PHYSICS–BASED MULTIAGENT SWARMS AND THEIR APPLICATION TO COVERAGE PROBLEMS

by Wesley N. Kerr

A thesis submitted to the Department of Computer Science
and the Graduate School of the University of Wyoming
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE
in
COMPUTER SCIENCE

Laramie, Wyoming
December 2005

DEDICATION

To my grandmother and grandfather.

ACKNOWLEDGEMENTS

This project has consumed the past two years, and as any such project has had many different contributors. I thank my advisor Diana Spears for her helpful comments and for initially coming up with the idea for the base algorithms. Diana has contributed to this project in multitude of ways: she is responsible for the derivations and equations found in Chapter 5 as well as the driving force behind our publications throughout the project. Without Bill Spears help, I would have been lost when attempting to organize a paper of this size in LaTeX2e. Both of the Spears are responsible for the "physicomimetics" work in presented in later chapters. I would also like to thank my additional committee member Douglas Smith for his support. I would also like to thank David Thayer for suggesting we read about the kinetic theory of gases and co-authoring my first publication. Finally, I thank my parents for all of their encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

Chapter 1

Introduction

1.1 Introduction

The traditional view of robotics has been to develop one large expensive robot capable of achieving a task. Unfortunately this approach has many weaknesses. For autonomous control, what happens when this robot has an accident or stops responding? For missions in a local laboratory, this is not a problem, since the researcher can simply reset and rerun the test. Unfortunately if this robot runs into problems and is in a remote location, this is a huge problem. Issues like this and others have led robot researchers to consider large groups of simple coordinated robots. If the controllers are decentralized then even if robots fail the remainder can still accomplish the task.

Designing multiagent systems (swarms) as decentralized entities is a challenging research field. The idea is to program the behavior of an individual robot and then put many of these robots together and witness the emergent behavior. In this thesis, we focus on creating two new swarm controllers to accomplish coverage tasks.

1.2 Task

The coverage task being addressed is to sweep a large group of mobile robots through a long bounded region. This could be a swath of land, a corridor in a building, a city

sector, or an underground passageway/tunnel. The goal of the robots is to perform a search, requiring maximum coverage. This search might be for enemy mines (demining), survivors of a collapsed building or, alternatively, the robots might act as sentries by patrolling the area. All of these different examples require the robot to search the entire region to guarantee detection. The most researched coverage tasks include vacuuming and lawn mowing. In these coverage tasks the goal is the same: they both require maximum coverage.

We want simple, inexpensive robots, so we assume the robots have a limited sensing range for detecting other robots or objects, though all robots can sense the global direction to move (e.g., with light sensors). These robots need to avoid obstacles of any size, possibly the size of buildings. With limited sensors, robots on one side of large obstacles cannot visually/explicitly communicate with robots on the other side.

We assume that the robots need to keep moving because there are not enough of them to simultaneously view the entire length of the region. The robots in our task begin at one end of a corridor for example, the entrance of a subway tunnel and then move to the opposite end of the corridor (considered the "goal direction"). We define this movement as a *sweep*, and consider a sweep finished once all robots have reached the goal.

The primary objective of the robots is to maximize the coverage for one sweep, while the second objective is to minimize the sweep time. We believe gases are excellent at achieving this task. The following properties of gases convinced us to choose them as a model for our algorithms for this task: they are easily deformed, they are capable of rejoining after parting around an object, and gases fill volumes.

1.3 What is Coverage?

While we have defined our task in terms of coverage, we have not yet defined what coverage is. We are concerned with two primary forms of coverage: *spatial* coverage, and *temporal* coverage.

1.3.1 Spatial Coverage

There are two forms of spatial coverage: longitudinal (in the goal direction) and lateral (orthogonal to the goal direction). Longitudinal coverage can be achieved by moving the swarm as a whole and lateral coverage is achieved by a uniform distribution of the robots between the side walls of the corridor. Recall the corridor defined for the task. As robots move towards the goal they increase longitudinal coverage, and as robots move orthogonal to the goal, they increase lateral coverage.

1.3.2 Temporal Coverage

Temporal coverage is determined by the time spent performing a sweep of the corridor. The longer it takes the swarm to perform a sweep, the worse the temporal coverage. For example, the worse the coverage the easier an intruder can slip by undetected. Greater temporal coverage can be achieved by increasing the average agent speed.

1.4 Preview

In this chapter we discussed the task that we will reference throughout this thesis. In Chapter 2, we will explore alternative and state–of–the–art methods to solving our coverage task. Chapters 3 and 4 are devoted to developing our new control algorithms.

The next section examines the success of our new algorithms. In Chapter 5 we cover theoretical foundations of the algorithms presented in Chapters 3 and 4.

In Chapter 6, we provide the baseline and state–of–the–art algorithms to compare against the performance of our algorithms. Chapter 7 contains a summary of our work and future plans.

Chapter 2

Related Work

2.1 Introduction

The task presented in Chapter 1 has been studied extensively for many different applications. There is also extensive research attempting to derive different types of robotic controllers. The following chapter discusses different approaches adopted by researchers not only to solve coverage tasks, but also to control large groups of autonomous agents.

We break the chapter into the following sections. First we investigate behavior–based control, a popular approach to designing swarms. We then discuss the most relevant work, which utilizes *biomimetics* – or robots that mimic life. Following that is cell decompositions and then approaches utilizing physics methods for control. Finally, we look at a similar task – ad–hoc sensor networks, and different approaches to solving this related problem.

2.2 Behavior–based Swarm Control

Behavior–based swarm control algorithms draw their inspiration from biology. They involve developing a suite of behaviors – i.e., different methods to interact with the environment that are combined with other behaviors in a bottom up fashion. Different

behaviors can subsume other behaviors; i.e. obstacle avoidance can overtake a path planning behavior when an obstacle collision is eminent. How to combine behaviors effectively has attracted many researchers. Brooks (1986) defined this behavior–based control framework using a layered control system where behaviors at a higher level can subsume lower level behaviors. This architecture allows the user to add more behaviors without directly affecting the behaviors at a lower level. Arkin (1989) created a similar framework structure based on schemas and draws its inspiration more from the brain and psychology than biology.

Other research has progressed in determining what are the proper sets of behaviors to complete certain tasks (Matarić 1995). Goldberg and Matarić (2002) have shown that the behavior–based controllers are able to solve "collection" tasks very well. Swarms used to solve the collection tasks had to find objects and return them to a home base. Goldberg and Matarić develop a set of behaviors to solve the task and evaluated the controllers based on time and interference metrics. Other researchers used the behavior–based model for combining behaviors in order to generate robotic formations (Balch and Hybinette 2000; Fredslund and Matarić 2002; Balch and Arkin 1998). This is accomplished by adding a behavior that maintains the currently selected formation to the basis set of behaviors. Although this work shows that the formations generated by the robotic teams are able to navigate small obstacle courses, no results are presented indicating how coverage was affected by these formations.

Robotics researchers have explored many different algorithms that are able to solve search tasks where the environment is unknown to the robot. Balch and Arkin (1993) worked on a behavior–based approach for robots to avoid *box canyons*. Figure 2.1 shows a box canyon with several robots unable to reach the goal. Box canyons require backtracking in order to successfully continue to navigate to the goal. Box canyons were explored because they provide an attractive local minimum for goal-

oriented robots. Simulation results show that the controllers developed were successful at navigating out of box canyons. As an extension to Balch's and Arkin's work Ranganathan and Koenig (2003) provide a reactive on-line planner to the behavior–based work. They are able to experimentally show that the paths taken by the robot controlled with their architecture is shorter than the controller developed by Balch and Arkin. Lee and Arkin also improve on box-canyon avoidance by using a method described as "learning momentum." This approach allows the robot to modify system parameters at run time. Typically if the robot is successfully moving towards the goal, the parameters are left alone. As the robot encounters difficulty, it must then modify parameters in an attempt to rectify the situation.

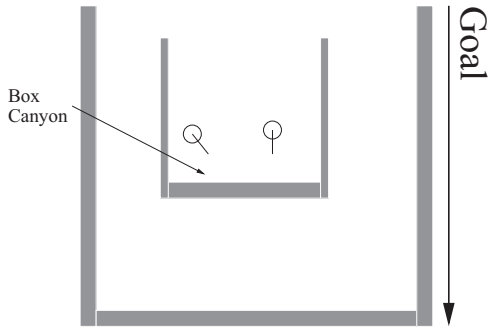


Figure 2.1: Box canyon.

Gage (1992) explored many different facets of multirobot coordination. He discusses different behaviors necessary for coverage, and even introduces the notion of a sweep behavior. This behavior is analogous to a row of soldiers forming a line from one side to the other of the search area. They then march in step to the goal, searching along the way. Although he argues that this achieves excellent coverage, he does not present experimental results. Other work by Gage (1993) explores random behaviors versus coordinated search for a de–mining task. He explores the idea of detecting mines with imperfect sensors, i.e. less than 100% probability of detection. Because of

this, any algorithm guaranteeing complete coverage cannot guarantee that all mines would be detected. He then shows that as the probability of detection decreases, so does the advantage of coordinated search.

Bruemmer *et al.* utilize the *potential fields* method in order to generate large–scale swarm formations (Bruemmer, Dudenhoeffer, McKay, and Anderson 2002; Dudenhoeffer, Bruemmer, Anderson, and McKay 2001) . Reif *et al.* (1999) first used potential fields as a control algorithm for swarms. The work is considered physics–based and we will discuss this more later, but essentially they used virtual forces to control the behavior of the swarm. Bruemmer *et al.* show that the potential field method alone is not an optimal algorithm for their spill finding task. They introduce ways to tune behaviors in order to be more successful via online learning. They also show that as the number of robots increase the coverage time decreases dramatically, but they find a crossover point where the work done by each robot also decreases once the search area has been saturated with robots. Although this work provides an algorithm able to perform their spill finding task, it provides no formal arguments of completeness, i.e. guarantees of complete coverage.

The work by Zelinsky *et al.* utilizes a map of the environment in order to determine the robots' desired paths (1993). The map provided of the environment divides the world into equally sized *cells*, or small regions of the environment. Zelinsky *et al.* define a new metric called *obstacle discomfort*, which is a measure of the discomfort of moving too close to an obstacle. The map then propagates a wave front which is a weighted sum of the distance and the obstacle discomfort. This measurement for each cell is called the "path transform." Since distance to the goal is included as part of the measurement, the values stored by the path transform will always decrease as the distance to the goal decreases. Utilizing this map, a robot can then traverse from a start cell to a goal cell, guaranteeing complete coverage. This is accomplished by

maintaining an internal representation of the world and what cells have been visited. Instead of following a path that leads directly to the goal, the robot follows a path visiting all cells that have a larger or equal path transform value. Once all cells with a larger or equal path transform value have been covered, the algorithm chooses a cell with a smaller path transform value and continues searching. This work has been demonstrated successfully on robots.

Another approach that requires a global map of the environment is given by de Carvalho *et al.* (1997). This work utilizes *a priori* information combined with *templates* in order to cover the area completely. Templates are sets of low–level behaviors that when combined perform a specific higher–level behavior, i.e. combining a forward behavior with a U-turn behavior to generate a sweeping behavior. The work describes the new sets of templates and how they can combine these templates in order to get complete coverage. The work described is also able to deal with small dynamic obstacles at run time. Finally, the work by Bayazit *et al.* (2002) attempts to combine both behavior–based models and global road maps. This work successfully navigates the world, but assumes global information known *a priori* and does not discuss the coverage aspect of the algorithm.

2.3 Biomimetic Swarm Control

As an extension to behavior–based approaches, biomimetic control more closely resembles biological creatures and communication techniques. Examples of this design principle include finding router policies for network traffic based on ant foraging. By leaving pheromone trails ants are able to determine the shortest paths to food sources. Routing software takes advantage of this fact in order to route traffic as quickly as possible. The biomimetic research that solves the task of coverage has also been based

on the behaviors of ants.

Koenig *et al.* (2001) have studied this problem for several years. The goal of their research is to design an ant robot capable of leaving short term traces on the environment also known as *stigmergy*. These traces then mark where the ant has been and inform itself and other ants that that area has been covered. By using this method, the robots need only limited computational resources and limited sensing capabilities. Koenig *et al.* have investigated different value update rules and created theoretical results showing that the ants are capable of achieving complete coverage. These results utilize proofs similar to those of A* (Koenig and Liu 2001; Koenig, Szymanski, and Liu 2001). Other experiments show that the results scale as more robots are added, complete coverage occurs even when robots are moved to new areas without realizing it, and coverage is achieved in time polynomial in the number of cells. Finally, these results have been strengthened with demonstrations shown on actual robots (Svennebring and Koenig 2003; Svennebring and Koenig 2002).

Work done by Wagner and Bruckstein (1995, 1998, 1999) is very similar to that of Koenig *et al.*. They both explore the usefulness of pheromone traces for ant-robots exploring an unknown environment. The algorithm presented in (Wagner and Bruckstein 1995) is then extended in (Wagner, Lindenbaum, and Bruckstein 1996). Here Wagner *et al.* present several new algorithms based on ant pheromone trails. Both algorithms divide the world into discrete cells. Upon executing the algorithm, the robot determines its next move by selecting the cell least visited from its surrounding neighbors. During the move, the robot deposits pheromone in the position it came from. An algorithm is presented that extends the basic algorithm by removing redundant exploration, based on a multilevel depth-first search. If the robot is presented with a section that is completely explored, it will back itself out along the same entrance route until it finds unexplored areas. Wagner *et al.* also

show that a single ant can cover a graph in time $O(nd)$ where n is the number of vertices and d is the diameter of the graph.

Vaughan *et al.* (2000) created a robotic controller based on the trail following behavior of ants and the waggle dance of honey bees. Rather than leaving pheromone trails explicitly as other research has done, this research seeks to pass this information via a shared *localization space*, i.e. a global map. This shared space can be achieved, for example, with the Global Positioning System (GPS). The robots' task is to navigate an environment from a start position to a goal position. The robots are transferring objects from the goal to the start positions. The performance metric is how many objects can be moved in a predetermined time. The ant algorithm uses pheromone trails in the shared space, but only updates this shared space when the goal is found. This updating resembles the waggle dance of honey bees. Experimental results show that given difficult obstacle courses where gradient searching fails, this robotic control algorithm performs well.

Later we will compare our new physics–based control algorithm with that of Koenig *et al.*. The primary difference between the work presented above and the work to follow is that all information is passed between robots by leaving traces on the environment. In the work by Vaughan *et al.* this trace is not explicit, but it does exist. We attempt to design a control algorithm that does not need such traces in order to generate sufficient coverage.

2.4 Cell Decompositions

Another classic approach to solving the coverage problem utilizes different decompositions of the corridor. In these decompositions, space is separated into cells that can be covered by sweeping back and forth. Once the world has been decomposed into

cells the robots just need to visit each cell and perform the sweep of that cell. An example of the decomposition method can be seen in Figure 2.2. By creating simple cell decompositions many of these approaches guarantee complete coverage. We will explore each of the decompositions in more detail.

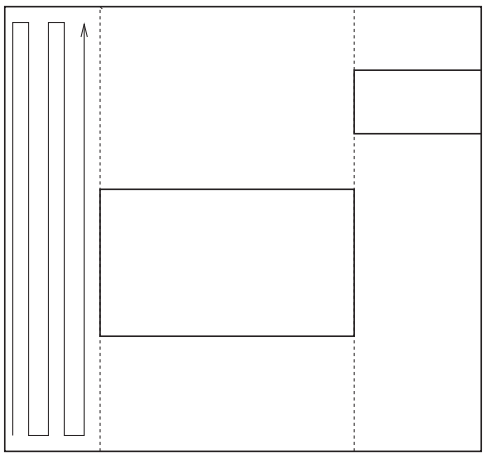


Figure 2.2: Decomposition schematic.

2.4.1 Trapezoidal Decomposition

One popular decomposition is the *trapezoidal decomposition* (Latombe 1991) (also known as the *slab method* (Preparata and Shamos 1985)). This process works by sweeping a line across the corridor. When this line encounters an obstacle boundary, a cell boundary is created. Upon completion of this method we are left with cells that are either trapezoidal or triangular as long as the obstacles within the corridor are also polygonal. In order to cover the decomposition, they create an adjacency relation between the cells.

