Robust and Reusable Methods for Shepherding and Visibility-Based Pursuit

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Christopher Vo
Master of Science
George Mason University, 2007
Bachelor of Science
George Mason University, 2005

Director: Dr. Jyh-Ming Lien, Professor
Department of Department of Computer Science

Fall 2014
George Mason University
Fairfax, VA

# Dedication

Dedicated to my mom and dad, for their love, support, and sacrifice throughout my life.

# Acknowledgments

Over the past 12 years I have been a student at George Mason University as both an undergraduate and graduate student, I have met many wonderful people who have contributed to this thesis, and to my personal development. First, I would like to thank my committee members: Jyh-Ming Lien, Zoran Durić, Jana Košecká, and Gerald Cook. In particular, I very deeply appreciate Jyh-Ming for serving as my advisor throughout the Ph.D. program and for his invaluable support on almost every academic work that bears my name. Jyh-Ming's wizardry in the art of problem solving is a humbling inspiration to all of his students, and I owe just about everything I know about the domain of Computational Geometry, as well as the foundational work of this thesis, to him.

Of course, I cannot forget how I began my career in robotics. In the spring of 2005, as a Computer Science undergraduate student at George Mason University, my classmates Brian Hrolenok, Garrett Barton, and Andrew Bovill dragged me by the feet to sign up for CS 499 *Autonomous Robotics*. It was an experimental course taught by Sean Luke. It was in this course that I had my first in-depth exposure to the notion of *robotic swarms*, and it was because of this course that I spent so many of my days and nights at the robotics lab. Eventually, I would spend the next few years working with lab-mates Brian Hrolenok, Keith Sullivan, and Faisal Abidi to develop a team of humanoid robots. To this day, I owe a great deal of my engineering knowledge about robots (or perhaps "robot repair skills") to the time I spent working at the lab with these brilliant colleagues and friends.

At the end of spring semester 2007, Sean Luke recommended that I join the CS Ph.D. program. At the time, I was not very interested in an academic career, and had no intention of submitting my application to the Ph.D. program. It was on the very due date that my mentor Dr. Paula Shaw pushed me to complete and submit my application. If it weren't for Sean's recommendation and Paula's nudging, I would not have even begun to explore this fascinating field. I began as an apprentice of Sean Luke, who taught me indispensable fundamentals and sensibilities of scientific research.

Thanks to the several grad students from the MASC research group who have lent considerable advice and support in this process: Evan Behar, Joey Harrison, Guilin Liu, Yanyan Lu, Arsalan Mousavian, and Zhonghua Xi; as well as the many others who have contributed directly to my work including Stephen Donnelly, Nikhil Garg, and Sam McKay. In particular, I would like to single out Joey Harrison, who lent a great deal of support and leadership in the shepherding work. Our work together in shepherding forms the basis for a large part of this thesis. Thanks to Lisa Huynh for editing and support. Also, thanks to the wonderful and diligent staff at the George Mason University Computer Science department for their help despite all the paperwork I've caused over the years.

Finally, I owe infinite thanks to my mom and dad. It is because of their love, encouragement, and support that I could dedicate my time and energy to academic research.

# Table of Contents

# List of Tables

# List of Figures

# Abstract

ROBUST AND REUSABLE METHODS FOR SHEPHERDING AND VISIBILITY-BASED PURSUIT

Christopher Vo, PhD

George Mason University, 2014

Dissertation Director: Dr. Jyh-Ming Lien

Algorithms for the control and monitoring of swarms of moving agents are important in a wide variety of real and virtual applications such as crowd control, livestock herding, and decentralized robot control architecture. In the presence of obstacles, these applications can be quite difficult, especially with large swarms. This thesis presents reusable and robust motion planning algorithms for the swarm control problem of *shepherding*, a task involving using a small set of mobile robots interact with a larger set of swarm agents; and the swarm monitoring problem of *visibility-based pursuit*, a task involving using a mobile robot to follow and maintain visibility of a moving swarm. For both problems, we developed algorithms to efficiently sample reusable geometric information in the environment to enable fast online planning and replanning. For the shepherding problem, we discuss several representations and abstractions for flocks to improve scalability and robustness to uncertainty. For the visibility-based pursuit problem, we discuss several methods for space decomposition that enable fast online planning to achieve visibility objectives. We validate our results with multi-agent simulation software to understand the tradeoffs between different techniques for these problems.

# Chapter 1: Introduction

Swarms exist in many fascinating forms. In nature, swarms refer to clusters of moving, interacting organisms. Emergent collective movement behaviors arise from the simple interactions between agents in a swarm. In some cases, swarm behaviors can improve the success of an organism in foraging for food or finding mates, or enhancing their defenses against predators (as in the "selfish herd" theory by Hamilton [68, 87]). Many birds fly collectively in formation to improve aerodynamic efficiency [105]. Ants and termites lay chemical trails or pheromones in their environment as they forage for food or build nests—a process of indirect communication called *stigmergy* [65]. As humans, we also exhibit swarm-like behavior, as we tend to align in heading and walk behind our nearest neighbors [128]. Swarm behaviors have caught the attention of roboticists because of their decentralized, robust, and self-organizing properties.

The control and monitoring of swarms is important in a multitude of applications and real life scenarios. For example, there has been much interest in how to use automated or virtual barriers and fences to manage free-ranging animals such as cattle [3, 26, 132]; or dynamically control the moment or placement of barriers, signs, or audio beacons to provide subtle control of a large group of people [74, 75], such as in a disaster response scenario [20]. A Penn State report in 2001 [85] highlighted the lack of research into effective strategies for crowd control and the catastrophic consequences that may result from flawed strategies. To harness the power of swarms in robotics, we need efficient motion planning algorithms for controlling and monitoring swarms of robotic agents. This thesis is about how to do that in a scalable and robust manner.

Motion planning is a fundamental area of robotics that involves using a robot's knowledge of its environment to make decisions about how it should move next through the

domain of its possible configurations, or *configuration space*. The literature in robotic motion planning is vast and impressive, with algorithms addressing a broad range of problems and scenarios. Unfortunately, traditional motion planning algorithms perform poorly and scale poorly in controlling and monitoring large swarms of interacting agents, especially due to the high dimensionality and complexity of the configuration space. In this thesis, I will show how we used geometric processing and space decomposition techniques to improve the performance of motion planning for two relevant swarm problems: *shepherding* for swarm control, and *visibility-based pursuit* for swarm monitoring.

## 1.1 Shepherding

The first motion planning problem we will examine in depth in this dissertation is *shepherding*. The shepherding task requires planning the motion for one or more moving *shepherd* agents to interact with a potentially large flock of the *sheep* agents and influence their movement. Here, a simple principle guides the interaction between each agent: shepherd agents influence the movement of the flock by moving close to it, which creates tension. The sheep react by moving away from the tension. Our goal is to come up with a sequence of movements for a small number of shepherds to keep the large flock of sheep intact, and also move it successfully from pasture to pasture.

There are a multitude of applications for shepherding, such as security (e.g., simulation of disaster scenarios and responses [20, 74, 136]); civil crowd control (e.g., planning evacuation routes for sporting or spectator events [75]); pollution control (e.g., collecting oil spills [52]); agriculture (e.g., sheep herding [132]); transportation safety (e.g., preventing bird strikes [53]); education and training (e.g., providing immersive museum exhibits and training systems); and entertainment (e.g., interactive games). The heterogeneous architecture of shepherding also enables a scalable architecture for controlling large swarms of robots [147]. For example, instead of a planning architecture where a central planner must monitor, coordinate, and command a large number of individual robots, we can use a

distributed architecture based on shepherding. In this distributed architecture, each agent in the swarm is equipped with simple shepherding behaviors, reducing the motion control problem to controlling a much smaller set of shepherd agents.

How do we plan the motion for such robotic shepherds? Assuming an obstacle free environment, and that we know that the flock follows a basic potential field model, it can be rather simple. In 2000, Vaughan et al. were successful in developing a "robotic sheepdog" and control algorithm that could move live ducks from an initial configuration to a given goal area [151]. However, these methods are not sufficient when the environment contains obstacles. While many current manipulation planning methods make use of approximate models of the geometry and dynamics of the world, their accuracy is unknown—that is, subject to *modeling uncertainty*. The movement of the robot and objects it manipulates is imperfect, introducing *action uncertainty*. This can also lead to uncertainty in the configuration of the objects relative to the robot, or *object pose uncertainty*. Some planning methods use information gathered from a sensor to decide what to do about unexpected situations during execution of a movement, but even this information is susceptible to *sensor uncertainty*. Sensors mounted onto robots such as cameras and laser rangefinders must deal with the problems of limited range and occlusion.

There are other useful variants of the shepherding problem, such as escort or protection [163]. In the escorting problem, the task of an agent in a "parent" role is to maintain or maximize a separation between its "children" agents and some "stranger" agents. In such a problem, it is necessary for the parent to react quickly to a possibly adversarial agent. In Chapter 5, I will discuss other possible variants such as manipulation of a rigid object.

## 1.2 Visibility-Based Pursuit

The second problem this dissertation will address in depth is *visibility-based pursuit*. The objective of visibility-based pursuit is to plan motions for a "supervisor" agent to follow and maintain visibility of a coherent flock of constantly-moving "target" agents that have

unknown trajectories. In this problem, it is necessary for the supervisor agent to predict occlusion risks and plan quickly to cover them so that it can maintain visibility of as many target agents as possible.

Our initial goal was to develop a way to monitor swarms of collaborating robots, but we found that visibility-based pursuit is useful in many real and virtual world applications. For example, several works have explored the use of autonomous unmanned aerial vehicles (UAVs) with camera sensors to follow and monitor endangered species populations in poaching prevention efforts [117]. In computer gaming, visibility-based pursuit can be used to plan motions for the player's camera to follow an army of virtual characters in an obstacle-filled battlefield. In crowd control, visibility-based pursuit could be used by automated surveillance cameras to track a crowd of protesters as they march through an urban environment. In robotics, visibility-based pursuit algorithms could enable an automated camera attached to robotic manipulators to observe live performers in a concert, monitor assembly of a mechanical system, or maintain task visibility during teleoperated surgical procedures. For an overview of the broad applications and studies on the topic of visibility-based pursuit, see the survey by Christie et al. [37].

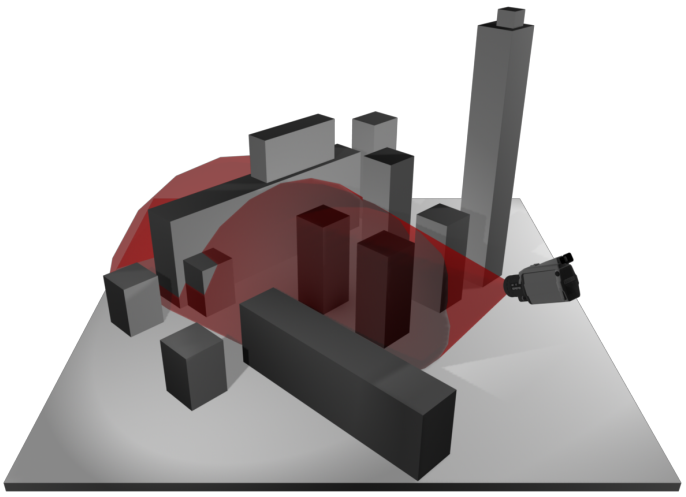


Figure 1.1: An illustration of visibility-based pursuit in a city-like environment.

Building a camera to pursue moving targets can be viewed as an online motion planning problem in which the camera must be able to predict the motion of the targets, and proactively move to ensure visibility of those targets in the future. In Chapter 4 of this dissertation, I will discuss the methods we developed to address this problem.

## 1.3 Methodology and Contributions

The main thread of this dissertation is the use of geometric methods to extract information from the environment and express it in such a way that helps us improve the robustness and effectiveness of shepherding and tracking among obstacles. In particular, I studied the shepherding and visibility-based pursuit problems. I used a simulator and ran numerous experiments to test design ideas. This thesis offers an analysis of these design choices and methodologies for the control and monitoring of swarms of agents.

For the shepherding problem, I developed motion planning methods for the shepherd robots that compute and reuse geometric information about the workspace to improve the efficiency of planning, replanning, and processing similar environments; and are more robust than current methods to uncertainties in the environment model, the localization of the flock, and their movements. While this research focused narrowly on the shepherding problem, an examination of these design decisions paved the way towards a better understanding of how to deal with uncertainty in other real-world manipulation contexts, and provided insight about how to develop efficient, reusable methods for robotic motion planning.

For the visibility-based pursuit problem, it is important for the robot to be able to quickly identify areas of the free space that represent occlusion risks so that the supervisory agent can quickly react to cover those risks. Therefore, in visibility-based pursuit I also applied a similar methodology as in shepherding: I compute reusable information about the environment model and partition the free space in a form that is less sensitive to small discontinuities or noise than previous space partitioning methods, creating a roadmap that accurately represents the visibility relationships between different areas of the space. Through this

method, it was possible to classify areas of the space that share similar visibility characteristics and determine a space decomposition and topology that is more coherent and robust for making decisions involving visibility than previous approaches. Since we are interested in approaches that improve the success rate, or the number of times the planner was successful in in generating a collision-free path, I evaluated the performance of the planner over repeated queries on the same or similar environments.

## 1.4 Organization

In Chapter 2, I provide a survey of the current literature related to shepherding and visibility-based pursuit as background material to understand the context, applicability, and foundations for the rest of the dissertation. Chapter 3 discusses the shepherding problem in depth and the strategies that we used in this research to create efficient algorithms for shepherding, from a simple PRM and RRT based planner to the use of geometric shapes to shepherd swarms of agents as an aggregate. Chapter 4 discusses the visibility-based pursuit problem in depth and the various strategies used in the pursuit problem. The final chapter synthesizes my contribution to the community of swarm motion planning and provides several examples of how the lessons learned in my analysis may be applied to many other areas and future avenues of research.

# Chapter 2: Background

This chapter provides basic background material to give the reader context for understanding the analysis discussed later. By the end of this chapter, the reader should have a broader conception of the field of swarm motion planning, shepherding, and visibility-based pursuit, and the work that has been done to understand those fields. The next chapter will describe the specifics of the framework upon which I have centered my analysis and experiments in shepherding and visibility-based pursuit.

## 2.1 Overview of Related Motion Planning Problems

The literature in motion planning is vast, covering a broad range of problems and scenarios. At the highest level, motion planning problems involve modeling the environment and the robots or moving objects within it, and then using this model to compute a sequence of admissible motions for the moving objects to reach a goal state. In this section, we will discuss the basic motion planning algorithms that form the basis for the analysis in Chapter 3 and Chapter 4, and relevant extensions that have been studied for achieving robustness, dealing with uncertainty, and manipulating objects.

### 2.1.1 Probabilistic Motion Planning

Probabilistic motion planning algorithms and data structures such as Rapidly Exploring Random Trees (RRTs) [98], Expansive-Spaces Trees (ESTs) [73], and Probabilistic Roadmaps (PRMs) [84] have proven to be elegant and powerful ways to find paths in high-dimensional configuration spaces. The general idea of these approaches is to create a topological representation of the connectivity of the free space. In the case of RRT and EST, this representation

takes the form of a tree-like data structure. These methods construct a tree with its root representing the initial configuration of the moving objects. The tree is expanded iteratively until a path is found. In the case of PRM, this representation takes the form of a graph, where any two configurations are connected if there is a feasible path between those configurations. Practitioners have extended these baseline probabilistic motion planning algorithms to cover a wide variety of complex scenarios including robots with various kinematic, dynamic, or non-holonomic constraints [91]; manipulation planning [110, 144]; and multi-agent planning [82, 83, 111]. Later in this dissertation, we will show how we extend these basic RRT, EST, and PRM motion planning algorithms to perform shepherding of swarms of robots in a robust and scalable manner.

### 2.1.2 Achieving Robustness in Motion Planning

One of the focuses of this dissertation work is to improve the robustness of motion planning for swarms. There exist many techniques for developing more robust plans in traditional motion planning. A common approach for dealing with uncertainty is to reduce the planning horizon so that it is possible to perform replanning quickly during execution of a plan. Along the same lines, some researchers have focused on implicitly dealing with uncertainty of future states through using a concept known as *feedback motion planning* [96, 164]. Unlike traditional motion planners, which typically generate a path and execute it in an open-loop fashion, feedback planners use information about the state space to generate a navigation function that maps every state of the system to an action. These mappings often resemble vector fields, compositions of funnels, or gradient maps. To execute a task, the robot must repeatedly estimate the state of the system, and execute the action that the navigation function prescribes for that state. By covering all possible states, feedback planners implicitly offer a recovery plan for unexpected situations that may arise in the motion of the robot.

Another popular approach to handling uncertainty in motion planning is to model the problems as *Partially Observable Markov Decision Processes (POMDPs)*. POMDPs model the problem as one of an agent moving through a state space by taking actions. The uncertainty

of the end state of actions, and the uncertainty of observation are explicitly modeled using conditional probability functions. At each step, the agent receives a reward. The solution of these POMDPs is an optimal policy that maximizes the expected total reward. Due to the "curse of dimensionality" and the general difficulty in representing such probability distributions, POMDPs are notoriously difficult to solve exactly. However, over the years, many approximation algorithms [70] and point-based algorithms such as HSVI2 [137] and SARSOP [93] have been shown to handle up to 100,000 states in reasonable time [92].

Another general approach to handle uncertainty is to use a hybrid (or "hierarchical") motion planning approach, such as in [34, 80]. This approach is similar to *three-layer architectures* [130], where control is separated into layers, each with a different level of precision and planning horizon. For example, it is typical to construct a planner that uses local reactive control to handle immediate geometric constraints, and a global symbolic planner that handles high-level task specifications. One class of methods that uses this technique is *temporal logic motion planning* [49]. In this technique, the planner obtains a discrete, topological abstraction using a projection—for example, through a cell decomposition method. The planner then computes a high-level plan in this discrete space using automata theory, and implements each step of the plan in the continuous space using local feedback control. The hierarchy of using a high-level planner with low-level controls allows such a planner to handle complicated high-level temporal logic constraints while also providing more robust local feedback control for the robot to move safely between space partitions.

Parker et al. [51, 71] handled this issue by adopting a heterogeneous approach i.e., by partitioning different roles to three distinct classes of robots. A similar heterogeneous robot architecture is also seen in in several other works. For example, Chaimowicz and Kumar [27] developed a system called "Aerial Shepherds" which employs a small number of unmanned aerial vehicles (UAVs) to escort groups of unmanned ground vehicles (UGVs). In Chaimowicz and Kumar's work, the "shepherds" do not interact with the ground vehicles through motion; instead "shepherding" simply refers to the relationship between UAVs and UGVs within this command and control architecture.

### 2.1.3 Dealing with Uncertainty in Motion Planning

Recently, probabilistic reasoning has been introduced for sensor-based grasping, such as the use of particle filters to estimate the changing probability distributions of the pose of the object and find reliable grasps [94]. In related work, Feiten et al. in [50] presented how to use a convex combination of Projected Gaussians in lieu of multivariate Gaussian distributions to represent the probability density function (pdf) of a 6D pose consisting of a 3D translation component and an orientation. Such a representation permits more efficient calculation of compositions of rigid motions from the pdfs of individual rigid motions.

Some planning methods handle uncertainty by explicitly modeling or taking advantage of a priori knowledge of the sensors and controller that will be used to execute the task. This kind of modeling typically allows the planner to measure the quality of a path, and select a path from a set of candidate paths with respect to that measure. For example, in [148, 149] Jur van den Berg, Pieter Abbeel, and Ken Goldberg presented an approach for assessing the quality of a path by assuming a linear-quadratic controller and Gaussian models of motion planning and sensing uncertainty (LQG-MP). This can be used to determine the probability that collisions will be avoided, or the probability that the robot will arrive at the goal.

Another relevant representation for uncertainty in motion planning is the use of *uncertainty roadmaps*, which are graph data structures for capturing the probability of successfully transitioning from one valid configuration (or set of configurations) to another. With this information stored in a roadmap, it is possible to query paths that balance cost and uncertainty using discrete path finding algorithms such as $A^*$ [25].

### 2.1.4 Manipulation Planning

We can also view the control of groups of agents as a robotic manipulation problem. For example, manipulation planning researchers have attempted to use multiple robots to cooperatively manipulate or move passive objects, such as pushing a disc [150], a box [162], or kicking a ball [139]. A passive object moves only if external force is applied to it. On the

other hand, swarm control sometimes attempts to manipulate the motion of active objects which may have some ability to move on their own. This combined with the amorphous nature of a swarm generally makes active objects more difficult to control. So far, few methods have focused on the idea of manipulating multiple active objects using multiple robots.

In general, the difficulty in estimating the configuration of the system in manipulation planning has led some researchers towards methods that attempt to greatly restrict the way the robot interacts with the obstacles. For example, to reduce the burden of estimating an object's state to move it, some algorithms require a robot system to *grasp* an object before moving it. Planning grasps is a difficult problem in itself, and there are some works that have considered manipulation via grasping. For example, Lozano-Pérez, Mason, and Taylor have developed the *preimage planning framework* for performing manipulation planning under uncertainty [47, 106]. A *preimage* is defined as a region in space such that if the robot executes a motion within that region, it is always guaranteed to reach a given goal region regardless of any source of uncertainty. This leads to the approach of *preimage backchaining*, where the planner begins by finding preimages of the goal, then preimages of those preimages, and so on recursively, until the initial region is covered. There are also other variations of the grasping problem. For example, Platt et al. considered the problem of *Simultaneous Localization and Grasping*, where the robot must attempt to estimate the position of the object while a grasp is occurring [124].

While grasping is a popular way to manipulate objects, it is also very restrictive, and in fact may not be possible for particularly heavy or unwieldy objects. There are other natural ways to manipulate objects, such as by pushing, shaking, sliding, or throwing them. Therefore, it is also of interest to consider *nonprehensile manipulation*, or manipulation without grasping. Recently, Dogar and Srinivasa introduced a framework for planning in clutter using a library of actions derived analytically from the mechanics of pushing [42]. A frequently cited planner for pushing manipulation is the work of Akella and Mason [1], who consider pushing a polygon using a flat edge (called the "fence"). To do so, they derive a *Push Stability Diagram (PSD)* for the polygon to be manipulated. This table describes

the stable edges of the polygon and the maximum angle that a fence can use for each edge to stably reorient it. There have also been works focused on estimation of state via manipulation despite the lack of reliable sensor information. For example, Erdmann and Mason have explored *sensorless manipulation* [48], where by dropping an object into a tray and performing a sequence of tilting operations, the number of possible orientations is reduced; sometimes leaving the orientation of the object completely determined. Similarly, Goldberg and Mason [58–60] have explored manipulation of objects by squeezing. That is, because of the way that the object complies to the squeezing operations, it may be possible to design a sequence of squeezing operations that leads to a known orientation of the object. Another relevant example is the work of Nieuwenhuisen et al. [41] who consider the problem of pushing a disc through an obstacle-filled environment while making use of *compliance*—that is, allowing the object to slide along the boundaries of the environment.

## 2.2 Shepherding

### 2.2.1 Multi-Agent Simulation and Modeling

We are interested in controlling large groups of active agents by providing external stimuli. Several works exist along the vein of understanding the emergent behavior of simulated crowds when significant changes are made to the environment, such as adding movement barriers or additional agents. For example, the effect of adding barriers into disaster scenario environments was studied in [20]. Schubert and Suzić [131] used a genetic algorithm with agent simulation to determine optimal barrier deployments for controlling rioting crowds. Other works have modeled the effect of adding agents with attractive or repulsive social forces [88] and agents with different roles (such as the presence of “leader” agents [6]). Yeh et al. experimented with composite agents [163] which can exhibit different behaviors such as “guidance”, using proxy agents as temporary obstacles.

There have been several experiments to understand how to control living organisms using robotic agents. For example, Halloy et al. [66] showed that robotic agents can influence

the collective behavior of a group of cockroaches through local interactions. Ogawa et al. [116] demonstrated the ability to track and manipulate *Paramecium caudatum* cells in a chamber using an electrical field. Some experiments have involved herding cows using "smart collars" which act as virtual fences [3, 26]. When a cow approaches a virtual fence, the collar produces a noise which influences the cow's movement away from the fence. In the study of telerobotic systems, Wilson and Neal [161] performed a case study where with varying degrees of autonomy, a human carries out the task of controlling a robotic sheepdog to direct robotic sheep. The results of the study imply that there is a diminishing return in the reduction of human effort to control the sheep in relation to a given investment in the engineering effort to produce more autonomous control. A biological study in the modeling of shepherding behavior and the movement trajectories of a live Austrian sheepdog and merino sheep by King et al. [87] illuminated a few properties of herding sheep. For example, their observations found that sheep exhibit a strong attraction towards the center of the flock under threat, an example of the "selfish herd" theory by Hamilton [68].

Several works in robotics and computer animation are related to modeling behaviors such as shepherding. For example, Schultz et al. [133] applied a genetic algorithm to learn rules for a shepherd robot. Its objective was to control the movement of other robots (sheep) that react by moving away from the shepherd. Vaughan et al. [151] simulated and constructed a robot that shepherds a flock of live geese in a circular, obstacle-free environment. In computer animation, Funge et al. [55] simulated a shepherding behavior in which a *Tyrannosaurus rex* chases raptors out of its territory. Potter et al. [125] studied a herding behavior using three shepherds and a single sheep in a simple environment. Many other models have been considered, such as cellular automata [18], behavior-based modeling [21, 55, 126, 146] and flow-dynamics-based simulation [33, 74, 141, 145]. Most recently, Strömbom et al. [140] developed an algorithm for shepherding based on real data collected from sheep-dog interactions. While some of these approaches have swarm control in mind, most of the existing simulations [90, 122, 159] have only considered simple scenarios, and none of the aforementioned methods have shown the ability to guide flocks through environments with obstacles.

Scalable group simulation is an important application of computer graphics, particularly in simulating crowds and pedestrians [5, 18, 108, 129]. A critical requirement of a natural looking simulation is to populate the environment with large sets of interacting agents. This requirement raises several scalability issues in behavior computation, trajectory control, rendering, and authoring. Several methods have been proposed to deal with the computational scalability issues using topological data structures [95], hierarchical structures [135], k-d trees, quad-trees [138], and the concept of "region of interest" [111]. Computational scalability can also be tackled using top-down approaches based on the observation that the dynamics of a crowd share many common properties with flow dynamics. Rather than simulating individual agents, the flow of a crowd is computed [33, 74, 141, 145].

Multi-robot systems may also move in formation [7] to accomplish a given task. For example, Lien et al. [102] observed that formations can be used effectively to control the motion of a flock of agents. Similar observations have also been found in sociological studies of crowd control where similar formations are used by police and military personnel to disperse, contain, or block crowds and prevent riots [4]. More recently, Shell and Matarić [136] have used multiple robots to deploy and assist with the evacuation of pedestrians.

In many existing approaches to shepherding, a bounding circle is used to model the flock. This is sufficient for relatively sparse workspaces, but is excessively restrictive in environments with many obstacles or narrow corridors, or when the flock size is large. On the other hand, some researchers have looked at how to use deformable shapes. Notably, Kamphuis and Overmars [82, 83] have considered the problem of finding paths for coherent groups. They used a hinged-box to represent a group of agents navigating through an obstacle-filled environment.

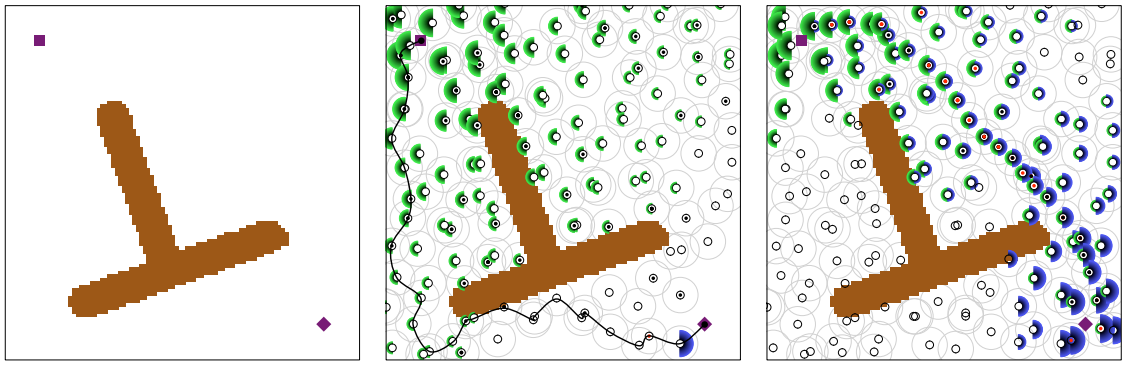
The general problem of planning the motion for one or more robots in dynamic environments is also relevant to our study. There are many previous approaches in this area which vary from speed optimizations in RRT, to algorithms that update roadmap data in real-time, to gradient descent methods [28]. For example, Bruce and Veloso [23] developed a planning algorithm called ERRT which uses optimizations such as a *waypoint cache* and an adaptive

bias schedule to increase the performance of RRT so that it can be used in real-time navigation. Sud et al. [143] use an adaptive roadmap that continuously updates in real-time to plan motions for multiple agents in a dynamic environment. Fulgenzi et al. [54] developed a navigation algorithm based on RRT which incorporates an estimation of the likelihood of obstacle trajectory, and probability of collision. These estimates are updated in real-time to handle dynamic obstacles. It is important to note that most work in multi-agent motion planning is focused on direct control of a group of agents, whereas we are interested in control via agent-agent interactions.

