4 evaluates both algorithms to demonstrate performance improvements in robot navigation. The next section describes a Bayesian approach to learn about crowd density online.

4.1 Learning a crowd density map online

This approach makes several fundamental assumptions. The robot’s laser range sensors are mounted at a uniform level near the floor, and it has a two-dimensional map of the static features in its environment. The robot can localize with its laser range scanner data. Within its sensor range, the robot can also detect local crowd data, the location and orientation of each pedestrian [Leigh et al., 2015]. The robot receives an ordered set of targets in its environment. Its task is to reach (arrive at a location less than 0.5m from) these targets in
their specified order. We also assume that the robot, as it visits its targets, learns online about the crowd, without separate phases for learning and testing.

For each cell in a grid superimposed on the footprint of the environment, this approach calculates a running average for crowd density and records it in a crowd density map. Here, I formulate the problem as a Bayesian online learning problem and give a learning algorithm derivation based on certain modeling assumptions. This framework both improves the understanding of the underlying assumptions and allows discovery of new algorithms when those assumptions are changed. Each cell in the crowd density map is modeled as a Poisson distribution with rate $\lambda$, because the crowd density is always non-negative. A Poisson distribution is also commonly used to model natural discrete events including crowd counting [Chan and Vasconcelos, 2009]. The likelihood that a crowd of size $z$ is present in an individual grid cell for a fixed duration is thus

$$P(CrowdSize = z | \lambda) = \frac{\lambda^z e^{-\lambda}}{z!}$$  \hspace{1cm} (4.1)$$

Within a given grid cell, let $z_{1:n}$ represent the sequence of $n$ observations $z_1, z_2, ..., z_n$ made by the robot. Each observation $z_i$ represents the number of pedestrians detected from the range scan data. To estimate the value of $\lambda$ for a particular grid cell from $z_{1:n}$, we assume that $z_1, z_2, ..., z_n$ are conditionally independent given $\lambda$. This yields a recursive Bayesian filter that computes $P(\lambda|z_{1:n+1})$ given $P(\lambda|z_{1:n})$ and a new observation $z_{n+1}$, with normalization
constants $\eta$, as follows.

$$P(\lambda|z_{1:n}) = \eta_1 P(z_{1:n}|\lambda)P(\lambda) \quad \text{(Bayes rule)} \quad (4.2)$$

$$P(\lambda|z_{1:n+1}) = \eta_2 P(z_{1:n+1}|\lambda)P(\lambda) \quad \text{(Bayes rule)}$$

$$= \eta_2 P(z_{1:n}, z_{n+1}|\lambda)P(\lambda)$$

$$= \eta_2 P(z_{1:n}|z_{n+1}, \lambda)P(z_{n+1}|\lambda)P(\lambda)$$

$$= \eta_2 P(z_{1:n}|\lambda)P(z_{n+1}|\lambda)P(\lambda)$$

$$= \eta_2 P(\lambda|z_{1:n})P(z_{n+1}|\lambda) \quad \text{(by (4.2))} \quad (4.3)$$

Let $P(\lambda)$ be a Gamma distribution with (prior) parameters $\alpha$ and $\beta$, and let $P(z_{n+1}|\lambda)$ be the Poisson distribution (evidence). Then given

$$P(\lambda|z_{1:n}) = \Gamma(\alpha + \sum_{i=1}^{n} z_i, \beta + n)$$

the update rule that tracks the crowd density in a given cell is

$$P(\lambda|z_{1:n+1}) = P(\lambda|z_{1:n})P(z_{n+1}|\lambda) \quad \text{(by (4.3))}$$

$$= \Gamma(\alpha + \sum_{i=1}^{n} z_i, \beta + n) \ast \left( \frac{\lambda^{z_{n+1}} e^{-\lambda}}{z_{n+1}!} \right) \quad \text{(by (4.1))}$$

$$= \Gamma(\alpha + \sum_{i=1}^{n} z_i + z_{n+1}, \beta + n + 1)$$

Thus the new values of $\alpha$ and $\beta$ after a new observation $z_{n+1}$ are readily computed from the
Algorithm 1: Learn crowd density map for CSA*

**Input:** grid, γ, zn;

/* Initialize */

for each cell in grid do
  (α, β) = (0, 1);
end

/* Online update step */

for each cell in grid do
  α\text{new} = (α\text{old} \times γ) + zn;
  β\text{new} = (β\text{old} \times γ) + 1;
  E(λ) = \frac{α\text{new}}{β\text{new}};
end

old values of α and β, and the expected value of λ is the ratio of α and β:

\begin{align}
\alpha_{\text{new}} &= \alpha_{\text{old}} + zn + 1 \\
\beta_{\text{new}} &= \beta_{\text{old}} + 1 \\
E(λ) &= \frac{α}{β}
\end{align}

For a given grid cell, update rules (4.4) through (4.6) estimate the crowd density as the running average of observations there, as described in Algorithm 1. The running total of the observations zn is α\text{new}, β\text{new} counts the number of observations collected from the laser range scanner. As the robot gathers more evidence, the accuracy of the crowd density estimate improves. This section has described an approach to learn the crowd density map online, but it ignores the flow of the crowd. The next section, adds crowd flow to the model.

### 4.2 Learning a crowd flow model online

People conform to multiple social and/or cultural norms by the way they navigate in an environment [Kruse et al., 2013]. A simple yet commonly observed phenomenon is lane
formation. Lane formation makes travel for everyone orderly and smooth, so that people
need make fewer collision avoidance maneuvers. A robot in a crowded environment could
also be required to follow such norms. For example, a museum-guide robot that escorts
people to their requested destinations should pick routes that follow the flow of the crowd
of visitors rather than routes that oppose it.

For each cell in a grid superimposed on the footprint of the environment, Flow-A* calcu-
lates a running average for crowd flow in different directions and records it in a *crowd-flow
map*. The crowd-flow map represents the global state of the crowd flow in the environment.
We assume that laser range sensors can detect (within sensor range) the orientation of a
pedestrian with a single observation. Each pedestrian is classified as facing one of 8 direc-
tions: north(*n*), south(*s*), east(*e*), west(*w*), northwest(*nw*), northeast(*ne*), southwest(*sw*)
and southeast(*se*). For a given observation and a given cell, let the number of pedestrians
facing direction *i* be *o* *i* where *i* ∈ {*n*, *s*, *e*, *w*, *nw*, *ne*, *sw*, *se*}. For each cell in the grid, *λ* *i*
represents the running average of the number of pedestrians facing direction *i* in that grid.
For every new observation the crowd flow map is updated as described in Algorithm 2 where,

\[
\text{total}_i = \text{total}_i + o_i \quad (4.7)
\]

\[
\text{count} = \text{count} + 1 \quad (4.8)
\]

\[
\lambda_i = \frac{\text{total}_i}{\text{count}} \quad (4.9)
\]

Algorithm 2 uses equations (4.7) through (4.9) to estimate *λ* *i* for each cell. In (4.9) *total* *i*
computes the running total of observations *o* *i*, while *count* counts observations. Observations
of crowd flow are collected from the laser range scanner and as the robot gathers more
evidence, the accuracy of the crowd flow estimate improves. In summary, this section has
described an approach that learns crowd flow maps to represent the global crowd flow. The
next section describes how CSA* and Flow-A* can use these models to generate crowd-aware
CHAPTER 4. CROWD-AWARE NAVIGATION

Algorithm 2: Learn crowd flow map for Flow-A*

Input: grid, γ, αi;
/* Initialize */
for each cell in grid do
    for each direction \(i \in \{n, s, e, w, nw, ne, sw, se\}\) do
        \((α_i, β_i) = (0, 1);\)
    end
end
/* Online update step */
for each cell in grid do
    for each direction \(i \in \{n, s, e, w, nw, ne, sw, se\}\) do
        \(α_i^{\text{new}} = (α_i^{\text{old}} * γ) + α_i;\)
        \(β_i^{\text{new}} = (β_i^{\text{old}} * γ) + 1;\)
        \(E(λ_i) = \frac{α_i^{\text{new}}}{β_i^{\text{new}}};\)
    end
end

plans.

4.3 Online crowd-aware planning

CSA* and Flow-A* use an A* planner to leverage the crowd-density map and the crowd-flow map to formulate crowd-aware plans. A regular grid is superimposed upon the map, and a weighted graph is built that represents each grid cell as a node. A node in the graph has edges to at most eight cells that adjoin it in the grid. (There are fewer than eight if the cell lies on the border of the grid or if a wall intervenes.) A* finds an optimal (shortest) path in this graph.

When presented with its first target in a new environment, the robot makes and follows a standard A* plan. As it travels and observes the crowd, however, the robot updates its cost map (either the crowd-density map or the crowd-flow map or a combination of them both). Then, before each subsequent target, the robot estimates from its cost map the likely impact of crowds on its navigation, and updates the graph’s edge weights. CSA* uses a
crowd-density map to update edge weights; Flow-A* uses both a crowd-density map and a crowd-flow map.