2.4.2 Boustrophedon Decomposition

Probably the most popular decomposition, the *boustrophedon decomposition* is an extension to the trapezoidal decomposition. The benefit of the boustrophedon decomposition is that it requires fewer cells than the trapezoidal decomposition and doesn't require the obstacles in the environment to be polygonal. Choset and Pignon first explored the boustrophedon cellular decomposition (Choset and Pignon 1997; Choset 2000). By merging cells from the trapezoidal decomposition this decomposition reduces the number of back and forth sweeping motions needed. Once the robot created the decomposition, it can then plan a path to guarantee complete coverage.

Because of the need for *a priori* information, Rekleitis et al. (2004) expanded this decomposition to multi-robot team coverage. By utilizing a multi-robot approach they are able to construct a decomposition while performing sensor based coverage (see Fig. 2.3). They even restrict the robots to line of sight communication. Using this approach the team is able to guarantee complete coverage of the environment while also guaranteeing minimum times an area is covered.

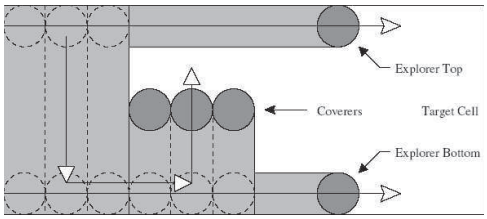


Figure 2.3: Multi-Robot team decomposition coverage. Figure source (Rekleitis, Lee, Shue, New, and Choset 2004).

2.4.3 Slice Decomposition

Another decomposition called a "slice decomposition" was created by Wong and MacDonald (2004). This decomposition is an online algorithm in which obstacles can be

either polygonal or curved. Sweep lines slice the world into segments, which are then used to determine critical points. This decomposition algorithm maintains a list of active obstacle and free space cells. Robots update the list when they encounter critical points. In order to guarantee coverage the robot must visit all of the free space cells.

2.4.4 Rectilinear Decomposition

Butler *et al.* (2000, 1999), show that the world can also be decomposed into rectilinear environments by incrementally constructing the decomposition of the environment. Rules are generated to handle "interesting" (*critical*) points, which are the x-values that determine the vertical boundary segments. The critical points and decomposition pattern can be seen in Fig. 2.4. They are then able to guarantee complete coverage by generating a finite state machine representing all the ways the environment can evolve. By showing that this FSM does not have any infinite loops and only terminates when it has attained complete coverage, Butler *et al.* are able to guarantee coverage. They are also able to guarantee complete coverage when using multiple cooperative robots (Butler, Rizzi, and Hollis 1999).

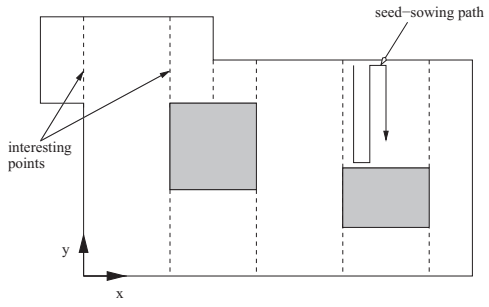


Figure 2.4: Rectilinear decomposition. The interesting points and cell coverage algorithm are highlighted. Figure source (Butler, Rizzi, and Hollis 1999).

2.4.5 Other Decompositions

Choset et al. (2000) show that there are many different compositions that yield complete coverage besides the boustrophedon and trapezoidal decompositions. They consider different shapes of slices instead of direct lines. By examining different slices they are able to achieve different tasks and ensure provable guarantees.

Wong and MacDonald (2003) present an online method of complete coverage by using a topological map of the world as opposed to a spatial map. The algorithm is implemented as a finite state machine with three states. The states are normal, boundary, and travel. The algorithm updates a topological map as it moves through the world. By using this topological map the robot requires much less storage space and can still guarantee coverage.

Seop Oh *et al.* (2004) propose a triangular cell based map. This approach guarantees complete coverage by building a map of the corridor as the robot explores the environment. This is accomplished by combining *template programming* (i.e. different behaviors based on sensor readings) and a triangular-cell-based decomposition of the world. While maintaining the minimum distances to uncovered area for each cell during exploration.

Most of these solutions require the robots to maintain an internal representation as well as perform localization. Some approaches go so far as to require the structure of the corridor *a priori*. We do not require any of these robot constraints with our approach, since it is assumed that they have limited sensors and *computational power*.

2.5 Physicomimetic Swarm Control

Another reactive robotic swarm strategy rooted in physics (as opposed to biology) has been termed "physicomimetics" and provides not only excellent control of swarms,

but also theoretical foundations for predicting the behavior of the swarms. Two different lines of related research use physics–based force laws to control swarms of agents (Reif and Wang 1999; Spears and Gordon 1999; Spears, Gordon-Spears, Hamann, and Heil 2004). Our work is an extension of the framework provided by Spears *et al.* This framework is discussed more thoroughly in Chapter 3. Other research based on physicomimetics determines the proper settings for a swarm to maintain its formation and navigate an obstacle course (Hettiarachchi and Spears 2005).

Other physics–based approaches to modeling fluid behavior besides our own are extremely rare. One exception is the work by Decuyper and Keymeulen (1991). Their work shows that the fluid model is capable of controlling robots. They use a path generation algorithm taken from fluid dynamics, and is described via a factory floor. The floor space is divided into grid cells and either is fluid or contains a piece of the wall. It models at the start position a pump and at the goal an outlet. Fluid particles are then pumped through the system until stability is reached. At this point the robots can do a gradient descent in order to reach the goal by way of a short path. This approach needs the global map and has to be updated real time when changes occur in the building. This is acceptable for a factory environment, but we would like a robot control algorithm that is able to cover unknown environments.

Jantz *et al.* worked with universal metrics for robotics (Jantz and Doty 1997; Jantz 1998), using behaviors from statistical mechanics to mimic ideal gas laws. They then use fluid theory in order to make predictions about the behavior of the swarm and the effusion of robots through a small opening, mean collision time, and the collision frequency of the robots. This work is in response to the need for evaluation tools for swarm robotics, and is very similar to our work, but our work extends the results found by Jantz *et al.* and applies more realistic kinetic theory behaviors to the robots.

2.6 Sensor Networks

Other work in this area deals with creating ad hoc sensor networks that achieve complete coverage. Batalin and Sukhatme (2002) created control algorithms to maximize coverage by increasing inter-agent distances. Each robot moves away from the other robots until coverage is maximized. Using this approach each robot maximizes the space covered by its sensors, but no results are shown when there are not enough robots to cover the entire environment.

The work by Howard *et al.* describes an incremental approach to maximize coverage by a mobile sensor network (2002). As robots are introduced into the environment, they attempt to maximize placement by utilizing the information from prior robots and also are guided by heuristics. The work is successful at maximizing coverage, but again assumes that there are enough robots to cover the area.

Megerian *et al.*(2005) utilize theoretical graph proofs to guarantee complete coverage for several approaches. Other research looks at the network sensor coverage problems (Cortes, Martinez, Karatas, and Francesco 2002; Horling, Vincent, Mailer, and Shen 2001; Kadrovach and Lamont 2002). These problems also implicitly assume enough robots for complete coverage without sweeping.

2.7 Conclusion

We have described a multitude of different approaches to the coverage problem presented in Chapter 1. Each had limitations that our new algorithms will attempt to overcome. The work in behavior–based was mostly heuristics or required global knowledge known *a priori*. The heuristic approaches suffer because they cannot guarantee anything and global knowledge approaches cannot adapt well to unknown environments. The biomimetic approaches required robots with the ability to leave

traces on the environment. These approaches suffer from sabotage by unfriendly robots because the traces left on the environment can be detected and altered by any robot with the sensors. This approach also requires special sensors that may or may not currently exist. Although decomposition methods are able to guarantee complete coverage, only one algorithm addresses the need for dynamic, unknown environments. This approach requires heterogeneous sets of robots and the robots must maintain an internal representation of the world. This also requires the robots to perform localization which is an expensive and challenging computation. Our work is an extension of the physicomimetic research. Finally, the sensor network research implicitly assumes that there are enough robots to cover the the area. We do not assume sufficient numbers of robots, in fact, our work requires a sweeping behavior since there are not enough robots. The research based on the artificial physics approach has primarily been concerned with structured formations, whereas our work deals with a gas behavior. In fact, our work requires a sweeping behavior since there are not enough robots.

Chapter 3

Artificial Physics

3.1 Introduction

Spears and Gordon (1999) created an "artificial physics" (AP) framework that controls groups of autonomous agents or robots through local interaction only. Using simple and elegant formulas they are able to demonstrate that AP is predictable. Utilizing these same ideas we set out to design a multiagent control algorithm that was able to solve the coverage tasks described in Chapter 1. Before we discuss our new algorithm, we first provide the background of AP and how it works.

3.2 Artificial Physics

Spears and Gordon (Spears and Gordon 1999; Spears, Gordon-Spears, Hamann, and Heil 2004) have provided a technique called *physicomimetics* (artificial physics) for controlling large groups of agents (modeled as particles), using virtual physics-based forces to move the agents into a desired formation, e.g., a hexagonal lattice. This technique scales well to large groups of agents and uses only local interactions. Using physicomimetics, agent swarms do well at staying in formation and avoiding obstacles, without the need for active communication, long-range sensing, or prespecified roles (Spears, Gordon-Spears, Hamann, and Heil 2004). Nevertheless, a problem still

exists when the agents encounter a very *large* obstacle, e.g., a building in a city. As the agents move around the obstacle, they are unable to detect the agents moving on the other side of the obstacle. Because of this, they are never able to regroup, and they leave an exposed and uncovered area downstream of the obstacle (see Fig. 3.1). The problem is that physicomimetics has traditionally been run in a mode that mimics the behavior of a crystalline solid. Yet solids are rigid and do not expand to fill/cover a region. This is the reason for investigating a gas approach to physicomimetics.

3.3 AP Solid

Creating a robotic control algorithm capable of mimicking a crystalline solid is useful in that the ordered structure is important for distributed sensing. Agents are able to act as a large distributed sensing network that is fault-tolerant, scalable, self-organizing, and capable of self-repair. This section describes how we are able to create formations resembling crystalline solids.

In AP, agents are controlled via forces. These forces, which are based on Newtonian physics, do not really exist in a physical sense, but the agents react to them as if they were real. The system acts as a molecular dynamics ($\vec{F} = m\vec{a}$) simulation. In the original AP system, Newton's gravitational force was used as the force law. Therefore forces are calculated using

$$F = \frac{Gm_1m_2}{r^p}$$

Where (m_1, m_2) are the masses of the agents, G is a user defined constant, r is the distance between the agents, and p is a user-defined parameter.

Each agent is described by a position vector $\vec{x}$ and a velocity vector $\vec{v}$. Time is maintained with the scalar variable t. The simulation can be run in either 2D

or 3D (to model swarms of micro-air vehicles). Agents in the system update their position, $\vec{x}$, in discrete time steps, Δt. At each time step, each agent updates its velocity, $\vec{v}$, based on the vector sum (resultant) of all forces exerted on it by the environment, which includes other agents within visibility range. This velocity, $\vec{v}$, determines $\Delta\vec{x}$, i.e. the next move of the agent. In particular, at each time step, the position of each particle undergoes a perturbation $\Delta\vec{x}$. This perturbation depends on the current velocity, i.e. $\Delta\vec{x} = \vec{v}\Delta t$. The velocity of each particle at each time step also changes by $\Delta\vec{v}$. The change in velocity is controlled by the force on the particle, i.e. $\Delta\vec{v} = \vec{F}\Delta t/m$, where m is the mass of that particle and $\vec{F}$ is the force on that particle. Note that this is the standard, Newtonian $\vec{F} = m\vec{a}$ equation. The goal of AP is reducing the potential energy of a system.

By setting system parameters in AP, we can mimic solid, liquid, or gas states, as well as phase transitions between these states (Gordon-Spears and Spears 2002). To generate a hexagonal lattice with ordered structure, each agent in the system experiences a repulsive force from other agents that are too close, and an attractive force from other agents that are too far away. Spears explains this in (Spears, Gordon-Spears, Hamann, and Heil 2004):

> At first blush, creating hexagons appears to be somewhat complicated, requiring sensors that can calculate distance, the number of neighbors, their angles, ets. However, only distance and bearing information is required. To understand this, recall an old high-school geometry lesson in which six circles of radius R can be drawn on the perimeter of a central circle of radius R. Figure 3.2 illustrates this construction. If the particles (shown as small circular spots) are deposited at the intersections of the circles they form a hexagon with a particle in the middle.

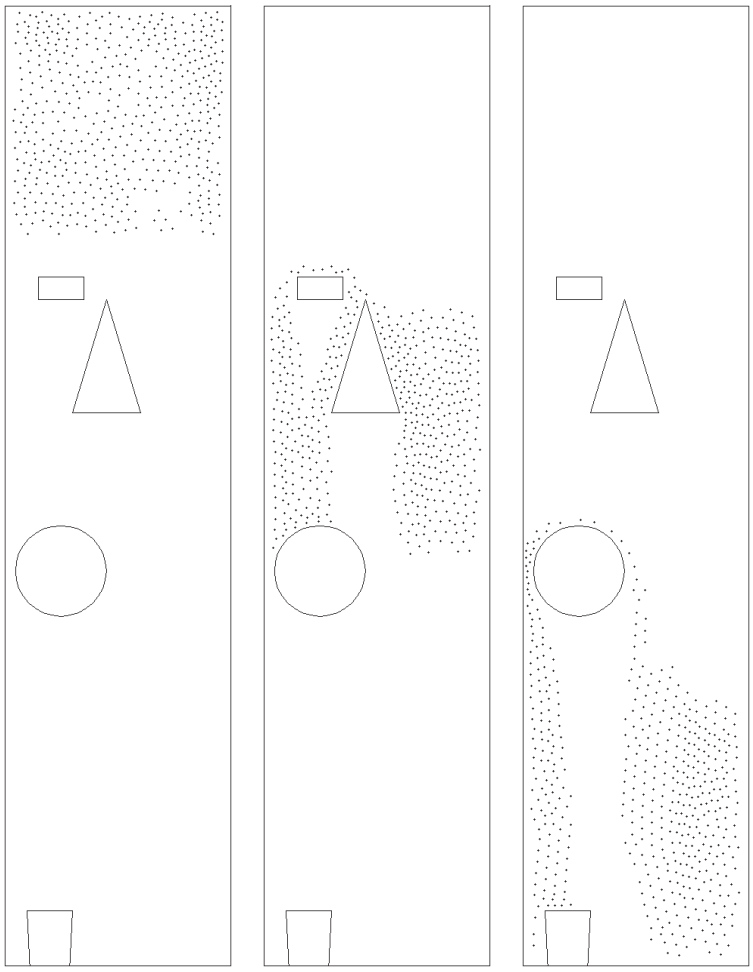


Figure 3.1: AP in solid form performs a sweep.

This result is then mapped to a force law, using R as the desired distances. Particles repel other particles closer than R and attract particles farther than R. Exploring the force law for AP, one notices that as the agents get close together, the force experienced grows incredibly fast. In order to counter this, the force is scaled to a user defined parameter F_{max}.

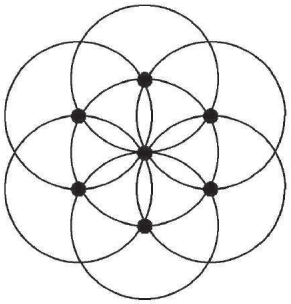


Figure 3.2: How circles can create hexagons.

3.4 AP Gas

Converting the AP crystalline control algorithm to an algorithm that mimics gas behavior involves converting forces. For AP solid there is a desired radius that allows that agent to attract other agents that are farther and repulse other agents that are closer. Recall Figure 3.2. When creating the AP gas algorithm we disabled any forces that are attractive. Most gas research has determined that the hard-sphere model works really well when modeling a gas. This hard sphere model provides agents with collisions when they get too close, much like the repulsive force used in AP gas. Again we only allow the force to get as large as F_{max}. Besides being repulsed from all other agents within its sphere of influence, the agent is repulsed from all obstacles and boundaries. Anything that can be sensed by the agent exerts a repulsive force on the agent.