### 2.2.2 Cooperative Multi-Agent Systems

Swarm control is also a problem that involves multi-agent cooperation. A survey from Parker [119] provides an overview of multi-robot systems. From the perspective of multi robot systems, the task of group control requires *inherent cooperation*, in which the success of a robot in the team depends on the actions of the other robots. Inherent tasks (such as swarm control) are distinguished from non-inherent tasks (such as covering) in that they cannot easily be decomposed into independent sub-tasks and thus are generally more difficult.

In Hrolenok et al. [72], we looked at the foraging problem, a collaborative multi-agent problem wherein the goal is for a set of robots to collaboratively find food in one location and deliver it to another location. The approach we used mimics the stigmergic use of pheromone trails by ants, but with a twist: the stigmergic information is stored in a sparse set of beacons distributed through the space, rather than storing the pheromones continuously in the environment or on a dense grid. These beacons can be placed by robots, moved around freely, and their value can be updated by the robots continuously. An illustration of the approach appears in Figure 2.1. The approach is stigmergic, meaning that the agents store information in the environment and act only locally. This feature allows the approach to be fully distributed, and also allows for quick response to dynamic changes in the environment despite using only local information. While the model we used explicitly placed physical beacons in the environment, these desirable properties also lent themselves to methods



(a) (b) (c)

Figure 2.1: An example of collaborative foraging with beacons. Figure 2.1(a) shows the problem, with the home position (square) and the food source (diamond). Figure 2.1(b) shows the environment after a few iterations, with beacons emitting pheromones pointing towards home (green half-circles) and emitting pheromones towards the food (blue half-circles). An initial path is found. Figure 2.1(c) shows the environment after the path and beacon placement has been optimized through collaborative foraging.

involving *virtual* trails of beacons. In the future, we believe that the advantages of this approach could improve the work in this dissertation. Using similar update rules, it may be possible to possible to continuously optimize paths and roadmaps, and account for dynamic obstacles by relocating and updating simple stigmergic information.

Swarm control is also related to competitive activities such as pursuit and evasion behaviors, and sports such as soccer. A simplified version of the pursuit and evasion problem that considers only one pursuer and one evader has been studied for decades using methods such as game theory [77], co-evolutionary algorithms [127], and neural networks [38] (see survey in [109] for details).

## 2.3 Visibility-Based Pursuit

### 2.3.1 Pursuing Targets with Known Trajectory

There are methods for pursuing a target with a known trajectory, such as work done by LaValle et al. [97] using dynamic programming, and by Goemans and Overmars [57] using

probabilistic motion planning. LaValle et al. [97] used dynamic programming to find optimal camera trajectories that maintain the visibility of a target whose trajectory is known. They use a regular grid for each target location and identify all cells in the grid where the target is visible. While their method can consider different objective functions and constraints easily (e.g., shortest path, view range, target speed), it is slow and not scalable especially in higher dimensions than 2D. In [97], they also consider partially-known paths. Goemans and Overmars [57] also proposed a probabilistic motion planner to follow an object with known path. They consider smoothness of the camera and apply the method on both 2D and 3D problems.

### 2.3.2 Pursuing Unknown Target Trajectories in Known Environments

There exists work considering the pursuit of targets with unknown trajectories in a known environment [14, 112, 113]. The general idea is to partition the space into non-critical regions in which the camera can follow the target without complex *compliant motion*—that is, rotating the line of visibility between the camera and the target around a vertex of an obstacle. The main benefit of this line of work is the ability to determine the *decidability* of the visibility-based pursuit problem [16]. Unfortunately, the decomposition usually results in many small components even for a very simple environment. The brute-force approach for computing a visibility graph, visibility polygons of all vertices, and the visible regions of all finely discretized grid cells, is not scalable enough for real-world environments.

Recently, Li et al. [100, 101] have proposed a real-time planner that tracks a target with unknown trajectory. Their main ideas include a *budgeted roadmap method* with lazy evaluation and a simple linear extrapolation to predict the target's motion. However, their method has no guarantees on the performance and quality of the camera path. Oskam et al. [118] developed a *visibility transition planning* method which exhaustively precomputes visibility on a roadmap of overlapping spheres in the free space. This visibility roadmap enables quick prediction of spheres that represent high risk of occlusion so that the pursuer can take proactive motion to prevent this occlusion. However, this grid-based sphere generation

step has difficulty in narrow passages, and exhaustive visibility computation is inefficient. Geraerts [46] proposed the idea of using a *corridor map* to track a single target. The corridor map is a decomposition of the environment computed by sampling the Generalized Voronoi Diagram of the free area of the workspace, and local motions are controlled by potential fields inside a corridor.

### 2.3.3 Pursuing Unknown Target Trajectories in Unknown Environments

Many researchers have also considered the case that both trajectory and environment are unknown. For example, Becker et al. [12] proposed a very simple planner to follow a target with unknown trajectory in an environment with landmarks. The idea is to predict the target's next position and place the camera at the position that can see most of the predicted target positions. Later, González-Baños et al. [62, 99] proposed the idea of greedily minimizing the *escaping risk* or maximizing the *shortest escaping distance* of the target. Recently, Bandyopadhyay et al. [8–10] improved the definition of *escaping risk* by introducing the idea of *vantage zone* and showed that this definition can improve the camera's ability to pursue the targets. All of these methods focus on pursuing a single target. There are also extensive studies on pursuing multiple objects, e.g., [134], in which the goal is to maintain a belief of where the target(s) are by using particle filters.

### 2.3.4 Methods Based on Constraint Satisfaction

In computer graphics, camera planning is often viewed as a constraint satisfaction problem, and so there have been attempts to represent the problem so that it can be solved efficiently with constraint satisfaction techniques. For example, some works use the idea of *screen space* or *image space* constraints, e.g., Blinn [17] and Gleicher and Witkin [56]. Given desired screen locations of observed points, these methods compute the necessary camera location and orientation. Gleicher and Witkin [56] also proposed an improved method based on *image space constraints* that involves tracking one or more static or dynamic points, or maintaining

a certain area constant within the image. This method relies heavily on control theory, and also requires information about the object's trajectory. Occlusion of the target is also not considered. There are a number of works which involve the use of metaheuristics to compute optimal positions or trajectories for the camera. For example, Drucker and Zeltzer [43] used an A* planner to compute a camera path. Along the path, the orientation of the camera is then solved frame-by-frame to satisfy given constraints. Burelli et al. [24] used particle swarm optimization to compute positions of the camera that satisfy certain constraints while optimizing other objective functions. However, this work generates only camera positions, not paths.

We can also represent constraints geometrically. For example, Bares et al. [11] proposed a method to find camera positions (rather than a path) to meet given constraints. In this method, the constraints are represented as convex objects. The solutions are in the intersection of these convex objects. Using the idea of relaxation (satisfying only a subset of the constraints), and decomposition (returning multiple camera positions that jointly satisfy the constraints), they describe fallback procedures in case no single position can be found to satisfy all constraints. Jardillier and Languénou [78] proposed an offline planner (assuming known object geometry and trajectory), where constraints are specified using a declarative model. The constraints are then solved using *interval mathematics*. Their method allows users to specify desired image locations using frames (rectangles). That is, objects can be fully or partially included, or fully excluded. Covering area of the target is also considered (otherwise, camera can be so far away that objects become points and still satisfy constraints). They solve constraints using *interval arithmetics*. Later, Christie and Languénou [35] proposed another declarative model approach, this time to describe trajectories of cameras as sequences of parameterized elementary movements called *hypertubes*. However, this method was designed for offline purpose, taking 3 to 6 seconds to solve for a single given trajectory. The aim is to determine the path of a camera to satisfy various declarative properties on the desired image, such as location or orientation of objects on the screen at a given time. Their

method uses an incremental solving process: (1) interval-based filterings on the first hypertube to remove areas without solutions, (2) choice of a solution among the set of remaining areas with respect to an optimization criterion to determine starting conditions for the next hypertube, and (3) backtracking for inconsistent hypertubes. Christie and Normand [36] used this idea to partition the space so that in each partition, the cinematographic properties remain the same. However, their method is designed to produce a camera view that satisfies certain criteria—not to follow targets. Recent surveys on camera planning in the area of computer graphics are also available from [36, 37].

Coleman et al. [39] presented two solutions to the camera interpolation problem. The first uses linear matrix interpolation to interpolate camera pose matrices. The interpolation can be a linear interpolation, a spline-weighted blend, or an interpolating blend. The second approach performs interpolation in image-space. Image-space interpolation was found to provide additional control over the trajectories and sizes of selected objects in screen space. This allows the user to use far fewer key frames.

### 2.3.5 Reactive and Real-time Behaviors

Some studies have focused on developing reactive behaviors for real-time camera motion. For example, Courty and Marchand [40] used visual servoing along with obstacle and occlusion avoidance. Prediction of a target's movement is also relevant for developing real-time camera behaviors. For example, Halper et al. [67] introduced a camera planner that predicts state based on the past trajectory and acceleration. They also proposed the idea of PVR (potential visibility region) for visibility computation, and a pipelined constraint solver.

Hughes and Lewis [76] also conducted a study on the performance of camera control on a mobile robot, using three control models: coupled camera controls, independent control (the camera and robot can move independently), and multi-camera control (one coupled camera and one independent camera). They found that using independent control provides better performance in finding unknown objects, and that human observation, supervision, and judgment remain critical element of robotic activity.

### 2.3.6 Other Related Problems

Some studies have focused on the problem of estimating the position of targets despite uncertainty in the sensor and frequent occlusions. In many such works, the approach is to assume a motion model for the targets, and then use that motion model (perhaps along with a probabilistic filter) to make plausible hypotheses for future locations of the targets. Recently, Gong et al. [61] have proposed a motion model for object tracking that is based on enumerating the most likely *homotopy classes* of trajectories (sets of trajectories that can be smoothly deformed into one another without intersecting obstacles).

Several works study novel ways to find and pursue relevant areas of the workspace. For example, Andujar et al. [64] proposed a method for looking at "interesting" spots in the environment. They define "interestingness" from the idea of entropy. They also proposed simple methods using 2D distance fields to decompose a given environment into rooms and passages. Morbidi, Freeman, and Lynch [89] developed a distributed control strategy for UAVs via nonlinear gradient decent to match a discrete set of particles that describe the occurrence of "events of interest" to monitor. The goal of their work is to match the geometric moments of the swarm with those of the ensemble of the particles.

Luke et al. [107] have applied methods using hill-climbing/k-means clustering and hybrid to the problem of CMOMMT (a NP-hard problem by Lynne Parker [120]). This problem has multiple moving targets (more targets than observers), but no obstacles. Nieuwenhuisen and Overmars [115] considered the first person view using motion planning with a focus on path smoothness and naturalness. Multi-objective planning is achieved by combining weights (penalties) and encoding them in the graph edges. Bhattacharya et al. [14, 15] studied the *decidability* problem for both 2D and 3D environments, which determines if the camera can always maintain visibility or will eventually will lose track of the target for a given environment. This work is relevant, but when the target has escaped, the camera has no way to regain visibility.

# Chapter 3: Shepherding

In the shepherding problem, there are two types of agents: a set $S$ of *shepherd* agents, and a set $F$ of *flock* agents, moving about workspace $W$ filled with obstacles. A shepherd agent has the ability to influence the movement of flock agents. The flock agents that form $F$ move together with some level of cohesion through the environment and away from shepherds. The task of the shepherds is to steer the flock to a desired location. The *configuration* of an agent is represented by its position and velocity—e.g. we specify the state of a shepherd $s \in S$ as $(x_s, v_s)$. Therefore, a shepherding problem with $n$ flock members and $m$ shepherds in a 2D workspace will have a configuration space in $4(n + m)$ dimensions. We refer to the joint configuration of the shepherds $S$ at time step $t$ as $S_C(t)$ and the $F$ as $F_C(t)$. A *valid* configuration is one in which no agents are in collision with obstacles or other agents.

Each flock member in $F$ adheres to a set of simple rules, similar to the Boids program developed by Craig Reynolds in 1986 to simulate the flocking behavior of birds [126]. In the Boids model, we assume each flock member steers itself according to three basic rules: *separation*, steering to avoid collision with nearby flock members; *alignment*, steering towards the average heading of nearby flock members; and *cohesion*, steering towards the center of mass of nearby flock members. On top of the basic Boids model, we add the following rules to determine the motion of the flock: *avoidance*, steering away from nearby shepherd agents; *damping*, reducing speed over time when other forces are absent (so that the agents come to a stop when there is no outside stimulus), and *entropy*, adjusting direction randomly. There also exist other models that we could use to model $F$, such as pedestrian dynamics [5, 79], but generally we seek methods for shepherding that are independent of $F$'s precise behavior.

**Problem Statement** The basic shepherding problem as follows: Given the initial state $\{S_C(0), F_C(0)\}$, find a sequence of feasible movements for the shepherds $S_C(t)$ for $0 < t \leq 1$ such that the flock ends up in one of the goal states, i.e. $F_C(1) \in GS$, where $GS$ is a set of user-specified goal states.

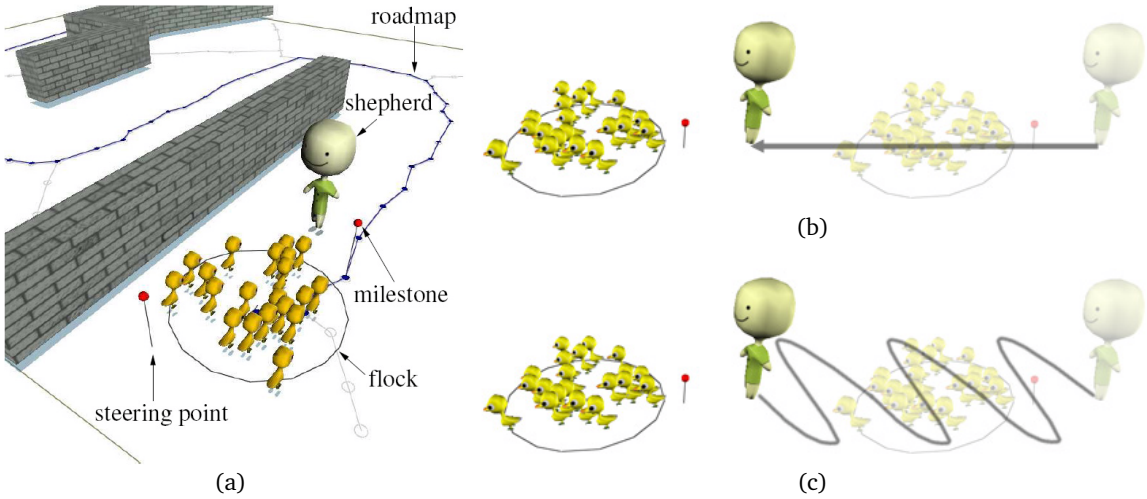



Figure 3.1: (a) An environment and associated terminology. Steering the flock using (b) a straight line (c) a side-to-side motion.

We use the term *steering point* to denote any position to which the shepherd moves itself to influence the movement of the flock, and the term *milestone* to denote any intermediate position that the shepherd attempts to steer the flock toward. A *roadmap* is an abstract topological representation of the feasible configuration space in a given environment, given as a directed graph $G = (V, E)$, where each node in $V$ represents a valid configuration of the flock, and each directed edge $(p, q) \in E$ denotes that it is possible for the shepherds to move the flock from configuration $p$ to configuration $q$. I will discuss various methods for building such a roadmap in the later sections. Figure 3.1(a) shows a visual representation of roadmaps, milestones, and steering points.

We also define a *shepherd's locomotion* as the manner in which a shepherd moves to

control the movement of a flock. The shepherd's locomotion remains an invariant in different shepherding behaviors and dramatically affects the quality of simulation. We divide the shepherd's locomotion into two sub-problems: *approaching* and *steering*. In an approaching locomotion, the shepherd attempts to get close to the flock. In a steering locomotion, the shepherd attempts to push the flock forward. Two examples of a shepherd's steering locomotion appear in Figures 3.1(b) and (c). We will describe the shepherd's locomotion in more depth in the following sections.

It is also possible for a flock to separate into multiple flocks due to repulsive forces exerted by obstacles or shepherds. Therefore, to improve chances for success, the shepherd is interested in not only steering the flock, but also keeping the flock in a cohesive group, i.e. where each member is within *visible range* of at least one other member of the flock.

## 3.1 Medial Axis Graph-Based Strategies

In this section, I present a framework of methods we developed called *Medial Axis Graph-Based* methods, or MAGB [152]. In traditional probabilistic motion planning methods, we use a *local planner* to connect pairs of nearby configurations. Examples of traditional local planners include: the straight-line planner (which simply connects configurations through direct interpolation); the rotate-at-$s$ planner [2]; and the $A^*$ planner. In MAGB, the local planner we used was a simulation defined by the set of shepherding behavior rules [102].

Our planner generally consists of two stages. In the first stage, a planner computes a high level global roadmap representing the workspace. For our experiments, we adopt the *medial axis* of the workspace as our high level roadmap. Figure 3.9 shows examples of these generated medial axes in several workspaces. We compute a path along the global roadmap from the *center* of the initial flock position to the goal position. In the second phase, the shepherd pushes the flock along the path by performing a sequence of locomotions determined by the current state of the flock. As with most traditional local planners, the shepherding local planner may fail to lead the flock to a given milestone. For example, the

shepherds may not have enough room to push the flock, or a passage may be too narrow for the flock to pass through.

We tested these algorithms on a variety of obstacle-filled environments, and compared them to a "simulation-only" approach which uses only simple behaviors with the help of the medial axis of the workspace. In general, we found that the exploration offered by motion planning sometimes improved the success rate versus a simulation-only approach, particularly in environments where the medial axis passes through narrow passages.

## 3.2 Tree-based Planners

The tree-based approaches we first experimented with extended the classic RRT [98] and EST [73] approaches through the use of shepherding behaviors [102] as a local planner to connect configurations and incrementally build a roadmap.

### 3.2.1 RRT-based planners

We began by developing a RRT-based planner (see Algorithm 1 and 2) which constructs the roadmap $G$ by repeatedly extending $G$ towards new randomly generated configurations in $W$. For RRT, SELECT-INTERMEDIATE GOAL chooses a random configuration $q \in W$, and the procedure SELECT-NODE-TO-EXPAND chooses the node $p \in G$ that is closest to $q$.

**Algorithm 1** RRT-Based Planner

```
procedure TREE-BUILD(q_init)
    G.init(q_init)
    for k = 1 to K do
        q ← SELECT-INTERMEDIATE-GOAL
        p ← SELECT-NODE-TO-EXPAND
        r ← SIMULATE(G,p,q,simsteps)
        if r = Success then
            G.add_edge(p, q)
        end if
    end for
    return G
end procedure
```

**Algorithm 2** Local Planner (SIMULATE Procedure)

```
procedure SIMULATE(G,p,q,simsteps)
    for i = 1 to simsteps do
        for shepherd s in S do
            Use workspace roadmap to find a path to q
            Select a milestone m on the path.
            Calculate new target t_new for s
            Move s towards t_new
        end for
        if flock is close to q then
            return Success
        end if
    end for
end procedure
```

### 3.2.2 EST-based planners

Our EST planner is similar to the RRT planner presented in Algorithm 1, with a few key differences. Instead of expanding the tree towards a randomly generated configuration, EST operates by first choosing an existing node $p \in G$ to expand based on a probability distribution $\pi_G$ (in the procedure SELECT-NODE-TO-EXPAND). Then, within a *neighborhood* of $p$, it selects an intermediate milestone $q$ to expand to (SELECT-INTERMEDIATE-GOAL). A key design decision in the implementation of EST algorithms is the choice of $\pi_G$ to prevent excessively dense sampling of configurations in the workspace. We implemented several variations of the EST method which vary only in the selection of $\pi_G$:

- BASICEST—The node to expand is selected randomly; that is, the distribution $\pi_G$ is uniformly distributed.
- NAIVEEST—The distribution $\pi_G$ is weighted so that nodes with fewer neighbors are more likely to be selected over nodes with more neighbors.
- MINEST—The node with the fewest neighbors is chosen every time.

## 3.3 Meta-Graph

The PRM [84] method generates the roadmap by repeatedly picking random free configurations and connecting them to the roadmap graph. For our PRM based approach, we took a similar approach, building a *meta graph* that represents the set of possible navigation routes in a *fuzzy* way. Each node in a meta graph defines a set of flock configurations (excluding the shepherd positions) that are conforming to the properties of the node. As in the other methods, edges in the meta graph approach are directed. The weight of an edge in a meta graph presents the probability of successful herding from the start node to the end node of the directed edge. In a similar fashion to fuzzy or lazy PRM approaches [19, 114], paths from the meta graph are extracted and evaluated until a path is found, or until no path can be found to connect the start and the goal configurations in the meta graph.

Each node in a meta graph represents a *meta configuration*. Each meta configuration $C$ is as an oriented disc in a four dimensional space, and is composed of four values: position ($p_C = (x_C, y_C)$), orientation ($\theta_C$) and radius ($r_C$). Each meta configuration intuitively defines a set of *conforming flocks*. We say a flock is conforming to a meta configuration $C$ if the minimum enclosing circle of the flock is enclosed by $C$ and the angle between the mean velocity of the flock and $\theta_C$ is less than a user defined threshold.

We say that a meta configuration is *free* if the disc does not intersect an obstacle. To generate one free meta configuration, we first assign a random position and orientation, whose ranges are fixed.

To assign a random radius, we compute the smallest enclosing disc as follows: First, we consider a group containing $N_f$ flock members where each individual member is enclosed in a radius $R_f$ circle. The compact area of a group is the smallest circle that one can put all group members into. Ideally, shepherds should control the flock so that all members are inside its compact area.

This is an inverse version of the computationally difficult *packing circles in a circle problem* [63]. The objective of this problem is to find the maximum radius of $K$ equally-sized

smaller circles that can be packed into a unit circle. Since we do not have to compute the exact value of the radius $R_c$ of the compact area, we approximate it by considering a bounding square of $N_f$ flock members. In this case, the compact area will be the minimum circle enclosing that square. That is, we approximate $R_c$ as:

$$R_c = R_f \sqrt{2N_f} \tag{3.1}$$

The radius of our meta configuration is set to be a random value between $R_c$ and $3R_c$.

The weight of an edge that connects meta configurations $C_1$ and $C_2$ represents the probability that a flock conforming to $C_1$ can be guided by the shepherds so that the flock is conforming to $C_2$. This can be estimated in several ways. For example, we can estimate this probability by generating a set of random flocks conforming to $C_1$, and, for each flock, we find a path to $p_{C_2}$ using one of the tree-based planners described above. While this approach is accurate, it is inefficient due to the requirement of running many (usually thousands or even tens of thousands) of simulations. The approach we use estimates the probability by using the differences in position and orientation between $C_1$ and $C_2$. The distance is defined as the follows:

$$\text{dist}(C_1, C_2) = \begin{cases} \infty & \text{if } \theta_{C_1,V} \geq \tau, \\ \infty & \text{if } \theta_{C_2,V} \geq \tau, \\ \frac{|\theta_{C_1} - \theta_{C_2}|}{\pi} + \frac{\text{dist}(p_{C_1}, p_{C_2})}{D} & \text{otherwise,} \end{cases}$$

where $\theta_{C_i,V}$ is the angle between $V$ and $C_i$'s facing direction, $V = p_{C_2} - p_{C_1}$, $\text{dist}(p_{C_1}, p_{C_2})$ is the Euclidean distance between $p_{C_1}$ and $p_{C_2}$ and $D$ is the diagonal distance of the bounding box. In each iteration of the query phase, a path is extracted from the meta graph. To check the feasibility of a consecutive pair of meta configurations in the path, we use simulation to determine if the final configuration of the flock conforms to the end meta configuration.

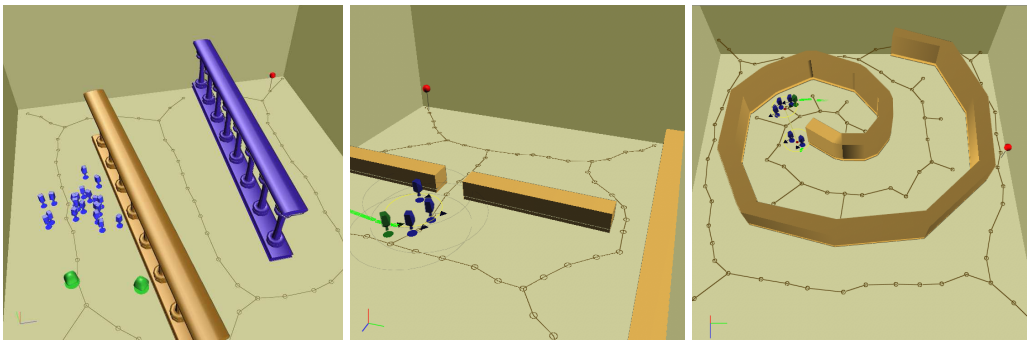
Note that this meta graph planner is not probabilistically complete. That is, there are some situations where our planner cannot find the answer even if the path exists—such as in problems that require the flock to form a long line. This is because we confine our radius of our meta configurations and disallow the discs from the meta configurations from intersecting workspace obstacles.

### 3.3.1 Experiments with MAGB

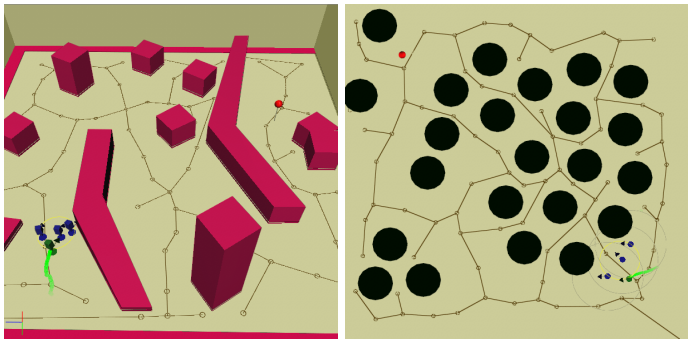
We tested the Medial Axis Graph-based motion planners on some of the environments shown in Figure 3.2. In the Spiral environment (Fig. 3.2(c)), the start configuration is at the center and the goal is outside the outer most wall of a spiral-shaped obstacle. In the S environment (Fig. 3.2(a)), the start and end configurations are at the ends of a S-shaped passage. In the Broken T environment (Fig. 3.2(b)), the free space is separated by three bars, with the middle bar equidistant to the bars on the side. This particular environment is intended to test the algorithm's ability to shepherd a flock through a narrow passage. We also tested the planners on two maze environments. The first maze (Fig. 3.2(d)) is similar to the S environment with additional road blocks in the S-shaped passage. The second maze (Fig. 3.2(e)) is composed of several cylinders—a larger cluster at the upper-right corner and a smaller cluster at the lower-left corner.

We compared our planners to a *simulation-only* approach (SIMONLY), which only uses the simulator to steer the flock with the help from the medial axis of the workspace. This simulation-only approach is the same method used in [102, 103]. In our test suite, we have 12 variants of EST-based planners: MINEST, NAIVEEST, and BASICEST with neighborhood sizes of 3, 5, 7 and 9. In our test suite we also have one RRT-based planner, and one fuzzy meta graph planner.

In Table 3.1, we show success rates for each of the algorithms. Each success rate in the table is an average over 50 runs, and all comparisons are statistically significant at the 95% confidence level. We found that planners do well in the Broken T, S and Spiral environments. We also found that planners do poorly on the maze environments. This



(a) S Env. (b) Broken T. (c) Spiral Env.



(d) Maze 1 Env. (e) Maze 2 Env.

Figure 3.2: Environments used for MAGB experiments. The workspace medial axes are shown in all figures. We use these medial axes as the workspace roadmap for local planning.

is because the planners cannot search sufficiently long enough. For example, the fuzzy meta graph planner will need to exhaustively check all the shorter but more difficult paths before it can try the longer path with larger clearance. This is clearly shown in the S and maze 1 environments. The main difference between maze 1 and S environment are the introduction of islands in the S shaped passage. In the S environment, the fuzzy meta graph planner clearly out-performs simulation-only. In the maze 1 environment, the performance of simulation methods degraded only slightly, whereas the success rate of the fuzzy meta graph planner drops significantly.