Let $e_{mn}$ be the edge cost to travel from node $m$ to node $n$, $\text{distance}_{mn}$ be the Euclidean distance between the nodes, and $\text{density}_m$ be the crowd density value at node $m$. The edge cost for CSA* is given by

$$e_{mn} = w_1 \times \text{distance}_{mn} + w_2 \times \left(\frac{(\text{density}_m + \text{density}_n)}{2}\right) \quad (4.10)$$

Let $\text{flow}_{mn}$ be the additional cost incurred by the robot when it is travelling from $m$ to $n$. Note that in (4.10) the average number of pedestrians facing direction $i$ as learned by the crowd flow model is averaged for nodes $m$ and $n$. These eight averaged direction vectors are projected on the edge $mn$. If the projection is negative, (i.e., the robot is going against the flow) the length of the projection is used as $\text{flow}_{mn}$. If the projection is positive (i.e., the robot is going with the flow) $\text{flow}_{mn}$ is revised to zero, because even a crowd that moves in the same direction as the robot cannot make travel easier than no crowd at all. The edge cost for Flow-A* is given by

$$e_{mn} = w_1 \times \text{distance}_{mn} + w_2 \times \left(\frac{(\text{density}_m + \text{density}_n)}{2}\right) + w_3 \times \text{flow}_{mn} \quad (4.11)$$

In summary, A* returns the shortest plan, while CSA* returns a plan that avoids crowded areas. Flow-A* adds nuance to CSA* by considering the direction of the crowd in addition to its density; it plans to avoid crowded areas and areas where the crowd flow opposes the robot’s intended direction. The next section evaluates CSA* and Flow-A*, and shows how they improve navigation performance.
This section describes simulation experiments conducted in accordance with Section 3.3. They evaluate CSA* and Flow-A* on navigation through crowded environments in the fourth and fifth floors of CUNY’s Graduate Center.

### 4.4.1 Fourth-floor experiments

The first experiment compared A* with CSA* and Flow-A* in the simulated fourth-floor environment. The robot’s task here is to visit targets at positions 2, 5, 3, 4, 1, 6, 2 in Figure 4.1 in that order until the robot has attempted to reach 40 targets. As the robot moves to complete its task, a group of 60 pedestrians begin (not all at same time) from the elevators, and move along the path described by the red arrows in Figure 4.1. Pedestrians return to the elevator area once they reach the end of the flow path; this simulates a steady stream of crowd flow. This experiment was repeated 10 times; the average results are reported in Table 4.1. All statistically significant differences ($p < 0.05$ unless otherwise specified) in this document are in boldface.
Figure 4.2: Plans in the fourth floor to move to target 4.
Table 4.1: CSA* and Flow-A* improve navigation performance in the fourth-floor map

<table>
<thead>
<tr>
<th></th>
<th>A*</th>
<th>CSA*</th>
<th>Flow-A*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Risky actions</td>
<td>avg 2285.80</td>
<td>1526.00</td>
<td>982.00</td>
</tr>
<tr>
<td></td>
<td>std 160.07</td>
<td>237.03</td>
<td>229.90</td>
</tr>
<tr>
<td>Failures</td>
<td>avg 4.10</td>
<td>3.70</td>
<td>1.80</td>
</tr>
<tr>
<td></td>
<td>std 1.73</td>
<td>2.58</td>
<td>1.23</td>
</tr>
<tr>
<td>Total time (sec)</td>
<td>avg 3924.77</td>
<td>3950.95</td>
<td>3926.94</td>
</tr>
<tr>
<td></td>
<td>std 136.61</td>
<td>155.57</td>
<td>174.51</td>
</tr>
<tr>
<td>Distance (m)</td>
<td>avg 2356.93</td>
<td>2327.25</td>
<td>2319.11</td>
</tr>
<tr>
<td></td>
<td>std 100.34</td>
<td>81.00</td>
<td>82.72</td>
</tr>
<tr>
<td>TCT (sec)</td>
<td>avg 591.74</td>
<td>775.28</td>
<td>974.95</td>
</tr>
<tr>
<td></td>
<td>std 13.28</td>
<td>48.06</td>
<td>42.65</td>
</tr>
<tr>
<td>Plan length (m)</td>
<td>avg 1850.87</td>
<td>1901.72</td>
<td>2084.98</td>
</tr>
<tr>
<td></td>
<td>std 30.45</td>
<td>38.42</td>
<td>28.05</td>
</tr>
</tbody>
</table>

The results in this environment demonstrate that, compared to A*, both CSA* and Flow-A* successfully avoid crowded areas and reduce both failures and risky actions. Compared to CSA*, Flow-A* demonstrates that the robot can construct plans that avoid opposing the flow of the crowd and thus reduce risky actions. There was no statistically significant difference in the distance and the total travel time among all three approaches, but the total computation time (time spent on deciding the next action and planning time) increased with both CSA* and Flow-A*. Flow-A* generates longer plans than CSA* ($p < 0.01$) or A* ($p < 0.01$), but the actual travel distance is similar for all three algorithms. This indicates that CSA* and A* produce plans that lead the robot toward crowds and require more travel to navigate around them. Figure 4.2 overlays the 10 plans (one for each trial) generated by A*, CSA* and Flow-A* when the robot moved to target position 4 from target position 1 (12th task), and illustrates how Flow-A* avoids going against the flow of the crowd.

4.4.2 Fifth-floor experiments

The second experiment was in the fifth-floor environment of Figure 4.3. The robot’s task was to visit targets 1, 2, 1, 3, 1, 4, 1 in that order, repeated until the robot had attempted to
reach 40 targets. This design focuses on the left side of the fifth-floor environment and tests
the algorithm in a different topology that has more parallel hallways compared to the fourth-
floor environment (four horizontal hallways on the right side of the fifth floor as opposed to
the three horizontal hallways on the left side of the fourth floor). As the robot moves to
complete its task, a group of 60 pedestrians begin (not all at same time) from the elevator,
and move along the hallway that follows the red arrows in Figure 4.3. Again, the crowd flow
is a steady stream, and each experiment here was repeated 10 times. Table 4.2 reports the
average results.

As in the fourth-floor environment, the results demonstrate that compared to A* both
CSA* and Flow-A* successfully avoid crowded areas and reduce both failures and risky
actions. Plan length was again longer in Flow-A* than in CSA* and A*. There was no
statistically significant difference in the distance, total computation time, or total travel time
among all three approaches. Figure 4.4 overlays the 10 plans (one for each trial) generated
by A*, CSA* and Flow-A* when the robot moved from target position 1 to target position
3 (8th task) and illustrates Flow-A*’s ability to contend with crowd flow in the fifth-floor
environment.
Figure 4.4: Plans in the fifth floor to move to target 3.


Table 4.2: CSA* and Flow-A* improve navigation performance in the fifth-floor map

<table>
<thead>
<tr>
<th></th>
<th>A*</th>
<th>CSA*</th>
<th>Flow-A*</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Risky actions</strong></td>
<td>avg</td>
<td>1068.47</td>
<td>591.20</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>130.39</td>
<td>115.28</td>
</tr>
<tr>
<td><strong>Failures</strong></td>
<td>avg</td>
<td>0.87</td>
<td>2.20</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>1.13</td>
<td>1.32</td>
</tr>
<tr>
<td><strong>Total time (sec)</strong></td>
<td>avg</td>
<td>3214.09</td>
<td>3492.78</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>128.06</td>
<td>160.88</td>
</tr>
<tr>
<td><strong>Distance (m)</strong></td>
<td>avg</td>
<td>2311.86</td>
<td>2556.03</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>73.66</td>
<td>105.30</td>
</tr>
<tr>
<td><strong>TCT (sec)</strong></td>
<td>avg</td>
<td>1051.02</td>
<td>1347.26</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>12.97</td>
<td>31.25</td>
</tr>
<tr>
<td><strong>Plan length (m)</strong></td>
<td>avg</td>
<td>1893.95</td>
<td>1980.32</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>9.43</td>
<td>24.56</td>
</tr>
</tbody>
</table>

4.4.3 Conclusion

In summary, this chapter has described two algorithms that work online and require only local sensor information. These methods for crowd-aware navigation do not require a global view of the world and hence overcome the limitations of earlier approaches. In simulation these approaches produce statistically significant improvements in navigation performance. An important limitation of these approaches, however, is that a densely crowded area does not always imply that the robot’s navigation will be hindered. In some environments, people may readily make way for a robot, so the global plans generated by CSA* or Flow-A* would be inefficient. The next chapter, describes an algorithm that addresses this problem.
Chapter 5

Risk-aware navigation

People behave differently toward robots than toward people; this behavior may include curiosity or abuse [Brscić et al., 2015]. These reactions vary with demographics [May et al., 2017], personality [Nomura et al., 2008, Walters et al., 2005], personal experience with pets or robots [Takayama and Pantofaru, 2009], perceived likeability and aggressiveness of the robot [Mumm and Mutlu, 2011], appearance and size of the robot [Butler and Agah, 2001], direction of approach [Dautenhahn et al., 2006], and gaze of the robot [Ríos-Martínez et al., 2015]. In real environments, where people behave differently in the presence of a robot, the crowd density map computed in Algorithm 1 is not an accurate reflection of the true cost of navigation. If, for example, people consistently make way for a robot as it travels, the crowd density map would overestimate navigation difficulty.

Instead, I estimate the expected number of risky actions in a given grid cell per unit time as a proxy for the cost of navigation through that cell. The expected risky action rate $\lambda_r$ in a grid cell depends both on the crowd density there, as estimated from the laser range data, and on the crowd behavior when a robot is present in that cell. The following section describes an online algorithm for learning $\lambda_r$. The subsequent sections describe changes to MengeROS that simulate variations in crowd behavior, and Risk-A*, an algorithm that
uses the learned risk model to generate global plans. The final section evaluates Risk-A* in simulation on two different environments and describes the results.

5.1 Robot-specific behaviors in MengeROS

In the Menge crowd simulator, pedestrians’ navigation and collision avoidance strategies remain unchanged when they encounter one another. I modified this in MengeROS to ensure a different collision avoidance behavior between a pedestrian and a robot. To ensure collision-free navigation, Menge uses ORCA or PedVO to generate a velocity command, a vector for the pedestrian to follow. In MengeROS, however, when a pedestrian is within 3m of a robot, MengeROS adds a change vector to the default velocity command generated by Menge. The change vector has magnitude $d \cdot p$, where $d$ is the distance between robot and the pedestrian and $p$ is a control parameter. The resultant pedestrian velocity command vector makes the pedestrian veer towards the robot if the magnitude of the change vector is positive, and veer away from the robot if the magnitude of the change vector is negative.