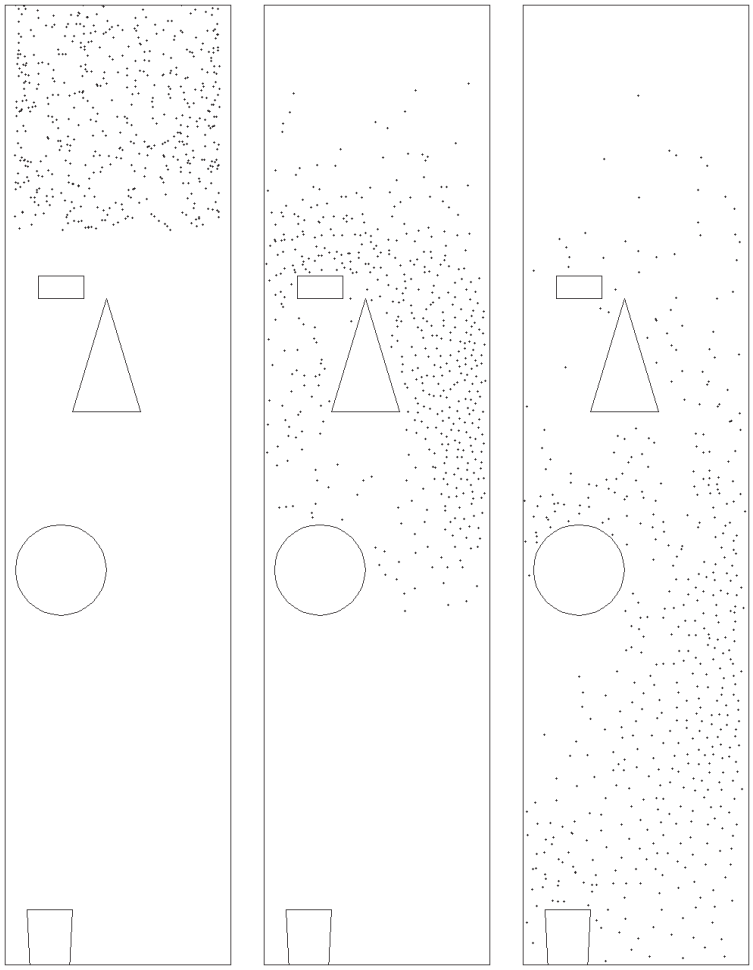


Figure 3.3: AP in gas form performs a sweep.

As discussed previously, this type of gas coverage works well for distributed sensor coverage. In order to achieve good coverage on a corridor, we need to add one force that is attractive. This force comes from the goal direction. The strength of the goal force is left constant for all of our experimental data.

3.5 Robot Port

Simulated robots as seen in Figure 3.3 share a known coordinate system and have perfect sensing. In order to overcome this, we ported the particle algorithm defined in the previous simulator to a robot–faithful simulator. In this simulator each robot contains a ring of 24 sonars positioned every 15°. Forces are then determined from the sonar readings. The goal sensor provides a bearing to the goal, and is also able to determine if the goal has been reached. This is analogous to a light sensor determining it is at the most intense point of light. This new algorithm can be seen in Fig. 3.4.

```
float distance, turn, vx, vy;
float delta_t = 1.0f;
float VMAX = 5.0f;
float FMAX = 1.5f;
float G = 240.0f;
float R = 24.0f;
float goalForce = 0.9f;

void move()
    float[] force = findForce();
    float[] goal = goalForce();
    float sum_fx = force[0] + goal[0];
    float sum_fy = force[1] + goal[1];
    float deltavx = delta_t * sum_fx;
    float deltavy = delta_t * sum_fy;
    vx = distance + deltavx;
    vy = deltavy;
    float deltax = vx * delta_t;
    float deltay = vy * delta_t;
    distance = sqrt(deltax*deltax + deltay*deltay);
    if (distance > VMAX) distance = VMAX;
    turn = atan2(deltay, deltax);

float[] findForce()
    Read Sonar Sensors
    float[] distances = sonar.getDistances();
    float[] bearings = sonar.getBearings();
    for i=0 to Number of Sonar
        float theta = bearings[0];
        float r = distances[0];
        float F = 0;
        if (r <= R)
            F = -G / (r*r);
            if (F < -FMAX) F = -FMAX;
        sum[0] += F*cos(theta);
        sum[1] += F*sin(theta);
    return sum;

float[] goalforce()
    Read Goal Sensor (Light Sensor)
    float bearing = sensor.getBearing();
    force[0] = goalForce*cos(bearing);
    force[1] = goalForce*sin(bearing);
    return force;
```

Figure 3.4: AP gas psuedocode.

Chapter 4

Kinetic Theory

4.1 Introduction

Chapter 2 provided the background needed to understand why we are looking to develop a new control algorithm. Most of the previous research focused on solving coverage tasks primarily used *stigmergy* in order to make predictions about the success of each algorithm. By allowing the robots to modify the environment several algorithms were able to guarantee complete coverage. Other attempts to solve the coverage tasks were mostly heuristic and provided no formal guarantees about the system.

The question remains though as to whether we can design an algorithm that is able to achieve good coverage without *stigmergy* and still has formal assurances. Stigmergy is undesirable since the traces left by those algorithms can be duplicated and used to deceive the algorithms. Chapter 3 then provided an introduction to "physicomimetics" and the benefits of designing distributed agent control algorithms with a solid physics foundation. This chapter discusses our new algorithm based on the physics of fluids.

4.2 Fluid Physics

Both liquids and gases are considered fluids, but this research focuses on gases. Gases offer excellent coverage, unpredictability of particle locations, and they can be bounded. In general, fluids (gases and liquids) are able to take the shape of their container and therefore are well suited to avoiding obstacles. Fluids are also capable of squeezing through narrow passages and then resuming full coverage when the passage expands. With gases, if we model a container, the gas will eventually diffuse throughout the container until it reaches an asymptotic state. Because gases have this property but liquids do not, gases are a more natural way to think of how to get particles around an obstacle, and why we chose to model a gas. Once the particles have moved around an obstacle, fluids have the ability to regroup. For example, consider releasing a gas from a container at the top of a room with obstacles. The gas inside the container is slightly heavier than the surrounding air. As the gas slowly falls to the ground, it separates around obstacles and diffuses - by either laminar or turbulent processes - to cover areas under the obstacles.

Agents capable of mimicking fluid flow will be successful at avoiding obstacles and moving around them quickly. By mimicking gas flow in particular, the agents will be able to distribute themselves throughout the volume once they have navigated around the obstacle, which will also increase coverage.

4.3 Kinetic Theory

There are two main methods for modeling fluids: the Eulerian approach, which models the fluid from the perspective of a finite volume fixed in space through which the fluid flows (typically the method of computational fluid dynamics), and the Lagrangian approach, in which the frame of reference moves with the fluid volume (typically the

kinetic theory approach) (Anderson 1995). Because we are constructing a model from the perspective of the agents, we choose the latter. Kinetic theory (KT) is typically applied to plasmas or gases, and here we model a gas. This overview of KT borrows heavily from Garcia (2000).

When modeling a gas, the number of particles is problematic, i.e., in a gas at standard temperature and pressure there are 2.687×10^{19} particles in a cubic centimeter. Utilizing today's computational power, we are unable to model a gas deterministically. Therefore, a typical solution is to employ a stochastic model that calculates and updates the probabilities of the positions and velocities of the particles, also known as a Monte Carlo simulation. This is the basis of KT. One advantage of this model is that it enables us to make stochastic predictions, such as the average behavior of the ensemble. The second advantage is that with real robots, we can implement this with probabilistic robot actions, thereby avoiding predictability of the individual agent.

KT has a rich history of research and any attempt to cover all aspects of KT is too ambitious. In response to this, we only touch on the areas of KT relevant to our research. In KT, particles are treated as possessing no potential energy (i.e., an ideal gas), and collisions with other particles are modeled as purely elastic collisions that maintain conservation of momentum.

Using some of the formulas for kinetic theory, we can obtain useful properties of the system. If we allow k to be Boltzmann's constant, such that $k = 1.38 \times 10^{-23}$ J/K, m to be the mass of the particle, T to be the temperature of the system, and $v = |\vec{v}|$, then we can define the average speed of any given particle (in 3D) as,

$$\langle v \rangle = \int_0^\infty v f(v) dv = \frac{2\sqrt{2}}{\sqrt{\pi}} \sqrt{\frac{kT}{m}}$$

where $f(v)$ is the probability density function for speed.

Another property we can define for KT is the average kinetic energy of the particles:

$$\langle K \rangle = \langle \frac{1}{2} m v^2 \rangle = \frac{3}{2} kT$$

4.4 Algorithm

A useful benefit of fluids is their ability to flow. By modeling our multiagent control algorithm as a fluid we are able to model different types of fluid flow. For the design of our algorithm we assume that we will treat the particles as being in a 2D Couette flow. The original algorithm for this one-sided Couette flow was provided by Garcia (2000). Figure 4.1 shows a schematic for this one-sided Couette flow, where we have a fluid moving between two walls – one wall moving with velocity v_{wall}, and the other stationary. Because the fluid is a Newtonian fluid and has viscosity, we see a linear velocity profile across the system. Fluid deformation occurs because of the shear stress τ, and the wall velocity is transferred because of molecular friction on the particles that strike the wall. On the other hand, the particles that strike the non-moving wall will transfer some of their velocity to it. This does not cause the wall to move, since in a Couette flow the walls are assumed to have infinite length and depth and therefore infinite mass. We chose a Couette flow so that we can introduce energy into the system and give the particles a direction to move. We created a 2D simulation

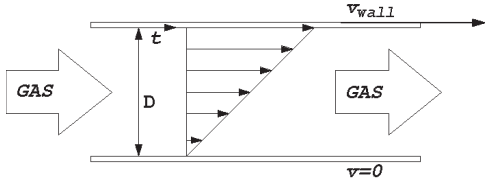


Figure 4.1: Schematic for a Couette flow.

world with a pair of corridor walls (which can be considered Couette walls), obstacles,

and agents (modeled as gas particles). The fluid flow is unsteady with no turbulence, i.e., an unsteady laminar flow.

Our KT approach models a modified (two-sided) Couette flow in which both Couette walls are moving in the same direction with the same speed. We invented this variant as a means of propelling all agents in a desired general direction, i.e., the large-scale fluid motion becomes that of the walls. Particle velocities start randomly and remain constant, unless collisions occur. (Note that with actual robots, collisions would be virtual, i.e., they would be considered to occur when the agents get too close. Wall motion would also be virtual.) The system updates the world in discrete time steps, Δt. We choose these time steps to occur on the order of the mean collision time for any given agent. Each agent can be described by a position vector $\vec{x}$ and a velocity vector $\vec{v}$. At each time step, the position of every agent is reset based on how far it could move in the given time step and its current velocity:

$$\vec{x} \leftarrow \vec{x} + \vec{v}\Delta t \ .$$

When updating the position, a check is performed to see if the movement would cause an agent-wall collision. If a collision would occur, then the agent's velocity is reset and its new position is determined. The agent's velocity is sampled from a biased Maxwellian distribution in the x-direction $\sqrt{-\log(1-U)}$ where U is a uniformly random sample. The agent's velocity is sampled from a Gaussian distribution in the y-direction (Garcia 2000). The agent's new position is determined based on where the agent would collide with the wall and how far the agent would have been able to move if the wall were not there. If the agent is about to strike a moving wall, then some of the energy from the wall is transferred to the agent. Once the agent has updated its position, the inter-agent collisions are then processed. The number of

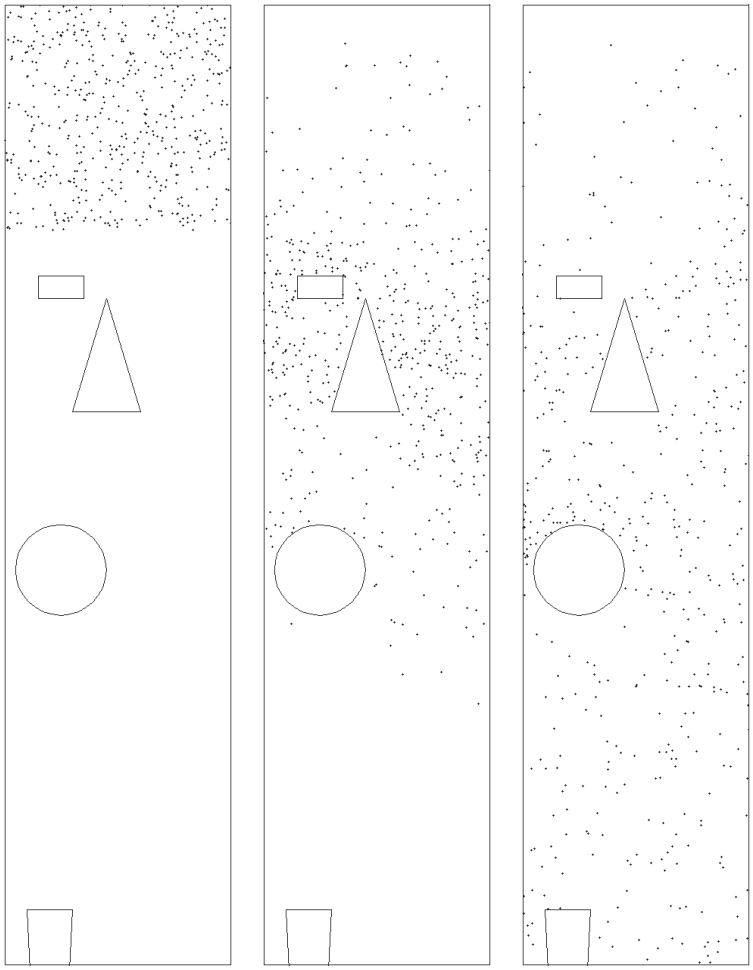


Figure 4.2: KT controllers perform a sweep.

collisions in any given region is a stochastic function of the number of agents in that region (Garcia 2000). This process continues until a desired state is achieved. Note that our algorithm performs a bucket sorting routine, which is not directly applicable to actual robots in the real world. Therefore, a variant of this algorithm is needed that is more robot-faithful.

4.5 Robot Port

In order to realize the potential of the KT algorithm, we decided to convert the original, shared-axes simulated algorithm into a purely local algorithm capable of being ported to robots. Our new KT robot algorithm is a distributed algorithm that allows each robot to act independently of the other robots. At each time step, t, every agent needs to determine an angle θ_t to turn, and a distance s_t to move. Each of these values derives from two types of collisions. Note that *all collisions are virtual.* The first set of collisions processed is those with walls. When this is complete, the robot has a desired velocity vector. The second set of collisions is between robots, which results in another velocity vector. If collisions do not occur, the robot will maintain its current heading and speed. Otherwise, the robot will combine the two new velocity vectors into one final resultant velocity vector. From this final vector, the robot is able to determine its new heading, θ_t, and speed, s_t.

All sensor data is collected before decision-making. The goal direction is the pivotal piece of information and is needed for solving the task. Therefore, the bearing to the goal is sensed first. If the robots are presently at the goal (e.g., based on light intensity), then they have completed the task. Otherwise, the robots sense other information, such as whether a collision is imminent. A robot processes wall collisions as long as there is not another robot currently along its heading. Robot collisions are processed for every time step. Figure 4.3 contains the pseudo code for the primary algorithm. Following this section, we describe wall collisions first, then inter-robot collisions.

```
float distance, turn, vx, vy;
float mpv = 1.11777f; (T=0.0030)
float stdev = 0.79038f;
float wall = 2.0f;
float safe = 15.0f;
float VMAX = 5.0f;
boolean[] col;
int[] timeCounter;

void move()
    vx = distance; vy = 0;
    incrementCollisionCounters();
    Read Goal Sensor (Light Sensor)
    float bearing = sensor.getBearing();
    Read Trilaterative Sensor
    int[] robotIds = trilaterative.getRobotIds();
    float[] robotDistances = trilaterative.getDistances();
    float[] robotBearings = trilaterative.getBearings();
    Read Sonar Sensors
    float[] sonarDistances = sonar.getDistances();
    float[] sonarBearings = sonar.getBearings();
    if ( !robotDirectlyInFront() )
        updateForWalls(bearing, sonarBearings, sonarDistances);
    updateForRobots(robotIds, robotDistances, robotBearings);
    distance = sqrt(vx*vx+ vy*vy);
    if (distance > VMAX) distance = VMAX;
    turn = atan2(deltay, deltax);
```

Figure 4.3: KT base pseudocode.