One important observation we had is that the motion planning in MAGB helps improve the success rate, albeit slowly. In Figures 3.3, 3.4, and 3.5, we show the success rate of the planning approaches on a sample of initial configurations. This plot confirms the hypothesis

that while simulation-only approaches often get stuck, the MAGB planning approaches help explore other path options and eventually find solutions given a sufficient simstep budget. This is particularly clear in the S and spiral environments as shown in Figure 3.3 and Figure 3.4. The planners have lower success rates before 80K simulation steps but end at much a higher success rate (60% vs. 18%). Even in the maze environments, in which the planners perform poorly, based on the plot shown in Fig. 3.5, we can still see the continuing improvement throughout the entire experiment.

## 3.4 Group Control as Deformable Shape Manipulation

It is impractical to handle each flock member individually, even when controlling a small flock. In most existing methods for swarm control, a simplified shape is often used as a high level representation to model the flock, such as an axis aligned bounding box, or a bounding circle. This model works well if the workspace is relatively sparse or has few obstacles. However, in environments with many obstacles or narrow corridors or for very large flock sizes, a bounding circle is excessively restrictive.

Therefore, we improved on the MAGB methods by developing a *deformable object* representation for the flock, which can split and merge [69]. During steering, the shepherd is viewed as a "deformer", continuously reshaping the contour of the flock to a target shape. The shepherds can represent the contour of the flock using $\alpha$-*shapes* [44, 45], which are updated as the flock moves. An example of $\alpha$-shapes is shown in Fig. 3.6(b). To prevent the shepherd from disturbing the flock unnecessarily, the value of $\alpha$ can be determined from the flock's sensing range. In this paper, the shepherds represent the flock using "pixel blobs" (see Section 3.4.2 for details) that represent the flock's deforming contour.

To efficiently create the flock model, we designed a new algorithm to efficiently update the $\alpha$-shapes [30–32] of moving points by exploiting the temporal and spatial coherence.

A shepherd needs to have a model to represent other shepherds. This model affects how we solve the task allocation problem that assigns steering positions to shepherds. In

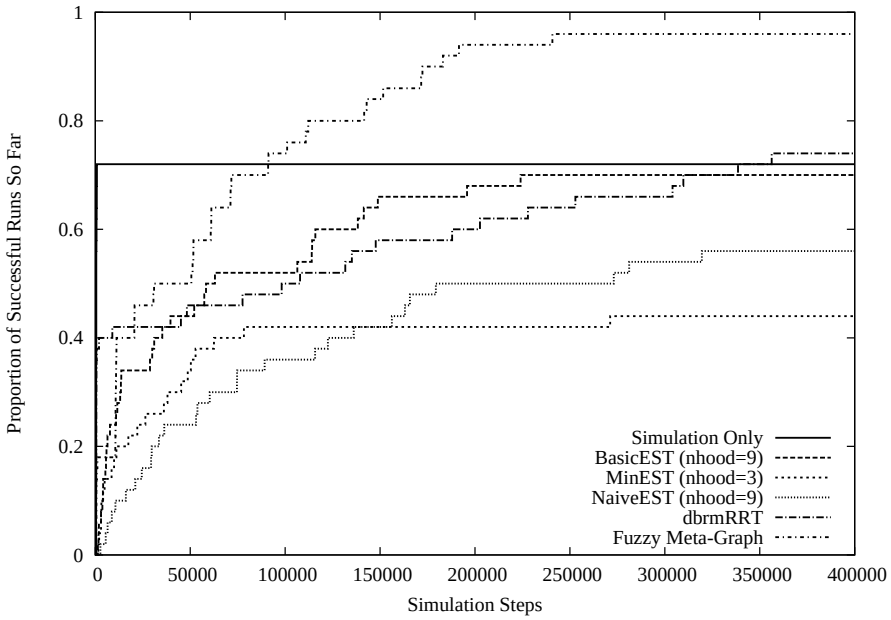



Figure 3.3: Plots showing the proportion of successful runs so far versus the number of simulation steps in MAGB, S environment.

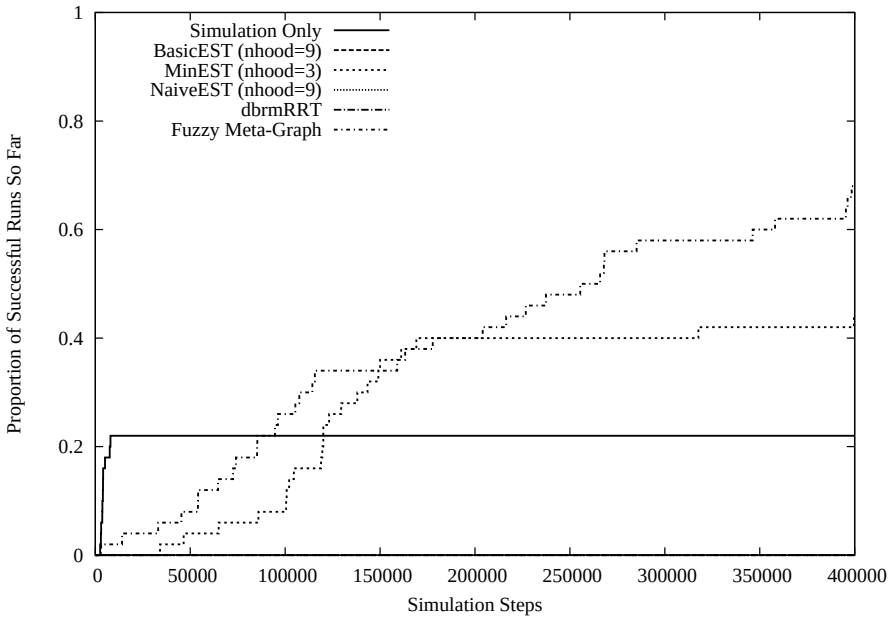



Figure 3.4: Plots showing the proportion of successful runs so far versus the number of simulation steps in MAGB, Spiral environment.

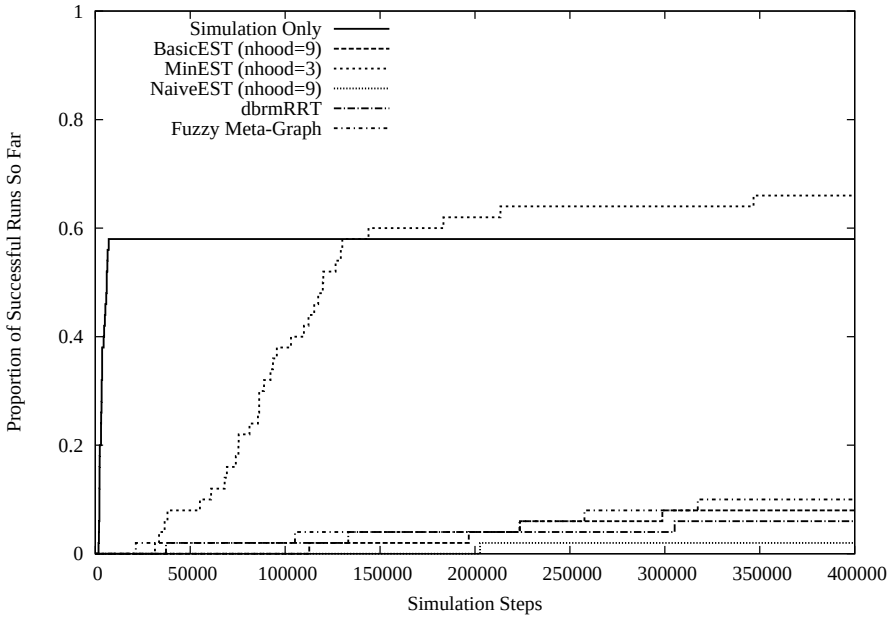



Figure 3.5: Plots showing the proportion of successful runs so far versus the number of simulation steps in MAGB, Maze 1 environment.

Table 3.1: MAGB Proportion of Successful Runs

| Method | S | Broken T | Spiral | Maze 1 | Maze 2 |
|---|---|---|---|---|---|
| Simulation Only | 0.72 | 0.58 | 0.22 | 0.58 | **0.82** |
| MinEST (nhood=3) | 0.44 | 0.48 | 0.44 | **0.66** | 0.68 |
| MinEST (nhood=5) | 0.48 | 0.64 | 0.12 | 0.38 | 0.52 |
| MinEST (nhood=7) | 0.5 | 0.56 | 0.08 | 0.22 | 0.4 |
| MinEST (nhood=9) | 0.6 | 0.48 | 0 | 0.2 | 0.3 |
| NaiveEST (nhood=3) | 0.26 | 0.46 | 0 | 0 | 0.18 |
| NaiveEST (nhood=5) | 0.4 | 0.58 | 0 | 0.02 | 0.12 |
| NaiveEST (nhood=7) | 0.44 | 0.82 | 0 | 0.02 | 0.28 |
| NaiveEST (nhood=9) | 0.56 | 0.84 | 0 | 0.02 | 0.32 |
| BasicEST (nhood=3) | 0.24 | 0.36 | 0 | 0.02 | 0.22 |
| BasicEST (nhood=5) | 0.46 | 0.62 | 0 | 0.04 | 0.32 |
| BasicEST (nhood=7) | 0.48 | 0.84 | 0 | 0.02 | 0.24 |
| BasicEST (nhood=9) | 0.7 | 0.84 | 0 | 0.08 | 0.28 |
| rbrmRRT | 0.72 | 0.3 | 0 | 0.02 | 0.06 |
| dbrmRRT | 0.74 | 0.56 | 0 | 0.06 | 0.22 |
| Fuzzy Meta Graph | **0.96** | 0.68 | **0.68** | 0.1 | 0.24 |

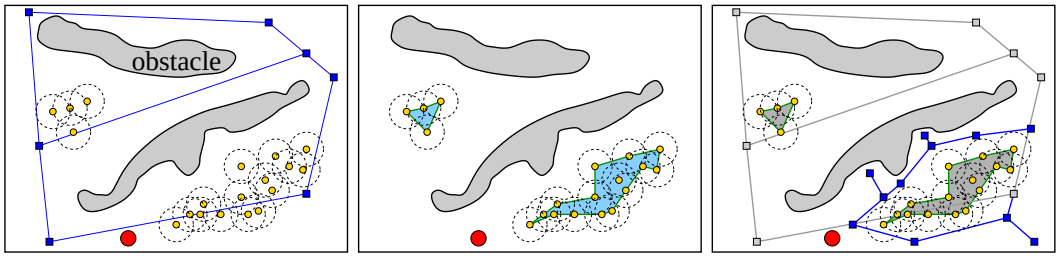



Figure 3.6: (a) An environment with a global roadmap, two obstacles, a shepherd (larger disc) and two sub-flocks. Each flock member is shown as a small disc enclosed in a dashed circle, which represents its visibility. (b) The shepherd's flock model is shown as two $\alpha$-shapes for the two sub-flocks. (c) A local roadmap is built around the flock nearest the shepherd.

this framework, we use a centralized system that knows the locations of all the shepherds. This is analogous to a large robot arm with many fingers (i.e. shepherds) manipulating a deformable object (i.e. flock). All these fingers are connected to and only communicate with a central command center, which coordinates the shepherds. Therefore, we reduce the task assignment problem to a *bipartite matching problem* between the steering points and the shepherds. Then, the edge weight of the bipartite graph is the geodesic distance (estimated using the global roadmap) between a shepherd and a steering point.

### 3.4.1 Overview of the DEFORM Method

By representing the flock as a deformable object, the shepherding problem becomes a deformable object manipulation problem. Given a desired shape, the shepherds' task is to move close to important contact points near the deformable object to change its shape. A solution to the problem is likewise reduced to determining a sequence of intermediate target shapes that will eventually lead the flock to the final goal. DEFORM, shown in Algorithm 3, outlines the steps necessary to achieve the goal.

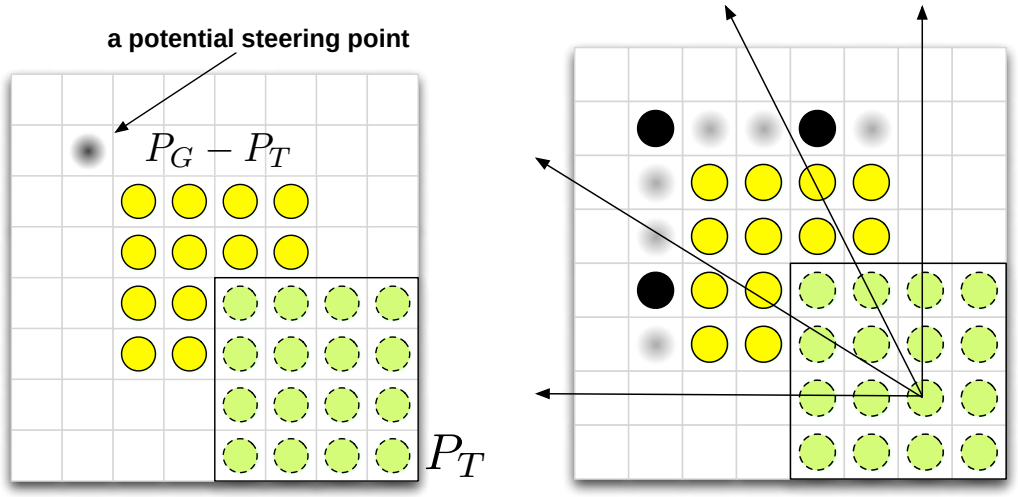
More precisely, DEFORM regards the flock as a deformable object with an *area conservative* constraint. That is, DEFORM will find a sequence of continuously deforming polygons $P_F(t)$, $t \in [0, \ldots, 1]$, so that $P_F(1)$ is near a goal. For each $P_F(t)$, the shepherds can tightly

**Algorithm 3** DEFORM($F$, $S$, $g$)

*Input*: $F$ (flock), $S$ (shepherds), and $g$ (goal)
**while** $g$ is not reached **do**
  Compute the contour polygon $P_F(t)$ of $F$ at time $t$
  Determine the next target polygon $P_T(t + \delta t)$
  Find $s_{steer}$ to morph $P_F$ to $P_T$
  Move $S$ to $s_{steer}$
**end while**

pack all flock members in $P_F(t)$. Because DEFORM may not consider shepherd positions when building $P_F(t)$, additional *soft* constraints or heuristics on $P_F(t)$ should be imposed to increase the chance of finding a successful control plan. For example, the shepherds should try to keep each polygon $P_F(t)$ as "fat" as possible and keep the medial or the principal axis of $P_F(t)$ close to the edges along the global roadmap. The fatness increases the controllability of the flock and the centeredness reserves room to maneuver. There are many ways to compute the target polygons $P_T(t)$. For example, we can grow a search tree along the edges of the global (workspace) roadmap. Instead, we adopted the approach of simply extracting a path with maximal clearance and building a sequence of area-conserving polygons along the path. Details of this approach are discussed in the next section.

Next, DEFORM computes the movements for the shepherds so that the flock can assume the shape of polygon $P_F(t)$ at each time step $t$. More specifically, given the desired shape $P_T(t + \triangle t)$ in the next time step and the current flock model $P_F(t)$, the behavior will determine the necessary deformation and transformation to morph $P_F(t)$ to $P_T(t + \triangle t)$ using, for example, the *Iterative Closest Point (ICP)* algorithm [13] when $\alpha$-shape is used to compute the contour. Here, we use boolean operators between pixel blobs to determine the necessary deformation. Finally, DEFORM calculates a steering state $s_{steer} \in S_C$ for the shepherds (recall that $S_C$ is the state space of the shepherds). To plan shepherd's motion, a search tree in $S_C$ is iteratively expanded by randomly selecting a feasible behavior until $s_{target}$ is reached (and therefore $G$ is 'shaped' like $Ply(t + \triangle t)$).




(a) (b)

Figure 3.7: (a) Target shape, current flock blob, and a potential steering point. (b) Finding steering points using a radial partitioning.

### 3.4.2 DEFORM Implementation Detail

In this section, we flesh out the framework discussed in the previous sections. In particular, we discuss in detail how a flock is represented as a deformable shape, how the target shape is determined, and how the steering points are chosen.

An important feature of our framework is that it can be implemented in many different ways. While using $\alpha$-shapes can provide high accuracy in flock contour representation, a new algorithm to efficiently update the $\alpha$-shapes [30–32] is required to efficiently create the flock model. This method will need to exploit the temporal and spatial coherence of the flock's movement.

To avoid this difficulty, we discretize the workspace into a regular grid and represent the contour of the flock using pixel blobs (described below). Each cell in the grid is classified as a *free* cell if a flock member in the cell is free of collision from the workspace obstacle. Otherwise, the cell is marked as an *in-collision* cell. The size of the cell in the grid is defined by the geometric size of an individual flock member. Therefore, each cell can only be occupied

by a single flock member. This approximation is very easy to implement and is very efficient (linear to the flock size). As we will see later in our experiments, this representation is also accurate enough to model and handle large flocks.

### Flock Blobs

The *pixel blob* of a single member $g \in F$ is simply the set of pixels that are within the distance $r - \epsilon$ of $g$, where $r$ is the sensor range of $g$ and $\epsilon$ is an arbitrarily small number. The pixel blob of a flock (called flock blob) is therefore the union of all pixel blobs of its members. The flock blob may have several connected components if the flock has separated into sub-flocks.

### Target Blobs

Given the current flock blob, the target shape is defined as another pixel blob, called a *target blob*, that includes all flock members at the next time step. It is straightforward to compute the target blob. First, we pick a pixel that is occupied by the closest flock member to the final goal position. The closeness to the goal is measured by geodesic distance. Once the center pixel $o$ is determined, we grow the blob using *free* pixels, starting with the 8-connected set and expanding in concentric rings around $o$. We stop when the blob contains the same number of pixels as the number of flock members. This ensures that the blob is large enough to contain all flock members and also conserve the area of the target blobs throughout the entire control behavior. We say a flock configuration *conforms* to a target blob if the target blob includes all flock members. Note that the target blob could be grown in many different ways. For example, one could make it grow away from obstacles or toward the medial axis. The version we used in this paper makes the blob grow as fat as possible.

### Shepherds' Steering Points

Given a flock blob $P_F$ and a target blob $P_T$, we compute the set of steering points $s_{steer}$. To compute $s_{steer}$, we first compute the differences between $P_F$ and $P_T$. This is simply done by

performing the boolean difference $P_F - P_T$ pixel by pixel.

Next, we determine a set of potential steering points, $s'_{steer}$. We say a pixel $p$ is in $s'_{steer}$ if (1) $p$ is neighboring (using 8-connectivity) to a pixel $c \in P_F - P_T$, (2) $p \notin (P_F \cup P_T)$, and (3) $p$ is on the opposite side of the pixel in $P_T$ closest to $c$. An example of $p$ is shown in Fig. 3.7(a).

Once the points in $s'_{steer}$ are identified, the last step is to choose $n$ points from $s'_{steer}$, where $n$ is the number of shepherds. If there are fewer than $n$ points in $s'_{steer}$, we are finished. In this case, some shepherds may not have steering points assigned and will stay stationary. If there are more points in $s'_{steer}$ than there are shepherds, then we partition $s'_{steer}$ into $n$ pie wedges whose apex is at the center of $P_T$ (the flock member closest to the global goal). In each pie wedge, the point in $s'_{steer}$ farthest from the global goal is selected as a steering point (see an example in Fig. 3.7(b)).

After $s_{steer}$ is computed, the steering points are assigned to the shepherds by forming a bipartite graph whose nodes are the points in $s_{steer}$ and the shepherd positions, and whose edge weights are geodesic distances between them. The steering point assignment is then solved using a bipartite matching algorithm.

Screen shots our simulation running DEFORM can be found in Fig. 3.8.

### 3.4.3 Behavior-based Motion Planners with DEFORM

DEFORM can theoretically be used as a local planner in higher-level planners such as PRM, RRT, and EST. Based on our results here and in [152], planning is not generally helpful. However, theoretically speaking, the proposed control behavior may not always successfully control the flock. For example, in some rare cases, the deformable polygon may block paths needed by shepherds to reach their steering points. For the sake of completeness, we will discuss two high-level planners: RRT and meta graph integrated with the DEFORM.

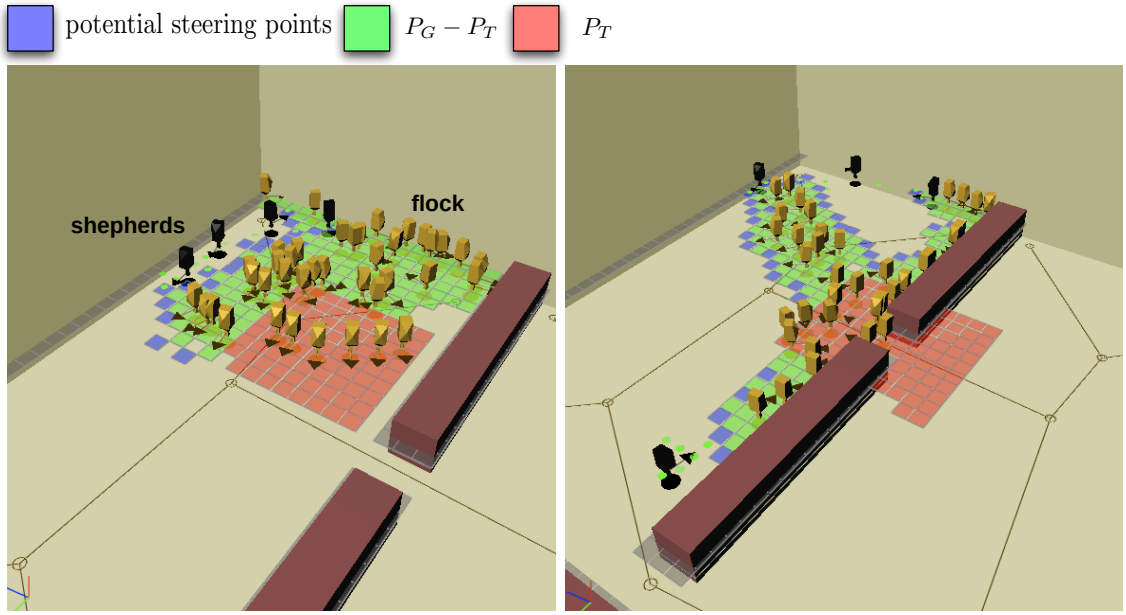



Figure 3.8: Screen shots of DEFORM planner in action.

### Distance Metrics

In both RRT and meta graph planners, distance metrics are important for estimating the probability of successfully herding from one flock state to another flock state. Using Euclidean distance is usually not enough to reflect this probability. In addition, we use the idea of *deformation energy* which defines the amount of effort needed to deform one polygon to another polygon. Again, deformation energy can be implemented in many ways depending on the representation of the deformable object. By representing the polygon as a pixel blob, we have the advantage of efficiently computing the overlay of two deformable polygons. More specifically, given two pixel blobs, $P$ and $Q$, we translate $P$ so that their centers coincide, then count the number of pixels in the boolean difference $P - Q$. This value approximates the number of pushes needed to deform $P$ to $Q$. The final distance between $P$ and $Q$ is simply a weighted sum of the Euclidean distance between their centers and the deformation energy estimated by boolean difference $P - Q$. In our implementation, we use the same weights for Euclidean distance and deformation energy.

#### RRT

Our RRT-based DEFORM planner proceeds as described before in Section 3.2.1 and Algorithm 1, constructing the roadmap $G$ by repeatedly attempting to extend $G$ using a local planner towards new randomly generated configurations in $W$. However, in this case, the local planner is the DEFORM behavior described in Section 3.4.2.

#### Meta Graph

Although meta graph as described in 3.3 was originally designed for directed circles, it was adapted for use with the DEFORM behavior. Each node in a meta graph represents a *meta configuration*. A meta configuration defines a set of group configurations (excluding the shepherd positions) that are conforming to the properties of the node. More specifically, each meta configuration $C$ is as a pixel blob that has twice as many pixels as the number of flock members. Each meta configuration intuitively defines a set of *conforming flocks*. We say a flock is conforming to a meta configuration $C$ if $x\%$ flock members overlap with the pixels of $C$. In our implementation, we let $x = 85$.

A meta configuration is generated in the same way as the *target blob* is generated (in Section 3.4.2). We first pick a random point $p$ in workspace and use $p$ two grow a blob level by level. These meta configurations are then connected to their $k$-nearest neighbors to form a meta graph. In a similar fashion to lazy PRM approaches [114], paths from the meta graph are extracted and evaluated using the proposed control behavior until a path is found, or until no path can be found that connect the start and the goal configurations.

### 3.4.4 DEFORM Results

We evaluated our DEFORM behavior against the earlier MAGB behavior across 6 different test environments shown in Figure 3.9. The next sections explain our experimental method. We evaluated its performance against MAGB specifically in terms of *scalability* (the ability to successfully herd large flocks to the goal within the given simulation time budget) and

*robustness* (the ability to herd agents despite random and unpredictable behavior). Each experimental sample consisted of 30 runs, and claims of statistical significance were based on 95% confidence.

### Scalability

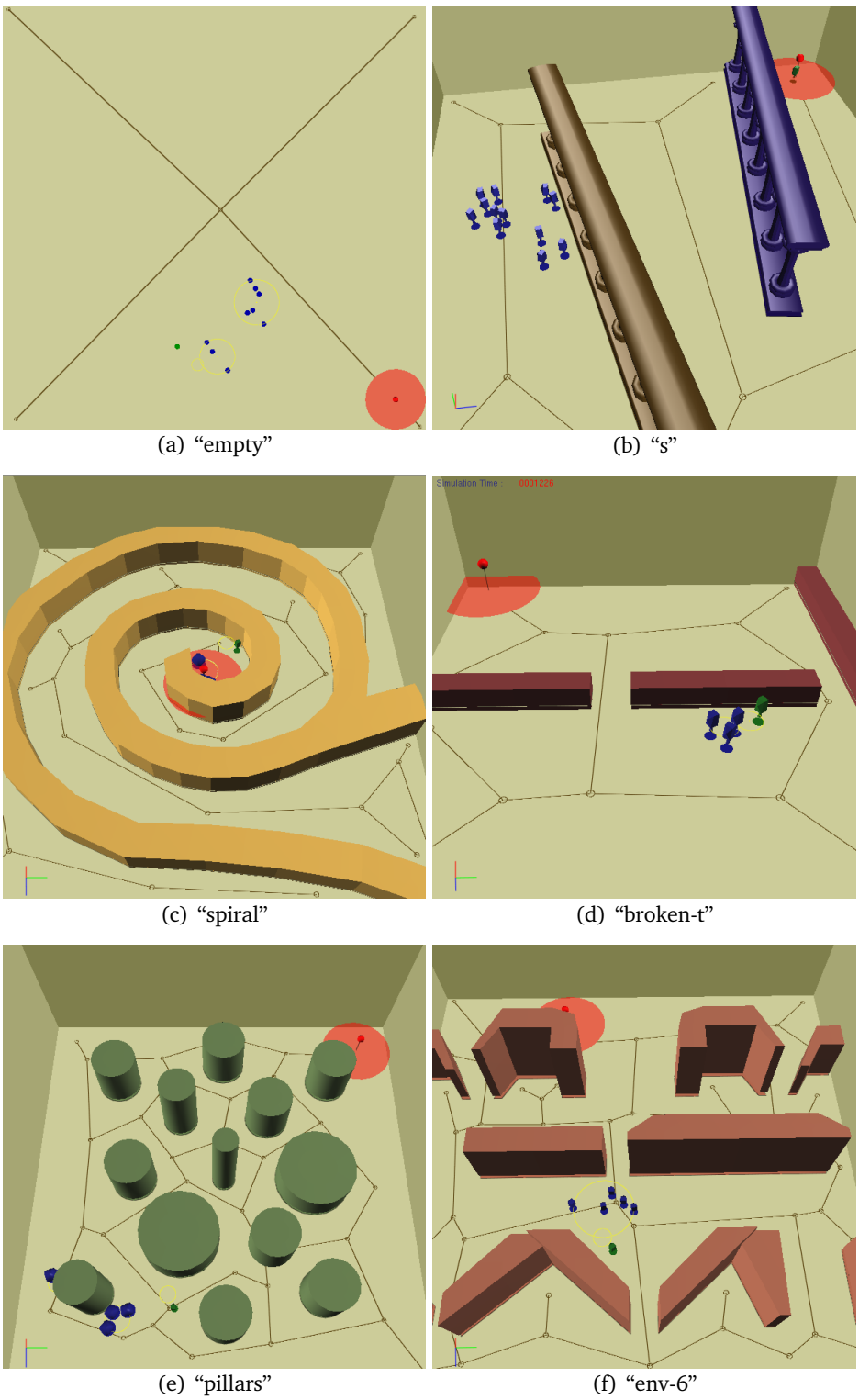
We presume that increasing the size of the flock should impact both the effectiveness and performance of the shepherding algorithm, and that more shepherds will be needed to move larger flocks to the goals in a timely manner. The *scalability* of these behaviors is defined by the ability for them to effectively herd increasingly large flocks to the goal within the given simulation time budget.