5.2 Learning a risk map

The set of $\lambda_r$ across all the grid cells in the environment is defined as a risk map. To learn the risk map, for each grid cell I count the risk experience $a_i$, the number of risky actions the robot took when it moved through that cell for the $i^{th}$ time. Let $a_{1:k}$ be the sequence of risk experiences that the robot observes in the same cell from time 1 to time $k$. We assume that the occurrence of one risky action does not influence the next, given $\lambda_r$ in that cell (i.e, given $\lambda_r$, $a_i \perp a_j$, where $\perp$ denotes independence). We then model the risky actions in any given grid cell as a Poisson distribution with rate $\lambda_r$. To learn $\lambda_r$, let $z_{1:n}$ be the sequence of crowd counts in that grid cell based on laser scans, and let $\lambda$ be the estimated crowd density
there. If we assume that $\lambda_r$ is conditionally independent of $z_{1:n}$ given $\lambda$, the probability $P(\lambda_r | \lambda, z_{1:n}, a_{1:k})$ of risky actions after $k$ experiences is equal to $P(\lambda_r | \lambda, a_{1:k})$. We can then compute $P(\lambda_r | \lambda, a_{1:k})$ recursively, again with normalization constant $\eta$, as follows.

Let the prior $P(\lambda_r | \lambda, a_{1:k})$ be a gamma distribution with parameters $\lambda$ and $c$, where $\lambda$ is the current crowd density estimate for the cell and $c$ indicates how much crowd density affects risky actions. Then, given a new experience $a_{k+1}$ where the robot moves through a crowd in that cell, we can construct a Bayes filter by reasoning similar to the derivation in Section 4.1:

\begin{align}
P(\lambda_r | \lambda, a_{1:k}) &= \eta_1 P(a_{1:k} | \lambda, \lambda_r) P(\lambda_r | \lambda) & \quad (5.1) \\
P(\lambda_r | \lambda, a_{1:k+1}) &= \eta_2 P(a_{1:k+1} | \lambda, \lambda_r) P(\lambda_r | \lambda) \\
&= \eta_2 P(a_{1:k+1}, a_{k+1} | \lambda, \lambda_r) P(\lambda_r | \lambda) & (Bayes rule) \\
&= \eta_2 P(a_{1:k+1} | \lambda, \lambda_r) P(a_{k+1} | \lambda_r) P(\lambda_r | \lambda) & (given \lambda, \lambda_r, a_i \perp a_j) \\
&= \eta_2 P(\lambda_r | \lambda, a_{1:k}) P(a_{k+1} | \lambda_r) & (substitution (5.1)) \\
&= \eta_3 P(\lambda_r | \lambda, a_{1:k}) P(a_{k+1} | \lambda_r) & (given \lambda_r, a_{k+1} \perp \lambda) & (5.2)
\end{align}

Equation 5.2 describes an online Bayes filter where the posterior $P(\lambda_r | \lambda, a_{1:k+1})$ can be computed as a product of the prior $P(\lambda_r | \lambda, a_{1:k})$ and the likelihood of new risky action data $P(a_{k+1} | \lambda_r)$. Since the prior is assumed to be a Gamma distribution and likelihood of
risky-action $a_{k+1}$ a Poisson distribution, the posterior becomes a new Gamma distribution:

$$P(\lambda_r | \lambda, a_{1:k+1}) = \Gamma(\lambda + \sum_{i=1}^{k} a_i, c + k) \ast \left( \frac{\lambda_r a_{k+1}^e e^{-\lambda_r}}{a_{k+1}!} \right)$$

$$= \Gamma(\lambda + \sum_{i=1}^{k} a_i + a_{k+1}, c + k + 1)$$

where the expected parameter value of the posterior Gamma distribution, $\lambda_r$ can be computed as:

$$E(\lambda_r) = \frac{\lambda + \sum_{i=1}^{k} a_i}{c + k}$$

$$E(\lambda_r) = \frac{\sum_{j=1}^{n} z_j}{n} + \sum_{i=1}^{k} a_i$$

(5.3)

Given current observations for crowd density $z_n$ and risky action count $a_k$, Algorithm 3 estimates the risk map with equation (5.3), where $\gamma$ is a discount factor in $(0, 1)$, $\alpha_r^{new}$ replaces the sum on $a_k$, and $\beta_r^{new}$ counts $k$. This approach balances the evidence collected from the laser range scanner about crowding in a given location against the realized cost of navigation. Given a cell where the robot has no risk observation $a_k$, the algorithm depends on evidence from the laser scanner to estimate navigation cost. As the robot gathers more evidence about how crowds in that cell behave, it progressively adjusts the risky action count.

### 5.3 Risk-A*

Risk-A* is an A*-based planner that uses both the crowd density and its past experience of risky actions in a given grid cell to update edge weights. When presented with its first target in a new environment, the robot makes and follows the standard A* plan. As it travels and observes the crowd, however, the robot updates its cost map, tracking $\lambda_r$ for each grid cell.
Algorithm 3: Learn risk map for Risk-A*

/* Initialize */
for each cell in grid do
  | (α, β, α_r, β_r) = (0, 1, 0, 1);
end

/* Given a new Observation z_n */
for each cell in grid do
  | α_{new} = (α_{old} * γ) + z_n;
  | β_{new} = (β_{old} * γ) + 1;
  | E(λ) = \frac{α_{new} * γ}{β_{new}};
end

/* Given a new Risk experience a_k */
for each cell in grid do
  | α_{new,r} = α_{old,r} + a_k;
  | β_{new,r} = β_{old,r} + 1;
  | E(λ_r) = \frac{E(λ) + α_{new,r}}{c + β_{new,r}};
end

Then, before each subsequent target, the robot estimates from its cost map the likely impact of crowds on its navigation, and updates the graph’s edge weights.

Let $e_{mn}$ be the edge cost to travel from node $m$ to node $n$, and $distance_{mn}$ be the Euclidean distance between the nodes. The edge cost for CSA* is given by

$$e_{mn} = w_1 * distance_{mn} + w_2 * ((λ_m + λ_n)/2)$$ (5.4)

A* returns the shortest plan, and CSA* returns a plan that avoids crowded areas. Risk-A* returns a plan that avoids crowds that are likely to cause collisions. The next section compares Risk-A* to CSA* and A* in the simulated fourth-floor and fifth-floor environments.
5.4 Evaluation

Risk-A* addresses risky actions in its cost map when it plans. We compared the performance of Risk-A* to A* and to CSA*, which only considers the crowd density estimate when it predicts the cost. The metrics were risky actions, failures, distance travelled, total travel time, total computation time, and plan length as defined in Section 3.4. Experiments ran in the fourth-floor and fifth-floor environments of The Graduate Center as described in Section 3.3.

5.4.1 Fourth-floor experiment

In the fourth floor, the robot’s task is to visit target positions 2, 5, 3, 4, 1, 6, 2 in Figure 5.1 in that order until the robot addresses 40 targets. Two groups of 35 pedestrians move as shown in Figure 5.2, where blue arrows describe the first group’s path and black arrows describe the second’s. Both groups begin and end at the elevators (E) but they visit different parts of the space. Pedestrians in both groups always repeat their trip when they return to the elevator area; this ensures a steady stream of crowd flow. The important difference between the two groups is that the pedestrian in the second group (black arrows) deviate from their normal path; when one comes within 1m of the robot, it generates a new path that brings it closer to the robot. This makes it more difficult for the robot to navigate through the second group.

As shown in Table 5.1, there were no statistically significant differences in distance or total time. CSA* takes fewer risky actions than A* ($p < 0.01$), and Risk-A*, takes fewer risky actions than CSA* ($p < 0.01$). Risk-A* has fewer failures compared to CSA* ($p < 0.01$). Risk-A*’s computation time is higher than CSA* ($p < 0.01$). Although it spends more time computing decisions, Risk-A* does not take more time to complete its task. This indicates that the time lost by waiting for decisions may have been be regained from travel through
Figure 5.1: (Repeated from Figure 4.1) The fourth floor with elevators at E and six targets

Table 5.1: Risk-A* improves navigation performance in the fourth floor

<table>
<thead>
<tr>
<th></th>
<th>A*</th>
<th>CSA*</th>
<th>Risk-A*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Risky actions</td>
<td>avg</td>
<td>6414.4</td>
<td>5213.3</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>781.9</td>
<td>495.04</td>
</tr>
<tr>
<td>Failures</td>
<td>avg</td>
<td>15.5</td>
<td>10.4</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>2.01</td>
<td>1.35</td>
</tr>
<tr>
<td>Total time (sec)</td>
<td>avg</td>
<td>4254.94</td>
<td>4073.21</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>211.37</td>
<td>104.08</td>
</tr>
<tr>
<td>Distance (m)</td>
<td>avg</td>
<td>2154.12</td>
<td>2260.97</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>110.8</td>
<td>106.4</td>
</tr>
<tr>
<td>TCT (sec)</td>
<td>avg</td>
<td>245.5</td>
<td>341.86</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>12.71</td>
<td>20.44</td>
</tr>
<tr>
<td>Plan length (m)</td>
<td>avg</td>
<td>1669.99</td>
<td>1780.56</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>40.92</td>
<td>47.22</td>
</tr>
</tbody>
</table>
Figure 5.2: All plans generated by A*, CSA* and Risk-A* in the fourth-floor environment. Arrows describe crowd behavior. (See text.)
Figure 5.3: (Repeated from Figure 4.3) The fifth floor with elevators at E and four targets

more accommodating crowds. Risk-A* generates longer plans then CSA* ($p < 0.01$) or A* ($p < 0.01$), but the actual travel distance is similar. This indicates that A*'s plan led the robot toward crowds and required more travel to navigate through them. Figure 5.2 compares the plans generated by A*, CSA*, and Risk-A* as the robot navigated to target position 4 from target position 3 (10th task). Risk-A* mostly generated plans that sought to avoid the second group of pedestrians.