4.5.1 Wall Collisions

When (virtually) colliding with a wall, a robot must determine its post-collision velocity vector. This new velocity depends on the speed and orientation of the wall with which the robot collided. A Couette wall is assumed to be in motion. Because robots cannot distinguish Couette walls from obstacle walls that are parallel to them, they assume that *any* wall parallel to the Couette walls (also parallel to the goal direction) is a wall in motion. Therefore, the first portion of our algorithm addressed here is the determination of the angle of the wall that the robot has collided with, in relation to the goal direction. If the wall is determined to be approximately parallel to the goal direction, then the robot assumes that this wall is in motion. For simplicity, in this subsection we call any wall that is parallel to the goal direction, and is therefore virtually considered to be in motion, a "Couette wall." Couette wall velocity, of course, affects the new velocity that is adopted by the robot after its collision. Furthermore, in the original KT algorithm, robots reset their velocities based on a shared coordinate system. For our new algorithm, each velocity is converted to a local coordinate system with a shared point of interest, namely the goal.

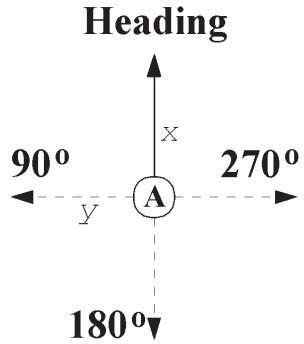


Figure 4.4: Diagram of robot.

For the following description of our algorithm, assume that robot A is the current

robot doing the processing. In order to determine if A has collided with a wall, it checks the values of three of its sensors. These primary sensors are along the robot's current heading and orthogonal to the heading as seen in Fig. 4.4. The robot will process a collision with a wall if any of these sensors detect less than a predefined safe distance. We require that the robot check the orthogonal sensors as a safety precaution – to prevent the robot from moving almost parallel to a wall and clipping that wall (because the front sensor never processes the collision). We chose this approach as opposed to increasing the safe distance for the heading sensor. Fig. 4.5 shows a collision with a wall occurring.

To determine if a wall is a Couette wall, the robot has to find the orientation of the wall with respect to the goal direction. For this calculation the robot is only concerned with the sonar sensors between $-45°$ to $45°$, since this is the direction that the robot is facing (see Fig. 4.5). First, the robot identifies the *essential endpoints*, which allow the robot to determine the corresponding plane of the wall. The essential endpoints are the endpoints of the wall as defined by the robot and will be used to determine the orientation of the wall. To determine the first of these two endpoints, A begins at $45°$ and moves through the sensors looking for a reading that is closer than the maximum range for the sonar sensors. The second endpoint moves positively through the sensors beginning at $-45°$. The essential endpoints are at $45°$ and $-45°$ in Fig. 4.5. The wall is then defined as a line between these two endpoints. Although this process of determining the corresponding line is error prone, it performs quite well in simulation.

Now that the robot has identified two endpoints for the wall, it creates a virtual line through these endpoints. When determining the orientation of the wall, A intersects this line with a second virtual line from itself to the goal (the direction to the goal is sensed). If the lines do not intersect, then we know that the wall must be parallel,

or at least close to parallel, to the goal direction. Otherwise, since our orientation algorithm is imperfect, A must calculate the intersection angle.

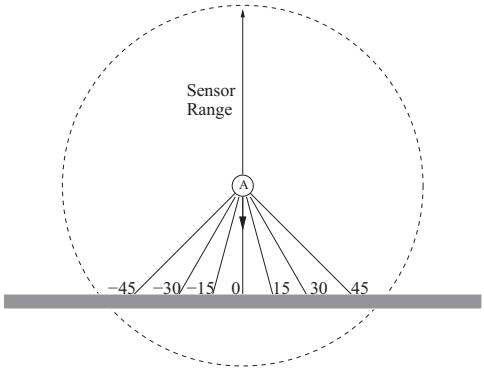


Figure 4.5: The sonar information used to determine the orientation of a wall. The sonar at 0° represents the robot's current heading.

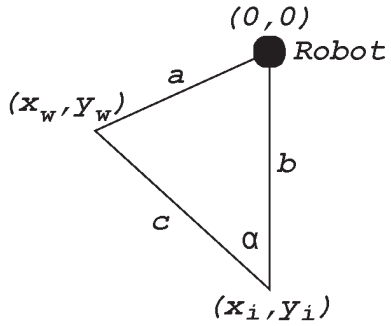


Figure 4.6: The information used to determine the correct angle of the wall relative to the goal.

At this point, three known endpoints aid robot A in determining the wall's orientation. Let (x_w, y_w) be one of the essential endpoints, and let (x_i, y_i) be the intersection between the virtual goal line and the wall line. Robot A's position is always $(0, 0)$ with respect to the other points. Using this information, A can create a triangle as seen in Fig. 4.6. The robot calculates the angle between the wall and the goal direction to find the orientation. If this angle is within a predefined error amount,

the wall is assumed to be a Couette wall; otherwise it is considered to be a stationary wall. The Cosine Law is used to calculate the angle α:

$$\alpha = \cos^{-1}\left(\frac{b^2 + c^2 - a^2}{2bc}\right)$$

Robot A defines all walls with $\alpha = 0° + \varepsilon$ or $\alpha = 180° - \varepsilon$ as parallel walls, where ε is a user defined error parameter based on the type of obstacles the robot expects to encounter. The pseudocode for this portion of the algorithm can be seen in Fig. 4.7.

Now that the robot has determined the orientation of the wall with respect to the goal direction, it needs to determine how to reset its velocity, if it has changed due to a collision with the wall. Ideally, in a global coordinate system, the velocity is reset in the x-direction to be a biased Maxwellian distributed velocity, and in y-direction to be a Gaussian distributed velocity. In the event of a collision with a Couette wall, the wall velocity v_{wall} is added to the y-direction.

These ideal (i.e., in global coordinates) velocities need to be converted into real (i.e., in local coordinates) velocities. First, robot A determines the speed s and angle ζ_1 of the ideal velocity, where ζ_1 is the amount needed to turn in a global coordinate system. To determine the velocity in local coordinates, A translates the ideal ζ_1 to the local β using the angle of the goal θ_g. With the information shown in Fig. 4.6, the robot is able to determine its new local velocity from its ideal velocity and bearing to goal. The first step is to figure out the angle between the ideal velocity vector and the robot's current heading:

$$\beta = \theta_g - 90 + \zeta_1$$

With angle β, the robot can now reset its (local) velocity based on the previously

```
float collisionParallel(sonarDistances, goalBearing)
    // Sonar i represents 15*i degrees off of the heading.
    // Find essential endpoint ee1
    theta = 45;
    for (i = 0; i < numSonars; ++i, theta -= 15)
        if (sonarDistances[i] = sonar.MAX)
            continue;
        ee1x = sonarDistances[i]*cos(theta);
        ee1y = sonarDistances[i]*sin(theta);
        break;
    // Find essential endpoint ee2
    theta = -45
    for (i = 0; i < numSonars; ++i, theta += 15)
        if (sonarDistances[i] = sonar.MAX)
            continue;
        ee2x = sonarDistances[i]*cos(theta);
        ee2y = sonarDistances[i]*sin(theta);
        break;
    // Find goal endpoint
    goalx = sonar.MAX*cos(goalBearing);
    goaly = sonar.MAX*sin(goalBearing);
    Find intersection point
    (intx, inty) = intersects(ee1x, ee1y, ee2x, ee2y, 0, 0, goalx, goaly)
    if ( noIntesectionPoint(intx, inty) )
        a2 = ee1x*ee1x + ee1y*ee1y;
        b2 = intx*intx + inty*inty;
        c2 = (ee1x-intx)*(ee1x-intx) + (ee1y-inty)*(ee1y-inty);
        top = b2 + c2 - a2;
        bottom = 2*sqrt(b2)*sqrt(c2);
        return acos(top/bottom);
    return 0;
```

Figure 4.7: KT wall processing pseudocode.

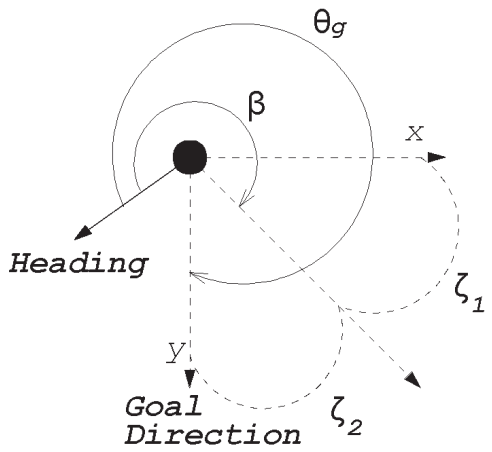


Figure 4.8: The information locally used to determine the correct angle to turn when recalculating velocity.

determined speed, s, and β:

$$\begin{aligned} v_x &= s \times \cos\beta \\ v_y &= s \times \sin\beta \end{aligned}$$

This concludes the collision processing for walls. The robot now has a new desired velocity vector, in its own local coordinate system that will direct it away from the wall. The pseudocode to generate this velocity vector can be seen in Fig. 4.9.

4.5.2 Robot Collisions

The second part of the algorithm consists of processing (virtual) collisions with other robots. By using our trilaterative localization algorithm, a robot can sense the distance and bearing to all other robots within sensing range. As stated before, collisions between robots require a communication module. The communication ensures that only one robot processes the collision, which is needed because of the random component inherent in KT.

```
void updateForWalls(goalBearing, bearings, distances)
    // Sonar i represents 15*i degrees off of the heading.
    if (distances[0] > safe and distances[6] > 2 and distances[18] > 2)
        return;

    thetaCollision = collisionParallel(distances, goalBearing);
    if (thetaCollision < 45 and thetaCollision > -45)
        parallel = true;
    if (thetaCollision > 135 and thetaCollision < 225)
        parallel = true;
    idealVx = sqrt(-log(1-random()))*mpv;
    idealVy = randomGaussian()*stdev;
    d = sqrt(idealVx*idealVx + idealVy*idealVy);
    xi = atan2(idealVy, idealVx);
    beta = goalBearing - 90 + xi;
    vx = d*cos(beta);
    vy = d*sin(beta);
    if ( parallel )
        vx += wall*cos(goalBearing);
        vy += wall*sin(goalBearing);
```

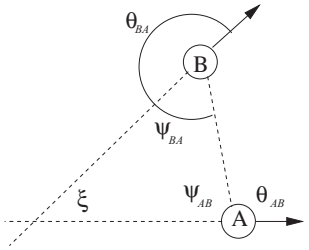
Figure 4.9: KT wall processing pseudocode.

A robot begins processing collisions by checking for messages from other robots. These messages contain pre-processed collision information. The structure for the messages will be apparent later, but for now assume that the robot has enough information for processing. Once the robot has completed the messages, it can begin determining if there are any other unprocessed robots left to collide with. One issue for the algorithm is that velocities are reset using a random collision angle; therefore, robots could collide repeatedly. To counter this situation, collisions can only be processed again for a pair of robots after a predetermined number of time steps have occurred.

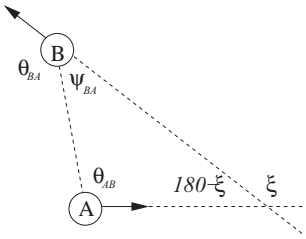
Let us walk through the inter-robot collision algorithm in detail. We will refer to the current robot as A, and the robot that A collides with as B. Therefore, A is

currently processing this inter-robot collision. Again, there is some predefined safe distance that allows robots to determine whether a collision has occurred. In order for A to be processing this collision, it has determined that B is closer than the safe distance. There is no requirement that states that the robots must have velocity vectors that will cause a collision in order for the collision to be processed.

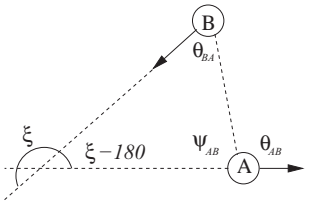
Assume that robot B has not already processed this collision. Robot A requests the bearing from B to A, called the reverse bearing, θ_{BA}, and the current speed of B, namely s_B. Using this information, A can determine the velocity of B relative to A. Since robot A always moves along its x-axis, its velocity is straightforward. Determining the relative velocity of robot B is not simple. Fig. 4.10 shows the available information and all of the cases, and provides a good reference for understanding the final equation.



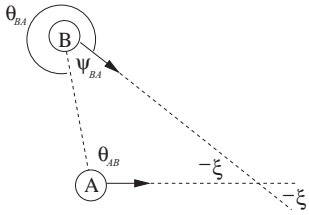
A. Case I. $\xi \geq 0°$ and $\xi < 90°$



B. Case II. $\xi \geq 90°$ and $\xi < 180°$



C. Case III. $\xi \geq 180°$ and $\xi < 270°$



D. Case IV. $\xi \geq 270°$ and $\xi < 360°$

Figure 4.10: Cases for determining robot B's velocity in robot A's coordinate system.

Beginning with Case I. we utilize the known information to derive our final equa-

tion:

$$\begin{aligned}
\psi_{AB} &= 180 - \theta_{AB} \\
\psi_{BA} &= \theta_{BA} - 180 \\
180 &= \xi + \psi_{AB} + \psi_{BA} \\
180 &= \xi + 180 - \theta_{AB} + \theta_{BA} - 180 \\
180 &= \xi - \theta_{AB} + \theta_{BA} \\
\xi &= 180 + \theta_{AB} - \theta_{BA}
\end{aligned}$$

Continuing with Case II we again utilize the known information to derive the final equation:

$$\begin{aligned}
\psi_{BA} &= 180 - \theta_{BA} \\
180 &= 180 - \xi + \theta_{AB} + \psi_{BA} \\
180 &= 180 - \xi + \theta_{AB} + 180 - \theta_{BA} \\
180 &= \xi - \theta_{AB} + \theta_{BA} \\
\xi &= 180 + \theta_{AB} - \theta_{BA}
\end{aligned}$$

Case III uses similar information for the derivation:

$$\begin{aligned}
\psi_{AB} &= 180 - \theta_{AB} \\
180 &= \xi - 180 + \psi_{AB} + \theta_{BA} \\
180 &= \xi - 180 + 180 - \theta_{AB} + \theta_{BA} \\
180 &= \xi - \theta_{AB} + \theta_{BA} \\
\xi &= 180 + \theta_{AB} - \theta_{BA}
\end{aligned}$$

Finally, we conclude with the derivation of Case IV.

$$\begin{aligned} \psi_{BA} &= 360 - \theta_{BA} \\ 180 &= -\xi + \theta_{AB} + \psi_{BA} \\ 180 &= -\xi + \theta_{AB} + 360 - \theta_{BA} \\ \xi &= 180 + \theta_{AB} - \theta_{BA} \end{aligned}$$

As we have shown, for each case it is possible to derive that the angle of the velocity vector for B in robot A's coordinate system as:

$$\xi = 180 + \theta_{AB} - \theta_{BA} \tag{4.1}$$

where θ_{AB} is the bearing to robot B from robot A obtained from the trilaterative localization algorithm and θ_{BA} is the bearing to robot A from robot B obtained through the communication module. Using this ξ, A calculates the velocity of robot B in its local coordinate system by:

$$\begin{aligned} v_x^B &= s_B \cos\xi \\ v_y^B &= s_B \sin\xi \end{aligned}$$

A then begins the kinetic theory portion of processing the collision utilizing B's velocity. A first calculates the relative speed of the two robots as

$$s_r = \sqrt{(v_x^B - v_x^A)^2 + (v_y^B - v_y^A)^2}$$

and then it calculates the average velocity, $\vec{v}_c$, of the two robots. Once it has these two quantities, it then chooses the colliding angle ϕ randomly from a uniform distribution. This colliding angle is the angle from the center that the two robots collide on. Using

this ϕ, A determines the relative velocity, $\vec{v_r}$, from the relative speed and the colliding angle, ϕ.