To test our hypotheses, we ran experiments with varying numbers of shepherds (1, 2, 3, and 4) and varying flock sizes (5, 10, 15, 20, and 25) on each of the 6 test environments. We computed *success rates* as the proportions of sample runs where the algorithm successfully moved the flock to the goal. Table 3.2 shows example results for the "broken-t" environment. For small flock sizes, DEFORM and MAGB perform similarly. However, as the size of the flock increases, MAGB shows significant decay in performance - it is clear that more shepherds are needed to control larger flocks using the MAGB behavior. On the other hand, DEFORM manages to achieve excellent results throughout (above 90% success rate) with no negative trend in performance up to 100 flock members.

We also ran experiments to test larger shepherd and flock sizes on the "s" environment. The results, shown in Fig. 3.10, show that DEFORM once again outperforms MAGB across the board, but especially with larger flock sizes.

### Robustness

We define *robustness* as the ability for the shepherds to control the flock in spite of unpredictable flock behavior. We modeled the unpredictable behavior by linearly mixing each flock agent's behavior with a random vector. We control this randomness by increasing and decreasing the magnitude of the random vector. Figures 3.11 and 3.12 illustrate the results



(a) “empty” (b) “s”

(c) “spiral” (d) “broken-t”

(e) “pillars” (f) “env-6”

Figure 3.9: Environments used in DEFORM experiments.

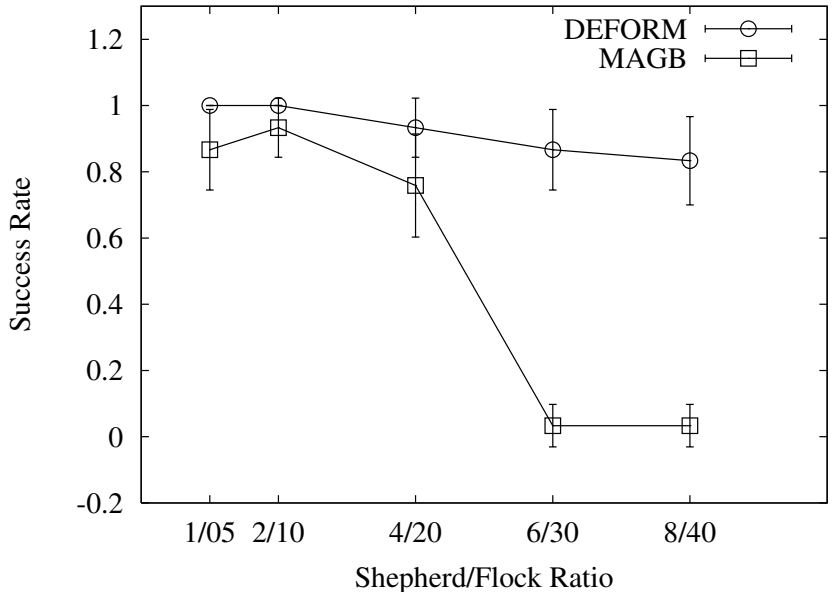



Figure 3.10: Success rate vs. Shepherd/Flock Ratio on s Environment.

Table 3.2: Success Rates for DEFORM and MAGB with Varying Shepherd and Flock Sizes (Broken-T Environment)

| | | **DEFORM** | | | | **MAGB** | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | # Shepherds | | | | # Shepherds | | | |
| | | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 |
| **Flock Size** | 5 | 0.93 | 1.00 | 0.90 | 0.83 | 0.90 | 1.00 | 0.93 | 0.97 |
| | 10 | 1.00 | 0.93 | 0.93 | 0.93 | 0.56 | 0.97 | 1.00 | 0.97 |
| | 15 | 1.00 | 1.00 | 1.00 | 0.93 | 0.47 | 0.80 | 0.77 | 0.83 |
| | 20 | 1.00 | 0.97 | 1.00 | 0.90 | 0.07 | 0.67 | 0.43 | 0.83 |
| | 25 | 1.00 | 1.00 | 0.97 | 0.97 | 0.00 | 0.40 | 0.33 | 0.43 |

(with 95% confidence bars) of success rate versus increasing randomness. In these figures, randomness is shown as a ratio from 0.0 to 1.0 where 0.0 represents fully deterministic behavior and 1.0 represents fully random behavior. In general, DEFORM performs significantly better than MAGB with increasing flock randomness. A notable effect is shown in Figure 3.11, where there is a bump in performance with increasing flock randomness for the MAGB behavior. We believe that this is because the random oscillation of the flock members helps the shepherds push them through narrow corridors in the broken-t environment similar to how salt granules pass through the openings of a salt shaker.

### High-Level Planning and DEFORM

With MAGB we found that in many cases, the planning methods we applied did not yield significant enough improvements to justify the additional computational expense. Indeed, our tests showed that with a fixed budget of simulation steps, planning-based approaches using these techniques were often unable to even match the performance of our behavior-only method. However, in almost all environments, planners using the DEFORM behavior performed significantly better than planners using the MAGB behavior.

Table 3.3: Success Rates for DEFORM and MAGB With Various High-Level Planners and Environments

| | | **DEFORM** | | | **MAGB** | | |
|---|---|---|---|---|---|---|---|
| | | Planner | | | Planner | | |
| | | None | Graph | Tree | None | Graph | Tree |
| **Environment** | broken-t | 0.97 | 0.13 | 1.00 | 0.86 | 0.00 | 0.07 |
| | empty | 1.00 | 0.00 | 0.77 | 1.00 | 0.00 | 0.33 |
| | env6 | 1.00 | 0.30 | 0.97 | 0.10 | 0.00 | 0.00 |
| | pillars | 0.97 | 0.96 | 1.00 | 0.93 | 0.00 | 0.00 |
| | s | 0.97 | 1.00 | 0.83 | 0.70 | 0.00 | 0.00 |
| | spiral | 0.83 | 0.70 | 0.53 | 0.00 | 0.00 | 0.00 |

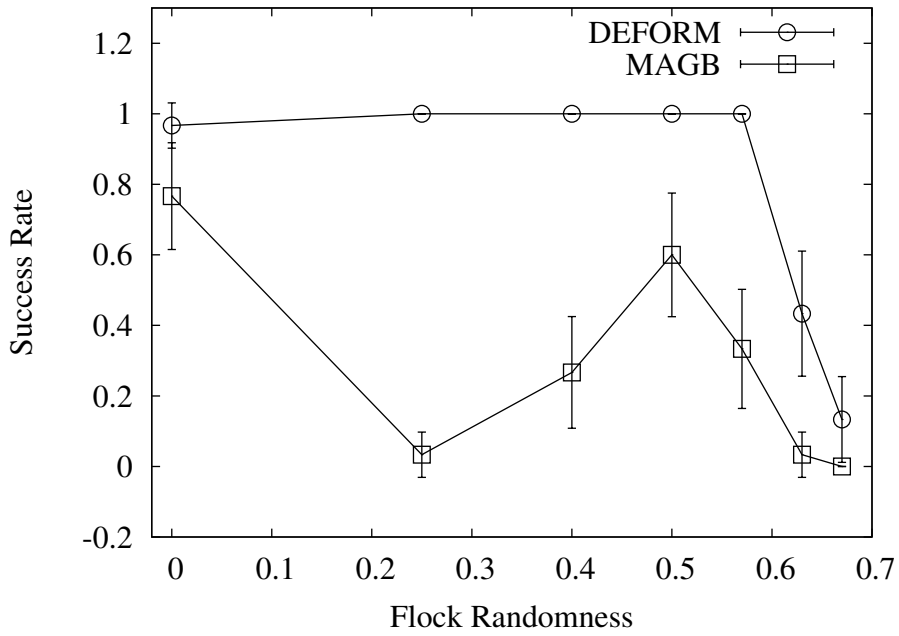



Figure 3.11: Success Rate vs. Randomness Ratio on the broken-t Environment.

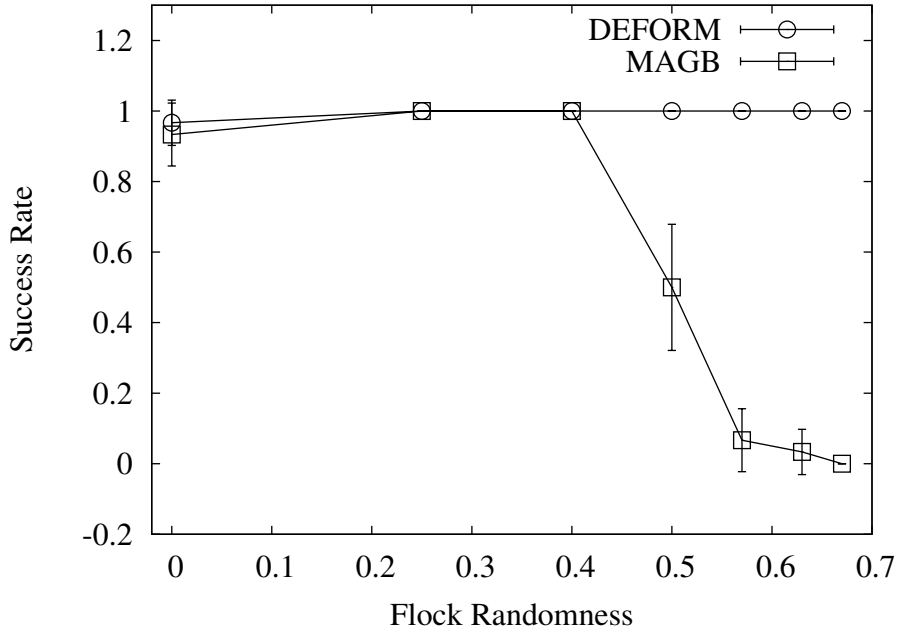


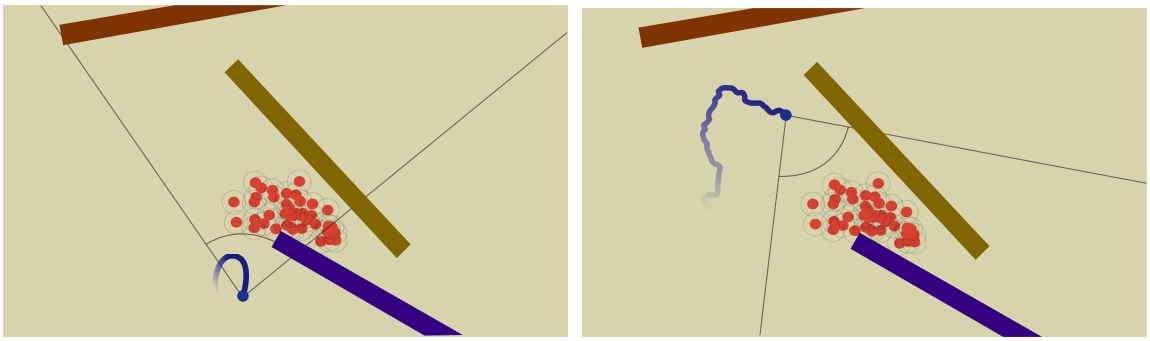

Figure 3.12: Success Rate vs. Randomness Ratio on the s Environment.

## 3.5 Conclusion

In this chapter introduced several methods that I have applied to the problem of shepherding. We began by combining traditional motion planning techniques such as RRT, EST, and PRM with shepherding behaviors, and showed that these extensions were beneficial in exploring the space of solutions. Based on the success of this method, we developed yet another abstraction called DEFORM where we represent the flock configuration as a discretized deformable object, and showed that shepherding could be done more effectively in the face of uncertainty without relying on the medial axis to be computed in advance. This makes DEFORM applicable to scenarios where perhaps only a partial map is available.

# Chapter 4: Visibility-Based Pursuit

*Visibility-Based Pursuit* is the problem of controlling a moving camera $C$ to pursue and maximize visibility of a group $T$ of target agents as they move coherently through an environment filled with obstacles. An illustration of the group tracking problem is shown in Fig. 4.1(a). In this figure, a coherent group of targets is shown entering a passage, such that some of the targets have become occluded from the camera's view. A more intelligent behavior is shown in Fig. 4.1(b), in which the camera has moved so that it can still see the entire group of targets and prepare itself better for the case that the targets move into the narrow corridor.



(a) (b)

Figure 4.1: (a) An illustration of the behavior of a pursuer agent using a reactive behavior. (b) An illustration of a pursuer using the sampling-based IO planner.

In our investigation of this problem, we assumed the environment to be known, but the trajectory of the targets to be unknown. This can be the case especially in virtual environments such as video games, where the geometric data may be available in advance. This can also be the case in many real world environments where floor plans or GIS data are available in advance. Given this information about the environment, it is possible for the planner to perform deeper *lookahead* in the search space of possible trajectories, and

therefore provide better real-time visibility-based pursuit strategies even when the motion of the targets is unknown.

Our investigation also focused on pursuing a coherent group of targets (such as a crowd or a flock) using a single camera. Can we apply existing single-target pursuit techniques to solve the visibility-based pursuit problem? Unfortunately, as I will explain, our preliminary results found that direct application of existing single-target pursuit strategies to track a coherent group usually performs poorly. We speculate that these strategies perform poorly because of the fundamental differences between single-target pursuit and group pursuit. For example, while maintaining the visibility of a coherent group in some aspects is easier than pursuing a single target since there is more than one target that the camera can pursue, pursuing a group can be more difficult if the objective of the camera is to *maximize* the number of visible targets over time. A group of targets can assume many forms; for example it could compress into a long, dense line within a narrow corridor, and then spread out into a sparse blob when it reaches an open area. Groups of targets could also engulf obstacles, or split into multiple sub-groups for a short period of time. Furthermore, since the trajectory of the targets is unknown, the trajectory of the camera must also be computed online to react to the movement of the targets.

The work presented in this chapter attempts to address these issues. First, I present three new strategies (in Section 4.2) extended from the existing single-target techniques—the reactive, sampling-based [12] (called IO), and escaping-risk-based [118] (called VAR) methods. I also present a method to preprocess the given environment offline to generate a data structure called *monotonic tracking regions* (MTRs) (defined later in Section 4.3) that can be used to assist real-time planning. I also present some optimizations to these methods including caching and incremental construction to allow these methods to perform well in larger environments or perhaps when the environment model is incomplete.

## 4.1 Problem Description

In visibility-based pursuit, we assume that the workspace is populated with known obstacles represented by polygons. These polygons are the projection of 3D objects that can potentially block the camera's view. This projection essentially reduces our problem to a 2D workspace. We refer to a group of target agents as $T$, and we assume that, in the initial state, the group $T$ is visible by the camera $C$. While the future movement of the targets is unknown, the maximum (linear) velocity of a target, $v_T^{max}$, is known. The current position at time $t$ of the target $x_T(t)$ is also known if $T$ is in $C$'s viewing range. Likewise, the camera $C$ has bounded linear velocity $v_C^{max}$. The camera $C$'s view range $\mathcal{V}_C$ is defined as a tuple: $\mathcal{V}_C = (\theta, r_{near}, r_{far})$, where $\theta$ is the view angle, and $r_{near}$ and $r_{far}$ define the near and far view range. The exact configuration of this view range at time $t$, denoted as $\mathcal{V}_C(t)$, is defined by the tuple and the camera's location $x_C(t)$ as well as the maximum angular velocity $w_C$. The position $x_C$ of the camera is simply governed by the following linear equation:

$$x_C(t + \triangle t) = x_C(t) + \triangle t \cdot v_C(t) \ .$$

The targets are either controlled by the user or by another program, so the trajectories of the targets are not known in advance to the camera. We modeled the target group as non-adversarial, and constantly moving toward a common goal waypoint. When a waypoint is reached, a new goal (unknown to the camera) is randomly selected, and the group will plan a path toward this new goal. When the target group is moving to a goal, each member in the group keeps itself close to other members (similar to the flocking behavior used in shepherding in Chapter 1.1 and in [126]). More specifically, at every time step, the coherence of the group is maintained by filling the space around a leader in the group. However, when one or more obstacles are nearby, the group may separate into multiple sub-groups.

Given the positions of the (visible) targets and the position of the camera, one can compute the camera's view direction so that the number of targets inside the view range

is maximized. Therefore, the problem of visibility-based pursuit then is reduced to find a sequence of velocities $v_C(t)$:

$$\arg\max_{v_C(t)} \left( \sum_t \text{card}(\{T' \subset T \mid X_{T'}(t) \subset \mathcal{V}_C(t)\}) \right) , \tag{4.1}$$

subject to the constraints that, for all $t$, $v_C(t) \leq v_C^{max}$, and $x_C(t)$ is collision free.

When the positions of $T$ are not known, the camera will be in the search mode. In general, searching is difficult even when the environment is known. In Section 4.3.5, we will briefly show that this problem can be significantly simplified when we consider the problem of finding the large crowd.

## 4.2 Overview of Pursuit Strategies

We developed four baseline visibility-based pursuit strategies. The first three strategies are extensions of the existing methods which are originally designed to track a single target. The fourth motion strategy is based on the idea of monotonic tracking region (MTR). Since there is no prior work on visibility-based pursuit, we will also compare these strategies against each other in Section 4.4.

The baseline strategy is called *reactive camera*. It simply determines its next configuration by placing the visible targets as centered in the view as possible based only on the targets' current positions. The motivation is that by placing the visible targets at the center of its view, the camera will have better chance to find invisible targets.

### 4.2.1 IO Camera

IO camera is a sampling-based method extended from [12]. At each time step, given the visible targets $T$, the planner first creates $k$ point sets $P_T$, where $k$ is a parameter. Each point set contains $|T|$ predicted target positions. The predicted position of each target $\tau$ is sampled

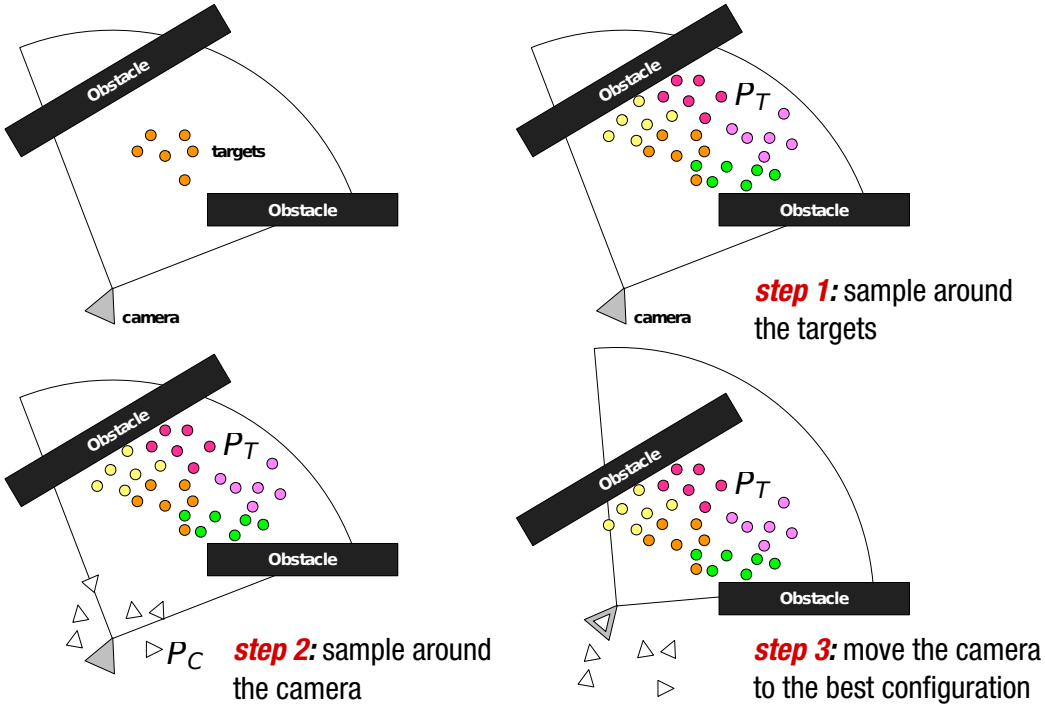



Figure 4.2: Illustration of the stages in IO camera

from the region visible from $\tau$, and is at most $(v_T^{max} \cdot \triangle t)$ away from $\tau$. The planner then creates a set $P_C$ of camera configurations (including position and view direction) at most $(v_C^{max} \cdot \triangle t)$ away from $C$. To decide the next camera configuration, we determine

$$\arg\max_{x \in P_C} \left( \sum_{X \in P_T} \text{vis}(x, X) \right) ,$$

where $\text{vis}(x, X)$ is the number of points in $X$ visible by $x$. These steps are illustrated in Fig. 4.2. To simplify our discussion, we will use the notation IO-$k$ to denote an IO camera that samples $k$ point sets for target prediction. It's important to remember that the IO cameras usually cannot be used as an online planner for tracking a large group (e.g., more than 50 targets) because of the large number of visibility checks between the sampled target positions and camera configurations in every time step.

### 4.2.2 VAR Camera

VAR camera is loosely based on [118]. Here, we preprocess the environment to obtain information about the environment in the form of a *roadmap* and *visibility graph*. Then, we use this pre-computed information to make proactive movement decisions in real-time.

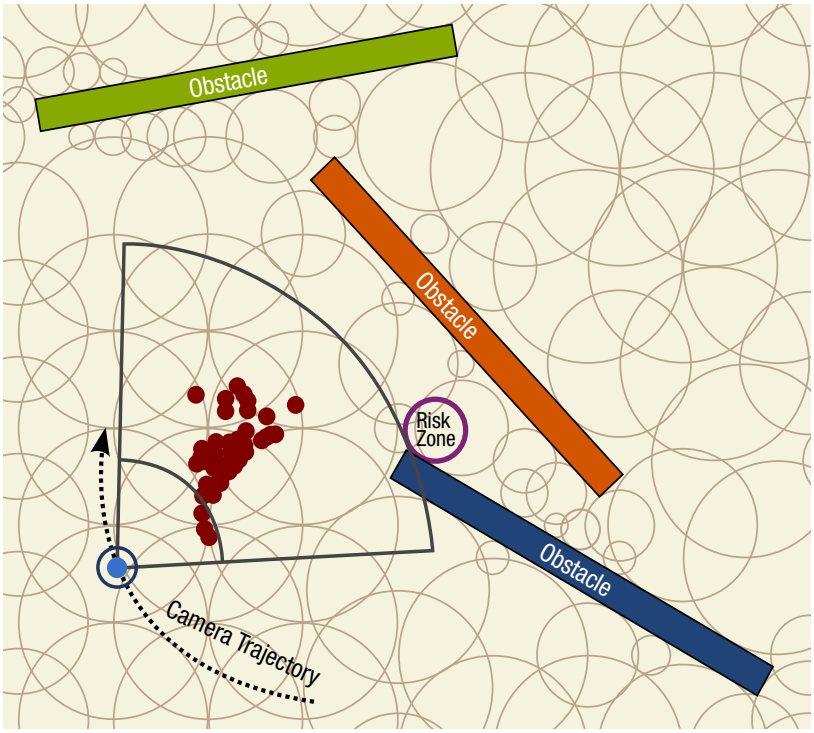



Figure 4.3: Illustration of VAR camera

This is done by first sampling a grid of discs $D$ in $\mathcal{C}_{free}$. To do this, a grid resolution $\Delta_{grid}$ is specified by the user to correspond to the size of the environment and obstacles. For each grid point, a disc $d$ is created with minimum radius is $r_{min}$, and expanded to its largest collision-free size, up to maximum radius $r_{max}$. If $d$ meets such requirements, it is added to $D$. The user selection of the resolution $\Delta_{grid}$, $r_{min}$, and $r_{max}$ is critical for ensuring sampling coverage inside narrow passages.

This grid-based sampling method produces many overlapping discs in $D$. The area of

overlap between a pair of discs is referred to a *portal*. To reduce number of portals and discs, we first remove discs whose aggregate overlap with other discs is very small. Second, an iterative pruning algorithm is used to remove the unnecessary discs. This algorithm works in a greedy fashion. Starting at a random disc, selecting neighboring discs with maximal overlap are selected, and adjacent unselected discs are pruned. This process is repeated until there remain no unselected discs.

With this reduced set of discs $D$, we construct a roadmap $R = (V, E)$ such that $V$ contains the portals formed by intersections of discs in $D$, and $E$ contains edges between two portals if they are are connected by a disc. Additionally, for each pair of discs, a visibility metric is computed using a method similar to Monte-Carlo raytracing. This process involves repeatedly drawing line segments between a random point inside one disc to a random point inside another, and checking if that line segment collides (hits) an obstacle. This ratio of misses to hits is stored for each pair as the visibility metric.

This visibility data structure, along the portal roadmap $R$, can be used in a variety of ways. Our VAR method is a hybrid approach that uses the constructed visibility information in a reactive behavior when the camera has good visibility of the targets (more than 20% of the targets are currently visible by the camera), and uses visibility-aware path planning from [118] to plan short, alternative paths to reach predicted locations of targets when the camera has poor visibility.

The reactive behavior in VAR works by computing a waypoint for the camera on each frame, and attempting to move towards it. First, we find a disc $d_c \in D$ that is closest to $C$, and a disc $d_r \in D$ that represents an imminent occlusion risk. That is, the disc $d_r$ is selected as one that is in the direction of the visible targets' velocity, closest to the location of the visible targets, and whose visibility from the closest disc to $C$ is less than 100%. An example of computing $d_r$, a risk zone, is shown in Fig. 4.2.2. Once $d_r$ is computed, two waypoints are selected along the ray extending from $d_r$ and passing through $d_c$. The first waypoint leads the group of targets (it is on the side of the group that is in the direction of the group's aggregate motion), and the second waypoint is selected trailing the group. The

camera attempts to move to the closest of these two waypoints. Using two waypoints in this way prevents $C$ from having to move to the other side of the group to chase a waypoint if the target group suddenly changes direction, allowing the camera to take advantage of risk prediction regardless of whether it is in front or behind the group.

The visibility-aware path planner, which is used to reconnect with the group of targets when the visibility is low, is the same as described in [118]. It uses an A* algorithm on the pre-computed roadmap with a heuristic function to compute a path to the $d_r$ that simultaneously minimizes distance traveled and maximizes visibility of the targets.

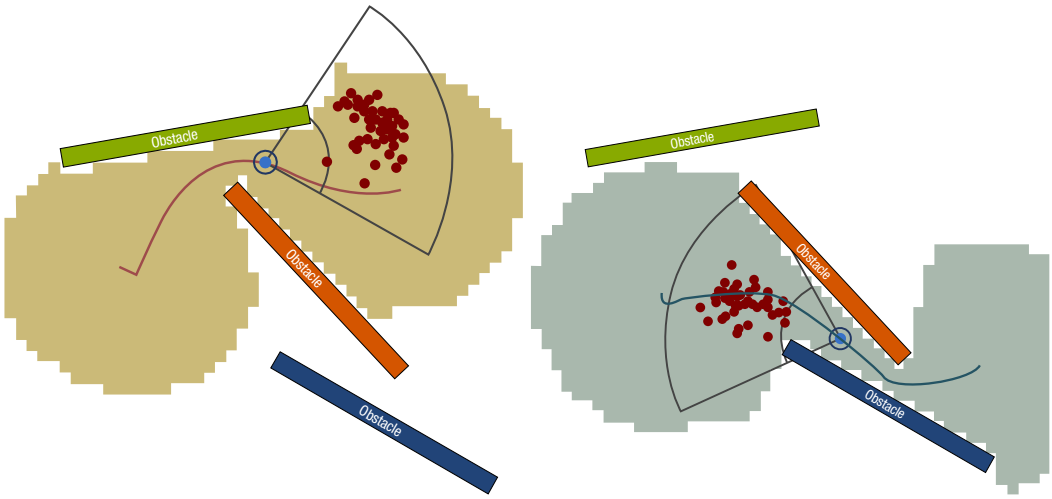
## 4.3 Monotonic Tracking Regions (MTRs)

The main idea of MTR [153, 156, 157] is to (1) partition the free space into regions with a simple (monotonic) topological feature so that the planner can reuse use the same data structure and strategy to follow the target group efficiently, and (2) utilize the fact the targets form a coherent group.

The first step of this method decomposes the environment into a set of *monotonic tracking regions* (MTRs). These regions usually look like tunnels and may overlap with each other. Intuitively, in these tunnel-like regions, the camera can *monotonically* maintain the visibility by moving forward or backward along a trajectory that *supports* the tunnel. More specifically, the main property MTR is that each MTR is topologically a *linear subdivision* so that the problem of visibility-based pursuit in a MTR can be represented as a *linear programming problem*. Two examples of MTR and their support paths are shown in Fig. 4.4.