5.4.2 Fifth-floor experiment

In the fifth floor the robot’s task is to visit target positions 1, 2, 1, 3, 1, 4, 1 in Figure 5.3 in that order until the robot addresses 40 targets. Two groups of pedestrians begin at the elevator E and move as shown in Figure 5.4. The blue arrows describe how 35 pedestrians travel along the inner loop. The black arrows describe how 50 pedestrians travel along the outer loop. The pedestrians in the outer loop come closer to the robot and make it harder for the robot to navigate. As on the fourth floor, pedestrians in both groups always repeat their trip when they return to the elevator area to ensure a steady stream of crowd flow.

As shown in Table 5.2, there were no statistically significant differences in distance, failures, total time, or plan length. Again the focus is on risky actions, and the increased
Figure 5.4: All plans generated by A*, CSA* and Risk-A* in the fifth-floor environment. Arrows describe crowd behavior. (See text.)
Table 5.2: Risk-A* improves navigation performance in the fifth floor

<table>
<thead>
<tr>
<th></th>
<th>A*</th>
<th>CSA*</th>
<th>Risk-A*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Risky actions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>avg</strong></td>
<td>3934.00</td>
<td>3804.90</td>
<td><strong>3383.10</strong></td>
</tr>
<tr>
<td><strong>std</strong></td>
<td>314.95</td>
<td>370.66</td>
<td>223.04</td>
</tr>
<tr>
<td>Failures</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>avg</strong></td>
<td>2.90</td>
<td>4.10</td>
<td>5.00</td>
</tr>
<tr>
<td><strong>std</strong></td>
<td>1.79</td>
<td>1.66</td>
<td>1.94</td>
</tr>
<tr>
<td>Total time (sec)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>avg</strong></td>
<td>3662.68</td>
<td>4093.17</td>
<td>4192.16</td>
</tr>
<tr>
<td><strong>std</strong></td>
<td>147.10</td>
<td>196.73</td>
<td>253.34</td>
</tr>
<tr>
<td>Distance (m)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>avg</strong></td>
<td>2486.64</td>
<td>2537.64</td>
<td>2545.12</td>
</tr>
<tr>
<td><strong>std</strong></td>
<td>56.28</td>
<td>52.96</td>
<td>2545.12</td>
</tr>
<tr>
<td>TCT (sec)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>avg</strong></td>
<td>281.65</td>
<td><strong>553.58</strong></td>
<td><strong>672.49</strong></td>
</tr>
<tr>
<td><strong>std</strong></td>
<td>11.48</td>
<td>27.75</td>
<td>39.86</td>
</tr>
<tr>
<td>Plan length (m)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>avg</strong></td>
<td>1882.08</td>
<td>1913.88</td>
<td>1903.24</td>
</tr>
<tr>
<td><strong>std</strong></td>
<td>31.00</td>
<td>43.63</td>
<td>26.74</td>
</tr>
</tbody>
</table>

computation time for which risk-aware navigation compensates. Risk-A* takes fewer risky actions than CSA* ($p < 0.01$). Figure 5.4 compares the plans generated by A*, CSA*, and Risk-A* as the robot navigates to target position 1 from target position 4 (11th task). Risk-A*, as in the fourth-floor environment, generates plans that seek to avoid the second group of pedestrians.

In summary, this chapter has described Risk-A*, which learns a risk map of the environment that captures the effect of human behavior near the robot. Specifically, it adapts the robot’s behavior based on the number of risky actions it has experienced. Since the choice of a local collision-avoidance planner directly impacts the number of risky actions, the choice of a local planner influences the performance of Risk-A*. If a robot uses a local planner that does not perform well in specific areas of the environment, the number of risky actions in those areas would increase, and Risk-A* would make plans that avoid those areas. In this sense, Risk-A* is an example of metacognition, where the robot would learn about its shortcomings in local planning and plan to avoid such scenarios.

Thus far, all crowd scenarios had crowd patterns that did not change with time. The next chapter describes and evaluates approaches to handle such dynamic crowd behavior.
Chapter 6

Change detection for crowd models

The spatial patterns of crowds tested thus far did not change over time. Online learning in CSA*, Flow-A* and Risk-A* was based on this assumption, which does not hold in true in many scenarios. For example, in a museum when a new exhibit is opened, the crowd patterns change, making the previously learned model stale. This chapter describes CUSUM-A*; an online technique to contend with changing crowd patterns. In CUSUM-A*, any sudden change in crowd patterns is detected by CUSUM, a statistical change detection algorithm [Page, 1954]. Once a change is detected in a specific part of the environment, CUSUM-A* resets the learned model and starts learning anew.

The following section describes discounting as a way to address dynamic crowds and identifies its drawbacks. The subsequent sections describe CUSUM and CUSUM-A*. The final section of this chapter describes the evaluation and reports the results.
6.1 Discounting

A straightforward approach to address this problem is discounting. Equations (4.4) through (4.6) can be modified to add a discount factor $\gamma \in (0, 1)$.

\[
\begin{align*}
\alpha_{\text{new}} &= \alpha_{\text{old}} \ast \gamma + z_{n+1} \\
\beta_{\text{new}} &= \beta_{\text{old}} \ast \gamma + 1 \\
E(\lambda) &= \frac{\alpha}{\beta}
\end{align*}
\]

This ensures that the model is biased to retain the most recent crowd patterns and forget older ones. Discounting has two drawbacks, however. First, the value of $\gamma$ is fixed, which means that the model forgets at a fixed rate irrespective of the changes in the underlying crowd behavior. Second, there is a single $\gamma$ for all parts of the environment, which means that it cannot account for spatial variations. Thus, the parts of the environment where crowd patterns remains constant will have the same $\gamma$ value as those parts where dynamic changes in crowd patterns are more frequent. Both these drawbacks are addressed in the next two sections, which describes CUSUM and CUSUM-A*.

6.2 Background: CUSUM

CUSUM is an online statistical change detector. Given a sequence $z_1, z_2, ..., z_n$ of observations of a random variable $Z$ that follows some probability distribution, CUSUM detects a change in $Z$’s distribution parameter $\theta$. Because $z_1, z_2, ..., z_n$ come from a probability distribution, some variation is to be expected. What CUSUM seeks to identify, however, is a persistent and significant change.

Assume that a change in the distribution parameter happens before the observation $z_c$. Let $\theta_0$ be the distribution parameter before that change and $\theta_1$ the distribution parameter
after it. CUSUM tracks the *summation score*, the sum of the log-likelihood ratios. For observations before \( z_c \) (when \( \theta_0 = \theta_1 \)), the likelihood ratio \( \frac{p(z_i|\theta_1)}{p(z_i|\theta_0)} \) is 1, and the log-likelihood ratio \( \ln \frac{p(z_i|\theta_1)}{p(z_i|\theta_0)} \) is 0. For observations from \( z_c \) onward, the likelihood ratio is \( \frac{p(z_i|\theta_1)}{p(z_i|\theta_0)} \), and the log-likelihood ratio \( \ln \frac{p(z_i|\theta_1)}{p(z_i|\theta_0)} \), will be positive if \( p(z_i|\theta_1) > p(z_i|\theta_0) \) and will be negative if \( p(z_i|\theta_1) < p(z_i|\theta_0) \). As the distribution parameter changes from \( \theta_0 \) to \( \theta_1 \) at \( z_c \), CUSUM’s summation score gradually increases as more positive log-likelihood ratios corresponding to observations after \( z_c \) are added. A larger summation score indicates stronger evidence for the change in the underlying distribution. Because the precise instant \( c \) of the change is unknown, CUSUM assumes \( c \) is the instant when the summation score is maximized:

\[
G[n] = \max_{1 \leq c \leq n} \sum_{i=c}^{n} \ln \frac{p(z_i|\theta_1)}{p(z_i|\theta_0)}
\]  

(6.4)

To be certain that the change at \( c \) is significant, CUSUM requires that \( G[n] \) exceed some threshold \( \tau \).

### 6.3 CUSUM-A*

To detect a change in the distribution of the crowd density in a given cell, we take \( \theta_0 \) as \( \lambda_0 \) and \( \theta_1 \) as \( \lambda_1 \), where \( \lambda_0 \) and \( \lambda_1 \) are the parameters of the Poisson distribution for that cell before and after the change. The instantaneous log-likelihood ratio \( s[i] \) of \( i^{th} \) observation is defined as

\[
s[i] := \ln \frac{p(z_i|\lambda_0)}{p(z_i|\lambda_1)}
\]
Substitution with the formulas for the Poisson distribution yields

\[ s[i] = \ln \left( \frac{\lambda_0 z_i e^{-\lambda_0}}{\lambda_1 z_i e^{-\lambda_1}} \right) = \ln(\frac{\lambda_0 z_i e^{-\lambda_0}}{z_i!}) - \ln(\frac{\lambda_1 z_i e^{-\lambda_1}}{z_i!}) = z_i \ln(\frac{\lambda_0}{\lambda_1}) - (\lambda_0 + \lambda_1) \]

A cumulative sum \( S[j] \) from 1 to \( j \) is defined as

\[ S[j] = \sum_{i=1}^{j} s[i] \]

\( S[j] \) can be used to reformulate \( G[n] \) as a minimization problem, a common CUSUM technique for more efficient computation:

\[ G[n] = S[n] - \min_{1 \leq j \leq n} S[j-1] \]

Before the robot travels, \( \alpha \) is initialized to 0 and \( \beta \) to 1 for each cell. Given a current observation \( z_n \), Algorithm 4 estimates the crowd density map with CUSUM, where \( \gamma \) is a discount factor in \((0,1)\).