There are now two velocities to assign to the two robots, from the following equation:

$$\begin{aligned} \vec{v}_1 &= \vec{v}_c + \tfrac{1}{2}\vec{v_r} \\ \vec{v}_2 &= \vec{v}_c - \tfrac{1}{2}\vec{v_r} \end{aligned}$$

Once A has determined what velocities to assign each robot, by avoiding future collisions, it transmits the information to robot B. In order for robot B to understand the new velocity, A first translates the velocity vector into local polar coordinates. A then transmits three pieces of information to robot B for processing. It sends the bearing, θ_{AB}, so that B knows the reverse angle to A, as well as the new speed of B, and finally the turn in robot A's coordinate system that B should move along. Combining these three pieces of information, B is able to determine its new velocity in its own coordinate system by using Equation 4.1.

This collision process is repeated for all robots within A's safe distance that have not already processed the collision. The final resultant velocity of each robot is a vector sum of all the new velocities from every collision. Figure 4.11 contains the pseudocode for the robot collision processing.

4.5.3 Finishing

Executing the new velocity requires turning and then moving. Robots use $\tan^{-1}$ of their new velocity to determine the turn, and *always* move with a predefined speed. We assume that the robots have low-level routines that prevent *actual* collisions. In the event that a robot cannot move at maximum speed, it moves as far as possible.

```
void updateForRobots(ids, bearings, distances)
    if ( noRobotsClose() )
        return;
    for (i = 0; i < numRobotsClose; ++i)
        if ( distanceToRobot(i) > safe ) continue;
        if ( alreadyCollided(i) ) continue;
        if ( receivedCollisionInfo(i) )
            calculateNewVelocity();
        processCollision();
```

Figure 4.11: KT wall processing pseudocode.

Chapter 5

Theoretical Properties of Physics Systems

5.1 Introduction

One of the key benefits of using a physics-based multiagent system is that extensive theoretical (formal) analysis tools already exist for making predictions and guarantees about the behavior of the system. Furthermore, such analyses have the added benefit that their results can be used for setting system parameters for achieving desired multiagent behavior. The advantages of this are enormous – one can transition directly from theory to a successful robot demo, without all the usual parameter tweaking. For an example of such a success (using AP solid), see (Spears, Gordon-Spears, Hamann, and Heil 2004). To demonstrate the feasibility of applying physics-based analysis techniques to physics-based systems, we make predictions that support some of our claims regarding the suitability of gas models for our coverage task.

Recall that our objectives are to sweep a corridor and to avoid obstacles along the way. A third objective for the *swarm* of agents is that of coverage. We utilize our previous definitions of spatial coverage: *longitudinal* (in the goal direction) and *lateral* (orthogonal to the goal direction). Longitudinal coverage can be achieved by movement of the swarm in the goal direction; lateral coverage can be achieved by a uniform spatial distribution of the robots between the side walls. The objective of the coverage task is to maximize both longitudinal and lateral coverage in the

minimum possible time (maximize temporal coverage). The number of particles, initial distribution of particles, and termination criterion are determined individually for each experiment, based on earlier studies.

To measure how well the robots achieve the task objective, we observe:

1. The distribution of velocities of all agents in the corridor. This is a measure of both temporal and spatial coverage (i.e., a wide distribution typically implies greater coverage of the corridor length and width).
2. The degree to which the spatial distribution of the robots matches a uniform distribution. This is a measure of lateral coverage of the corridor
3. The average agent speed (averaged over all agents in the corridor). This is a measure of spatial and temporal coverage.

Before describing the experiments, let us first present the metric used for measuring error between the theoretical predictions and the simulation results. *Relative error* is used, which is defined as:

$$\frac{|\ theoretical - actual\ |}{theoretical}$$

The theory and experiments presented in this section assume a 2D environment. The theoretical formulas assume no obstacles.

We ran two sets of experiments. The first set of experiments explicitly tests the theory versus experimental results, to see how predictive the theory is. For each experiment, one parameter was perturbed (eight different values of the affected parameter were chosen). For each parameter value, 20 different runs through the simulator were

executed, each with different random initial agent positions and velocities. The average relative error (over the 20 runs) and the standard deviation from the average were determined from this sample.

The second set methodically adds obstacles to the environment to determine how well the theory predicts as its assumptions are increasingly violated for KT only. Explanations for why AP is not considered in these experiments are handled in the experimental sections.

Again for each of the second set of experiments, one parameter was perturbed (eight different values of the affected parameter were chosen). For each parameter value, 20 different runs through the simulator were executed, each with different random initial agent positions and velocities. The average relative error (over the 20 runs) and the standard deviation from the average were determined from this sample. An obstacle generator places obstacles into the world in randomly-chosen locations. This was repeated five times, for five obstacle densities ranging from 10% to 50% resulting in five curves.

Measurement of each of these three aspects of the system (velocity distribution, spatial distribution, average speed) corresponds to each of our three experiments. Recall (above) that for each experiment, we vary the value of one parameter. The reason for varying such parameter values is to allow a system designer to optimize the design – by understanding the trade-offs involved. In other words, we have observed that there is a trade-off between the degrees of longitudinal coverage, lateral coverage, and temporal coverage – greater satisfaction of one can lead to reduced satisfaction of the others, making this a Pareto-optimization task. By varying parameter values and showing the resulting velocity and spatial distributions and average speed, a system designer can choose the parameter values that yield desired system performance. Finally, why show *both* theory and simulation results for each experiment and each

parameter value? Our rationale is that it is far easier for a system designer to work with the theory when deciding what parameter values to choose for the system. The designer can do this *if* the theory is predictive of the system. In our experimental results below, we show that the theory is indeed predictive of experimental results using our simulation.

For the first experiment, the agents are placed uniformly along the beginning of a long corridor and allowed to perform one sweep. In the second experiment, the agents are placed in a square container in an initially tight Gaussian distribution and allowed to diffuse to an asymptotic state. For the final experiment, the agents are placed at the beginning of a long corridor once again, and allowed to run for a predetermined number of time steps, after which the average speed is measured. In the second and third experiments, there is no goal force or wall movement, and therefore there is no directed bulk movement (transport) of the swarm.

5.2 Experiment 1: Velocity Distribution

The first theoretical prediction for our system is devoted to longitudinal coverage and sweep speed via movement. The theory predicts the velocity distribution for each of the approaches, AP and KT. It is assumed that fluid flow is in the y-direction (downward toward the goal), as in Fig. 5.1. Obstacles are not added to the AP system because of the challenge of balancing the forces needed to keep the AP particles from moving through obstacles. In order to test the theory of AP, we do not apply any friction and we do not limit the maximum speed of the particles in the system. Because of this particles are not able to stop before moving through an obstacle. In one case we can keep the AP particles from moving through obstacles by stopping them if they attempt it, but this causes a loss of energy in the system and will make

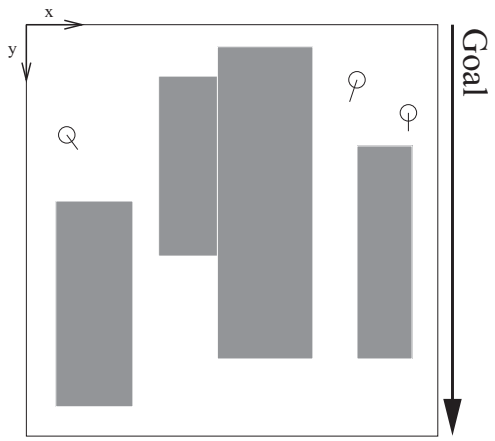


Figure 5.1: Fluid flow simulation diagram.

the theory grossly incorrect.

5.2.1 AP Theory

Recall that the AP approach is an implementation of $\vec{F} = m\vec{a}$. Assuming $F_y = g$, where g is the magnitude of the goal force, which is constant for all particles and is strictly in the goal direction, and assuming $m = 1$ (which is assumed throughout this thesis), we have the following derivation (where v_y is the magnitude of the velocity in the y-direction, and v_x is assumed to be 0):

$$\begin{aligned} g &= \frac{dv_y}{dt} \\ g \cdot dt &= dv_y \\ g \int dt &= \int dv_y \\ g \cdot t &= v_y \\ v_y &= gt \end{aligned}$$

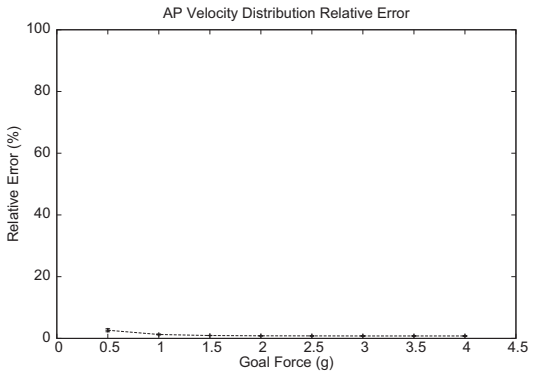


Figure 5.2: Relative Error for AP Velocity Distribution.

This shows that the velocity in the direction of the goal is just the force of the goal times the amount of time that has elapsed.

5.2.2 AP Experimental Results

We set up an experiment using this theoretical formula to determine the relative error for our experiments. The experiment placed 500 agents in the simulator and terminated in 100 time steps, since by this time the agents reach the maximum velocity that can be achieved on real robots. The parameter being varied is the goal force. The results are plotted in Fig. 5.2, and the relative error is roughly 1%.

5.2.3 KT Theory

For KT, a traditional one-sided Couette drives the bulk swarm movement. The complete derivation for the velocity profile of a Couette flow can be found in (Anderson 1995) (pages 417–420), but here we present a more concise version.

For steady, 2D flow with no external forces, there is a classical "Governing Equation" that predicts the y-direction momentum of the fluid. This Governing Equation is:

$$\rho v_y \frac{\partial v_y}{\partial y} + \rho v_x \frac{\partial v_y}{\partial x} = -\frac{\partial P}{\partial y} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{xy}}{\partial x}$$

where ρ is the fluid density, v_x and v_y are the x- and y-components of velocity, P is the fluid pressure, and τ_{yy} and τ_{xy} are the normal and shear stresses, respectively. We can use this equation for momentum to derive the velocity. However, first we need to specialize the equation for our particular situation. We assume that we have a Newtonian fluid, parallel flow, and zero pressure gradient, therefore for Couette flow, the equation becomes:

$$0 = \frac{\partial}{\partial x}(\mu \frac{\partial v_y}{\partial x})$$

where μ is the fluid viscosity. Assuming an incompressible, constant temperature flow with constant viscosity, this becomes:

$$\frac{\partial^2 v_y}{\partial x^2} = 0 \tag{5.1}$$

Equation 5.1 is the Governing Equation for steady, 2D, incompressible, constant temperature Couette flow. Integrating twice with respect to x to find v_y, we get:

$$v_y = c_1 x + c_2 \tag{5.2}$$

We can solve for c_1 and c_2 from the boundary conditions. In particular, at the stationary Couette wall ($x = 0$), $v_y = 0$, which implies that $c_2 = 0$ from Equation 5.2. At the moving wall ($x = D$), $v_y = v_{wall}$, where D is the Couette width and v_{wall} is the velocity of the moving wall, which is in the y-direction (toward the goal). Then $c_1 = v_{wall}/D$ from Equation 5.2.

Substituting these values for c_1 and c_2 back into Equation 5.2, we get:

$$\begin{aligned} \frac{v_y}{v_{wall}} &= \frac{x}{D} \\ v_y &= \frac{x}{D} v_{wall} \end{aligned}$$

This is a linear profile.

5.2.4 KT Experimental Results

We set up an experiment to measure the relative error generated by our simulation, with each particle behaving as if it were part of a one-sided Couette flow. Each experiment contained 3,000 particles, and ran for 50,000 time steps. When determining the error, we divided the world into seven discrete cells. For each cell, we determined the average velocity of the particles located in that cell. The relative error was averaged across all cells and plotted in Fig. 5.3(A) for eight different wall speeds. One can see that the error is below 20%, with a reduction in error for KT as the wall speed is increased. Note that the original algorithms from Garcia (Garcia 2000) also have error between theory and simulation that is slightly below 20%. Reasons for this discrepancy between theory and simulation are elaborated later. When determining the longitudinal coverage via swarm movement, we are able to predict very accurately for both algorithms in the simple scenario, except at slow wall speeds for KT.

The second set of experiments contained 500 particles, and ran for 5,000 time steps. To determine the error, we divided the world into eight discrete columns. For each column, we determined the average velocity of the particles located in that column. The relative error was averaged across all columns and plotted in Fig. 5.3(B) for eight different wall speeds and five different obstacle percentages. The error increases

dramatically as obstacles are added, with a reduction in error as the wall speed is increased.

Why is there a discrepancy between the theory and experimental results in our second set of experiments? The reason is that theory predicts a linear velocity profile, but assumes that particles never move backwards (back up the corridor). In the experiments, on the other hand, particles *do* move backwards, regardless of whether or not there are obstacles. In fact, as obstacles are introduced into the simulated world, the frequency of backward moving particles increases substantially. This phenomenon accounts for the significant discrepancy between theory and experiment.

Upon inspection of the graph, we noticed that between 0% and 10% obstacles, at wall speed 1.0, our predictions are actually better for 10% obstacles. This is an interesting observation that warranted further investigation. We formed the following hypothesis to explain the reason for this phenomenon. Recall that we expect a linear velocity profile. KT essentially models collisions between walls and other particles. When a particle collides with a Couette wall, then that particle receives an increase in its goal velocity. In order to recreate the linear velocity profile, this velocity is transfered across the system through inter-particle collisions. At a low (1.0) wall velocity, inter-particle collisions play a larger role in maintaining the correct velocity profile than the wall speed does. With a low wall speed, horizontal movement is more predominant than vertical movement for the agents/particles. Recall that collisions only occur locally – between particles in the same cell. Unfortunately, with low wall speed and no obstacles these collisions are too infrequent to generate the desired linear velocity profile. On the other hand, once we introduce a few obstacles, the obstacles cause the particles to collide more frequently, thereby increasing the linearity of the velocity profile and reducing the error between theory and simulation.

To test this hypothesis, we created a second experiment. The experiment tested

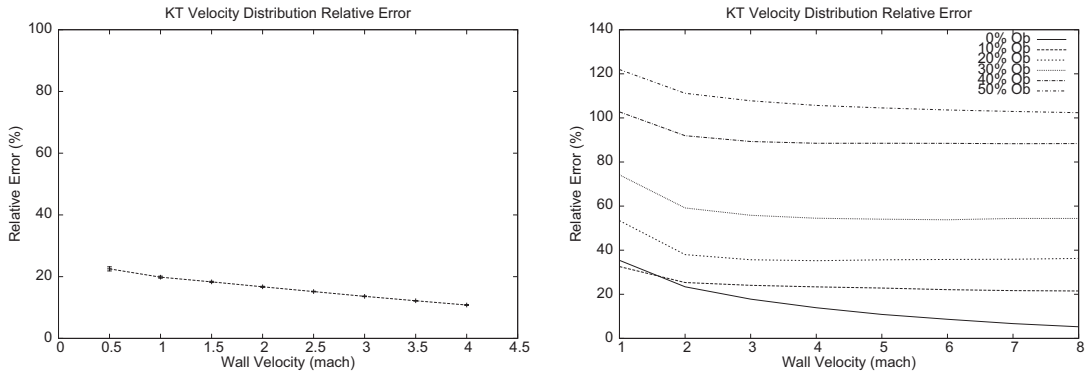
the belief that at lower speeds, inter-particle collisions were the primary distributing force for the velocity profile. In order to accomplish this we disregarded the y-position of the particles when determining inter-particle collisions. So, given the right conditions particles within the same column could collide, thus increasing the number of inter-particle collisions. This experiment more closely simulates Garcia's (Garcia 2000) initial model. When we ran the experiment to verify the hypothesis, the results confirmed what we believed. We saw that we can predict the velocity distribution for 0% obstacles with relative error at roughly 16%, confirming our hypothesis about the role of both the inter-particle collisions and the placement of the obstacles.

In conclusion, without obstacles, the theory becomes highly predictive as the wall velocity increases. Furthermore, this very predictive theoretical formula can also be used to achieve a desired swarm velocity distribution, i.e., to *control* the swarm – simply set the value of v_{wall}, the virtual wall speed, to achieve the desired distribution, using the formula.

On the other hand, with an increasing number of obstacles, the predictiveness of the theory is increasingly reduced. Therefore, future work will focus on increasing the sophistication of the theory for predicting the swarm velocity distribution in situations with obstacles present.