Note that such a MTR needs not to be convex or star shaped, and, in fact, it can have an arbitrary number of turns (like a sinuous tunnel). Moreover, MTR decomposition usually creates much fewer components than convex or star-shaped decompositions but, as we will see later, still provide similar functionality in visibility-based pursuit. However, one can draw similarities between the support path of a MTR and the *kernel* of a star-shaped object $S$. The kernel of $S$ is a set of points that can see every point in $S$, thus by placing the camera in the

kernel we can always track the targets. Similarly, with more relaxed restriction, the points along support path of a MTR can *collectively* see every point in the MTR, thus, by moving the camera along the path, we can always track the targets.




(a) (b)

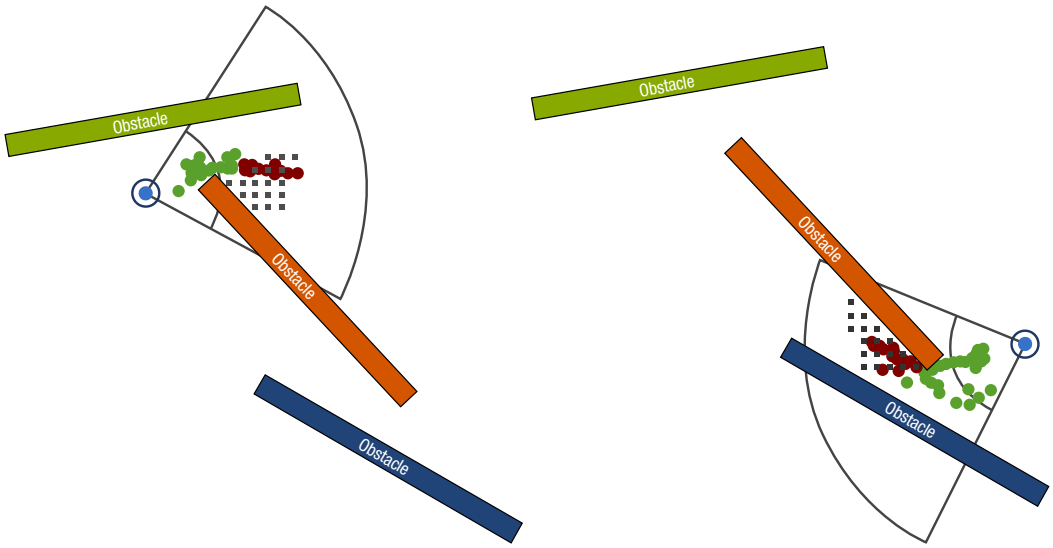
Figure 4.4: Two examples of MTRs (the shaded areas) and their support paths.

Once all the MTRs are extracted from the given environment, the online real-time planner will (implicitly) construct a global graph, called a *transition graph*, whose nodes are the *subregions* of MTRs. The nodes are connected if their subregions overlap. Since the graph is constructed upon the MTRs, the size of the graph is rather small even for a complex, realistic scene. Initially, the planner uses the transition graph to construct a breadth-first search (BFS) tree rooted at the MTR that the camera and the target reside in. The depth of the tree is proportional to $h$, a user-specified time horizon. During the online planning, the planner will update the BFS tree when the camera and target move to different regions.

In addition, we identify the intersections of MTRs. These intersections can help us to form critical areas in which the camera can regain visibility if the target has escaped. We call these intersections *kernels* in the sense similar to the kernel of a star-shaped polygon that can significantly increase the chance of seeing every point in the polygon. When the camera

is in the kernel of two MTRs, the camera can see (and follow) the target regardless which region the target is in.

Our planners also take advantage of the fact that the camera is pursuing a group of somewhat coherent targets. When the number of the targets visible by the camera is smaller than the total number of the targets, the planner will generate a set of *ghost targets* in the invisible regions of the workspace nearby the visible targets. Fig. 4.5 shows two examples of ghost targets. Note that the planner does not distinguish if a target is visible or is a ghost. Therefore, the planning strategy described in the previous sections remains the same. The positions of the ghost targets are estimated based on the following assumptions: (1) targets tend to stay together as a group, and (2) invisible targets are in $\mathcal{C}_{free}$ outside the visibility region of the camera. Therefore, even if targets are invisible, they must be in some occluded regions nearby. Our experiments (in Section 4.4) show that the idea of ghost targets significantly increases the visibility.




(a) (b)

Figure 4.5: Two examples of ghost targets in MTR camera. The invisible targets are shown as dark (red) circles, and the ghost targets are shown as small dots near the invisible targets.

We first provide a more precise definition of MTR and then we describe the process of computing these regions in Section 4.3.1. In Sections 4.3.2 and 4.3.3, we will discuss how to track the target in a single MTR and then in multiple MTRs.

### 4.3.1 Building Monotonic Tracking Regions (MTRs)

We let a 2D region $\mathcal{M}_\pi$ be a 2D generalized cylinder defined with respect to a *supporting path* $\pi$. We say $\pi$ is a supporting path of $\mathcal{M}_\pi$ if every point $x \in \mathcal{M}_\pi$ can see a subset of $\pi$. Because of this property, $\mathcal{M}_\pi$ can essentially be viewed as a linear object and the camera can see every point in $\mathcal{M}_\pi$ by moving along $\pi$.

**Definition 4.3.1.** *$\mathcal{M}_\pi \subset \mathcal{C}_{free}$ is a region supported by a path $\pi$ if $\mathcal{M}_\pi = \{x \mid \exists y \in \pi$ s.t. $\overline{xy} \subset \mathcal{C}_{free}\}$, where $\overline{xy}$ is an open line segment between $x$ and $y$, and $\mathcal{C}_{free}$ is the free space (i.e., the area without obstacles).*

Furthermore, we define the subset of $\pi$ visible by $x$ as: $\mathcal{V}_\pi(x) = \{y \in \pi \mid \overline{xy} \subset \mathcal{C}_{free}\}$ . Note that $\mathcal{V}_\pi(x)$ can have one or multiple connected components. Finally, we define MTR:

**Definition 4.3.2.** *A region $\mathcal{M}_\pi \in \mathcal{F}$ is a* MTR *supported by $\pi$ if $|\mathcal{V}_\pi(x)| = 1, \forall x \in \mathcal{M}_\pi$, where $|\mathcal{X}|$ is the number of connected components in a set $\mathcal{X}$.*

Because each $x \in \mathcal{M}_\pi$ can see only an interval of $\pi$, we can compactly represent the visible region (called visibility interval) of $x$ as a tuple $\mathcal{V}_\pi(x) = (s, t), 0 \leq s \leq t \leq 1$, if we parameterize $\pi$ from 0 to 1. Fig. 4.6 shows an example of MTR and its supporting path $\pi$.

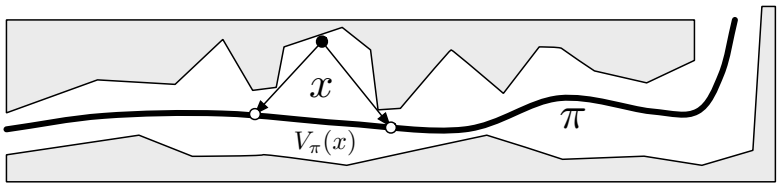



Figure 4.6: An example of monotonic tracking regions defined by $\pi$

With the definition in hand, our task here is to first find a set of supporting paths whose MTRs will cover $C_{free}$, and, next, from a given path $\pi$, we compute the associated MTR and the visibility interval $V_\pi(x)$ for every point $x$ in the MTR. We will describe these two steps in detail next.

**Constructing supporting paths**. Our strategy here is to find the homotopy groups $G$ of the $C_{free}$. We propose to use the medial axis (MA) of the $C_{free}$ to capture $G$ because of its several relationships with MTRs. First of all, we can show that the *retraction region* of every edge $\pi$ on the MA forms a MTR (supported by $\pi$).

**Lemma 4.3.3.** *The retraction region $R \subset C_{free}$ of an edge on the* MA *forms a* MTR.

*Proof.* Let $\pi$ be an edge on the medial axis MA of $C_{free}$. The retraction region $R \subset C_{free}$ of $\pi$ is simply a set of points that can be continuously retracted to $\pi$ by a retraction function $r : R \to \pi$ [160]. By definition, given an arbitrary point $x \in R$, the largest circle $c$ centered at the point $r(x)$ with $x$ on $c$'s boundary must be empty. Therefore, it follows naturally that each point in $R$ must be able to see at least a point on $\pi$.

Now we briefly show that each point $x$ can only see a consecutive region of $\pi$. We prove this by contradiction. Assuming that $x$ can see multiple intervals of $\pi$. This means that there must be an obstacle between $x$ and $\pi$. However, this contradicts the fact that $x$ can be retracted to $\pi$ in a straight line. Thus, we conclude that $R$ forms a MTR. □

The supporting paths are simply constructed by extracting the edges from the MA of a given environment. These paths only intersect at vertices, i.e., no paths share any common edges. More specifically, we extract the paths by iteratively finding the shortest path from the MA that does not overlap with the previous extracted path. This can be done by constructing the short path trees rooted at every vertex using Floyd-Warshall algorithm.

**Constructing MTRs**. Given an edge $\pi$ of MA, its retraction region $R$ forms a MTR supported by $\pi$. However, simply using $R$ as $\pi$'s MTR can be overly conservative. The set of points that satisfy Eq. 4.3.1 and 4.3.2 is usually larger than $R$. To address this issue, we iteratively expand $R$ by considering the points adjacent to $R$ until no points can be included

without violating the definition of MTR. Next, we compute the visibility interval for every point in a MTR. The brute force method for computing the visibility interval for each point time consuming. To speed it up, we use the following observation.

**Observation 4.3.4.** *If $x$ and $x'$ are (topological) neighbors, and $x$ is further away from $\pi$ than $x'$ is, then $V_\pi(x) \subset V_\pi(x')$.*

For example, in Fig. 4.6, imagining a point $x'$ below $x$, $x'$ can see a larger interval of $\pi$ than $x$ does. That is if we can compute the visibility intervals $V_\pi(x')$ for all the points $x'$ on $\pi$, then we should be able to obtain the visibility intervals for $x$ that are $\triangle d$ away from $x'$ by searching inside $V_\pi(x')$.

**Dominated MTRs**. The exact MA of a given environment can contain small (and in many cases unnecessary) features and result in many small MTRs. In many cases, these MTRs are unnecessary and should be removed. This is when a MTR is dominated by another MTR. We say MTR $M'$ is dominated by another MTR $M$ if $M' \subset M$. In our implementation, we use an approximate MA [160] to avoid small features, and identify and remove dominated MTRs.

### 4.3.2 Pursuing the Targets in a MTR

The motivating idea behind decomposing the environment into a set of MTRs is that the visibility-based pursuit problem in MTR can be solved much easily than that in the original environment. In fact, as we will see in this section, the camera can solve a long time horizon plan in MTR using linear programming.

**Follow a single target**. To simplify our discussion, we will first describe how a single target can be tracked in MTR. Let $x_\tau(t)$ be the current position of the target $\tau$ at time $t$. Since we know the current speed of the target, we can estimate the positions $x_\tau(t + \triangle t)$ in the next time step, i.e., the intersection of $\mathcal{F}$ and a disc with radius $\triangle t \cdot v_T^{max}$. To keep the target in the view, the camera's next position $x_C(t + \triangle t)$ must be:

$$x_C(t + \triangle t) \in \mathcal{V}_\pi\left(x_\tau(t + \triangle t)\right) = \bigcap_{x \in x_\tau(t+\triangle t)} \mathcal{V}_\pi(x) \ .$$

Note that this estimation can be applied to an arbitrary value of $\triangle t$. However, when $\triangle t$ is bigger, the position of the target becomes less accurate. Let $I_i = V_\pi(x_\tau(t + i \cdot \triangle t)) = (s_i, t_i)$. Here $i$ is an integer from 1 to $h$, where $h$ is the user-defined time horizon. Recall that both $s_i$ and $t_i$ are parameters on the path $\pi$. In order to follow the target for $h$ steps, the planner needs to find a sequence of camera locations $x_i$ from a sequence of parameterized intervals such that every point $x_i$ is in its corresponding interval $I_i$ without violating the constraint on the camera's max speed i.e., $|x_i - x_{i-1}| \leq v_C^{max}$. In addition, one may desire to minimize the distance traveled by the camera. Taking all of these into consideration, this problem can be formulated as an $h$-dimensional linear programming (LP) problem:

$$\begin{aligned} \min \quad & t_h - x_h \\ \text{s.t.} \quad & s_i \leq x_i \leq t_i \\ & 0 \leq (x_{i+1} - x_i) \leq \tfrac{v_C^{max}}{|\pi|}, \forall x_i \ , \end{aligned} \tag{4.2}$$

where $v_C^{max}/|\pi|$ is the maximum *normalized* distance that the camera can travel on $\pi$. Finally, the camera's future locations are simply $x_C(t + \triangle t \cdot i) = \pi(x_i)$.

Note that the rationale behind the minimization of $(t_h - x_h)$ is that when the target moves further away beyond $h$ steps in the future, the camera will have better chance of keeping the target in the view when it is located closer to $t_h$ along the path $\pi$. We call the above linear programming problem the *canonical pursuit problem*. Solving a canonical pursuit problem can be done efficiently since $h$ is usually not large ($h = 20$ is used in our experiments) given that modern linear programming solvers can handle thousands of variables efficiently. It is possible that the linear programming problem has no feasible solution, in which case we reduce the plan horizon iteratively until a solution is found.

**Follow multiple targets**. Now, we will extend this canonical pursuit problem to handle multiple targets $T$. Let $x_T(t)$ be the current positions of the targets $T$. Similar to the case of a single target, we estimate the positions $x_T(t + \triangle t)$ in the next time step. In order to see a

least one target, the camera must move so that

$$x_C(t+\triangle t) \in \bigcup_{\tau \in T} \mathcal{V}_\pi(x_\tau(t+\triangle t)) = \bigcup_{\tau \in T} \left( \bigcap_{x \in x_\tau(t+\triangle t)} \mathcal{V}_\pi(x) \right) .$$

To simplify our notation, let $I_i = \bigcup_{\tau \in T} \mathcal{V}_\pi(x_\tau(t) + i \cdot \triangle t) = (s_i, t_i)$. By placing the camera in $I_i$, we can guarantee that at least one target is visible. However, our goal is to maximize the number of visible targets, at least over the planning horizon. To do so, we segment $I_i$ into $j$ sub-intervals $I_i^j$, each of which can see $n_i^j$ targets. Fig. 4.7(a) shows an example of $I_i$ defined as the union the all the visibility intervals $\mathcal{V}_\pi(x_\tau)$ of the targets $\tau$. Note that $I_i$ may contain multiple connected components. Fig. 4.7(b) shows the subdivision of $I_i$ (i.e., subintervals $I_i^j$) bounded by the end points of $\mathcal{V}_\pi(x_\tau)$. Each $I_i^j$ is associated with the number of visible targets $n_i^j$. When the velocity of the camera is unlimited, then the optimal strategy is to pick the subinterval $I_i^j$ with the largest $n_i^j$ in each $I_i$, i.e., $I_i$ is shrunk to $I_i^j$. Thus, instead of solving the pursuit problem using $I_i$, the subintervals $I_i^j$ will be used. See Fig. 4.8.

From Fig. 4.8, one can also see that the distance that the camera has to travel from $x_2$ to $x_3$ is quite long, thus the camera will need to move very fast to maintain the maximum visibility. When the camera speed is bounded, this may not always be possible. Therefore, we need a way to select a subinterval from each $I_i$ so that the total number of visible targets is maximized while still maintaining the constraint that the minimum distance between $I_i^j \subset I_i$ and $I_{i+1}^k \subset I_{i+1}$ is smaller than $\triangle t \cdot v_T^{max}$. More specifically, we would like to find a solution to the following problem:

$$\arg\max_{\{j_i\}} \left( \sum_{i=1}^{h} n_i^{j_i} \right) \text{ s.t. } \text{dist}(I_i^{j_i}, I_{i+1}^{j_{i+1}}) \leq \triangle t \cdot v_T^{max}, \forall i ,$$

where $j_i$ is the index of the $j_i$-th subinterval in interval $I_i$, and $\text{dist}(x, y)$ is the closest distance

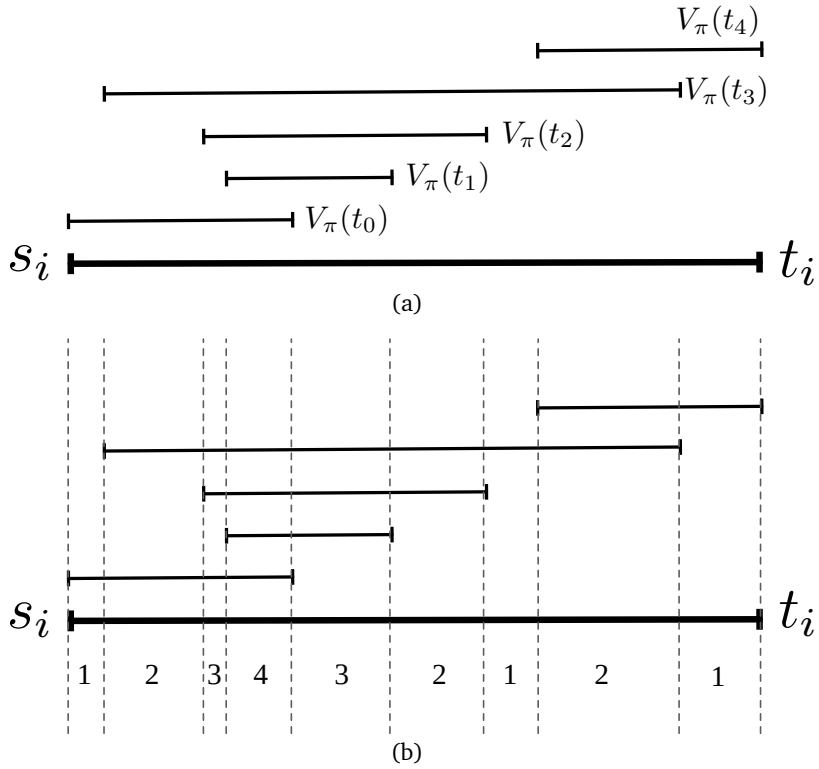



Figure 4.7: (a) The interval $I_i = (s_i, t_i) = \cup_{\tau \in T} \mathcal{V}_\pi(x_\tau)$. (b) The interval $I_i$ is segmented into 9 subintervals, each of which is a set of points in $\pi$ that can see the same number of targets, which is shown below each interval.

between two subintervals $x$ and $y$. Although, at the first glance, this problem seems to be another LP problem, fortunately, Lemma 4.3.5 shows that the optimal subintervals can be found in $O(hn^2)$ time, where $n$ is the number of targets and $h$ is the time horizon.

**Lemma 4.3.5.** *Finding all $I_i^{j_i}$ takes $O(hn^2)$ time for $n$ targets and $h$ planning time horizon.*

*Proof.* The main idea is to construct a directed graph from these subintervals and the current position of the camera, and show that this graph must be a DAG with $O(hn)$ vertices and $O(hn^2)$ edges. Then the problem of finding a sequence of optimal subintervals become the longest path search problem in the DAG, which can be solved in time linear to the size of the graph.

To construct such a graph, we first define the idea of *reachability*. Given two subintervals

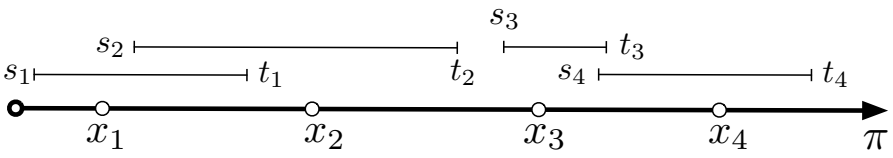



Figure 4.8: Making predictions for the next $h = 4$ future steps.

$u$ and $v$ from two consecutive intervals, the reachable interval $r(u, v) \in v$ is a set of points in $v$ that the camera can reach from $u$ in one step without violating the speed constraint. If $r(u, v)$ is not empty, then we say $v$ is reachable from $u$. Note that the reachability can be nested, i.e., given three subintervals $u$, $v$, and $w$, we say that $w$ is reachable from $u$ if $r(u, w) = r(r(u, v), w)$ is not empty. In the graph that we will construct, we ensure that every node in the graph is reachable from the current position $x_C$ of the camera. Finally, we say that a subinterval $v$ is reachable by the camera if $r(x_C, v)$ is not empty.

Specifically, we let the current position $x_C$ of the camera be the source of the graph and let the subintervals be the rest of the nodes in the graph. The source are then connected to the subintervals in $I_1$ that are reachable by the camera. The each reachable subinterval in $I_1$ are connected to the subintervals in $I_2$ that are reachable by the camera. The process repeats until the reachable intervals in $I_h$ are connected by those in $I_{h-1}$. Note that since we only need to pick one subinterval from each interval, the subintervals within each interval are not connected. Finally, we let the edge weight be the number of visible targets in the destination node. The graph constructed this way must be a DAG since there is no back edge. Any path the connects the source to a sink will contain a sequence of valid subintervals. Thus, finding the maximize number of targets visible from these subintervals is equivalent to finding the longest path in the DAG, which can be solved in linear time using topological sort. Since each interval will have $\Theta(2n)$ subintervals and two consecutive intervals will have $4n^2$ edges, this DAG has $O(hn)$ vertices and $O(hn^2)$ edges. □

**Regaining visibility of an escaped target**. Since the topology of the MTR is linear, regaining the visibility of the escaped target in a MTR is straight forward. When the target

escapes the camera's view, the camera can move along the supporting trajectory of the MTR toward target's last visible point in its maximum speed. Note that this strategy has two benefits. First, it allows the camera to gain visibility faster than moving the camera to the target's last visible position (due to the triangular inequality). Second, staying on defining trajectory allows the camera to maintain larger view range to prevent the target to escape from one blind spot to another blind spot. This simple idea works well when the escaped target remains in the MTR. The more difficult situation is when the target has potential of escaping to the other MTR. This leads us to the strategies for following the target across MTRs.

### 4.3.3 Following the Target Between MTRs

In this section, we will discuss strategies to follow the targets across MTRs. Without loss of generality, we only consider the case with two MTRs. The same approach can be naturally extended to handle three or more intersecting MTRs by decomposing the problem into pairs.

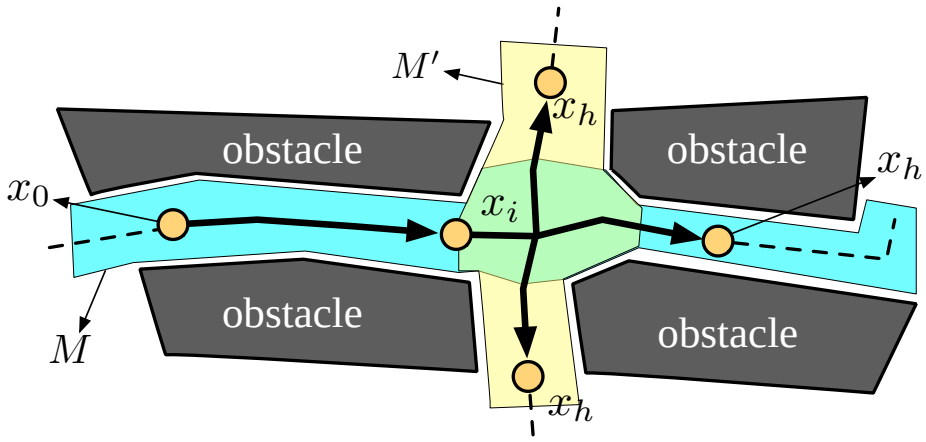



Figure 4.9: Tracking in multiple MTRs

Given two MTRs, $M$ and $M'$, we let $X = M \cap M'$. The intersection $X$ of two MTRs plays two critical roles. On the one hand, when the target enters $X$, the possibility for the target

to escape increases. On the other hand, when the camera enters $X$, the possibility of seeing the target increases.

Recall that we plan the camera's location by predicting the targets' future locations $x_T(t + \triangle t)$ and by computing the visible intervals $I(\triangle t)$ of $x_T(t + \triangle t)$. When $x_T(t + \triangle t)$ reaches $X$, each $x_T(t + \triangle t)$ can have two intervals, one in $M$ and the other one in $M'$. To compute the future locations of the camera, we solve at most *four* canonical following problems. Two of these are in $M$ and the other two are in $M'$. Fig. 4.3.3 shows an example, in which the targets may move from $M$ to $M'$ after time step $i$. Therefore, the camera will solve one canonical following from step 0 to $i$ and three canonical following problems from step $i$ to $h$. Finally, the best solution that maximizes the visibility will be used to move the camera.

To generalize the case described above, we consider the case that the time horizon is long enough so that $x_T(t + \triangle t)$ cross more than two MTRs. It is natural to maintain our plans in a planning tree. The root of the tree is camera's current location and has exactly two children, which define two sub-trees: one for $x^+(t + \triangle t)$ and the other one for $x^-(t + \triangle t)$. The internal nodes in the tree are critical points in $X$ and the edges are visible intervals in the corresponding MTR. The degree of the node is at most two times the number of MTRs intersecting at $X$. The number of canonical following problems is therefore equal to the number of edges in the tree. As the camera moves but stays in the same MTR, the tree will be updated in a way similar to Section 4.3.2, but updates are applied to the *leaves* of the tree. When the camera enters the intersection zone $X$ in one time step, the root of the tree moves between a child node $n$ of the root and $n$'s child, i.e., the structure of the tree changes. The canonical problems corresponding to the new edges created near the root will need to be solved.

**Regaining visibility of the target**. When the target has escaped "near" the intersections of MTRs, regaining the visibility of the target is not as easily as the case we described in Section 4.3.2. There are two difficulties here. First, when should we decide when the target may escape (and try to prevent that) and what we should do after the target has escaped.

To address the first issue, our idea is similar to escape risk estimation [9, 62], which computes the possibility of the target escapes around a corner. We estimate the risk of the target escapes to another MTR. More specifically, we compute the *minimum escape time* (MET) from each point in MTR. We first define the MET for each point $x$ in $M$ w.r.t $M'$ as:

$$\tau(x \in M, M') = \min_{y \in \mathcal{M}'} \text{dist}(x, y)/V_T \ ,$$

where $\mathcal{M}' \subset M'$ is a set of points in $M'$ that cannot see the supporting trajectory of $M$. Finally, we define the (MET) in $M$ as:

$$\tau(x) = \min(\tau(x, M')), \forall M', M' \cap M \neq \emptyset \ .$$

To prevent the target escaping into other MTRs, the strategy for the camera is to reach the *safe zone* before the MET of the target becomes zero. We define the safe zone in $M$ as:

$$\text{safe-zone}(X) = \{x \in M \mid \text{d}(x, X) \leq \tau(x) \cdot v_C^{max}\} \ .$$

From the definition, it is clear that when $\tau(x)$ is zero, the camera must be in $X$. It is also obvious that this estimation of the safe zone is conservative. That is, some points with zero MET are visible outside the safe zone. Therefore, we only encourage the camera to stay in the safe zone if $\tau(x)$ is smaller than $\epsilon$, a user define value. When target has escaped, the best strategy for the camera is to enter the safe zone at max speed.

### 4.3.4 MTR Architecture

**Offline computation**. Instead of a continuous space, our implementation is built on a regular grid which uniformly discretizes the workspace. The dimension of the (square) grid cell is the size of a target. We first approximate the workspace's medial axis MA using [160] and compute a MTR for each MA edge. Each MTR is a collection of cells that can see part of its

supporting path $\pi$. Each cell in the grid stores (1) a boolean variable which indicates if it is in $C_{free}$, and (2) a list of linked MTR *cells*, denoted as $\sigma_M$. Each $\sigma_M$ belongs to the MTR, $M$ and stores the visibility interval $V_\pi(\sigma_M) = (s_i, t_i)$. An illustration in Fig. 4.10 illustrates this. Advantages of using grid-based representation also include efficiency and applicability to video games and mobile robots, which often store the environmental data in bitmaps. Although we choose regular grid as our main data structure, other representations, such as topological graphs, can also be used, for example, when memory is strictly limited.