When presented with its first target in a new environment, the robot makes and follows a standard A* plan. As it travels and observes the crowd, however, the robot updates its cost map, tracking \( \lambda \) for each grid cell. Whenever CUSUM detects a change in the crowd for a cell, the parameters \( \alpha \) and \( \beta \) of that cell are reinitialized. This CUSUM-based cost map is used to update the graph’s edge weights using equation (4.10). This variant of A* that uses a CUSUM-based cost map is called CUSUM-A*.

In summary, while A* returns a shortest plan, and CSA* returns a plan that avoids crowded areas, CUSUM-A* returns a plan that avoids crowds, and can address dynamic
crowd patterns. The next section compares CUSUM-A*’s performance with A*, CSA* and discounted CSA* in the fourth-floor and fifth-floor environments, and reports the results.

6.4 Evaluation

To compare CUSUM-A* to CSA* and A*, I again used the simulated fourth-floor and fifth-floor environments described in Section 3.3, and the metrics described in Section 3.4. Three versions of CSA* were tested; with no discount (CSA*) and with the discount factor $\gamma$ set to 0.66 (CSA*(0.66)) and 0.33 (CSA*(0.33)).

6.4.1 Fourth-floor experiment

In the fourth floor (Figure 6.1), the robot’s task is to visit targets 2, 5, 3, 4, 1, 6, 2 in Figure 6.1 in that order, repeatedly until the robot has attempted to reach 40 targets. A

Algorithm 4: Learn crowd density map with CUSUM

/* Initialize */
for each cell in grid do
    $(\alpha, \beta, S[0], SMIN^{old}) = (0, 1, 0, 0)$;
end

/* Online update step */
for each cell in grid do
    $\alpha^{new} = \alpha^{old} + z_n$;
    $\beta^{new} = \beta^{old} + 1$;
    $S[n] = S[n-1] + z_n \ln(\lambda_0/\lambda_1) - (\lambda_0 + \lambda_1)$;
    $SMIN^{new} = \min(S[n-1], SMIN^{old})$;
    $G[n] = S[n] - SMIN^{new}$;
    /* Reset if CUSUM detects change */
    if $G[n] \geq \tau$ then
        $\alpha^{new} = 0$;
        $\beta^{new} = 1$;
    end
end

$E(\lambda) = \frac{\alpha^{new}}{\beta^{new}}$;
Figure 6.1: (Repeated from Figure 4.1) The fourth floor with elevators at E and six targets group of 50 pedestrians begins from the elevators, moves along the black arrows in Figure 6.2, and returns to their starting position. Whenever a simulated pedestrian returns to the elevators, however, it repeats the trip with probability 0.98; otherwise, the pedestrian leaves the environment. As a result, the pedestrian population gradually decreases. When the robot has addressed about half its targets, a second group of 50 pedestrians enters the environment. They too begin and end at the elevators, but cycle instead along the path shown by the blue arrows in Figure 6.2, and always repeat their trip.

This dynamic change in crowd pattern, where the first group gradually leaves and another group is suddenly introduced, tests an algorithm’s ability to track changes in crowd behavior efficiently. CUSUM’s parameters were set to detect a sudden increase in the crowd in any cell. The threshold $\tau$ was 10, $\lambda_0$ was the current mean arrival rate at the cell, and $\lambda_1 = \lambda_0 - 4$. Targets were chosen to force the robot to interact with the crowd, and there was always more than one path from one target to the next.

The results of these experiments, averaged over 10 trials, appear in Table 6.1. Distance traveled, time taken, total computation time, and plan length were not statistically significantly different with any of the planners. The most noteworthy differences were in risky
Figure 6.2: (left) On task 10, plans generated by A*, CSA* and CUSUM-A* in the fourth floor. (right) On task 33, plans after the crowd changes at task 20. Arrows describe crowd behavior.
Table 6.1: CUSUM-A* improves navigation performance in the fourth floor

<table>
<thead>
<tr>
<th></th>
<th>A*</th>
<th>CSA*</th>
<th>CSA* (0.66)</th>
<th>CSA* (0.33)</th>
<th>CUSUM-A*</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Risky actions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td>830.20</td>
<td>647.60</td>
<td>696.70</td>
<td>611.80</td>
<td><strong>486.10</strong></td>
</tr>
<tr>
<td>std</td>
<td>150.09</td>
<td>128.89</td>
<td>179.79</td>
<td>152.79</td>
<td>122.48</td>
</tr>
<tr>
<td><strong>Failures</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td>0.70</td>
<td>0.40</td>
<td>0.50</td>
<td>0.40</td>
<td><strong>0.20</strong></td>
</tr>
<tr>
<td>std</td>
<td>1.16</td>
<td>0.70</td>
<td>0.71</td>
<td>0.52</td>
<td>0.42</td>
</tr>
<tr>
<td><strong>Total time (sec)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td>2349.34</td>
<td>2221.55</td>
<td>2504.27</td>
<td>2467.14</td>
<td>2109.54</td>
</tr>
<tr>
<td>std</td>
<td>212.27</td>
<td>148.05</td>
<td>139.00</td>
<td>185.97</td>
<td>94.63</td>
</tr>
<tr>
<td><strong>Distance (m)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td>2098.59</td>
<td>2184.89</td>
<td>2223.75</td>
<td>2216.35</td>
<td>2134.31</td>
</tr>
<tr>
<td>std</td>
<td>158.55</td>
<td>58.60</td>
<td>57.63</td>
<td>72.56</td>
<td>16.20</td>
</tr>
<tr>
<td><strong>TCT (sec)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td>165.97</td>
<td>218.75</td>
<td>435.74</td>
<td>439.18</td>
<td>208.06</td>
</tr>
<tr>
<td>std</td>
<td>11.49</td>
<td>9.84</td>
<td>37.90</td>
<td>34.43</td>
<td>11.10</td>
</tr>
<tr>
<td><strong>Plan length</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>std</td>
<td>109.57</td>
<td>22.53</td>
<td>23.27</td>
<td>10.52</td>
<td>8.77</td>
</tr>
</tbody>
</table>

actions and failures. CSA*, CSA*(0.66), CSA*(0.33) took fewer risks than A* \((p < 0.01)\) and CUSUM-A* took even fewer than all tested versions of CSA* \((p < 0.01)\). Variations in the discount factor had no significant effect on the number of risky actions. This is because discounting relies on a single parameter across the entire grid, and therefore fails to capture the granularity of changes. CUSUM-A* also failed less often than A* and all tested versions of CSA*.

Figure 6.2 (left) overlays the 10 plans (one for each trial) generated by A*, CSA*, and CUSUM-A* to move from target position 3 to target position 4 (10th task) in the fourth floor as the original crowd circled along the arrows. CSA* and CUSUM-A* both plan to avoid the crowd. On the 20th task, the second group of pedestrians entered and circled along the blue arrows in Figure 6.2 while the first group began to leave the environment. Figure 6.2 (right) overlays all 10 plans generated by A*, CSA*, and CUSUM-A* on the 33rd task. CUSUM-A* had quickly adapted to the changed crowds and planned to avoid the new crowd flow, while CSA* plans went through the crowded areas.
6.4.2 Fifth-floor experiment

In the fifth floor, The robot’s task is to visit targets 1, 2, 1, 3, 1, 4, 1 in Figure 6.3 in that order. This is repeated until the robot has attempted to reach 40 targets. A group of 60 pedestrians begin from the elevators, move along the black arrows in Figure 6.4, and return to their starting position. Whenever a simulated pedestrian returns to the elevators, it repeats the trip with probability 0.98; otherwise, the pedestrian leaves the environment. As a result, the pedestrian population gradually decreases. When the robot has addressed half its targets, a second group of 60 pedestrians enters the environment; they too begin and end at the elevators, but they cycle instead through a smaller inner path, shown by the blue arrows in Figure 6.4, and always repeat their trips. As on the fourth floor, the threshold \( \tau \) was 10, \( \lambda_0 \) was the current mean arrival rate at the cell, and \( \lambda_1 = \lambda_0 - 4 \). The results of these experiments, averaged over 10 trials, appear in Table 6.2.