5.3 Experiment 2: Spatial Distribution

For the second experiment, we predict the lateral coverage via the spatial distribution. During the experiment, there is not a goal direction and obstacles are only added for KT. The agents' task is to diffuse throughout the system.



A. *B.*

Figure 5.3: Relative Error for KT Velocity Distribution.

5.3.1 Theory

Regardless of the approach a gas model predicts a uniform distribution throughout the system. For the experimental performance metrics, we measured the distance from the uniform distribution once the gas reached an asymptotic state. This was accomplished by dividing the system into discrete cells and counted the number of particles in each cell. Theory predicts that the number of particles in each cell should be n/c, where n is the total number of particles and c is the total number of grid cells that cover our system.

Our experimental system serves as a simple container to hold a gas. The gas should diffuse within the container until it reaches an asymptotic state and contains equal numbers of particles in each cell. We allowed the system several thousand time steps, starting from a tight Gaussian distribution about the center of the container, to reach this state and then measured the number of particles in each cell. This measurement was averaged over many time steps, since particles were still moving through the system and diffusion did not imply particles ceased to move. Both experiments were the same for AP and KT.

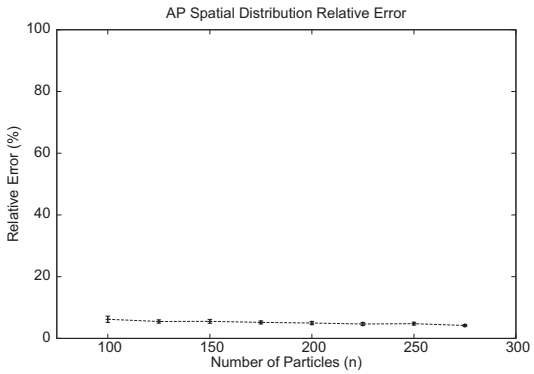


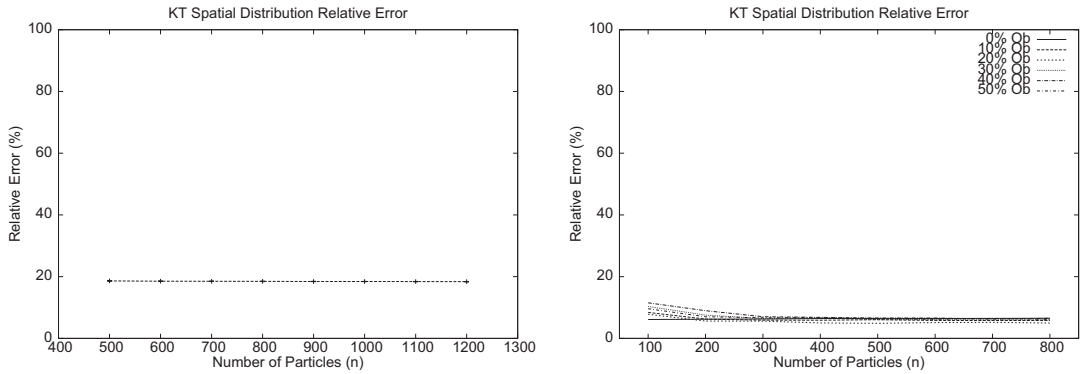
Figure 5.4: Relative Error for AP Spatial Distribution.

5.3.2 AP Experimental Results

There is a noticeable downward trend for the relative error in the AP system as more particles are added to the system. Recall that in AP we use forces to affect other particles as well as forces from the walls to keep the particles inside the simulation. This requires that particles have a desired radius such that when another particle enters this radius, it is repelled away. As more particles are added to the simulation, the space is filled with particles that are constantly pushing each other away and moving into the only formation that will allow them all to fit, which is a uniform distribution.

5.3.3 KT Experimental Results

The results can be found in Fig. 5.5. The results for KT show that we are able to predict the spatial distribution of the particles in the system with a relative error of 20%. Note that despite the introduction of as much as a 50% particle regional coverage, we are able to predict the spatial distribution with a relative error of less than 20%, which is excellent.



A. *B.*

Figure 5.5: Relative Error for KT Spatial Distribution.

5.4 Experiment 3: Average Speed

For the third experiment, we predict the average speed of the particles in the system. The average speed of the particles serves as a measure of how well the system will be able to achieve complete coverage, because higher speed implies greater coverage. Again for this experiment we do not include obstacles in the corridor. The reasons are similar to the reasons provided for the first experiment. We do not provide a maximum speed for the AP particles and therefore keeping them from moving into obstacles becomes a very challenging problem. In one time step at maximum speed from the theory they could move through the entire obstacle without even registering ever noticing it.

5.4.1 AP Theory

The derivation for AP's prediction of average speed begins with a theoretical formula for AP system potential energy (PE) from (Spears, Spears, and Heil 2004). This theory assumes that the particles start in a cluster of radius 0. There are two different situations, depending on the radial extent to which F_{max} dominates the force law

$\vec{F} = m\vec{a}$. Recall that agents use F_{max} when $F > F_{max}$. This occurs when $\frac{G}{r^2} > F_{max}$ or, equivalently, $r \leq \sqrt{\frac{G}{F_{max}}} \equiv R'$. The first situation is when F_{max} is used only at close distances, i.e., when $0 \leq R' \leq 1.5R$. The second situation occurs when $R' > 1.5R$. Here we assume the first situation, i.e., a low value of G is used such that $G \leq F_{max}(1.5R)^2$, and F_{max} is only used at close distances. Because we are using AP *gas*, there is no friction and all forces are repulsive. We begin with a two-particle system. In this case, the formula is the sum of two integrals. The first represents the force felt by one particle as it approaches another, from a distance of $1.5R$ to R'. The second is the force F_{max} that is experienced when $0 \leq r \leq R'$. Then, using R' as defined above, with r the inter-agent distance, we have (V is the standard symbol for PE):

$$
\begin{aligned}
PE &= V \\
&= \int_0^{R'} F_{max}dr \; + \; \int_{R'}^{1.5R} \frac{G}{r^2}dr \\
&= F_{max}R' \; + \; G\int_{R'}^{1.5R} r^{-2}dr \\
&= F_{max}R' \; + \; G(-r^{-1})|_{R'}^{1.5R} \\
&= F_{max}R' \; + \; G\left(-\frac{1}{1.5R} + \frac{1}{R'}\right) \\
&= \sqrt{GF_{max}} \; + \; G\left(\sqrt{\frac{F_{max}}{G}} - \frac{1}{1.5R}\right) \; \textit{because } R' = \sqrt{\frac{G}{F_{max}}} \\
&= \sqrt{GF_{max}} \; + \; \sqrt{GF_{max}} \; - \; \left(\frac{G}{1.5R}\right) \\
&= 2\sqrt{GF_{max}} \; - \; \left(\frac{G}{1.5R}\right)
\end{aligned}
$$

Now we generalize V to N particles. V_N is our abbreviation for total potential energy, and

$$V_N = \sum_{i=0}^{N-1} iV = \frac{VN(N-1)}{2}$$

Note that all the potential energy transforms into kinetic energy (since there is no friction energy dissipation), i.e., $V_N \rightarrow KE$. Also, the total kinetic energy, KE, is equal to $\frac{1}{2}\sum_{i=1}^{N}(v(i))^2$, assuming m=1 and $v(i)$ is the speed of particle i. This formula for KE is equal to $\frac{N}{2}\langle v^2\rangle$, where $\langle v^2\rangle$ is the average of the particle speeds squared. Setting V_N = KE, we get:

$$\begin{aligned} \frac{VN(N-1)}{2} &= \frac{N}{2}\langle v^2\rangle \\ V(N-1) &= \langle v^2\rangle \end{aligned}$$

Substituting for V we get

$$\langle v^2\rangle = V(N-1) = (N-1)\left[2\sqrt{GF_{max}} - \left(\frac{G}{1.5R}\right)\right]$$

From (Stark and Woods 1986), we know that the relationship between $\langle v\rangle$ and $\langle v^2\rangle$ is the following:

$$\langle v\rangle = \sqrt{\langle v^2\rangle - \sigma^2}$$

where σ^2 is the variance of the velocity distribution. However, because the variance of the velocity distribution is not typically available when making a theoretical prediction, one approximation (which is an upper bound on the true theoretical formula because it assumes 0 variance) that we can use is:

$$\langle v\rangle \approx \sqrt{\langle v^2\rangle} = \sqrt{(N-1)\left[2\sqrt{GF_{max}} - \left(\frac{G}{1.5R}\right)\right]}$$

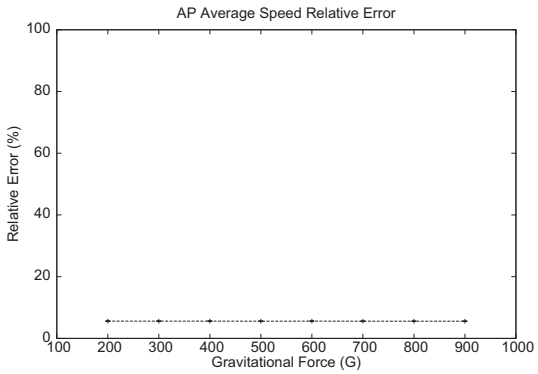


Figure 5.6: Relative Error for AP Average Speed.

5.4.2 AP Experimental Results

Using this equation for AP, we ran through the experiments (starting with the particles in a tight cluster to match the theory), allowed the gas to reach an asymptotic state, and measured the relative error. For each experiment, there were 100 agents in the system. The total number of time steps required to reach this asymptotic state is different for each value of G since it requires that the agents are no longer interacting with each other. This terminating state can be found when all the agents have ceased to change their velocity. The parameter being varied is the gravitational force, G. As seen in Fig. 5.6, the error is less than 6%. Furthermore, if the system designer has *any* clue as to what variance to expect in speeds, the theoretical prediction will be greatly improved.

In addition to verifying the formula for $\langle v \rangle$, we also verified the predictiveness of the formula above for $\langle v^2 \rangle$, which is precise because it does not involve variance. The relative error in this case is less than 0.07% for all values of G, which is *extremely* low.

5.4.3 KT Theory

We next show how we derive a KT formula for average speed by modifying the derivation for 3D $\langle v \rangle$ in (Garcia 2000) to a 2D formula for $\langle v \rangle$ (so it applies to our simulation). Assuming a system in thermodynamic equilibrium (since there is no bulk transport), with velocity components within the ranges $v_x + dv_x$ and $v_y + dv_y$, and k is Boltzmann's constant, m is the particle mass, v is the magnitude of the particle velocity (i.e., the particle speed), and T is the initial system temperature (a simple, settable system parameter), then the probability, $f(v_x, v_y)dv_x dv_y$, that a particle has velocity components in these ranges is proportional to $e^{(-mv^2/2kT)}dv_x dv_y$. In particular, we have:

$$\begin{aligned} f(v_x, v_y)dv_x dv_y &= Ae^{(-mv^2/2kT)}dv_x dv_y \\ &= Ae^{(-m{v_x}^2/2kT)}e^{(-m{v_y}^2/2kT)}dv_x dv_y \end{aligned}$$

because $v^2 = {v_x}^2 + {v_y}^2$, and A is a normalization constant that is fixed by the requirement that the integral of the probability over all possible states must be equal to 1, i.e.,

$$\int_0^\infty f(v_x, v_y)dv_x dv_y = 1$$

Therefore,

$$A = \frac{1}{\int_0^\infty e^{(-m{v_x}^2/2kT)}dv_x \int_0^\infty e^{(-m{v_y}^2/2kT)}dv_y}$$

To simplify the expression for A, we can use the fact (from pages 40-46 of (Feynman, Leighton, and Sands 1963)) that:

$$\int_0^\infty e^{(-m{v_x}^2/2kT)} dv_x \;=\; \sqrt{\frac{2\pi kT}{m}}$$

and then do likewise for v_y. Therefore:

$$f(v_x, v_y)dv_x dv_y \;=\; \left(\frac{m}{2\pi kT}\right)(e^{(-m({v_x}^2+{v_y}^2)/2kT)})dv_x dv_y$$

where $\frac{m}{2\pi kT}$ is A.

Note, however, that $f(v_x, v_y)dv_x dv_y$ is a probability for a velocity *vector*, but we want average *speed*. To get average speed, the math is easier if we go from Cartesian to polar coordinates. In particular, to go from velocity to speed, we integrate over all angles.

In polar coordinates, $2\pi v dv$ is the area of extension (annulus) due to Δv. In other words, the area of an annulus whose inner radius is v and outer radius is $v + dv$ is $2\pi v dv$. Then the Maxwell-Boltzmann distribution of speeds, $f(v)dv$, is obtained by integrating the velocity distribution, $f(v_x, v_y)dv_x dv_y$, over all angles from 0 to 2π. This integration yields:

$$f(v)dv \;=\; 2\pi v\left(\frac{m}{2\pi kT}\right)(e^{(-mv^2/2kT)})dv$$

Canceling terms, the right-hand side becomes:

$$=\; v\left(\frac{m}{kT}\right)(e^{(-mv^2/2kT)})dv$$

Because $\langle v \rangle$ is an expected value,

$$\langle v \rangle \;=\; \int_0^\infty v f(v)dv \;=\; \frac{m}{kT}\int_0^\infty v^2(e^{(-mv^2/2kT)})dv$$

From (Reif 1965)(page 609), we know that $\int_0^\infty e^{-ax^2}x^2dx = \frac{1}{4}\sqrt{\pi}a^{-\frac{3}{2}}$. Substituting v for x and $\frac{m}{2kT}$ for a, we get:

$$\langle v \rangle = \left(\frac{m}{kT}\right) I(2) = \frac{1}{4}\sqrt{\frac{8\pi kT}{m}}$$

5.4.4 KT Experimental Results

Once again, we set up an experiment to measure the actual average speed of the particles in the system. We allowed the system to converge to an asymptotic state for 50,000 time steps measuring the average speed. For each of the 500 particles in the system, we found the average speed, $\langle v \rangle$, which we used to find the relative error for the system. Since temperature drives changes in speed, we varied the temperature. Note that by setting T, a system designer can easily achieve desired behavior. The results can be found in Fig. 5.7(A) for the different temperatures. Our ability to predict the average speed of the particles is shown, by errors less than 10%.

For the second set of experiments we compare this theoretical formula with the actual average speed of the particles in the system, averaged over all 100 particles. We allowed the system to converge to an asymptotic state for 10,000 time steps, then measured the average speed, $\langle v \rangle$. Since temperature drives changes in speed, we again varied the temperature. The results can be found in Fig. 5.7(B). The results are striking. Our ability to predict the average speed of the particles is less than 10% error, which is outstanding considering that as much as a 50% obstacle coverage has been introduced. Finally, note that we can use our theory to not only predict behavior, but also to control the swarm behavior. In particular, by setting T, a system designer can easily achieve desired behavior.

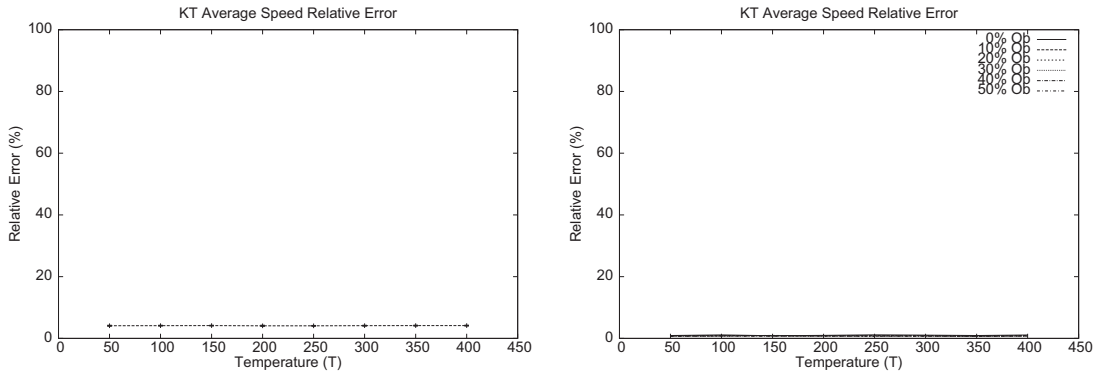


Figure 5.7: Relative Error for KT Average Speed.