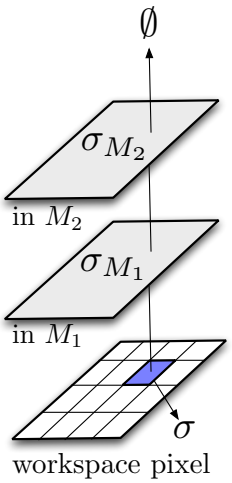
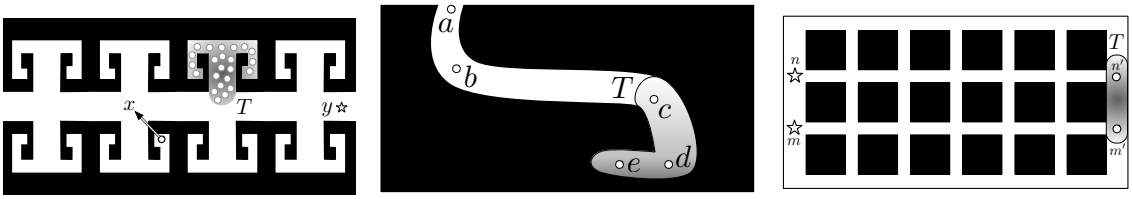



Figure 4.10: Data structure used in implementation.

**Online computation**. This is where the actual visibility-based pursuit takes place. In every time step, the camera obtains a list of visible targets, and computes the ghost targets accordingly (see below). To update the existing plan, the positions of the visible and ghost targets are used to retrieve occupied MTRs and visibility intervals from the pre-computed grid, which acts like an open hash table. Then, the canonical following problems, if any, associated with the new MTRs are solved. The camera then simply follows the trajectory that maximize the visibility.

### 4.3.5 Finding the Group in MTRs

In this section, we briefly show that finding a group of coherent targets with unknown trajectories is easier in general and can be done *asymptotically* more efficiently than finding a single target. The main idea in our analysis is that by considering the *minimum* area occupied by the group, we can reduce (sometimes drastically) the difficulty of the searching problem, both geometrically and topologically.




(a) indoor: local concavity (b) tunnel: global concavity (c) city blocks: topology simplification

Figure 4.11: Three cases showing that searching a group of coherent targets can be done asymptotically more efficiently. See text in Section 4.3.5 for more details.

Consider the two environments shown in Fig. 4.11. Each environment presents a scenario in which searching for a group can be done more efficiently than a single target. First, in an indoor-like environment, there can be a long corridor with many rooms (offices or cubicles), which we call *local concavity*. Small local concavity can also appear quite often in modeling errors in workspace geometry (due to numerical, sensor or human errors). In the case of a single target, the camera needs to go into every local concavity in order to ensure that the target is not hiding inside the concavity (e.g., $x$ in Fig. 4.11(a)). Moreover, without multiple collaborative cameras, it may be impossible to exhaustively search the entire workspace in Fig. 4.11(a) because when the camera enters one of the rooms, other rooms may be (re-)contaminated. However, when we consider a coherent group $T$ that is large enough to compactly occupy the local concavity such as $T$ in Fig. 4.11 (a), the entire workspace can be cleared by simply placing the camera at $y$ looking toward its left.

Second, we see a similar scenario in an environment with sinuous tunnel(s). For example, Fig. 4.11(b) requires the camera to move from $a$ to $d$ in order to search the entire tunnel for a single target; while in the case of a group $T$, the camera may only needs to move to $b$. We call this situation *global concavity*, in which a single camera can always clear the workspace for a single target but might take more time in doing so than looking for a target group.

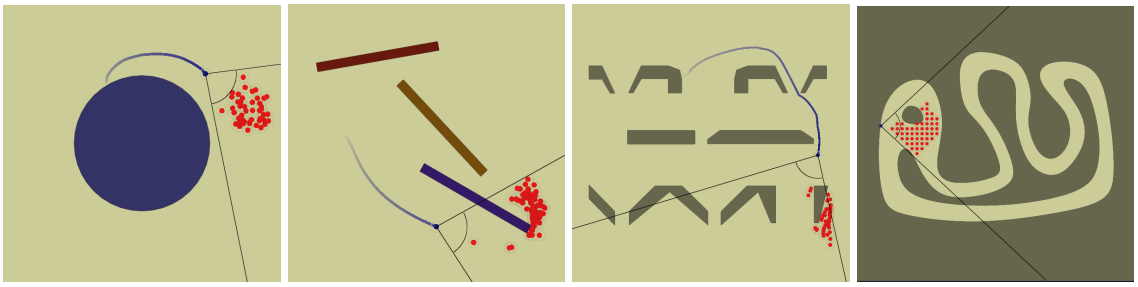
Third, in the city-blocks-like environment shown Fig. 4.11(b), we can see another example where no algorithms can guarantee to find a single target since the environment has many cycles. Therefore, a target can always hide behind an obstacle. In the case of multiple targets, if the group $T$ is large enough to span over two (horizontal) Homotopy classes (as shown in Fig. 4.11(b)), the camera can clear the environment by traversing any vertical or horizontal corridor, from one end to the other. The reason why the camera does not need to walk around each obstacle can be shown easily. By placing the camera at $n$ and then at $m$, if the camera cannot see any targets to its right in both places, e.g., at $n'$ and $m'$, (within a certain time) this indicates that there cannot be any targets between $n'$ and $m'$ because there is not enough space to fit the entire group $T$.

The above mentioned strategies reduce the search complexity to constant and to $O(n)$ for Figs. 4.11(a) and (b), respectively, where $n$ is the complexity of the environment. Both scenarios can be efficiently identified offline since we assume that the environment and the size of the targets are known. More specifically, in every MTR in the environment, we place the group at the end points of its supporting path and determine if the (either local or global) cavities are filled by the group, or if MTR is connected with the neighboring MTRs.

## 4.4 Experiments with IO, VAR, and MTR

In our experiments, the target group is constantly moving toward a random goal, which is not known by the camera. If all targets are invisible, the camera will stay still. Throughout the experiments, we measure the performance of the cameras by computing the *normalized visibility*, or the ratio of visible targets during the entire simulation. Every data point

presented in this section is an average over 32 runs, each of which is a 10000 time-step simulation. The planning horizon is $h = 20$ for all MTR cameras.



(a) disc (b) bars (c) islands (d) tunnel

Figure 4.12: Environments used in MTR experiments

We performed our experiments in the four workspaces shown in Fig. 4.12. These workspaces are designed to test the performance of the cameras in various conditions, such as large open space (disc), open space with narrow gaps (bars), small irregular obstacles with many narrow gaps (islands) and long sinuous narrow passages (RSS). Both islands and tunnel environments are considered difficult as the targets tend to separate around the small obstacles or hide behind a bend in the passage.

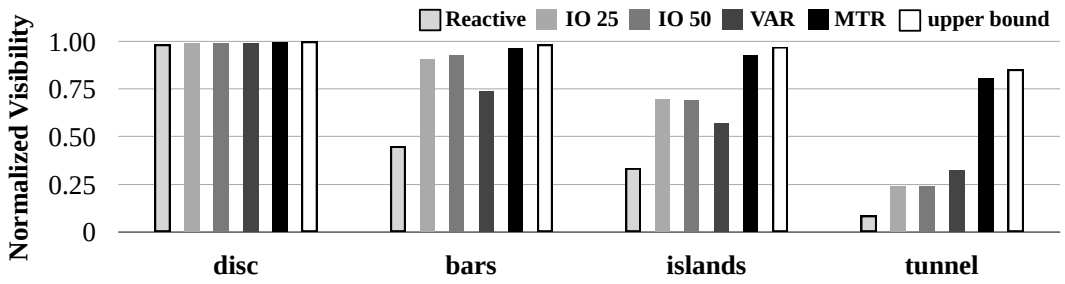



Figure 4.13: Visibility results for pursuing 50 targets in four environments using reactive, IO, and MTR approaches.

**MTR generally outperforms other strategies.** Our first experiment in Fig. 4.13 shows

strong evidence that MTR camera consistently achieves better normalized visibility than the other cameras when following 50 targets in all environments. It is also clear that the reactive camera has the worst performance. Note that we also include data called *upper bound* obtained from a MTR camera that has no speed limitation, i.e., it can move to the best configuration instantly. The strong performance of MTR is further supported by the small difference between the MTR camera and the upper bound in all four environments.

In the simple disc environment, all of the algorithms perform similarly. However, as the complexity of the environment increases in different ways, the difference between the algorithms becomes clear. For example, it is clear that the reactive camera performs worst, except in the disc environment. The VAR camera is the second worst, except in the RSS environment. However, as we will see, the VAR camera seems to handle large groups and faster moving targets better because of its enhanced ability to estimate risks. Furthermore, although IO cameras perform well in some situations, IO-25 in the bar environment is more than 200 times slower than VAR (≈2600 fps) and 12 times slower than MTR (≈157 fps), and thus cannot be used in many applications, such as real-time task monitoring and video game. There is no significant difference between IO-25 and IO-50.

**Experiments with ghost targets**. In the experimental results shown in Fig. 4.14, we attempt to estimate quantitatively the performance gain due to the idea of *ghost targets* (GS). Our result shows that the performance gain is more significant in more difficult environments. If compared to the (very time consuming) IO cameras in Fig. 4.13, MTR without GS is only slightly better in the island and RSS environments, but there are significant differences between the IO cameras and MTR with GS.

**Varying target group size and target speed**. We also attempted to gain better understanding of the MTR camera in relation to the other camera methods by varying the size and the maximum speed of the targets. In Fig. 4.16(a), we vary the sizes of the targets from 10 to 100, and can see that the size change has little effect on their performance, except in the RSS environment. By further examining the simulations, we observed that this performance drop is in fact inevitable (unless perhaps multiple cameras are used) because it is simply

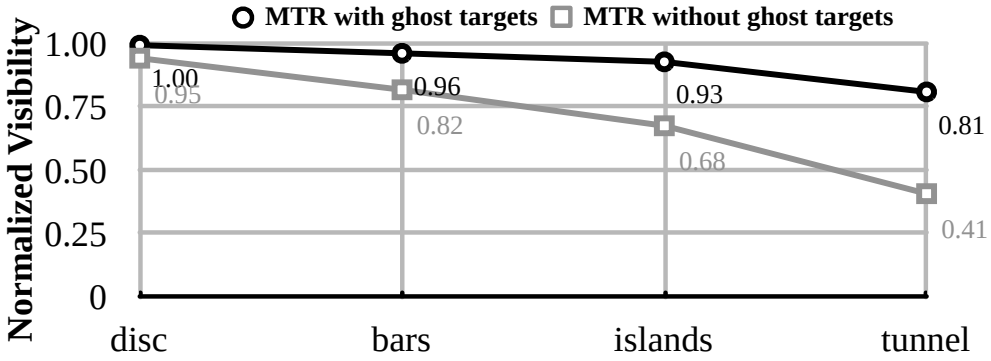



Figure 4.14: Comparing MTR cameras with and without ghost targets.

impossible for a single camera to see the entire group, e.g., when the group contour bends with the environments.

In Figure 4.16(b), we vary the speed of the target from $\frac{1}{2}$ (which is what used in the previous experiments) to twice the camera speed. When the targets and camera have the same maximum velocity, MTR keeps more than 70% visibility of the targets in all environments and is still significantly better than other cameras. However, when the targets are 1.5 times faster than the camera, the performance of MTRdrops quite significantly, especially in the island and RSS environments. Again, we found that this performance drop is due to the environmental constraints (e.g., when the targets escaped, the RSS environment provides no shortcut for the camera to re-capture the targets), and is inevitable unless multiple cameras are used.

## 4.5 Cached Intelligent Observers

In this section, we describe two methods for visibility-based pursuit we called *cached intelligent observers*: CIO$_g$ and CIO$_c$. Both of these methods extend the Intelligent Observer (IO) [12] (detailed in previous work [156]) by caching visibility information. The main

structure of both CIO$_g$ and CIO$_c$ is to separate the method into two parts: *pre-processing*, executed before the simulation starts to gather information about the environment and $\mathcal{C}_{free}$; and *behavior rule*, which is executed at each time step to control the motion of the camera $C$. The main difference between the CIO$_g$ and CIO$_c$ approaches is the data structure that is used to store the information about visibility in the workspace. The CIO$_g$ camera uses uniform grid while CIO$_c$ covers the workspace with overlapping discs.

In summary, the CIO$_g$ method begins by creating a two dimensional grid. For each grid point, CIO$_g$ caches a certain amount of information about not only itself (such as its distance from the nearest obstacle), but also about other grid points in the network (such as its visibility to other points). Like its predecessor, IO, at each time step, CIO$_g$ uses prediction and evaluation of camera and target positions to try and find an ideal location for the camera to maintain visibility of as much of the flock as possible for the next time step.

In CIO$_c$, a slightly different approach is taken to storing visibility information in the space. First, using disc-like partitions, a roadmap is computed as well as a visibility graph among overlapping partitions of the workspace. As with CIO$_g$ and its predecessor IO, these data structures are used with successive cycles of prediction and evaluation of future target and camera positions to make decisions about where to move next.

More details about these two methods are given in the next two sections.

### 4.5.1 Uniform Partitioning Method: CIO$_g$ Camera

**Pre-processing**. The pre-processing stage of the CIO$_g$ camera starts by first setting up a 2D grid $G$ in the workspace with user-specified grid resolution $\Delta_G$. Each grid point is checked to determine whether it resides in an obstacle, and the visibility between every pair of grid points in the workspace is computed and cached. This simple pre-processing is used so that visibility can be quickly queried between areas of the space. To save time, it is possible to incrementally compute this visibility data as-needed.

**Behavior Rule**. The behavior rule for CIO$_g$ attempts to locally determine the best grid

point for the camera to move to. To do this, at each time step, given the targets $T$, the CIO$_g$ method first creates $k$ point sets $P_T$, where $k$ is a parameter set by the user. Each point set in $P_T$ essentially represents $|T|$ predictions, or forecasts for the targets. The predicted position of each target $\tau$ is randomly sampled from the region visible from $\tau$, and is at most $(v_T^{max} \cdot \triangle t)$ away from $\tau$.

At each time step, a set $P_C$ of candidate camera positions is selected. Initially, $P_C$ contains all grid points within a user-specified distance $r$ from the camera. Grid points that are either in collision with the obstacles or not visible from the current camera position are removed from $P_C$. In a sense, the final set $P_C$ at each time step represents the candidate positions for the camera to move to in the next time step. For each point $p$ in $P_C$, a score is computed by counting the number of predicted target points in $P_T$ that are visible if the camera is placed at $p$. An additional weight (a *distance bias*) is applied to the scores of each point in $P_C$ based on its distance to the targets, to break ties in case two camera positions have a similar score (this can happen frequently in open spaces, where many camera positions may have the same visibility of the flock). The grid point in $C$ with the highest score is selected as the waypoint for the camera in the next time step. In other words, we select the next waypoint for the camera based on the following formula:

$$\arg\max_{x \in P_C}((\sum_{X \in P_T} \text{vis}(x, X)) - \alpha \|X - x\|) \ ,$$

where vis$(x, X)$ is the number of points in $X$ visible by $x$, and $\alpha$ is a scaling factor for the distance bias. The viewing direction of the camera is selected as the centroid of the visible members of the targets.

### 4.5.2 Unstructured Partition Method: CIO$_c$ Camera

**Pre-processing**. The pre-processing stage of CIO$_c$ camera is based on the previous method VAR [156], which is itself based on [118]. Here, rather than compute visibility on-the-fly, we

preprocess the environment to obtain information about the free space of the environment in the form of a *roadmap* and *visibility graph*. After pre-processing, we can use this pre-computed information to make movement decisions. The main difference between CIO$_c$ and its predecessor VAR, is in the behavior rule—whereas VAR uses this information to compute occlusion risks, CIO$_c$ uses this information instead to evaluate future camera positions.

The pre-processing is done by first sampling a grid of discs $D$ in $\mathcal{C}_{free}$. To do this, a grid resolution $\Delta_G$ is specified by the user to correspond to the size of the environment and its obstacles. At each grid point, a disc-shaped partition $d$ is created with minimum radius $r_{min}$, and expanded to its largest collision-free size, up to maximum radius $r_{max}$. If $d$ meets such requirements, it is added to $D$. The user selection of the resolution $\Delta_G$, $r_{min}$, and $r_{max}$ is critical for ensuring sampling coverage inside narrow passages.

This grid-based sampling method produces many overlapping discs in $D$. The area of overlap between a pair of these discs is referred to as a *portal*. To reduce the number of portals and discs, we first remove discs whose aggregate overlap with other discs is very small. Secondly, an iterative pruning algorithm is used to remove the unnecessary discs. This algorithm works in a greedy fashion. First, a random disc $d_r$ is selected from $D$. While there remain no unselected discs, a disc $d_s$ is selected with maximum overlap with $d_r$ (largest portal size). All unselected discs that are adjacent to $d_s$ are removed. This process is repeated until there remain no unselected discs.

With this reduced set of discs $D$, we construct a roadmap $R = (V, E)$ such that $V$ contains the portals formed by intersections of discs in $D$, and $E$ contains edges between two portals if they are are connected by a disc. Additionally, for each pair of discs $d_p$ and $d_q$ in $D$, a *visibility metric* $P_V(d_p, d_q)$ is computed which represents the average visibility between points in $d_p$ with points $d_q$. First, a basic test is performed by checking the visibility between the centers and at the extents of the discs. If these are mutually visible, the areas represented by the discs are assumed to be completely mutually visible and no further checks are performed. However, if any of these checks pass, Monte-Carlo raytracing is used

to determine the visibility metric. This process involves repeatedly drawing line segments between a random point in $d_p$ to a random point in $d_q$, and checking if that line segment collides (hits) an obstacle. The simple ratio of misses to hits is stored for each pair of circles as the $V_P(d_p, d_q)$.

As with the older VAR method, this visibility data structure along the portal roadmap $R$ can be used in a variety of ways. The CIO$_c$ method uses this information in a reactive behavior when the camera has good visibility of the targets (more than 20% of the targets are currently visible by the camera), and uses visibility-aware path planning as in [118] on the roadmap to plan short, alternative paths to reach predicted locations of targets when the camera has poor visibility. The next section describes the behavior rule used by CIO$_c$ when the visibility is good.

**Behavior Rule**. The CIO$_c$ method uses a behavior rule that is similar to the previous method IO. Just like the CIO$_g$ approach above, at each time step, given the visible targets $T$, the CIO$_c$ method (based on [12]) first creates $k$ point sets $P_T$, where $k$ is a parameter, which represents a set of predicted target positions. Next, the planner creates a set $P_C$ of camera configurations that are at most $(v_C^{max} \cdot \triangle t)$ away from $P_C$. To decide the next camera configuration, we simply determine

$$\arg\max_{x \in P_C} \left( \sum_{X \in P_T} \text{vis}(x, X) \right) ,$$

where vis$(x, X)$ is the number of points in $X$ visible by $x$.

### 4.5.3 Incremental Group Visibility: IGV

A major shortcoming of CIO$_g$, CIO$_c$ and its predecessor VAR is that they require a significant amount of offline pre-processing of the environment. We developed an incremental version of CIO$_c$ and CIO$_g$ which allows us to build up and cache visibility data online rather than offline. This visibility roadmap consists of a set of partitions connected to its nearby partitions

if it is at least partially visible from those partitions. At each time step, a query is made to determine whether the roadmap contains all of the information needed to compute visibility for candidate camera and target positions. If the roadmap does not contain sufficient information, a new partition in the space is created and the visibility between that new partition and the existing partitions in the roadmap is computed and stored in the edges of the roadmap. This roadmap is used at each time step to quickly compute an approximate visibility between candidate camera positions and predicted target positions.

Like CIO$_g$ camera, the IGV camera attempts to capture the visibility between areas of the space by first setting up a 2D grid $G$ in the workspace with user-specified grid resolution $\Delta_G$. However, unlike CIO$_g$, we do not compute the visibility between each grid point. Instead, we propose a simple partitioning approach that allows us to aggregate sets of nearby and similar grid cells into partitions. As we iterate through each grid cell, it is checked to determine whether it resides in an obstacle, and if it is already associated with a partition. If not, a new partition is created. Each partition is defined as a circular area in the space with radius that is proportional to its distance to the closest obstacle. Therefore, near obstacles and narrow passages, there will be many such partitions, and in open spaces, there will be fewer and larger partitions. This follows the notion that a higher resolution of visibility is important near occlusion risks, and a lower resolution is acceptable for open areas of the space where there are fewer obstacles. After a new partition $c$ is created, every grid cell inside $c$ that is not already associated with another partition is marked as associated with $c$. Finally, the visibility between the new partition and nearby partitions is computed and stored in the roadmap for future use.

## 4.6 Experiments with IO, CIO$_g$, and IGV

In this section, we compare the results for the methods IO, CIO$_g$, and IGV. In particular, IGV is an improvement over the baseline methods because it is able to achieve similar visibility performance to its predecessors while improving significantly on initialization time,

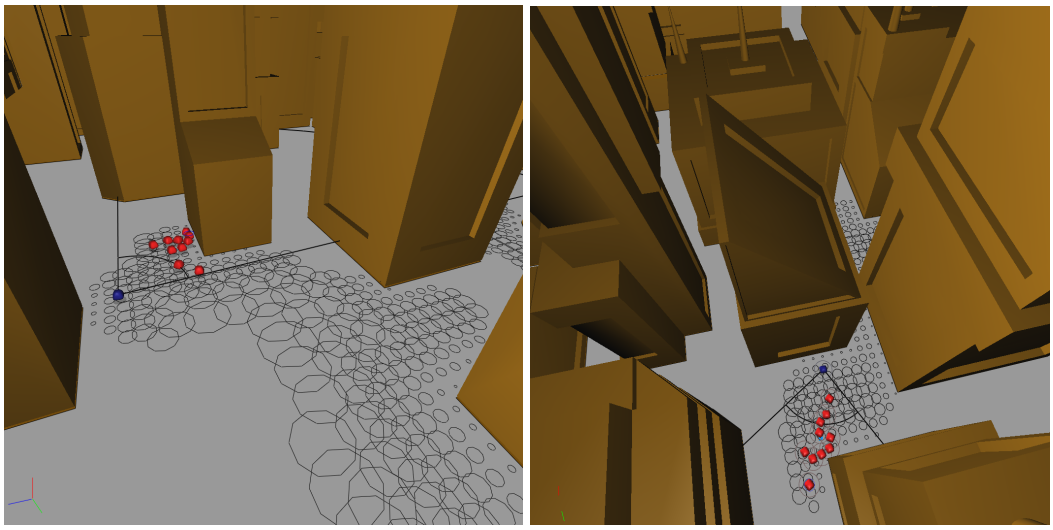


Figure 4.15: Screenshots of IGV in a city environment.

particularly on larger environments. In our experiments, we ran our algorithms on several simulated environments, shown in Figure 4.17. The maximum speed of the targets $v_T^{max}$ and the camera $v_C^{max}$ remain the same for all of the experiments. In each scenario $|T| = 30$. For the IO and CIO$_c$ algorithms, the number of samples for each target and the camera are set to 25 and 50 respectively. The time spent in initialization and each application of the behavior rule are presented as well as the average visibility of the targets after 10000 time steps. Figure 4.15 shows screenshots of IGV operating in a city environment.

Table 4.1 shows the results for visibility, rule time, and initialization time for each of the three algorithms on six different environments. What is particularly notable is the increase in initialization time for the CIO$_g$ approach as the environments become larger or more complex. In the city environment, initialization of the visibility data structure for CIO$_g$ takes nearly two minutes. However, with IGV all of the initialization times are zero, with insignificant difference in the rule time versus CIO$_g$ for most environments. In all environments, IGV achieves a faster rule time than IO, with no insignificant difference in visibility performance from IO. In these ways, IGV is naturally a replacement for both IO and CIO$_g$ approaches, achieving the best speed and performance advantages of each.

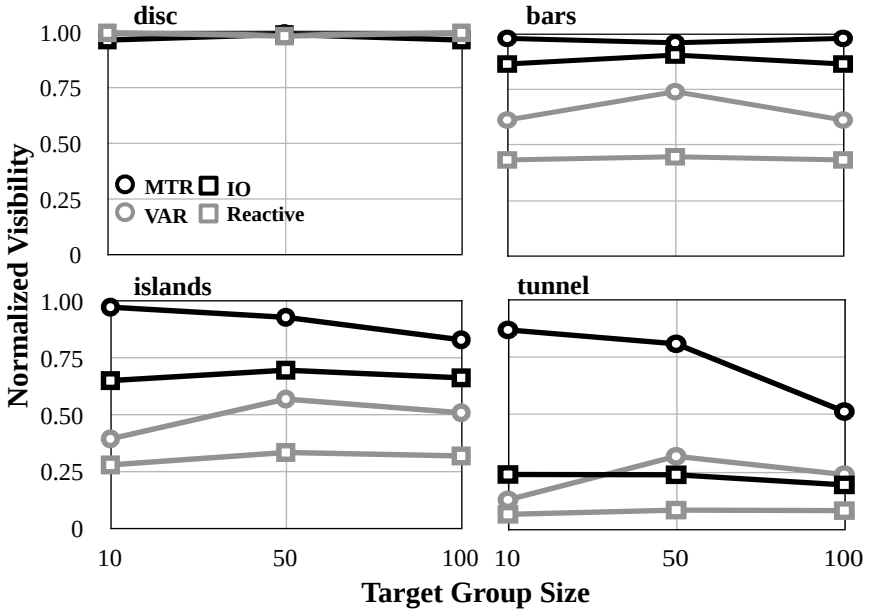
Table 4.1: Table of results for tracking 30 targets across several environments. Visibility score is an average visibility of the targets after 10000-steps of simulation across 30 runs. Visibility ranges from 0 (no visibility) to 1 (all targets visible). The initialization and rule times are reported in seconds and milliseconds respectively.

| | **Algorithm** | **Visibility** | **Rule (ms)** | **Init (s)** |
|---|---|---|---|---|
| **bars** | IO | 0.900 | 10.8 | — |
| | CIO$_g$ | 0.870 | 1.52 | 0.7577 |
| | IGV | 0.889 | 1.47 | — |
| **filter** | IO | 0.769 | 13.4 | — |
| | CIO$_g$ | 0.778 | 2.99 | 5.1199 |
| | IGV | 0.770 | 3.59 | — |
| **tunnels** | IO | 0.541 | 12.8 | — |
| | CIO$_g$ | 0.533 | 5.79 | 3.3247 |
| | IGV | 0.550 | 5.96 | — |
| **column** | IO | 0.987 | 11.1 | — |
| | CIO$_g$ | 0.987 | 1.45 | 0.5407 |
| | IGV | 0.984 | 1.65 | — |
| **pachinko** | IO | 0.631 | 22.1 | — |
| | CIO$_g$ | 0.622 | 9.55 | 28.5543 |
| | IGV | 0.611 | 11.4 | — |
| **city** | IO | 0.749 | 6.38 | — |
| | CIO$_g$ | 0.803 | 1.17 | 130.965 |
| | IGV | 0.768 | 1.19 | — |

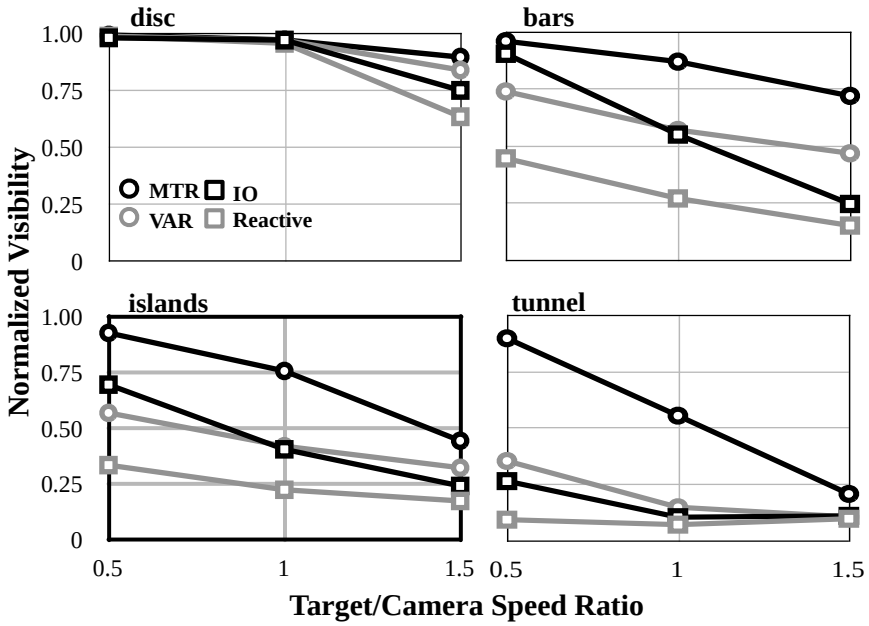
## 4.7 Conclusion

In this chapter, we presented robust and reusable camera planning techniques for visibility-based pursuit designed around a similar principle to the shepherding work from Chapter 3—namely, the pre-computation and caching of important geometric information about the workspace. To evaluate our techniques, we first developed baseline methods IO (based on the work from [12]) and VAR (based on the work by [118]). In our experiments, we found that IO was effective, but quite slow due to a large amount of sampling performed online, whereas VAR was efficient, but required a large amount of recompilation. We also developed a method called MTR, which is based on pre-processing the known environment offline

to obtain *monotonic tracking regions* which can be used the improve the efficiency of the online planner. Along the way, we also developed several extensions to this work to improve performance, such as the idea of *ghost targets*. We developed a class of methods called *Cached Intelligent Observers*, which attempt to combine the pre-processing and cached nature of VAR with the online behavior of IO. Finally, in IGV, we designed a camera system whose data structure is incrementally generated. These improvements enable planners to tackle very large workspaces, and provide a better balance between efficiency and performance than existing methods for large workspaces.