Again, in the fifth-floor environment, distance traveled, time taken, total computation time and plan length were not statistically significantly different with any of the planners; the only noteworthy differences was in risky actions. CSA* and CSA*(0.66), CSA*(0.33) took fewer risks than A* \((p < 0.01)\), and CUSUM-A* took even fewer than CSA* and CSA*(0.33) \((p < 0.05)\).
Figure 6.4: (left) On task 13, plans generated by A*, CSA* and CUSUM-A* in the fifth floor. (right) On task 32, plans after the crowd changes on task 20. Arrows describe crowd behavior.
Table 6.2: CUSUM-A* improves navigation performance in the fifth floor

<table>
<thead>
<tr>
<th></th>
<th>A*</th>
<th>CSA*</th>
<th>CSA*(0.66)</th>
<th>CSA*(0.33)</th>
<th>CUSUM-A*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Risky actions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td>2614.70</td>
<td>2206.90</td>
<td>2103.90</td>
<td>2608.60</td>
<td>1893.90</td>
</tr>
<tr>
<td>std</td>
<td>940.83</td>
<td>292.16</td>
<td>425.37</td>
<td>283.98</td>
<td>256.37</td>
</tr>
<tr>
<td>Failures</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td>9.50</td>
<td>11.00</td>
<td>12.80</td>
<td>12.80</td>
<td>9.90</td>
</tr>
<tr>
<td>std</td>
<td>3.10</td>
<td>2.00</td>
<td>2.35</td>
<td>2.35</td>
<td>2.38</td>
</tr>
<tr>
<td>Total time (sec)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td>4047.98</td>
<td>4395.02</td>
<td>4568.85</td>
<td>4802.88</td>
<td>4439.21</td>
</tr>
<tr>
<td>std</td>
<td>1019.25</td>
<td>264.31</td>
<td>301.84</td>
<td>282.48</td>
<td>276.93</td>
</tr>
<tr>
<td>Distance (m)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td>2179.95</td>
<td>2295.21</td>
<td>2208.01</td>
<td>2282.27</td>
<td>2389.50</td>
</tr>
<tr>
<td>std</td>
<td>431.46</td>
<td>186.39</td>
<td>176.39</td>
<td>162.05</td>
<td>262.32</td>
</tr>
<tr>
<td>TCT (sec)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td>239.75</td>
<td>417.20</td>
<td>708.43</td>
<td>593.34</td>
<td>470.96</td>
</tr>
<tr>
<td>std</td>
<td>68.85</td>
<td>25.55</td>
<td>42.20</td>
<td>75.24</td>
<td>61.78</td>
</tr>
<tr>
<td>Plan length (m)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg</td>
<td>1705.81</td>
<td>1723.43</td>
<td>1657.33</td>
<td>1693.12</td>
<td>1766.09</td>
</tr>
<tr>
<td>std</td>
<td>66.75</td>
<td>67.04</td>
<td>139.21</td>
<td>71.32</td>
<td>125.40</td>
</tr>
</tbody>
</table>

Figure 6.4 (left) overlays the 10 plans (one for each trial) generated by A*, CSA*, and CUSUM-A* on the 13th task (target position 2 to target position 1) for fifth floor as the original crowd circles along the black arrows. On the 20th task, the second group of pedestrians enters and circles along the blue arrows. Figure 6.4 (right) overlays all 10 plans generated by A*, CSA*, and CUSUM-A* on the 32nd task (target position 1 to target position 3). CUSUM-A* plans more often avoid the crowd flow, while CSA* plans go through the inner hallway, even though it is crowded.

In summary, this chapter has described an approach to dynamic crowd patterns using CUSUM. Simulated experiments showed that CUSUM-A* can adapt faster to environments with changing crowds as compared to a discounting approach. CUSUM-A* can be easily extended to make planners adaptive to changes in the flow and the risk models described in the previous chapters. The CUSUM computation is fast and memory efficient. If required, additional CUSUM’s can be run in parallel to track these model changes for different frequencies, which makes this approach applicable in diverse scenarios. The next chapter addresses the exploration-exploitation tradeoff when planning with crowd models.
Chapter 7

Planning under uncertainty

In the previous chapters, path planning addressed the cost of distance, crowd density, crowd flow, and risky actions. Thus far, however this work has ignored the degree of uncertainty in these models. This can lead to inefficient plans. Given two different paths with same costs with respect to distance, crowd density, crowd flow, and risky actions, the planner should choose the path whose underlying models have not been sufficiently explored. In environments where crowds evolve, exploration permits the discovery of better paths. For example, if a highly crowded path eventually becomes much less crowded, exploration allows the robot to revisit that path and update its crowd model. This chapter addresses this exploration-exploitation tradeoff in the context of planning with crowd models. The next section describes Explore-A*, an exploratory planner based on Thompson sampling. The final section empirically evaluates it.

7.1 Explore-A*

The navigation approaches described in Chapters 4, 5, and 6 combine future state prediction and graph search to learn crowd density, crowd flow, and risk models of the environment.
These models predict crowd behavior that cannot be directly perceived. Predictions are then used by graph-search algorithms to make plans that account for anticipated future crowding. A drawback of this approach however, is that planning does not consider opportunities to learn better models. To overcome this drawback, the CSA* crowd-aware planning algorithm described in Chapter 4 is modified in this chapter to produce more exploratory plans. Rather than greedily attempt to minimize risk and distance travelled, the objective in exploratory planning is to construct plans that reduce uncertainty in the learned models. This ensures that, even if there is no exploration at the level of actions, the plans themselves are exploratory.

Both the RL-based and the MAB-based approaches described in Section 2.3 select actions that are valued for the learning opportunities they afford. Such actions are exploratory and sacrifice short-term rewards for longer-term ones. The problem is formulated with MAB, rather than RL because it is assumed that the current plan chosen by the robot will have no influence on the robot’s state at the beginning of the next planning cycle. The robot’s state has four components: static features (e.g, unchanging walls), dynamic features (e.g, changing crowds), the robot’s current target, and the robot’s current pose. The robot’s location at the end of the current plan will be the same (the current target) whatever plan is chosen. The robot’s next target is random and thus uninfluenced by the current plan. The static obstructions are similarly uninfluenced by the current plan. The future crowd behavior might change based on the current choice of the robot’s plan in some scenarios, but for most large scale environments it can be assumed that the presence of a single robot is unlikely to significantly alter future crowd behavior.

This independence assumption makes crowd-aware planning, a MAB problem, where every choice of plan has a reward \( r \), which is a function of the length of the path and crowding along it. In the classic multi-arm bandit problem setting, one has to learn an action value for every action. That is infeasible in this setting because each plan is a possible action.
CHAPTER 7. PLANNING UNDER UNCERTAINTY

Instead this work uses a crowd density map to approximate the action-value for a given plan as the sum of expected rewards of its negative edge costs.

Let \( e_{mn} \) be the edge cost to travel from grid cell \( m \) to grid cell \( n \), \( \text{distance}_{mn} \) be the length of the edge, and \( \text{density}_m \) be the crowd density value at grid cell \( m \). The edge cost is given by

\[
e_{mn} = w_1 \times \text{distance}_{mn} + w_2 \times ((\text{density}_m + \text{density}_n)/2)
\]  

(7.1)

The expected action-value for a plan \( p \) with edges \( e_1, \ldots, e_k \) is \( \sum_{i=1}^{n} (-e_i) \). A drawback of this approximation is that rewards of the bandit arms are no longer independent and are spatially correlated because selection of a plan provides information about the action value of other plans with overlapping sections [Wu and Meder, 2017]. Despite this drawback, the multi-arm bandit formulation allows the use of well-studied heuristic exploration strategies. In this work, I use Thompson sampling as the heuristic exploration strategy.

In Thompson sampling, instead of using the crowd density value at each grid cell, one sample is drawn from the underlying distribution that represents the crowd density at a particular cell. Such samples are used to compute the edge costs. Then, according to the Thompson sampling strategy, the plan with the highest action value (lowest cost) is chosen. A* search can be used to select this plan, where we replace the crowd density value in the grid cells with a sampled crowd density value. The resulting path of this A* search on the modified graph will not be greedy; instead it will explore parts of the environment where the uncertainty in the crowd density model is relatively higher. Algorithm 5 describes this process.

My hypothesis is that this approach will avoid local minima and improve navigation performance over the long term. This hypothesis is tested experimentally in the simulated fourth-floor environment as described in the next section.
Algorithm 5: Planning with Explore-A*

**Input:** \texttt{crowd\_density\_model, graph, current, target};

/* Sample from crowd density model */
for each cell in \texttt{crowd\_density\_model} do
  \[ X \sim \Gamma(\alpha, \beta); \]
  \[ \text{sampled\_grid}[\text{cell}] = X; \]
end
/* Use the sampled grid to update edge cost the graph */
\text{updated\_graph} = \text{update}(\text{graph, sampled\_grid});
\text{plan} = A^*(\text{updated\_graph, current, target});
\text{return} \text{plan};

---

**Figure 7.1:** fourth-floor environment with elevators at E, and two patterns for pedestrian

### 7.2 Evaluation

This section compares CSA*, CSA* with discounts, and CUSUM-A* to Explore-A* in the simulated fourth-floor environment described in Section 3.3, and under the metrics described in Section 3.4, while the crowd changes its flow pattern across time. The scenario is designed to highlight the advantages of exploration.

In the fourth floor, 50 simulated pedestrians represented by black arrows begin from the elevators (E), and visit target points 1,2,1,3,1 in Figure 7.1, and return to their starting position. Whenever a pedestrian returns to the elevators, it chooses to repeat its trip with
probability 0.95; otherwise, it leaves the environment. As a result, the pedestrian population gradually decreases. The choice of 0.95 ensured that this group of pedestrians has substantially diminished when the robot has completed half its tasks. A second group of 30 pedestrians, represented by blue arrows, also begins from the elevators; they visit target points 5,6,5,4,5 in Figure 7.1, but always repeat their trips. The robot begins at the elevator and has to make 40 trips between target points 1 and 2 shown in Figure 7.2. At the beginning of this scenario, the hallway with black arrows is more crowded; as time passes the hallway with blue arrows becomes relatively more crowded. Greedy planners (e.g., CSA*) that do not explore would initially prefer the hallway with the blue arrows to travel between 1 and 2, and would continue to choose it. A planner with long-term behavior (e.g., Explore-A*), however, should occasionally explore an alternative route through the hallway with black arrows, to reduce its uncertainty about crowds there.