5.5 Conclusions

We are capable of predicting three different properties of the system, all of which affect coverage, with an accuracy of less than 20% error, and most with error less than 10%. A 10% error is low for a theoretical prediction.

By looking at the relative error graphs of both the AP and KT approaches, one notices that the AP error is always lower than that of KT (except in the case of $\langle v \rangle$, where the AP formula is a rough approximation). In fact, only KT gets 20% errors – AP errors are always substantially lower than 20%. Our rationale for AP having lower errors between theory and simulation is that AP uses a deterministic agent-positioning algorithm, whereas KT uses a stochastic algorithm for updating particle positions. Therefore, AP predictions are precise, whereas KT predictions are only approximate. Furthermore, as stated in (Garcia 2000), Monte Carlo simulations such as KT need very long runs and huge numbers of particles to acquire enough statistical data to produce accurate (theoretically predictable) results. We cannot guarantee this, since we are developing control algorithms for robotic swarms with a few to a few thousand robots. Therefore, our experiments show a higher error than

desired for a Monte Carlo method but they are realistic for real-world swarms.

As we can see, our prediction for the velocity distribution performs very poorly when obstacles are increased, but our other predictions appear to be independent of the number of obstacles in the world. We are capable of predicting two different properties of the system, all of which affect coverage, with an accuracy of less than 10% error, which is very low for a theoretical prediction.

In conclusion, there appears to be a trade-off. AP systems are more predictable – both on the macroscopic swarm level and on the level of individual agents. Therefore, if swarm predictability is a higher priority, then AP is preferable. On the other hand, if it is important that individual agents not be predictable (e.g., to an enemy), then KT is preferable.

Chapter 6

Performance Evaluation

6.1 Introduction

This chapter is dedicated to showing the performance of our new controllers on the coverage task described in Chapter 1. In order to provide a baseline and an upperbound, we have implemented several other algorithms. The Random controller is our baseline and represents the minimal behavior needed for the controller task. The FSM controller allows a genetic algorithm to learn a better approach than our AP and KT approaches. The final controller is the Ant controller, used as an upper bound since it is known to be complete and able to guarantee complete coverage.

6.2 Evaluation Algorithms

6.2.1 Random

The first algorithm we compared against is a baseline random controller. Although we call this a random controller it still attempts to exploit known information about searching "shadow" areas – regions just downstream (w.r.t. goal direction movement) of obstacles. The robots used to compare are the exact duplicates of the robots used by the KT controller. The primary difference is that the random robot only has eight possible moves available to it. These moves are selected in 45° increments as shown

in Fig. 6.1). The controller weights each move as more or less desirable based on the information known to the robot. Specific weights are given to maintaining the same direction, moving towards the goal, and exploiting shadow areas. We designed the Random algorithm in this fashion since the shadow areas represent the hardest areas to explore.

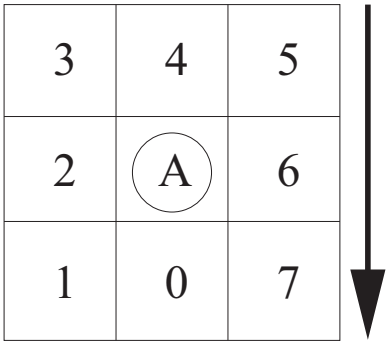
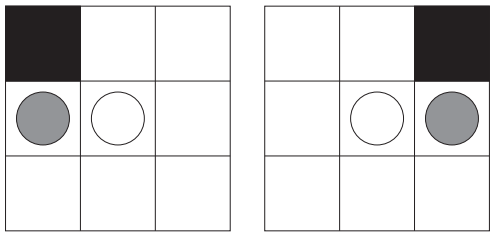


Figure 6.1: Random Controller Choices.

The first portion of the algorithm needs to determine the available moves by sensing the environment. The robot assigns a weight to each available move, and then decides to continue along the same path 80% of the time if the move is available. It then divides the remaining weight into three parts. Two of the three parts go to shadow areas (see Figure 6.2) if they are available. Then the remaining one third is again split two ways. Half of this weight goes to the goal direction and the other half is divided evenly between all the open moves. The sum of these weights is equal to one. Once this is accomplished, the robot randomly selects a number between zero and one from a uniform distribution. The position of the number inside our weighted choices determines the next move for the robot. The pseudocode for the Random algorithm can be seen in Fig. 6.3.



A. Left Shadow *B. Right Shadow*

Figure 6.2: Shadow preferences for the Random controller. The gray robot represents the preferred choice.

6.2.2 Trained Finite–State Machine

This work is a continuation of the Spears' research on strategies for protecting resources. Most of the presented work was derived from their implementation and attempts to solve a different problem using the same techniques (Spears and Gordon-Spears 2002). We attempt to train a finite–state machine (FSM) to be capable of achieving perfect coverage when presented with random obstacle–laden corridors. A FSM is a model of computation composed of states, actions, and transitions. The FSM maintains two transition functions. The first function determines the next state of the FSM given the current state and the current input. The second function determines the desired action of the FSM given the current state and current input. We will use a genetic algorithm (GA) in order to train or FSM. A genetic algorithm mimics natural evolution by maintaining a population of individuals. In our case these individuals are represented as finite-state machines. Each individual is given a fitness representing the likelihood of survival. Those individuals with the highest fitness are then selected to survive and reproduce undergoing recombination and mutation. The system then evolves through many generations searching for the desired solution.

Each robot is equipped with eight sensors, one located every 45°; thus we have eight inputs into the system. The sensors are very primitive, and only capable of

```
float distance, turn;
float VMAX = 5.0f;

void move()
    Read Goal Sensor (Light Sensor)
    Read Robot Sensor (Trilateration)
    Read Sonar Sensor
    float bearing = goalSensor.getBearing();
    int choice = randomAlgorithm();
    turn = bearing + (45*choice);

int randomAlgorithm( float goal )
    min = 0.0; left = 0.0
    previous = (goal / 45) == 8 ?  0 :  goal / 45;
    if ( available(previous) )
        weight(previous, min, 0.8)
        min = 0.8; left = 0.2;

    split = left / 3;
    if ( available(2) and notAvailable(3) and notBlockedByRobot(3) )
        weight(2, min, min+split)
        min += split; left -= split;
    if ( available(6) and notAvailable(5) and notBlockedByRobot(5) )
        weight(6, min, min+split)
        min += split; left -= split;

    split = left / 2;
    if ( available(0) )
        weight(0, min, min+split)
        min += split; left -= split;

    totalAvailable = countAvailable();
    split = left / totalAvailable;
    for (int j = 0; j < 9; ++j)
        if ( available(j) )
            weight(j, min, min+split)
            min += split; left -= split;
    return makeRandomWeightedChoice();
```

Figure 6.3: Random pseudocode.

seeing a distance equal to the speed of our robots. Each sensor can only detect if an object is blocking the path or if the path is clear, so each sensor can only have two states. Because the sensors can only see a limited distance in every direction, the world becomes a discretized version of the real world. Therefore we have 2^8 possible inputs.

Representation We chose to represent the FSM in a tabular arrangement, the same as (Spears and Gordon-Spears 2002), the efficiency gain again being primary motivation for this method. For each state s_i and input x_j, the table entry (i, j) contains the next state of the FSM, $\delta(s_i, x_j)$, and the action a to take. The number of states S will be variable, but initial tests will be performed on $S = 8$. As discussed above the number of different possible inputs is 256. The size of our table is $S \times 256$. The initial state of our world is state 0.

Initialization The goal is to allow the GA to generalize a strategy that yields good results regardless of the number of obstacles and the size of the obstacles. Therefore, we introduce as few biases as possible. When initializing the elements of the table, we choose a state from a uniform distribution between 0 and S. When initializing the action we are careful to choose only from those actions that are allowed for a given input. If the sensor has detected a wall or obstacle then the action that would take us to that position is not allowed.

The only biases initially introduced attempt to ensure that the shadow areas are searched fully. Figure 6.4 shows these sensor readings and the preferred directions associated with them. We are attempting to train the FSM to search the backside of obstacles when they have been detected.

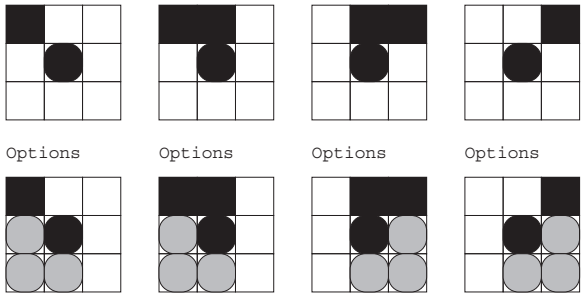


Figure 6.4: Biases for the FSM. The options are shown as the gray scale. The center square represents the current position of the robot. For each bias there are three preferred moves.

Operators Modification of the table entries by the genetic algorithm is accomplished via mutation and recombination. In order to keep the GA simple we are using uniform recombination with a probability of swapping alleles at $p = 0.5$. The probability of recombination is set at 0.6 as standard from other research. Mutation is performed the same as discussed in (Spears and Gordon-Spears 2002). Each allele is chosen for mutation with probability

$$p_m = \frac{1}{S \times 256}$$

Since each allele represents a (state/action) pair, we flip a coin to determine whether the state or the action is mutated. If the state is chosen for modification then we choose a uniformly random state from the set of all states. If instead we choose the action to mutate we choose a uniformly random action from the set of preferred or available actions for the given input.

Fitness Evaluation In order to determine the fitness of an individual, we ran the individual through our robotic simulator to measure its performance. This simulator scales well to large numbers of agents and can represent an arbitrarily sized corridor.

The corridor is then divided into discrete cells in order to determine coverage. The evaluation consists of three parts. The first part measures the percent of complete coverage in the world, p_c. This is calculated by storing a two-dimensional array of visited cells. If a particle enters a cell, then we clear that cell and assume that the cell has been searched. The second part of the fitness comes from the "shadows," as discussed earlier. In our simulation we only use square obstacles; therefore we can determine the cells that are directly behind the obstacles. The percentage of these cells explored become the second measure of fitness, p_b. Since exploration time is important, the agents are only allotted a certain number of time steps to explore the world. The percentage of those agents that have finished the corridor in the allotted time will act as a third measure of the fitness, p_t.

Each FSM is allowed to run through several different obstacle courses with the actual values for p_c, p_b, and p_t averaged over these runs. To get one measure of the fitness we combine these intermediate fitness values weighted with a constant defining their importance. The final fitness function becomes

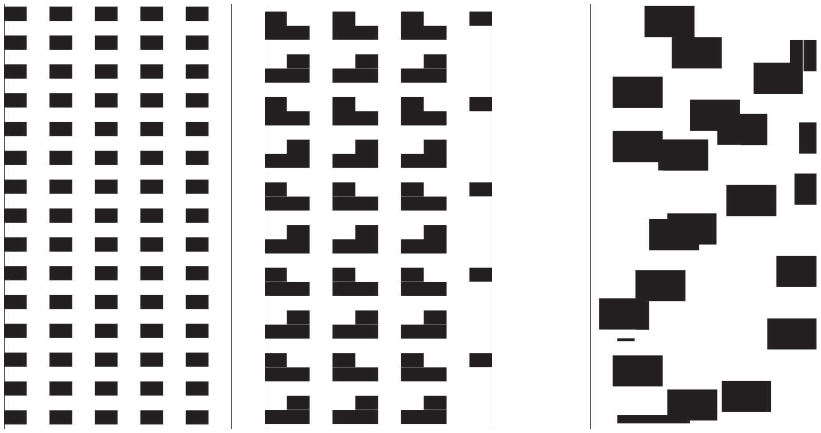
$$f = (c_1 \times p_b) + (c_2 \times p_c) + (c_3 \times p_t)$$

where

$$c_1 + c_2 + c_3 = 1.0$$

The constants c_1, c_2, and c_3 serve as parameters for the user to tune in order to achieve the desired behavior.

Selection and Termination We are using fitness proportional selection without elitism. For a termination criterion we ran the GA a pre-defined number of generations (200). Selection is accomplished using Baker's stochastic universal sampling.



A. Checkerboard *B.* "L" *C.* 30% Random Coverage

Figure 6.5: Obstacle courses used for evaluation.

GA Results There is a trade-off when determining how to evaluate the fitness for an individual. Ideally we would like to be able to run over all possible obstacle courses to get a true measure of fitness. Since there are infinitely many different obstacle courses, this is not possible. This poses an interesting problem. We need to pick a representative subset of these courses. The challenge is to develop a small number of obstacle courses that represent a very large number. Several attempts were made at accomplishing this and can be seen in Fig 6.5.

The first attempt was to use a checkerboard course for evaluations. This course contained a lot of shadow space and would really require the agents to learn to search both the shadow space and the surrounding space. Once the GA had trained a FSM individual, we realized that the GA had trained an individual that could excel at the checkerboard course, but when the same controller ran through a random obstacle course particles were stuck and performed very poorly at searching. Therefore, it was determined that our GA was learning on a specific subset of the obstacle courses and wasn't generalizing.

The second attempt tried a different obstacle course based on what appeared to be capturing the FSM in a trap. We will refer to this course as the "L" course. The course can be seen in Fig 6.5(B). The GA was allowed to run and the best individual produced was tested. Again, when the FSM was run on a random obstacle course performance was very poor.

The final attempt took a combination of the previous attempts and added another course. We ran the final GA on a combined three courses: checkerboard, "L", and a random obstacle course. All three courses can be viewed in Fig 6.5. The final fitness for the individual is an average of the scores for each course. The FSM produced by this appears more capable of generalizing than the other attempts.

Upon completion of the GA we have the best controller found, and we use this FSM controller to compete with the other controllers.

6.2.3 Ant

To experimentally compare our algorithm against what appeared to be the best performing and most appropriate alternative found in the literature, we duplicated the algorithm designed by Koenig, Szymanski, and Liu (2001). The Ant algorithm described by Koenig et al. employs graph theory to guarantee complete coverage. It assumes *stigmergy*, i.e., robots that can detect and leave markings/traces on the environment. It is important to note that the Ant robot described has an explicit advantage over our algorithms since it is able to mark its surroundings.

The Ant algorithm determines its next move by investigating the areas surrounding itself in four directions: North, East, South, and West. The four directions are determined by the orientation of the goal, which is always the South direction. Each direction is examined and the direction with the minimum covered (marked) value is chosen as the direction to move. The robot then updates its current position before

```
float distance, turn;
float VMAX = 5.0f;
boolean state1 = false;
int stateTimer = 0;

void move()
    Read Goal Sensor (Light Sensor)
    float bearing = sensor.getBearing();
    if (state1)
        int choice = antAlgorithm();
        turn = bearing + (90*choice);
        state1 = false;
    else
        turn = 0;
        if (++stateTimer == 3)
            stateTimer = 0;
            state1 = true;

int antAlgorithm( float goal )
    the direction of u[i] is goal+(90*i)
    Read Trail Values
    int[] u = trail.getTrailValues();
    for i=0 to 4
        if (u[i] < min)
            minIndex = i;
            min = u[i];
        else if (u[i] = min and rand() < 0.5)
            minIndex = i;
            min = u[i];
    return minIndex;
```

Figure 6.6: Ant pseudocode.

repeating the process. The value of the environment is incremented by one, based on Node Counting as described in (Koenig and Liu 2001). The pseudocode for the Ant algorithm can be seen in Fig. 6.6.

6.3 Experiment Setup

To better understand the performance of our new algorithms, we ran two sets of combinatorial experiments, varying the obstacle percentage in one set of experiments and varying the number of robots in the second set of experiments. For all of the combinatorial experiments we will be comparing our algorithms as well as the algorithms discussed earlier. First, we formalize the problem for each of the experiments and define how the parameters will be varied.

Measurements The simulated world is divided into 50×250 cells. We applied the following performance metrics:

1. w, the *sweep time*, which measures the number of simulation time steps for *all* agents to get to the goal location. These experiments assume only one sweep. Sweep time is a measure of temporal coverage.

2. c_c, the *total spatial coverage*, which is the fraction of all cells that have been visited at least once by at least one agent, counted over all time.

3. c_s, the *shadow coverage*. This is the same as c_c, except that we take this measure over a small region downstream (w.r.t. the goal location) of the obstacles (see Fig. 6.7). The shadow region is hardest to cover.