(a)




(b)

Figure 4.16: Camera performance with varying target sizes and maximum speeds.

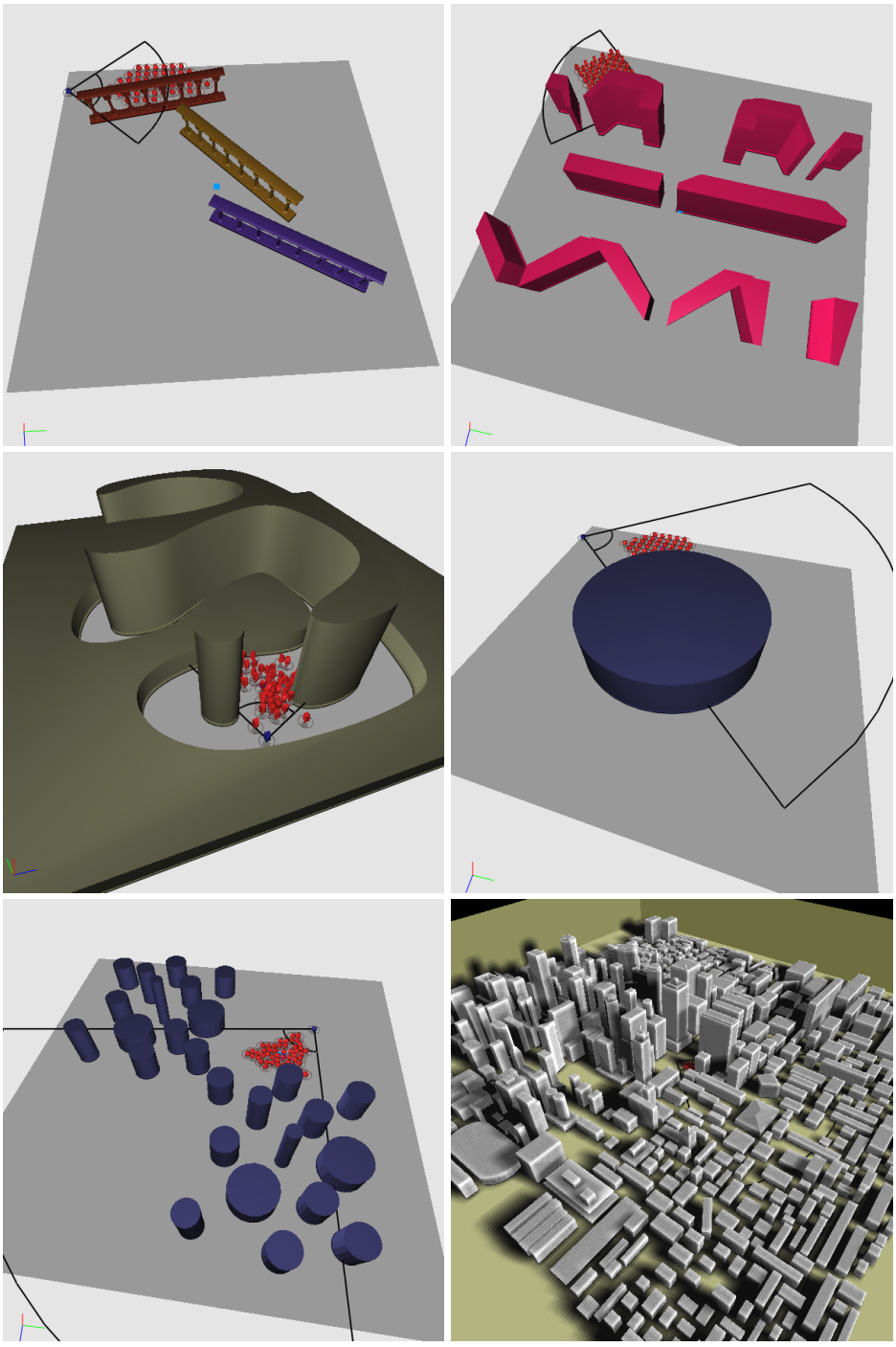


Figure 4.17: Images of the six environments used for the $\text{CIO}_c$ and IGV experiments. In reading order from top left: bars, filter, tunnels, column, pachinko, and city. For the most part, these environments are similar to those used in previous experiments except for city, which demonstrates the utility of IGV.

# Chapter 5: Conclusions

The main thread of this dissertation was to explore several ways to achieve better robustness and scalability for motion planning involving the control and monitoring of swarms by looking at two related problems—shepherding and visibility-based pursuit. In the shepherding scenario, we found that through geometric abstractions such as meta nodes, deformable shapes, and space partitioning, it was possible to make more intelligent decisions and produce coherent plans despite significant uncertainty in the pose of agents. In the pursuit scenario, we found ways to use offline computation on the geometry of the environment to greatly increase the effectiveness of online planning decisions, despite uncertainty about the future trajectory of agents.

## 5.1 Future Research

The work in this dissertation still has many open problems and concerns. Here, I discuss future trajectories of work with this framework and some of the preliminary work that I have done towards those trajectories. For example, meta graph techniques still require significant amounts of additional sampling to determine edge costs. It is not yet understood the relationship between the low-level planning techniques and the high-level planners, and whether a high-level planner can be designed that makes better use of low-level planners. Deformable pixel blobs to abstract large groups and space partitioning methods like VAR and MTR are useful, but these methods still rely on the use of 2D grid representations instead of other potentially more applicable representations such as polyhedra, $\alpha$-shapes, or 3D representations. All of the methods I have used in this work still do require significant offline computation on the environment and for the environment to be at least partially known a

priori. In most cases, this is justified by the ability to reuse computed data for many queries, but does not gracefully lend itself towards dynamic environments, where much sampled data may not necessarily be reusable.

### 5.1.1 Reusable Manipulation Planning Under Uncertainty

In Chapter 1.1, and in [69, 152], I described how meta graph techniques can be used to improve the robustness of shepherding. In a sense, the same technique may be applied towards the more general manipulation problem of posing rigid objects in the plane through push interactions. The most robust approaches to this problem usually involve grasping [29, 94,124], caging, using special manipulators [22], or require the design of rigorous analytical models for each new object [42]. Instead, I seek ways to perform this manipulation using *nonprehensile manipulation* (only pushing), considering dynamics, and through sampling-based methods rather than of analytical models. For simplicity, I consider a circular object, but one with significant dynamics and drifts (for example, a hockey puck resting on ice).

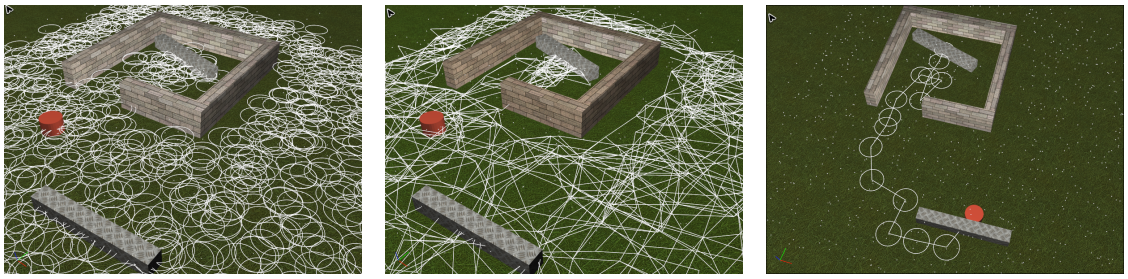
Furthermore, current planners for solving problems with kinodynamic constraints are usually tree-based planners (e.g., RRT [98], EST [73], PDST [142], and DSLX [123]). The general strategy behind such approaches is to grow an exploring tree rooted at an initial state until one of the goal states is (approximately) reached. However, if the initial or an intermediate state is inaccurate, or shifted a bit (e.g., due to various kinds of uncertainty and simulation and model inaccuracies), then the plan usually becomes invalid and a new tree will be built from scratch. Unlike these approaches, I seek to create an approach that is able to reuse prior computation to enable fast replanning. In addition, I hope to be able to reuse this data in pursuit problem (Section 5.1.2) so that pursuit and manipulation can be performed simultaneously (such as in the Simultaneous Localization and Grasping work in [124]) using the same data.

To achieve robust planning for this scenario, I have developed a graph-based planner based on the meta graph approach from [152]. This kind of planner generates a reusable uncertainty roadmap that can be used to perform multiple queries, for both short and long

horizon planning. The meta graph approach prescribes encoding the configuration of the target object in a fuzzy manner—that is, instead of a single configuration, each node of the roadmap acts as a *meta node*, representing a set of configurations.

I began preliminary research to determine the feasibility of this approach. More specifically, I considered an instance of a similar polygon pushing problem. This problem poses the question of how to plan the motion for a cylindrical robot $R$ to move polygonal target object $P$ from a given initial pose to a given goal pose using only push interactions. The workspace may be filled with known obstacles that the robot and the object are permitted to touch, but not penetrate. We assume that $R$ is a holonomic cylinder, and that $P$ is a polygonal shape. We also assume that a black-box kinodynamic simulator is provided, containing the obstacles, the robot, and the object. The simulation provides the resulting trajectories of $P$ and $R$ given control inputs for $R$.

A configuration of the robot $R$ can be represented as point $(x, y) \in \mathbb{R}^2$. Likewise, a configuration for $P$ is represented as a point $(x, y, \theta) \in \mathbb{R}^2 \times \mathbb{S}^1$, and the set of all such points forms the *configuration space* $C$ of the polygonal target object. As such, the set of all configurations where the object is colliding with an obstacle is referred to as $C_{obst}$, and the free space is referred to as $C_{free} = C - C_{obst}$.



(a) (b) (c)

Figure 5.1: An example pushing scene. The robot is depicted as a red cylinder, and the target object is the gray bar. In 5.1(a), white circles depict the meta nodes sampled from the configuration space after overlap reduction is performed. In 5.1(b), white lines depict the connections in the roadmap between meta nodes. In 5.1(c), a safe corridor is computed for the target object to move inside the room. The corridor is depicted by a series of connected circles (meta nodes).

My method consisted of several phases. First, we create a local planning function $LP$ that given a starting configuration for $P$ ($p_{start}$), and a desired destination configuration for $P$ ($p_{dest}$), outputs a push manipulation that supports moving $P$ from $P_{start}$ to $P_{dest}$. To do so, we sample information about how $P$ will move when it is pushed by $R$ from various angles. We store this information as points in a KD-tree so that when we are given the inputs $p_{start}$ and $p_{dest}$, we can quickly look up the closest sampled push manipulation that supports pushing $P$ to $p_{dest}$. Furthermore, for each manipulation, we also sample many times with random perturbations, to simulate uncertainty that may arise from imperfectly executing pushes. This information will be used later in roadmap generation to compute probabilities.

Next, we build the roadmap using this data. In general, we first seek to generate *meta nodes* which capture sets of states rather than single states, and then we connect them with edges whose weights reflect the probability of successful transit from one meta node to the next. We begin by sampling from the free space. Many sampling techniques can be used; we use the sampling technique from Gaussian PRM [104], mixed with uniform sampling. We convert each configuration $s$ from the set of samples into a meta node by building a sphere, centered at $s$, with pre-defined radius. Configurations that lie within the sphere defined for a configuration $s$ are thus considered to be *conforming* to the meta node for $s$. An example environment with sampled meta nodes is shown in Figure 5.1(a). This sampling process may lead to many overlapping meta nodes, so we also attempt reduce the number of meta nodes in the roadmap by removing meta nodes that have significant overlap.

Finally, to complete the roadmap, we connect meta nodes that are in close proximity. The edge between two nodes, $s_a$ and $s_b$ is given a weight that is computed based on sampling the number of feasible, successful pushes between uniformly random configurations conforming to $s_a$, and *any* configuration conforming to $s_b$. The number of successful pushes versus the number of attempted pushes at an edge provides us with a metric that can be used to estimate the probability of successfully pushing an object from some state conforming to node $s_a$ to some state conforming to $s_b$, in the presence of the obstacles. Other metrics can be used, such as the average probability of success. Figure 5.1(b) shows our example

environment with a constructed roadmap.

Given the weighted roadmap, we compute an all-pairs shortest path. This information is stored so that upon any query, a sequence of meta nodes can be quickly obtained that has the greatest probability of successful manipulation. We call this sequence of meta nodes a *corridor*, representing many possible paths for pushing $P$. Figure 5.1(c) shows an example corridor extracted from a query in the example scene. This corridor represents a *robust path* and can be used online quickly and efficiently to generate a final trajectory. Since the corridor is a significantly reduced subset of the free space, it can be used to focus search in another simple planning algorithm such as A-star, PRM, or RRT. Another relevant possibility is to apply the previous foraging work in [72] to continuously improve the meta graph. Meta nodes in the graph take the place of beacons, and can be continually re-arranged and assigned updated fitness as new paths through the space are sought and evaluated.

It is possible to develop a method that "learns" how to manipulate the target object by sampling repeated pushing interactions. This is motivated first by the notion that caching manipulation data may be useful if the same object may be manipulated many times by the same robot, and secondly by the notion that sampling manipulation results from either a simulated or real robot is easier and more generalizable than attempting to analytically derive control laws for pushing objects. I have completed preliminary work where many push interactions are sampled on a polygonal object and the collected data is used to build a roadmap that is used for manipulation planning. Currently, I have implemented a baseline approach which uses the sampled push data as a local planner for a simple high level EST approach. However, because the sampled push data does not consider obstacles, this approach sometimes fails to find successful plans. However, I have also implemented a method that works by grouping potentially similar push results together (a meta graph style approach), and developing a roadmap that captures the probability that an object can be successfully manipulated from one set of configurations to another. By doing so it is possible to compute the path with greatest probability of success, and focus sampling where it is needed most to solve the problem efficiently.

Such work can lead to the development of a framework for the Simultaneous Tracking and Manipulation Problem, or STAMP. When manipulating an object, the movements used by a shepherd to manipulate a flock may not take into consideration the fact that the robot can lose sight of the flock. Likewise, the movements made by the robot to ensure visibility of the flock may cause the robot to lose control of it. Our objective in STAMP would be to find ways to balance these objectives so that all of the goals can be met simultaneously.

### 5.1.2 Robust Pursuit With Visibility and Uncertainty in 3D

As we have described in Chapter 1, to manipulate an object, it is important to be able to track its position. Therefore, in [153–156], we previously experimented with several different methods to solve an instance of the visibility-based pursuit problem. Our two most successful techniques, VAR and MTR, involved partitioning the free space of the environment into topologically connected regions which provide our planner a view of both the mobility of agents in the space as well as the visibility between different regions of the free space.

Nevertheless, there is much room for improvement. For example, the roadmap in VAR is constructed based on partitioning the free space into a network of overlapping hyperspheres, and computing the visibility between those spheres. The sphere centers are selected by picking free configurations from a fixed grid, and the maximum radius of the spheres generated from those configurations is given as input by the user. This method of partitioning the environment has several limitations. First, to ensure coverage of narrow passages of the space, it is required to have a fine resolution grid, which can greatly increase the number of partitions and the sampling required to compute the roadmap. Furthermore, it is important that each sphere in the roadmap accurately represents points of the space that have similar visibility properties, so that clear decisions can be made about which areas of the space are visible from other areas of the space. However, by sampling on a rigid grid, it is common to end up with spheres containing points that have very different visibility characteristics. To reduce these problems, we seek a space partitioning method that can be performed on 3D environments, and is less sensitive to discontinuities or noise in the environment model.

Fortunately, the VAR method is amenable to improvement in this area. Therefore, to address the issues, it may be possible to compute the visibility-aware roadmap in VAR based on a combination of sampling and clustering. That is, to produce a space partitioning such that each node in the roadmap represents a set of points with similar visibility. This is done by first sampling $n$ points $\mathcal{S}$ from the 3D space. For each point $p \in \mathcal{S}$, we determine two types of connections: a visibility connection $v(p, q)$ if two points $p$ and $q$ are visible from each other, and a neighborhood connection $n(p, q)$ if two points $p$ and $q$ are neighbors.

We then cluster the points. To do so, we must first define a relevant metric for clustering. Here, we define the visibility integrity, which measures how accurately a set of points are indeed visible by a cluster of points. We compute and update visibility integrity to estimate the loss of visibility that occurs after merging two clusters. Let $C$ be a cluster of points. If $C$ contains only a single point $p$, its visibility $V(C)$ is simply all points visible from $p$. If $C$ contains multiple points, then $V(C)$ is the union of all points visible from the points in $C$, i.e., $V(C) = \bigcup_{p \in C} V(p)$. For each point $q$ in $V(C)$, we measure the visibility integrity of $q$ with respect to $C$. We denote this measure as a function $\mathbf{vi}(q, C)$ that maps a point $q$ and a cluster $C$ to a scalar value between 0 and 1. When $\mathbf{vi}(q, C)$ is 1, then all points in $C$ can see $q$. Likewise, if $\mathbf{vi}(q, C)$ is 0, $q$ is invisible from any point in $C$. Then, we define the visibility integrity of $C$ as the average of visibility integrity of all points in $V(C)$, i.e.,

$$\mathbf{vi}(C) = \frac{\sum_{p \in V(C)} \mathbf{vi}(q, C)}{|V(C)|} .$$

Intuitively, when visibility integrity is low, that means there are regions that are visible by some points in the cluster but are not visible from most of the points. Therefore, a cluster with high visibility integrity is more desirable. Note that the visibility integrity is always 1 when $C$ contains only one point.

Then, for each neighborhood connection $e = \{U, V\}$, we assign a weight to $e$ by computing the visibility integrity $\mathbf{vi}(U \cup V)$. We cluster points based on the order of their visibility

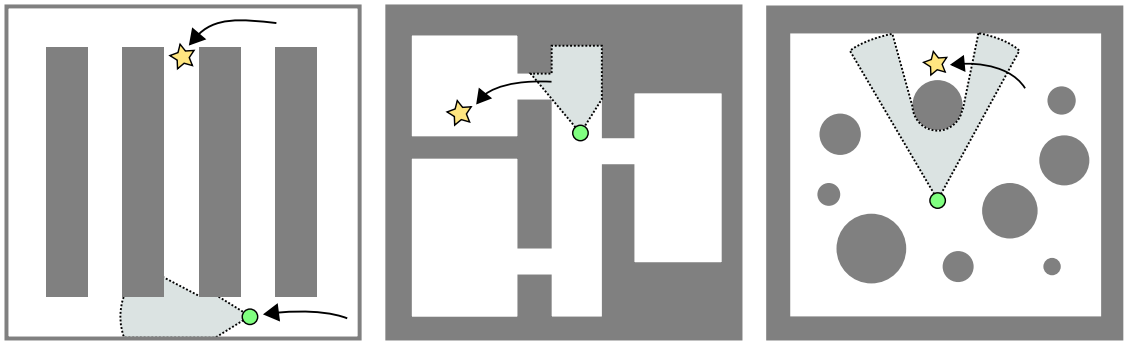
integrity. Our approach maintains neighborhood connections in a max-heap and iteratively collapses neighborhood connections with the highest visibility integrity. This process repeats until the highest visibility integrity is lower than a user specified value. We believe that this value is intuitive enough to be specified by a naïve user.

Each point $s$ in the original sample $\mathcal{S}$ now has the information about the clusters that are likely to be visible from $s$. Moreover, the cluster that has the highest visibility integrity to $s$ is usually the cluster that contains $s$. When the camera is pursuing a single target $t$, our planner simply finds the $k$ closest points around $t$ and identifies the clusters that are commonly associated with these $k$ points. The camera then determines its next target configurations by selecting a cluster and a configuration in the cluster which minimizes the travel distance and maximizes the visibility integrity. The motion of the camera is simply determined by a smooth and collision free path that brings the camera to the target. When the camera is pursuing multiple targets $T$, the planner will select $K$ points that are closest to $\mathcal{T}$. From these $K$ points in the roadmap, the camera decides its next position based on the same strategy.

One way to make such decisions is to use the computed visibility meta graph computed from the clustering approach to estimate which parts of the space represent risks not just in loss-of-visibility but also loss-of-localization. This information permits our camera sensor to localize objects in known environments without maintaining constant visibility, and potentially reduce the amount of movement necessary by the sensor to keep track of a target. These abilities are important for robustness because even if the visibility of the target is lost (leading to pose uncertainty). It ensures that sensor still has retained enough information to make decisions about where the mobile sensor can move, and optimize the probability of rendezvous with the target and predict the likelihood of escape. The idea of finding areas of the space that represent visibility risks is similar to the idea of finding *relocalization zones*, a concept used in [121] to plan paths that reduce localization uncertainty by maximizing visibility of important landmarks in the environment.

### 5.1.3 Pursuit Without Constant Visibility

Another limitation of the basic pursuit problem is that existing methods require that the mobile sensor maintains visibility of its target at all times. In reality, if the environment model is partially known, it may be possible to localize a moving target without maintaining constant visibility. If we are able to successfully localize the target without constant visibility, then we may be able to move the robot more freely and reduce the effort required to meet the combined objective of pursuing and manipulating the flock. For example, If a target is observed entering a partition of the space which is known to have only a single exit, it is not necessary to follow the target into the partition to localize the target. Instead, the robot may be able to trap the target inside a partition by moving to a position near the exit of the partition. Some examples of environments which illustrate these types of situations are shown in Figure 5.2. To achieve this, it is possible to use the above space partitioning method to discover areas that represent loss-of-localization risks and create paths that optimize the probability of future visibility or rendezvous with the target.



(a) Aisles (b) Rooms (c) Columns

Figure 5.2: Examples of pursuit without constant visibility. The sensor is depicted by a green circle, the target is depicted by a yellow star. The free space is the white area, and the obstacles are dark gray. In Figure 5.2(a), the target must be either in an aisle, or behind an obstacle. Determining which part of the space the target is in is only a matter of walking back and forth along the bottom of the aisle. In Figure 5.2(b), it is not necessary for a target to enter the room; the room is "covered" simply by waiting outside of it. In Figure 5.2(c), the target is obstructed by the column. However, the camera still covers the area adequately.

### 5.1.4 Combining UAVs and UGVs for Surveillance and Crowd Control

As mentioned in Section 2.1.2, there are several works [51, 71] adopting a heterogeneous approach to the multiagent control and monitoring problem. In particular, Chaimowicz and Kumar [27] implement a hybrid UAV/UGV command and control architecture. One very important feature importance of such an architecture is that UAV and UGV agents have complimentary abilities. For example, UAVs have the advantage of sensing from high perspectives and freedom to move around an environment in 3D. However, UAVs generally have limitations such as: low sensor resolution/fidelity, severely limited computing resources, short flight time, and inability to manipulate the environment. UGVs, on the other hand, have the ability to carry heavier and more powerful sensor and computing payloads, and can manipulate the environment, but lack the ability to see over obstructions and plan globally.

Lately, in [81, 86, 158], we have been exploring control architectures for a collaborative robot system that can take advantage of the complimentary roles of UAVs and UGVs to perform collaborative surveillance and crowd control. In particular, we developed a software and hardware test bed involving a custom-built quadrotor helicopter and differential drive UGV platform pictured in Figure 5.3. So far, we are in the process of developing an agent-based hardware-in-the-loop simulation for the multi-vehicle system and have done some preliminary data capture and communication tests.

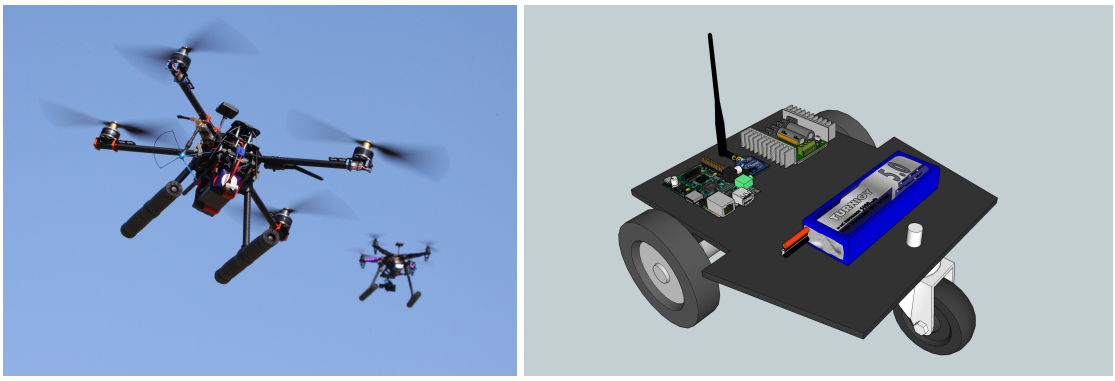


Figure 5.3: UAV and UGV developed for Surveillance and Crowd Control

# Bibliography


[1] S. Akella and M. T. Mason. Posing polygonal objects in the plane by pushing. *The International Journal of Robotics Research (IJRR)*, 17(1):70–88, 1998.

[2] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. *IEEE Transactions on Robotics and Automation*, 16(4):442–447, August 2000.

[3] D. M. Anderson. Virtual fencing–past, present and future. *The Rangeland Journal*, 29:65–78, 2007.

[4] R. Applegate. *Riot control: materiel and techniques*. Stackpole Books, 1969.

[5] K. Ashida, S. Lee, J. Allbeck, H. Sun, N. Badler, and D. Metaxas. Pedestrians: Creating agent behaviors through statistical analysis of observation data. In *Computer Animation*, pages 84–92, 2001.

[6] F. Aubé and R. Shield. Modeling the effect of leadership on crowd flow dynamics. In *ACRI*, pages 601–621, 2004.

[7] T. Balch and R. Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, 1998.

[8] T. Bandyopadhyay, M. H. A. Jr, and D. Hsu. Motion planning for 3-d target tracking among obstacles. In *Proc. Int. Symp. on Robotics Research (ISRR)*, 2007.

[9] T. Bandyopadhyay, Y. Li, M. H. A. Jr, and D. Hsu. A greedy strategy for tracking a locally predictable target among obstacles. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 2342–2347, 2006.

[10] T. Bandyopadhyay, Y. Li, M. H. A. A. Jr, and D. Hsu. Stealth tracking of an unpredictable target among obstacles. *Algorithmic Foundations of Robotics VI*, 17:43–58, Oct. 2005.

[11] W. H. Bares, J. P. Grégoire, and J. C. Lester. Realtime constraint-based cinematography for complex interactive 3D worlds. In *AAAI '98/IAAI '98*, pages 1101–1106. AAAI, 1998.

[12] C. Becker, H. González-Baños, J.-C. Latombe, and C. Tomasi. An intelligent observer. In *The $4^{th}$ International Symposium on Experimental Robotics IV*, pages 153–160, London, UK, 1997. Springer-Verlag.

[13] P. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb 1992.

[14] S. Bhattacharya, S. Candido, and S. Hutchinson. Motion strategies for surveillance. In *Proceedings of Robotics: Science and Systems*, Atlanta, GA, USA, June 2007.

[15] S. Bhattacharya and S. Hutchinson. On the existence of nash equilibrium for a visibility based pursuit evasion game. In *Proceedings of Workshop on Algorithmic Foundations of Robotics*, 2008.

[16] S. Bhattacharya and S. Hutchinson. On the existence of nash equilibrium for a two-player pursuit-evasion game with visibility constraints. *The International Journal of Robotics Research (IJRR)*, 57:251–265, 2009.

[17] J. Blinn. Where am i? what am i looking at? *IEEE Computer Graphics and Applications*, 8(4):76–81, July 1988.

[18] V. Blue. and J. Adler. Cellular automata model of emergent collective bi-directional pedestrian dynamics. In *Artificial Life VII, The Seventh International Conference on the Simulation and Synthesis of Living Systems*, pages 21–30, 2000.

[19] R. Bohlin and L. E. Kavraki. Path planning using Lazy PRM. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 521–528, 2000.

[20] M. Brenner, N. Wijermans, T. Nussle, and B. de Boer. Simulating and controlling civilian crowds in robocup rescue. In *Proceedings of RoboCup 2005: Robot Soccer World Cup IX*, 2005.

[21] D. C. Brogan and J. K. Hodgins. Group behaviors for systems with significant dynamics. In *Autonomous Robots*, pages 137–153, 1997.

[22] E. Brown, N. Rodenberg, J. Amend, A. Mozeika, E. Steltz, M. R. Zakin, H. Lipson, and H. M. Jaeger. Universal robotic gripper based on the jamming of granular material. *Proceedings of the National Academy of Sciences (PNAS)*, 107(44):18809–18814, 2010.