Under the metrics described in Section 3.4, an experiment compared the performance of CSA* without discounting ($\gamma = 1$), CSA* with discounting ($\gamma = 0.33$ and $\gamma = 0.66$) and
Figure 7.3: All plans generated by CSA* (left), and CUSUM* (right) in the fourth floor.

Figure 7.4: All plans generated by Explore-A* in the fourth floor.
Table 7.1: Explore-A* improves navigation performance in the fourth floor compared to CUSUM-A* and to CSA* with/without discounting

<table>
<thead>
<tr>
<th></th>
<th>CSA*</th>
<th>CSA* (0.66)</th>
<th>CSA* (0.33)</th>
<th>CUSUM-A*</th>
<th>Explore-A*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Risky actions</td>
<td>avg</td>
<td>1290.70</td>
<td>1373.40</td>
<td>1446.10</td>
<td>1499.80</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>441.91</td>
<td>152.61</td>
<td>197.11</td>
<td>306.71</td>
</tr>
<tr>
<td>Failures</td>
<td>avg</td>
<td>2.30</td>
<td>1.60</td>
<td>1.20</td>
<td>1.50</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>1.34</td>
<td>1.90</td>
<td>0.79</td>
<td>1.08</td>
</tr>
<tr>
<td>Total time (sec)</td>
<td>avg</td>
<td>3506.27</td>
<td>3363.32</td>
<td>3202.72</td>
<td>3408.35</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>198.44</td>
<td>194.10</td>
<td>113.01</td>
<td>148.42</td>
</tr>
<tr>
<td>Distance (m)</td>
<td>avg</td>
<td>2645.24</td>
<td>2349.98</td>
<td>2250.53</td>
<td>2450.35</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>241.49</td>
<td>113.51</td>
<td>90.29</td>
<td>134.18</td>
</tr>
<tr>
<td>TCT (sec)</td>
<td>avg</td>
<td>519.75</td>
<td>677.52</td>
<td>664.75</td>
<td>539.53</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>53.45</td>
<td>19.31</td>
<td>13.92</td>
<td>27.76</td>
</tr>
<tr>
<td>Plan length (m)</td>
<td>avg</td>
<td>2029.92</td>
<td>1711.66</td>
<td>1794.76</td>
<td>1923.58</td>
</tr>
<tr>
<td></td>
<td>std</td>
<td>229.96</td>
<td>50.98</td>
<td>25.54</td>
<td>108.21</td>
</tr>
</tbody>
</table>

CUSUM-A* to Explore-A*. (Flow-A* and Risk-A* are not relevant in this scenario.) The results, averaged over 10 trials, appear in Table 7.1. There was no statistically significant difference in total computation time, failures, or total time. Explore-A*, however, takes fewer risky actions than every other method ($p < 0.05$). It also travels farther than CSA*(0.66), CSA*(0.33) and CUSUM-A* ($p < 0.01$). The plans generated by Explore-A* are longer than those generated by every other method ($p < 0.05$). This indicates that Explore-A* plans traveled further to discover less crowded routes. Figures 7.2, 7.3 and 7.4 compare the plans generated by Explore-A*, CSA*, CSA*(0.33), CSA*(0.66) and CUSUM-A* as the robot navigates to target position 1 from target position 2 (10th task). Explore-A* is often able to explore, detect, and exploit the decreased crowd in the hallway with the black arrows.

In conclusion, this chapter described CSA* as a hybrid model of decision making that combines the advantages of graph-based search with the advantages of model learning. While CSA* in Chapter 4 overcame the drawback of A* by learning an online crowd model, this chapter addressed CSA*’s failure to explore. It formulated the robot’s task as a MAB problem and devised a planning approach based on Thompson sampling called Explore-A*. 
Explore-A* was shown to have the desired effect, to reduce risky actions compared to other crowd-based planners in the simulated environment. The next chapter explores another application of crowd models beyond the simple navigation task addressed thus far.
Chapter 8

Planning for availability

The previous chapters used crowd models to improve navigation performance, by avoiding crowded areas. This chapter demonstrates that these learned crowd models have applications beyond the simple navigation task described in Chapter 3. For example, one can imagine a multi-purpose museum robot that not only navigates to targets but also accepts voice-based commands from museum visitors. That requires the robot to be physically close to the simulated pedestrian. Here, a robot is available if it is physically close enough for a pedestrian to issue voice commands. It is assumed that whenever a pedestrian is less than 0.5m away from a robot, he or she will issue a command to the robot with probability 0.50. Such a robot has two objectives: to reach its target locations and to be available to as many people as possible. Although availability, by definition, increases the risk of collision, it can be assumed that the use of either a conservative local planner that stops near people or a soft structure or slow speed for the robot, can mitigate that risk.

Previous work on museum robots focused on tour-guide robots, but did not adapt global plans based on a learned model of crowd behavior [Thrun et al., 1999, Nourbakhsh et al., 2003, Kuno et al., 2007]. A robot was also used to distribute pamphlets in a shopping mall [Shi et al., 2018]. While this is similar to the availability problem, that robot was stationary;
and the paper focused on robot-arm control algorithms to effectively hand out pamphlets.

In scenarios where the robot’s only task is to be available, the robot should travel so that it encounters the maximum number of people. One solution is to cover all areas evenly; this problem can be addressed with coverage path planning algorithms [Galceran and Carreras, 2013]. Another option is to formulate the problem as an environmental monitoring problem where the robot has to generate trajectories to monitor the current state of the world [Lan and Schwager, 2013]. Neither approach considers the task of targets along the way. Another option is to formulate the problem as a traveling salesperson problem (TSP) [Kruskal, 1956], but here, unlike TSP, the points are to be visited in a predefined sequence.

A naive solution is to construct shortest path plans from one target location to the next. Given a crowd model, this solution can be improved by preferring routes through crowded areas. Further improvements to availability can be made by moving along with the flow of the crowd. The crowd density model and crowd flow model described in Chapter 4 can be used to generate such paths. The next section describes Help-A*, a planner that uses the crowd-density model for the multi-purpose museum robot problem. The final section explains the experimental setup and describes the results.

8.1 Help-A*

Help-A* is a crowd-density model-based planner that prefers crowded areas over other, less crowded areas to improve the robot’s availability. In this work, availability is measured by the number of voice-based commands received by the robot. The robot must both be available and must visit a sequence of pre-ordered target locations. These targets can be used either to deliver goods or as a way to ensure each location some minimum of service. A solution to this problem requires knowledge of the crowd’s behavior in the environment.

Given a sequence of targets, the robot can use the A* search algorithm to construct a
plan. In Help-A*, however, the graph underlying A* is modified so that the cost of an edge through a crowded area is reduced to make travel there more attractive. These changes must ensure that the edge cost remains positive. Let $e_{mn}$ be the edge cost to travel from grid cell $m$ to grid cell $n$, $distance_{mn}$ be the length of the edge, and $density_i$ be the crowd density value at grid cell $i$. The modified edge cost is given by

$$e_{mn} = \max(0, 1, \frac{w_1}{w_2} \cdot distance_{mn} - \frac{w_2}{w_1} \cdot ((density_m + density_n)/2))$$ (8.1)

While a standard A* plan would blindly choose the shortest route to the next target, Help-A* uses the modified edge cost to choose more crowded routes deliberately, to improve the robot’s availability. I compare the performance of Help-A* with A* The next section evaluates this method in simulation on two floors from The Museum of Modern Art in New York City (MOMA).

### 8.2 Evaluation

Unlike the planners described in Chapters 4, 5, 6 and 7, Help-A* seeks out crowd. This section compares standard A* with Help-A* in the simulated fourth-floor and fifth-floor environments at MOMA, described in Section 3.3. Although A* is not designed to navigate towards crowds, to the best of my knowledge, all other approaches that learn a crowd model either assume full observability or use a pedestrian trajectory dataset as described in Chapter 2. CSA*, Flow-A* and Risk-A* are inappropriate benchmarks here because they are designed to avoid crowds. On both floors, 20 simulated pedestrians begin at the lower right in the environment, travel through it along the blue arrows in Figure 8.1, and exit at the upper right. This crowd simulates a museum tour. Whenever a pedestrian exits the floor, it immediately restarts the trip; this ensures a constant crowd flow and is similar to the timed tickets that
Figure 8.1: (Repeated from Figure 3.7) Robot’s initial location and pedestrian movement in the fourth floor (left) and fifth floor (right) of MOMA popular museums offer. Some rooms are ignored in the museum tours, simulating rooms with negligible crowd movement. This avoids scenarios where all areas are equally crowded, because such a scenario is not apt to demonstrate the effectiveness of Help-A*. The robot’s goal in both environments is to visit 40 random targets while it maximizes availability. The robot fails on a target if it uses more than 500 decision cycles on it.

Evaluation metrics were those described in Section 3.4, but with commands received substituted for risky actions, as the robot visits 40 target locations. Each time a pedestrian came within 0.5m of the robot, the pedestrian issued one voice command with probability 0.50. To address the variance related to the choice of targets, 5 sets of 40 points were generated. This experiment was repeated 5 times for each of the 5 target sets on each floor.
### Table 8.1: Help-A* receives more commands on MOMA’s fourth floor

<table>
<thead>
<tr>
<th></th>
<th>A*</th>
<th>Help-A*</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total time (sec)</strong></td>
<td>avg 996.37</td>
<td>1022.61</td>
</tr>
<tr>
<td></td>
<td>std 127.54</td>
<td>167.43</td>
</tr>
<tr>
<td><strong>Distance (m)</strong></td>
<td>avg 1004.92</td>
<td>1010.50</td>
</tr>
<tr>
<td></td>
<td>std 100.74</td>
<td>119.46</td>
</tr>
<tr>
<td><strong>Commands received</strong></td>
<td>avg 89.94</td>
<td>104.74</td>
</tr>
<tr>
<td></td>
<td>std 25.76</td>
<td>25.39</td>
</tr>
<tr>
<td><strong>Failures</strong></td>
<td>avg 0.04</td>
<td>0.16</td>
</tr>
<tr>
<td></td>
<td>std 0.20</td>
<td>0.47</td>
</tr>
<tr>
<td><strong>TCT (sec)</strong></td>
<td>avg 38.42</td>
<td>37.47</td>
</tr>
<tr>
<td></td>
<td>std 5.42</td>
<td>6.19</td>
</tr>
<tr>
<td><strong>Plan length (m)</strong></td>
<td>avg 817.81</td>
<td>827.48</td>
</tr>
<tr>
<td></td>
<td>std 53.29</td>
<td>57.48</td>
</tr>
</tbody>
</table>

### Table 8.2: Help-A* receives more commands on MOMA’s fifth floor

<table>
<thead>
<tr>
<th></th>
<th>A*</th>
<th>Help-A*</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total time (sec)</strong></td>
<td>avg 896.69</td>
<td>1021.30</td>
</tr>
<tr>
<td></td>
<td>std 114.69</td>
<td>110.97</td>
</tr>
<tr>
<td><strong>Distance (m)</strong></td>
<td>avg 887.81</td>
<td>908.71</td>
</tr>
<tr>
<td></td>
<td>std 66.53</td>
<td>64.02</td>
</tr>
<tr>
<td><strong>Commands received</strong></td>
<td>avg 97.08</td>
<td>180.08</td>
</tr>
<tr>
<td></td>
<td>std 27.15</td>
<td>37.89</td>
</tr>
<tr>
<td><strong>Failures</strong></td>
<td>avg 0.08</td>
<td>0.12</td>
</tr>
<tr>
<td></td>
<td>std 0.28</td>
<td>0.44</td>
</tr>
<tr>
<td><strong>TCT (sec)</strong></td>
<td>avg 30.91</td>
<td>36.10</td>
</tr>
<tr>
<td></td>
<td>std 3.89</td>
<td>5.68</td>
</tr>
<tr>
<td><strong>Plan length (m)</strong></td>
<td>avg 665.85</td>
<td>677.83</td>
</tr>
<tr>
<td></td>
<td>std 13.10</td>
<td>16.00</td>
</tr>
</tbody>
</table>
CHAPTER 8. PLANNING FOR AVAILABILITY

The performance of the robot averaged over these 25 runs appear in Tables 8.1 and 8.2, where boldface denotes statistically significant differences.

There were no statistically significant differences in failures, total travel time, total computation time, or plan length. On both floors, Help-A* received more commands compared to A* ($p < 0.05$). This demonstrates increased availability. The percentage increase in the commands received with the Help-A* planner in the fifth-floor environment is greater than that on the fourth-floor environment. This could be explained by the difference in the topology of the two floors. The fifth floor has more doors and hence more ways to reach a target. This hinders Help-A* compared to A* in the fourth-floor environment; when there is only one way to reach a target, Help-A*'s plan will be no better than A*'s plan.

In conclusion, this chapter demonstrated the application of crowd models beyond collision avoidance for a new problem, the multi-purpose museum robot. There are many other such scenarios where completion of a task would require knowledge of the crowd density, crowd flow, or other crowd behaviors. For example, a security robot could use the crowd flow and crowd density models to automatically generate efficient patrolling patterns. A floor-cleaning robot could use the crowd models to generate navigation plans to vacuum the floor with minimum hindrance to people. The methods developed in this thesis can be applied to learn those behaviors online without expensive global sensing capabilities, thus broadening the set of problems where service robots can be successfully deployed. The next chapter summarizes this thesis and describes opportunities to improve this work.
Chapter 9

Conclusion

This chapter describes the limitations of this thesis and lists opportunities for future work. The chapter ends with a summary of the research questions and the contributions of this work.

9.1 Limitations

The experimental scenarios in this thesis were designed to ensure that robot performance could be improved through global planning. For example, in an environment where crowd density is uniform across the environment, CSA* would not lead to any performance improvements over A*. Thus the experiments demonstrated the best case performance of the algorithms. The real performance of these planning algorithms would depend on the topology of the environment, the crowd density and flow in the environment, and the target locations.

The performance of these algorithms might be sensitive to the choice of the parameters. Choices for the crowd size, the targets, the speed and size of the simulated pedestrians, the size of the map, the robot’s travel speed and laser scanner, CUSUM’s change-detection
parameters, the risky action threshold, and the decision limit could all have an impact on
the performance of the algorithms.

The algorithms in this thesis assume that crowd behavior in one grid cell is independent
from crowd behavior in all the others. This assumption, although it simplifies the algorithms,
has a cost, because the resultant model may be less accurate. A model that accounts for
spatial correlation in crowd behavior could further improve this work.

Finally, in Chapter 4, CUSUM-A\(^*\) completely abandons the old model and relearns a new
one. Although that makes the approach adaptive, there may be useful long-term temporal
crowd patterns to learn as well (e.g., a shopping mall’s movie theater might become more
crowded every Friday for new releases).

\section{9.2 Future work}

This section identifies open avenues for further research.

- Currently, MengeROS assumes that all robots are circular (with a configurable radius)
  and that the laser scan data and the robot’s actions are noise-free. Future work could
  introduce noise, simulate robots of different shapes, process data from sensors other
  than range finders, reduce decision cycle time, and introduce individual pedestrian’s
  reactions to robots.

- Although, significant performance gains were demonstrated through simulation for each
  of the algorithms described in this thesis, real-world experiments can provide further
evidence of these methods’ effectiveness. Evaluation could be extended to test across
  a variety of different maps, crowd scenarios, and tasks.

- In this thesis, different proposed algorithms solve different aspects of the global path
  planning problem. This formulation is useful to understand the effect of one algorithm
without interference from others. For real-world environments, however, the work in this thesis could consider a combined approach that would be effective for all scenarios.

- This thesis considers crowd density and crowd flow. Future work that maps the velocity of the crowd movement in different parts of the environment could also be useful to ensure efficient navigation.

- In environments with multiple service robots, each robot could learn about different parts of the environment and share knowledge to build a more accurate global crowd model.

- Automatic discretization of the environment into a grid whose cells are not of uniform size is another open problem. This would allow the robot to build more granular models in particular parts of the environment.

- Navigation through human crowds is a multi-objective problem; this thesis balances those objectives with weights for the edge costs. Those weights, however, are specific to the task, the crowd, and the environment. Given a new environment, a method that automatically discovers those weights could further improve this work and eliminate the need for manual fine-tuning in each new environment.

### 9.3 Summary

Chapter 3 introduced SemaFORR and MengeROS. SemaFORR is a decision-making architecture for robot navigation. MengeROS is a hybrid 2-D crowd-robot simulator, that has been developed as an independent module to work with any ROS-compliant controller. This allows it to be reused in the development and testing of different crowd-based navigation algorithms (e.g., surveillance or vacuum cleaning).
Chapter 4 described CSA* and Flow-A*. Both are online methods that rely on local on-board sensor data to learn and plan over a crowd model. Flow-A* performed statistically significantly better than CSA* and A* with respect to risky actions and failures, with no significant change in the total time or the distance the robot traveled.

Risk-A*, described in Chapter 5, addressed the important challenge of non-uniform human behavior in the vicinity of a mobile robot. In environments where pedestrians’ behavior hindered the robot’s navigation, Risk-A* combined the crowd data from its laser scanner with its actual travel experience through the crowded area to build a hybrid model. Risk-A* performed statistically significantly better with respect to risky actions, and the number of failures (only in the fourth-floor environment), with no significant change in the total time, or the distance the robot traveled.

CUSUM-A*, described in Chapter 6, detects and adjusts for temporal changes in crowd patterns. This allowed the robot to forget stale models and learn new ones that reflected the current state of the world. This becomes important in scenarios where the robot is deployed for the long term and must continuously adapt to a changing world. CUSUM-A* performed statistically significantly better with respect to risky actions, and the number of failures (only in the fourth-floor environment) with no significant change in the total time, or the distance the robot traveled.

Explore-A*, described in Chapter 7, uses Thompson sampling to introduce exploratory behavior into plan generation. Such plans demonstrate long-term thinking. Rather than optimize performance for only the current task, these plans intentionally explore different parts of the environment. Exploration allows the robot to build a better model of the crowd, and thereby improve its performance over the long term on a sequence of tasks. Explore-A* performed statistically significantly better with respect to risky actions while taking longer routes, with no significant change in the total time or the number of failures.

Finally, Chapter 8 demonstrated the broad applicability of learning crowd behavior pat-
terns. It posed a new task, where the robot must visit a sequence of targets while it simultaneously maximizes its availability. Service robots in crowded environments will be required to address many such problems, where knowledge of crowd behavior in the environment would be beneficial. Help-A* performed statistically significantly better with respect to the number of commands received, with no significant change in the total time, the number of failures, or the distance the robot traveled.

Service robots are increasingly deployed in everyday environments and are expected to complete tasks in the vicinity of people. This requires robots that learn and respond appropriately to crowd behavior. Historically, this problem was addressed with end-to-end pedestrian datasets. The collection of such datasets, however, requires expensive hardware, risks privacy violations, and can become stale as the environment evolves. The algorithms in this thesis learn crowd models online, overcome the limitations of earlier work, and thus make service robots more effective in environments with crowds.
Bibliography


of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction Extended Abstracts, HRI’15 Extended Abstracts, pages 63–64, New York, NY, USA. ACM.