[23] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2383–2388 vol.3, 2002.

[24] P. Burelli, L. Gaspero, A. Ermetici, and R. Ranon. Virtual camera composition with particle swarm optimization. In *SG '08: Proceedings of the 9$^{th}$ international symposium on Smart Graphics*, pages 130–141, Berlin, Heidelberg, 2008. Springer-Verlag.

[25] B. Burns and O. Brock. Sampling-based motion planning with sensing uncertainty. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 3313–3318, 2007.

[26] Z. Butler, P. Corke, R. Peterson, and D. Rus. Virtual fences for controlling cows. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, volume 5, pages 4429–4436 Vol.5, April-1 May 2004.

[27] L. Chaimowicz and V. Kumar. Aerial shepherds: Coordination among uavs and swarms of robots. In R. Alami, R. Chatila, and H. Asama, editors, *Distributed Autonomous Robotic Systems 6*, pages 243–252. Springer Japan, 2007.

[28] L. Chaimowicz, N. Michael, and V. Kumar. Controlling swarms of robots using interpolated implicit functions. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 2487– 2492, 2005.

[29] L. Chang and N. Pollard. Posture optimization for pre-grasp interaction planning. In *In Proceedings of the International Conference on Robotics and Automation (ICRA) 2011 Workshop on Manipulation Under Uncertainty*, Shanghai, China, 2011.

[30] H. Cheng, H. Edelsbrunner, and P. Fu. Shape space from deformation. In *PG '98: Proceedings of the 6th Pacific Conference on Computer Graphics and Applications*, page 104, Washington, DC, USA, 1998. IEEE Computer Society.

[31] H.-L. Cheng, T. K. Dey, H. Edelsbrunner, and J. Sullivan. Dynamic skin triangulation. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 47–56, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[32] S.-W. Cheng, H. Edelsbrunner, P. Fu, and K.-P. Lam. Design and analysis of planar shape deformation. *Computational Geometry*, 19(23):205 – 218, 2001. Combinatorial Curves and Surfaces.

[33] S. Chenney. Flow tiles. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 233–242, New York, NY, USA, 2004. ACM Press.

[34] J. Choi and E. Amir. Combining planning and motion planning. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 238–244, May 2009.

[35] M. Christie and E. Languénou. A constraint-based approach to camera path planning. In *Smart Graphics*, pages 172–181, 2003.

[36] M. Christie and J.-M. Normand. A semantic space partitioning approach to virtual camera composition. *Computer Graphics Forum*, 24(3):247–256, 2005.

[37] M. Christie, P. Olivier, and J.-M. Normand. Camera control in computer graphics. *Computer Graphics Forum*, 27(8):2197–2218, 2008.

[38] D. Cliff and G. F. Miller. Co-evolution of pursuit and evasion II: Simulation methods and results. In P. Maes, M. J. Mataric, J.-A. Meyer, J. B. Pollack, and S. W. Wilson, editors, *From animals to animats 4*, pages 506–515, Cambridge, MA, 1996. MIT Press.

[39] P. Coleman, K. Singh, L. Barrett, N. Sudarsanam, and C. Grimm. 3D screen-space widgets for non-linear projection. In *Proc. of the 3$^{rd}$ international conference on Computer graphics and interactive techniques*, pages 221–228, 2005.

[40] N. Courty and E. Marchand. Computer animation: a new application for image-based visual servoing. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation, 2001.*, volume 1, pages 223–228, 2001.

[41] A. Dennis Nieuwenhuisen, F. van der Stappen, and M. H. Overmars. Pushing a disk using compliance. *IEEE Transactions on Robotics and Automation*, 23:3:431–442, 2007.

[42] M. R. Dogar and S. S. Srinivasa. A framework for push-grasping in clutter. In *In Proceedings of the International Conference on Robotics and Automation (ICRA) 2011 Workshop on Manipulation Under Uncertainty*, Shanghai, China, 2011.

[43] S. M. Drucker and D. Zeltzer. Intelligent camera control in a virtual environment. In *In Proceedings of Graphics Interface '94*, pages 190–199, 1994.

[44] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, IT-29:551–559, 1983.

[45] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Trans. Graph.*, 13(1):43–72, Jan. 1994.

[46] A. Egges, R. Geraerts, and M. Overmars. Camera planning in virtual environments using the corridor map method. In A. Egges, R. Geraerts, and M. Overmars, editors, *Proceedings of the International Conference on Motion in Games (MiG)*, volume 5884 of *Lecture Notes in Computer Science*, pages 194—206, 2009.

[47] M. Erdmann. On motion planning with uncertainty. Master's thesis, Massachusetts Institute of Technology, Aug. 1984.

[48] M. Erdmann and M. T. Mason. An exploration of sensorless manipulation. *IEEE Journal of Robotics and Automation*, 4(4):369–379, Aug. 1988.

[49] G. Fainekos, H. Kress-Gazit, and G. Pappas. Temporal logic motion planning for mobile robots. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 2020–2025, 2005.

[50] W. Feiten and M. Lang. MPG - a framework for reasoning on 6 dof pose uncertainty. In *In Proceedings of the International Conference on Robotics and Automation (ICRA) 2011 Workshop on Manipulation Under Uncertainty*, Shanghai, China, 2011.

[51] F. Fernandez, D. Borrajo, and L. E. Parker. A reinforcement learning algorithm in cooperative multi-robot domains. *Journal of Intelligent and Robotic Systems*, 43(2-4):161–174, 2005.

[52] M. F. Fingas. *The basics of oil spill cleanup*. Lewis Publishers, 2nd edition, 2001.

[53] A. Froneman. Towards the management of bird hazards on south african airports. In *Proceedings of the International Bird Strike Committee IBSC25/WP-SA5, Amsterdam, Netherlands*, 2000.

[54] C. Fulgenzi, C. Tay, A. Spalanzani, and C. Laugier. Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1056–1062, 2008.

[55] J. Funge, X. Tu, and D. Terzopoulos. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In *Computer Graphics*, pages 29–38, 1999.

[56] M. Gleicher and A. Witkin. Through-the-lens camera control. In *SIGGRAPH '92: Proceedings of the 19$^{th}$ annual conference on Computer graphics and interactive techniques*, pages 331–340, New York, NY, USA, 1992. ACM.

[57] O. Goemans and M. Overmars. Automatic generation of camera motion to track a moving guide. In *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 187–202, 2004.

[58] K. Y. Goldberg. *Stochastic Plans for Robotic Manipulation*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, Aug. 1990.

[59] K. Y. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10:201–225, 1993.

[60] K. Y. Goldberg and M. T. Mason. Bayesian grasping. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1990.

[61] H. Gong, J. Sim, M. Likhachev, and J. Shi. Multi-hypothesis motion planning for visual object tracking. In *International Conference on Computer Vision (ICCV)*, 2011.

[62] H. H. Gonzalez-Banos, C. yu Lee, and J.-C. Latombe. Real-time combinatorial tracking of a target moving unpredictably among obstacles. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 1683–1690, 2002.

[63] R. Graham, B. Lubachevsky, K. Nurmela, and P. Östergård. Dense packings of congruent circles in a circle. *Discrete Mathematics*, 181(13):139 – 154, 1998.

[64] C. A. Gran, P. P. V. Alcocer, and M. F. González. Way-finder: Guided tours through complex walkthrough models. *Computer Graphics Forum*, 23(3):499–508, 2004.

[65] P.-P. Grassé. La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. la théorie de la stigmergie: Essai d'interprtation du comportement des termites constructeurs. *Insectes Sociaux*, 6(1):41–80, 1959.

[66] J. Halloy and colleagues. Social integration of robots into groups of cockroaches to control self-organized choices. *Science*, 318(5853):1155–1158, 2007.

[67] N. Halper, R. Helbing, and T. Strothotte. A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence. *Computer Graphics Forum*, 20(3):174–183, 2002.

[68] W. Hamilton. Geometry for the selfish herd. *Journal of Theoretical Biology*, 31(2):295–311, 1971.

[69] J. F. Harrison, C. Vo, and J.-M. Lien. Scalable and robust shepherding via deformable shapes. In R. Boulic, Y. Chrysanthou, and T. Komura, editors, *Proceedings of the International Conference on Motion in Games (MiG)*. Springer, 2010.

[70] M. Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.

[71] A. Howard, L. E. Parker, and G. S. Sukhatme. The SDR experience: Experiments with a large-scale heterogeneous mobile robot team. In *Proceedings of $9^{th}$ International Symposium on Experimental Robotics (ISER)*, 2004.

[72] B. Hrolenok, K. Sullivan, S. Luke, and C. Vo. Collaborative foraging using beacons. In *Proceedings of the International Conference on Autononomous Agents and Multiagent Systems (AAMAS)*, 2010.

[73] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, volume 3, pages 2719–2726vol.3, Apr. 1997.

[74] R. L. Hughes. A continuum theory for the flow of pedestrians. *Transportation Research Part B: Methodological*, 36(6):507–535, 2002.

[75] R. L. Hughes. The flow of human crowds. *Annual Review of Fluid Mechanics*, 35(1):169, 2003.

[76] S. Hughes and M. Lewis. Robotic camera control for remote exploration. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 511–517, New York, NY, USA, 2004. ACM.

[77] R. Isaacs. *Differential games: A mathematical theory with applications to warfare and pursuit and optimization*. John Wiley, 1965.

[78] F. Jardillier and E. Languénou. Screen-space constraints for camera movements: the virtual cameraman. *Computer Graphics Forum*, 17(3):175–186, 2001.

[79] P. Kachroo, S. J. Al-nasur, S. A. Wadoo, and A. Shende. *Pedestrian Dynamics: Feedback Control of Crowd Evacuation*. Springer, 2008.

[80] L. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. In *Proceedings of the International Conference on Robotics and Automation (ICRA) Workshop on Mobile Manipulation*, 2010.

[81] A. M. Kaleghi, D. Xu, S. Minaeian, M. Li, Y. Yuan, J. Liu, Y.-J. Son, C. Vo, and J.-M. Lien. A dddams-based uav and ugv team formation approach for surveillance and crowd control. In *Proceedings of the 2014 Winter Simulation Conference*, 2014.

[82] A. Kamphuis and M. Overmars. Motion planning for coherent groups of entities. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3815 – 3822, April 2004.

[83] A. Kamphuis and M. H. Overmars. Finding paths for coherent groups using clearance. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 19–28, New York, NY, USA, 2004. ACM Press.

[84] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, Aug. 1996.

[85] J. M. Kenny, C. McPhail, D. N. Farrer, D. Odenthal, S. Heal, J. Taylor, S. Ijames, and P. Waddington. Crowd behavior, crowd control, and the use of non-lethal weapons. Technical Report A274644, Penn State Applied Research Laboratory, January 2001.

[86] A. M. Khaleghi, D. Xu, S. Minaeian, M. Li, Y. Yuan, C. Vo, A. Mousavian, J.-M. Lien, J. Liu, and Y.-J. Son. A comparative study of control architectures in uav/ugv-based surveillance system. In *Proceedings of the Industrial and Systems Engineering Research Conference*, 2014.

[87] A. J. King, A. M. Wilson, S. D. Wilshin, J. Lowe, H. Haddadi, S. Hailes, and A. J. Morton. Selfish-herd behaviour of sheep under threat. *Current Biology*, 22(14):R561 – R562, 2012.

[88] J. Kirkland and A. Maciejewski. A simulation of attempts to influence crowd dynamics. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 5, pages 4328–4333, 2003.

[89] L. Korte and M. S. University. *The Spatial and Social Organization of Forest Buffalo (Syncerus Caffer Nanus) at Lope National Park, Gabon*. Michigan State University, 2007.

[90] A. Kozyrev, V. Leonov, and V. Selivanov. Computer simulation of critical behavior of localized masses (crowd). In *3rd European Symposium on Non-Lethal Weapons*, 2005.

[91] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Motion planning for humanoid robots under obstacle and dynamic balance constraints. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 692–698, 2001.

[92] H. Kurniawati, Y. Du, D. Hsu, and W. S. Lee. Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research (IJRR)*, 30(3):308–323, 2011.

[93] H. Kurniawati, D. Hsu, and W. S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems (RSS)*, 2008.

[94] J. Laaksonen and V. Kyrki. Probabilistic approach to sensor-based grasping. In *In Proceedings of the International Conference on Robotics and Automation (ICRA) 2011 Workshop on Manipulation Under Uncertainty*, Shanghai, China, 2011.

[95] F. Lamarche and S. Donikian. Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum*, 23(3):59, 2004.

[96] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

[97] S. M. LaValle, H. H. Gonzalez-Banos, C. Becker, and J.-C. Latombe. Motion strategies for maintaining visibility of a moving target. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, volume 1, pages 731–736. IEEE, 1997.

[98] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 473–479, 1999.

[99] C.-Y. Lee, H. Gonzalez-Banos, and J.-C. Latombe. Real-time tracking of an unpredictable target amidst unknown obstacles. In *Control, Automation, Robotics and Vision, 2002. ICARCV 2002. 7$^{th}$ International Conference on*, volume 2, pages 596–601vol.2, Dec. 2002.

[100] T.-Y. Li and C.-C. Cheng. Real-time camera planning for navigation in virtual environments. In *SG '08: Proceedings of the 9$^{th}$ international symposium on Smart Graphics*, pages 118–129, Berlin, Heidelberg, 2008. Springer-Verlag.

[101] T.-Y. Li and T.-H. Yu. Planning tracking motions for an intelligent virtual camera. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1353–1358vol.2, 1999.

[102] J.-M. Lien, O. Bayazit, R. Sowell, S. Rodriguez, and N. Amato. Shepherding behaviors. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, volume 4, pages 4159–4164Vol.4, 26-may 1, 2004.

[103] J.-M. Lien, S. Rodriguez, J.-P. Malric, and N. M. Amato. Shepherding behaviors with multiple shepherds. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 3413–3418, April 2005.

[104] Y.-T. Lin. The gaussian prm sampling for dynamic configuration spaces. In *Control, Automation, Robotics and Vision, 2006. ICARCV '06. 9$^{th}$ International Conference on*, pages 1–5, Dec. 2006.

[105] P. Lissaman and C. Shollenberger. Formation flight of birds. *Science*, 168(3934):1003–5, 1970.

[106] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor. Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research*, 3(1):3–24, 1984.

[107] S. Luke, K. Sullivan, L. Panait, and G. Balan. Tunably decentralized algorithms for cooperative target observation. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 911–917, New York, NY, USA, 2005. ACM.

[108] R. A. Metoyer and J. K. Hodgins. Reactive pedestrian path following from examples. In *CASA '03: Proceedings of the 16$^{th}$ International Conference on Computer Animation and Social Agents (CASA 2003)*, page 149, Washington, DC, USA, 2003. IEEE Computer Society.

[109] G. Miller and D. Cliff. Co-evolution of pursuit and evasion I: Biological and game-theoretic foundations. Technical Report CSRP311, School of Cognitive and Computing Sciences, University of Sussex, Brighton, UK, 1994.

[110] K. Miyazawa, Y. Maeda, and T. Arai. Planning of graspless manipulation based on rapidly-exploring random trees. In *(ISATP 2005) The 6$^{th}$ IEEE International Symposium on Assembly and Task Planning From Nano to Macro Assembly and Manufacturing 2005*, page 7, 2005.

[111] F. Morini, B. Yersin, J. Maim, and D. Thalmann. Real-time scalable motion planning for crowds. In *Proceedings of the International Conference on Cyberworlds*, pages 24–26, Hannover, Germany, 2007.

[112] R. Murrieta-Cid, T. Muppirala, A. Sarmiento, S. Bhattacharya, and S. Hutchinson. Surveillance strategies for a pursuer with finite sensor range. *The International Journal of Robotics Research*, 26(3):233–253, 2007.

[113] R. Murrieta-Cid, B. Tovar, and S. Hutchinson. A sampling-based motion planning approach to maintain visibility of unpredictable targets. *Autonomous Robotics*, 19(3):285–300, 2005.

[114] C. Nielsen and E. Kavraki. A two level fuzzy prm for manipulation planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 1716–1721vol.3, 2000.

[115] D. Nieuwenhuisen and M. H. Overmars. Motion planning for camera movements. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3870–3876, 2004.

[116] N. Ogawa, H. Oku, K. Hashimoto, and M. Ishikawa. Microrobotic visual control of motile cells using high-speed tracking system. *IEEE Transactions on Robotics*, 21(4):704–712, Aug. 2005.

[117] M. Olivares-Mendez, T. F. Bissyandé, K. Somasundar, J. Klein, H. Voos, and Y. Le Traon. The NOAH project: Giving a chance to threatened species in africa with UAVs. In T. F. Bissyand and G. van Stam, editors, *e-Infrastructure and e-Services for Developing Countries*, volume 135 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 198–208. Springer International Publishing, 2014.

[118] T. Oskam, R. W. Sumner, N. Thuerey, and M. Gross. Visibility transition planning for dynamic camera control. In *SCA '09: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 55–65, 2009.

[119] L. Parker. Current research in multirobot systems. *Artificial Life and Robotics*, 7(1-2):1–5, 2003.

[120] L. E. Parker. Distributed algorithms for multi-robot observation of multiple moving targets. *Autonomous Robotics*, 12(3):231–255, 2002.

[121] R. Pepy and A. Lambert. Safe path planning in an uncertain-configuration space using rrt. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5376–5381. IEEE, 2006.

[122] H. Piland, Q. Nguyen, F. McKenzie, and B. Silverman. Incorporating weapon effects injury model into the crowd federate. In *Proceedings of the 11$^{th}$ AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2006.

[123] E. Plaku, L. E. Kavraki, and M. Y. Vardi. Discrete search leading continuous exploration for kinodynamic motion planning. In *Robotics: Science and Systems (RSS)*, 2007.

[124] R. Platt, L. Kaelbling, T. Lozano-Perez, and R. Tedrake. Simultaneous localization and grasping using belief space planning. In *In Proceedings of the International Conference on Robotics and Automation (ICRA) 2011 Workshop on Manipulation Under Uncertainty*, Shanghai, China, 2011.

[125] M. A. Potter, L. Meeden, and A. C. Schultz. Heterogeneity in the coevolved behaviors of mobile robots: The emergence of specialists. In *IJCAI*, pages 1337–1343, Dec. 2001.

[126] C. W. Reynolds. Flocks, herds, and schools: A distributed behaviroal model. In *Computer Graphics*, pages 25–34, 1987.

[127] C. W. Reynolds. Competition, coevolution and the game of tag. In *Artificial Life IV*, 1994.

[128] K. W. Rio and W. H. Warren. Visually-guided collective behavior in human swarms. *Journal of Vision*, 13(9):481, 2013.

[129] S. J. Rymill and N. A. Dodgson. Psychologically-based vision and attention for the simulation of human behaviour. In *GRAPHITE '05: Proceedings of the 3$^{rd}$ international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 229–236, New York, NY, USA, 2005. ACM Press.

[130] G. Saridis. Intelligent robotic control. *IEEE Transactions on Automatic Control*, 28(5):547–557, May 1983.

[131] J. Schubert and R. Suzic. Decision support for crowd control: Using genetic algorithms with simulation to learn control strategies. In *Military Communications Conference, 2007. MILCOM 2007. IEEE*, pages 1–7, Oct. 2007.

[132] A. C. Schultz and W. Adams. Continuous localization using evidence grids. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, volume 4, pages 2833–2839, 1998.

[133] A. C. Schultz, J. J. Grefenstette, and W. Adams. Robo-shepherd: Learning complex robotic behaviors. In *In Robotics and Manufacturing: Recent Trends in Research and Applications, Volume 6*, pages 763–768. ASME Press, 1996.

[134] D. Schulz, W. Burgard, D. Fox, and A. B. Cremers. People tracking with mobile robots using sample-based joint probabilistic data association filters. *The International Journal of Robotics Research*, 22(2):99–116, 2003.

[135] W. Shao and D. Terzopoulos. Autonomous pedestrians. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 19–28, New York, NY, USA, 2005. ACM Press.

[136] D. A. Shell and M. J. Matarić. Directional audio beacon deployment: an assistive multi-robot application. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 2588–2594, 2004.

[137] T. Smith and R. G. Simmons. Point-based POMDP algorithms: Improved analysis and implementation. In *Proc. Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*, 2005.

[138] A. Steed and R. Abou-Haidar. Partitioning crowded virtual environments. In *VRST '03: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 7–14, New York, NY, USA, 2003. ACM Press.

[139] P. Stone and M. Veloso. A layered approach to learning client behaviors in the RoboCup soccer server. *Applied Artificial Intelligence*, 12:165–188, 1998.

[140] D. Strömbom, R. P. Mann, A. M. Wilson, S. Hailes, A. J. Morton, D. J. Sumpter, and A. J. King. Solving the shepherding problem: heuristics for herding autonomous, interacting agents. *Journal of The Royal Society Interface*, 11(100):20140719, 2014.

[141] S. Stylianou, M. M. Fyrillas, and Y. Chrysanthou. Scalable pedestrian simulation for virtual cities. In *VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 65–72, New York, NY, USA, 2004. ACM Press.

[142] I. A. Sucan and L. E. Kavraki. Kinodynamic motion planning by interior-exterior cell exploration. In *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2008.

[143] A. Sud, R. Gayle, E. Andersen, S. Guy, M. Lin, and D. Manocha. Real-time navigation of independent agents using adaptive roadmaps. In *VRST '07: Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, pages 99–106, New York, NY, USA, 2007. ACM.

[144] J. C. Thierry Siméon, Jean-Paul Laumond and A. Sahbani. Manipulation planning with probabilistic roadmaps. *The International Journal of Robotics Research (IJRR)*, 23(7-8):729, 2004.

[145] A. Treuille, S. Cooper, and Z. Popović. Continuum crowds. *ACM Trans. Graph.*, 25(3):1160–1168, 2006.

[146] X. Tu and D. Terzopoulos. Artifical fishes: physics, locomotion, perception, behavior. In *Proceedings of ACM SIGGRAPH*, pages 43–50, 1994.

[147] A. E. Turgut, H. Çelikkanat, F. Gökçe, and E. Şahin. Self-organized flocking in mobile robot swarms. *Swarm Intelligence*, 2(2-4):97–120, 2008.

[148] J. van den Berg, P. Abbeel, and K. Goldberg. LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. In *Robotics: Science and Systems (RSS)*, Zaragoza, Spain, June 2010.

[149] J. van den Berg, P. Abbeel, and K. Goldberg. LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. *The International Journal of Robotics Research*, 30(7):895–913, 2011.

[150] D. N. A. van der Stappen and M. Overmars. Pushing a disk using compliance. *IEEE Transactions on Robotics and Automation*, 23(3):431–442, 2007.

[151] R. T. Vaughan, N. Sumpter, J. Henderson, A. Frost, and S. Cameron. Experiments in automatic flock control. *Journal of Robotics and Autonomous Systems*, 31:109–117, 2000.

[152] C. Vo, J. Harrison, and J.-M. Lien. Behavior-based motion planning for group control. In N. Papanikolopoulos, S. Sugano, S. Chiavernini, and M. Meng, editors, *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2009.

[153] C. Vo and J.-M. Lien. Following a large unpredictable group of targets among obstacles. In R. Boulic, Y. Chrysanthou, and T. Komura, editors, *Proceedings of the International Conference on Motion in Games (MiG)*. Springer, 2010.

[154] C. Vo and J.-M. Lien. Following a large unpredictable group of targets among obstacles. Technical report, George Mason University, Jan. 2010.

[155] C. Vo and J.-M. Lien. Following multiple unpredictable targets among obstacles. Poster: ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D 2010), 2010.

[156] C. Vo and J.-M. Lien. Visibility-based strategies for searching and tracking unpredictable coherent targets among known obstacles. Poster: Proceedings of the International Conference on Robotics and Automation (ICRA) Workshop: Search and Pursuit/Evasion in the PhysicalWorld: Efficiency, Scalability, and Guarantees, 2010.

[157] C. Vo and J.-M. Lien. Group following in monotonic tracking regions. In F. B. Atalay and E. Ezra, editors, *Proceedings of the 22nd Fall Workshop on Computational Geometry*, 2012.

[158] Z. Wang, M. Li, A. M. Khaleghi, D. Xu, A. Lobos, C. Vo, J.-M. Lien, J. Liu, and Y.-J. Son. Dddams-based crowd control via uavs and ugvs. In *Proceedings of the 2013 International Conference on Computational Science*, 2013.

[159] E. Weisel, F. McKenzie, Q. Nguyen, M. Petty, J. Camp, J. Anthony, and R. Albright. Crowd federate implementation for maneuver support simulation. In *Simulation Interoperability Workshop*, 2006.

[160] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1024–1031, 1999.

[161] M. Wilson and H. Neal. Diminishing returns of engineering effort in telerobotic systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 31(5):459–465, Sep 2001.

[162] A. Yamashita, T. Arai, J. Ota, and H. Asama. Motion planning of multiple mobile robots for cooperative manipulation and transportation. *IEEE Transactions on Robotics and Automation*, 19(2):223–237, 2003.

[163] H. Yeh, S. Curtis, S. Patil, J. van den Berg, D. Manocha, and M. Lin. Composite agents. In M. Gross and D. James, editors, *Proceedings of Eurographics / ACM SIGGRAPH Symposium on Computer Animation*, 2008.

[164] L. Zhang, S. LaValle, and D. Manocha. Global vector field computation for feedback motion planning. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 477–482, May 2009.

# Curriculum Vitae

Christopher Vo is a robotics and mechatronics specialist. He received his M.S. degree in Computer Science from George Mason University in 2007. His academic research is in the area of robustness, scalability, and performance for multi-agent motion planning algorithms and simulation, with the goal of enabling robots to perform complex motion and manipulation tasks in real-time.

## List of Peer-Reviewed Publications

1. Amirreza M. Khaleghi, Dong Xu, Sara Minaeian, Mingyang Li, Yifei Yuan, Christopher Vo, Arsalan Mousavian, Jyh-Ming Lien, Jian Liu, and Young-Jun Son. A Comparative Study of Control Architectures in UAV/UGV-based Surveillance System. *Proceedings of the Industrial and Systems Engineering Research Conference*, 2014.

2. Amirreza M. Kaleghi, Dong Xu, Sara Minaeian, Mingyang Li, Yifei Yuan, Jian Liu, Young-Jun Son, Christopher Vo, and Jyh-Ming Lien. A DDDAMS-Based UAV and UGV Team Formation Approach for Surveillance and Crowd Control. *Proceedings of the 2014 Winter Simulation Conference*, 2014.

3. Zhenrui Wang, Mingyang Li, Amirreza M. Khaleghi, Dong Xu, Alfonso Lobos, Christopher Vo, Jyh-Ming Lien, Jian Liu, and Young-Jun Son. DDDAMS-based Crowd Control via UAVs and UGVs. *Proceedings of the 2013 International Conference on Computational Science*, 2013.

4. Christopher Vo and Jyh-Ming Lien. Group Following in Monotonic Tracking Regions. *Proceedings of the 22nd Fall Workshop on Computational Geometry*, 2012.

5. Christopher Vo and Sam McKay and Nikhil Garg and Jyh-Ming Lien. Following a Group of Targets in Large Environments. *Proceedings of the Fifth International Conference on Motion in Games*. Springer, 2012, Invited Paper.

6. Brian Hrolenok, Keith Sullivan, Sean Luke and Christopher Vo. Collaborative Foraging Using Beacons. *Proceedings of the Ninth International Conference on Autonomous Agents and Systems (AAMAS 2010)*. 2010.

7. Joseph F. Harrison, Christopher Vo, Jyh-Ming Lien. Scalable and Robust Shepherding via Deformable Shapes. *Proceedings of the Third International Conference on Motion in Games*. Springer, 2010.

8. Christopher Vo and Jyh-Ming Lien. Following a Large Unpredictable Group of Targets Among Obstacles. *Proceedings of the Third International Conference on Motion in Games*, Springer, 2010.

9. Keith Sullivan, Christopher Vo, and Brian Hrolenok. RoboPatriots: George Mason University 2009 RoboCup Team. *Proceedings of 2009 RoboCup Workshop*. 2009.

10. Christopher Vo, Liviu Panait, and Sean Luke. Cooperative Coevolution and Univariate Estimation of Distribution Algorithms. *Proceedings of the 10th International Workshop on Foundations of Genetic Algorithms (FOGA 2009)*. ACM Press, 2009.

11. Christopher Vo, Joseph Harrison, and Jyh-Ming Lien. Behavior-Based Motion Planning for Group Control. *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009)*. IEEE, 2009.

12. Keith Sullivan, Christopher Vo, Sean Luke, and Jyh-Ming Lien. RoboPatriots: George Mason University 2010 RoboCup Team. *Proceedings of 2010 RoboCup Workshop*, 2009.