Experiment 1

- Ψ, a bounded 2D spatial corridor whose length is much greater than its width.
- Γ, a swarm of 25 simulated robots.
- Λ, a set of randomly placed obstacles, varying 0-40% coverage.

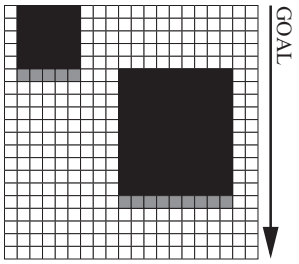


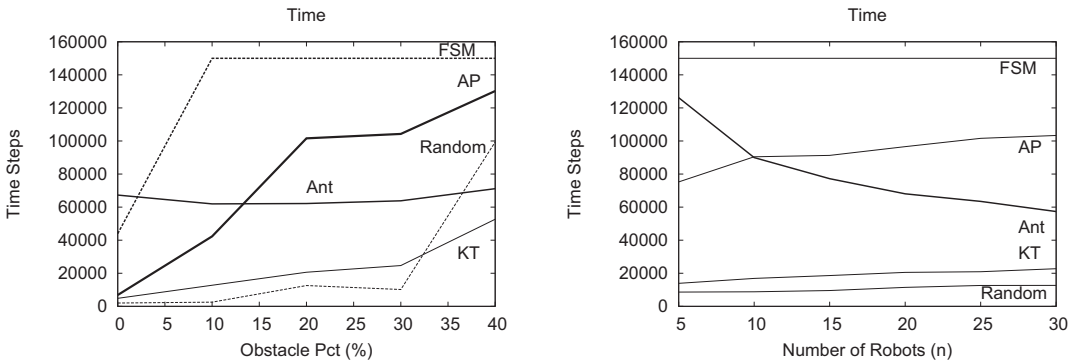
Figure 6.7: Measurements taken by the simulation. Gray squares measure c_s. Obstacles are colored black. Free space is white.

We vary the percent of obstacles Λ, in order to determine how coverage is affected. We generated 5 corridors for each obstacle percentage. Each individual experiment consisted of 10 trials through each of the 5 corridors with random initial locations and orientations of the robots. We chose to run through each corridor 10 times because most of the algorithms are stochastic. The mean performance for each metric has also been graphed. Since each algorithm is not guaranteed to have each robot finish the obstacle course, we limit the total number of time steps to 150,000.

Experiment 2

- Ψ, a bounded 2D spatial corridor whose length is much greater than its width.
- Γ, a swarm of n simulated robots, varying from five to thirty robots.
- Λ, a set of randomly placed obstacles at 30% coverage.

In the second set of combinatorial experiments we vary the number of robots in order to see how coverage is affected when robots fail or are not available. Similar to Experiment 1, we generated 5 corridors for each different number of robots n. Each individual experiment consisted of 10 runs through each of the 5 corridors with random initial locations and orientations of the robots. Again each corridor was run 10 times because most of the algorithms are stochastic and the mean performance is



A. Experiment 1 *B.* Experiment 2

Figure 6.8: Temporal Coverage.

graphed. Again to ensure the completion of the algorithms we limit the total time steps to 150,000.

6.4 Experimental Results

In order to understand the results more fully we present them according to measurement as opposed to combinatorial experiment. This allows us to look at different variations of parameter settings for each algorithm and how these settings affect specific metrics.

6.4.1 Temporal Coverage (w) Results

The first form of coverage the experiments measured was temporal coverage w, also known as the sweep time. The results can be seen in Figure 6.8.

KT KT is able to complete consistently in a short time. As obstacles are added to our corridor Ψ, KT begins to slow down. Although navigation becomes much more tough KT's temporal coverage performance degrades slowly. This leads to an

emergent behavior of the KT system. Obstacle "wells" or box canyons appear to affect most of the other controllers, but KT has an inherent behavior that allows it to escape these wells rather quickly.

For the second set of experiments, robots are added and performance is monitored. As robots are added to the corridors it appears that KT is able to continue to finish in roughly the same time. This means that KT will finish in roughly the same number of time steps regardless of the number of robots.

AP Regardless of the experiment AP's performance isn't optimal. When there are no obstacles in the environment Ψ, AP appears able to navigate as quickly as KT, but its performance quickly drops. Why does this drop occur? When one of the trials exceeds the amount of time allotted for a run, an image of the final robot positions is used in order to figure out why the time was needed. AP consistently ran over the allotted time and upon examining these images, we quickly realized that AP was becoming stuck in the obstacle wells discussed previously. An example of this can be seen in Figure 6.9.

Since the second set of combinatorial experiments were run on obstacle courses with 30% obstacle coverage, the results for AP are very similar to the first set of experiments. Note though that as more robots are added it doesn't appear to affect AP as much as changing the obstacle percentage.

Ant The ant algorithm is known to explore all areas. This causes the ant algorithm to take much more time than the other algorithms. As obstacles are added to the world Ψ, it doesn't appear to affect Ant. This is understandable since obstacles just cover additional cells and Ant no longer needs to visit those cells.

An interesting result occurs when looking at Ant's performance on the second

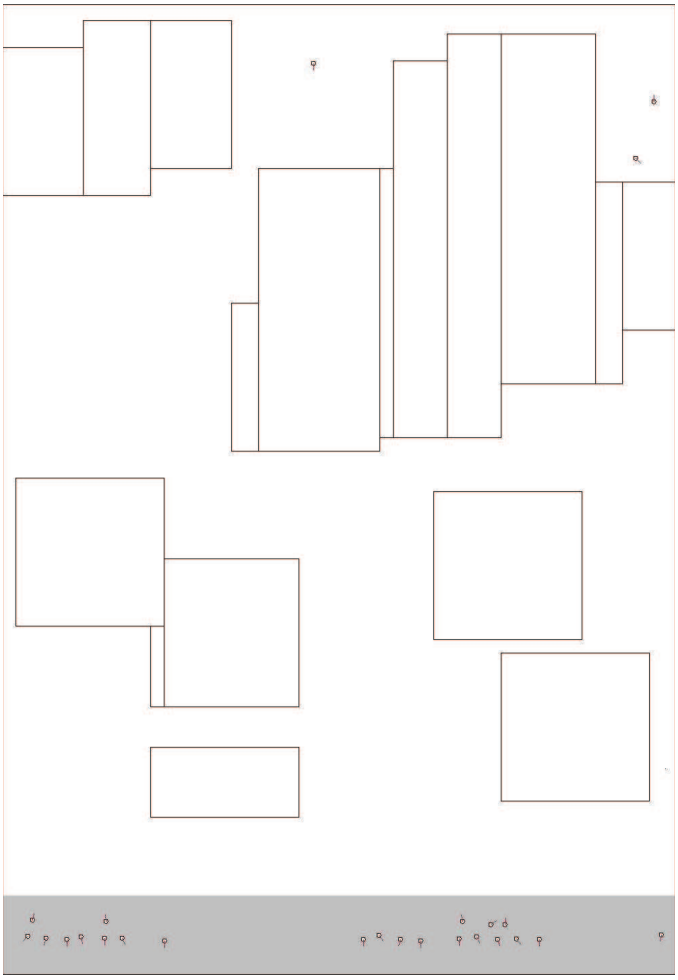


Figure 6.9: Robot simulation showing AP controller stuck in obstacle wells. The gray area is the goal and the direction the goal force is felt.

set of combinatorial experiments. As n is increased the time required for Ant to complete actually decreases. Ant is the only algorithm studied that we observed this phenomena. It is easily explained though by the Ant algorithm. The algorithm visits those surrounding cells that have been visited the least. As more robots are added the surrounding cells increase faster causing the individual robots to be able to explore faster.

FSM Our trained FSM algorithm performed very poorly even though the number of robots finished was a measurement in the learning process. With extended learning this may be able to overcome, but we can only comment on the current results. As soon as obstacles are added to Ψ, the FSM controlled algorithm is unable to finish any runs. The obstacle wells that plagued AP appear to also affect FSM.

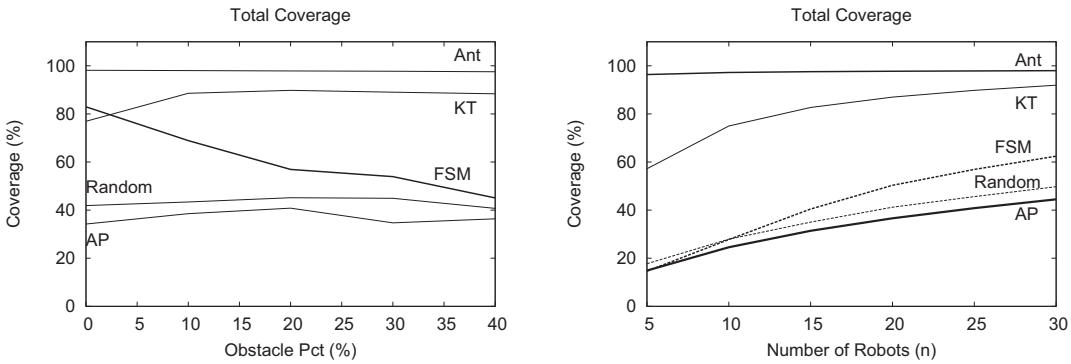
Since all of the second set of combinatorial experiments are run on obstacle courses with obstacle coverage of 30%, the FSM controller is unable to finish in the alloted time for all runs therefore achieving the worst possible temporal coverage.

Random Our baseline controller actually performed the best on temporal coverage up to a point. Once the obstacle percentage passed 40% the Random controller quickly dropped in performance. At 40% obstacle coverage the courses generated randomly are primarily obstacle wells and are difficult to navigate. By making purely stochastic decisions that emphasize exploration under obstacles the Random algorithm performs poorly.

The second set of experiments shows that as more robots are added to the Random controller performance is unaffected. This is intuitive since the robots controlled by the Random controller are unaffected by what other robots do.

6.4.2 Total Spatial Coverage (c_c) Results

We have seen how the control algorithms perform with respect to temporal coverage. The next two sections discuss how the algorithms fare with respect to two different measurements of spatial coverage. The first measurement is the total spatial coverage. It is important to note that although 100% coverage is expected, it cannot be achieved in the current robot simulator. This is because the robots are shut down once they pass the goal line. This means that there are areas behind the goal that may not be



A. Experiment 1 *B.* Experiment 2

Figure 6.10: Total Spatial Coverage.

able to be explored. These spaces could be explored on a second sweep, but for our experiments we are only concerned with a single sweep. The results for c_c can be seen in Figure 6.10.

KT Inspecting the graph found in Figure 6.10, we noticed that as the obstacle percent is increased, KT actually performs better. It increases by several percent and slowly tapers as the corridors reach 40% obstacle coverage. This can be explained by the extra obstacles in the corridor. As the obstacle percentage is increased the number of collisions with obstacles is also increased. This slows the robots longitudinal movement (goal direction) and increases the lateral movement (orthogonal to the goal). This increase in the lateral movement increases spatial coverage.

The second graph shows that as more robots added to the KT controller the coverage increases. This is as expected and it is important to note that with 30 robots, KT is able to achieve roughly 90% coverage and close to the successes of the Ant algorithm. This is even accomplished without the need for leaving traces on the environment.

AP Spatial coverage challenged AP in much the same way as temporal coverage. Different causes are believed to account for this problem. The first thought is that regardless of the location of a robot controlled by AP it always feels a force towards the goal. This leads to a very greedy robot and regardless of what other forces are felt, eventually and rather quickly the goal force will control movement. This lack of exploration shows up rather quickly since even without any obstacles in the environment AP performs more poorly than even our baseline Random. This will be explained later, but there is belief that implementation issues may also be a cause of the problem.

The second set of experiments shows that with more robots AP's performance improves, but nothing significant and AP still defines our baseline for performance on spatial coverage.

Ant Ant is known to be a complete solution to the task problem. Therefore, it was no surprise to see that for both sets of experiments Ant set the standard and achieved the best coverage. We discussed earlier that Ant would be unable to achieve complete coverage since robots are turned off as the finish their task.

FSM The FSM performance is a bit of a surprise. The most important results come from the corridors without obstacles. When watching the robots move through the corridor, it appears that the GA learned a sweeping motion to increase coverage. This sweeping motion can be seen in Figure 6.11. We notice that the FSM controller actually performs the second best when there are no obstacles, but KT quickly regains ground as soon as obstacles are added to the corridor. As more obstacles are added to the environment FSM's performance deteriorates until it performs roughly as well as Random. It is also important to show the temporal coverage along with the spatial

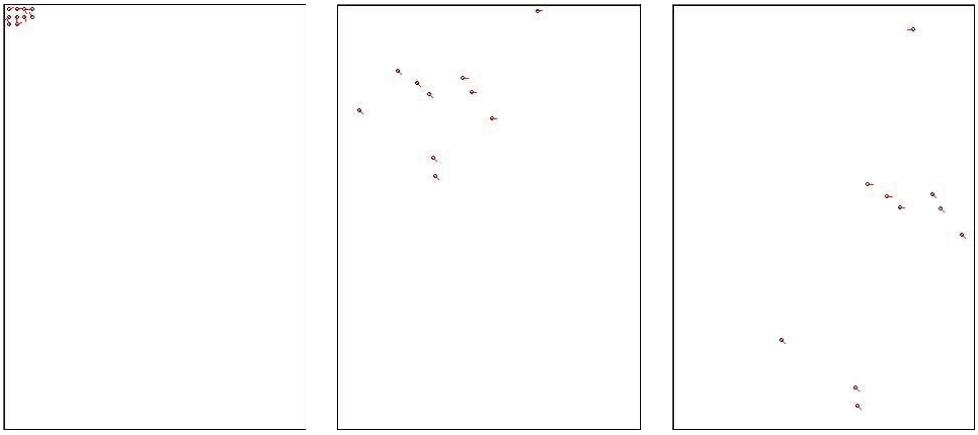


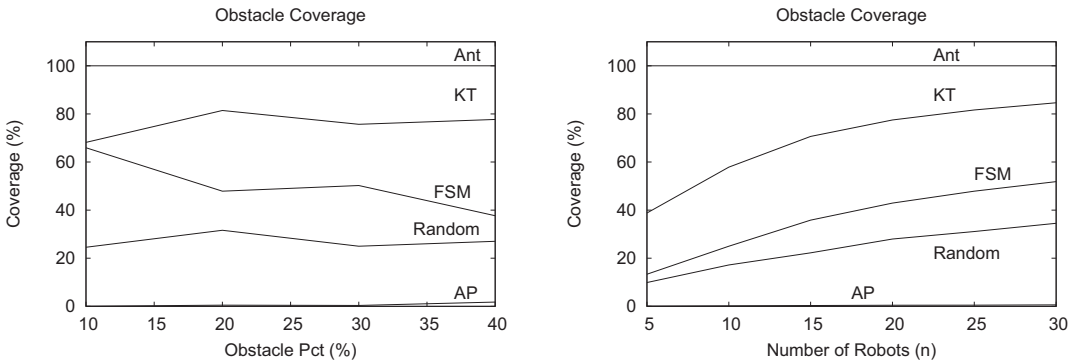
Figure 6.11: Learned FSM behavior.

coverage. The robots are only allowed 150,000 time steps in order to complete, yet the FSM controller never finished as soon as obstacles were added to the environment. This means that there must have been cycles that caught some of the robots. These cycles will directly affect the amount of coverage that the FSM controller is able to achieve. Future work will attempt to overcome the cycle problem by first adopting the solution provided by Spears and Gordon–Spears (2002).

In the second set of experiments it is clear that performance for FSM increases as more robots are added to the environment. Again, much the same as KT, this is not a surprising results. More robots means more independent exploration.

Random Our final algorithm was supposed to show the baseline performance. The Random controller attempts to combine greedy exploitation with exploration in a hand crafted manner. This algorithm performs well covering roughly 45%, regardless of the number of obstacles.

Again the second set of experiments provide no new insights for the Random controller. The results improve linearly as more robots are added. Since the robots'



A. Experiment 1 *B. Experiment 2*

Figure 6.12: Shadow Coverage.

decision making ability is independent of other robots, this isn't surprising.

6.4.3 Shadow Coverage (c_s) Results

The final measurement is shadow coverage. This is a measurement of how well the controller is able to explore. If the controller goes directly towards to the goal always it will achieve a 0% shadow coverage. If the controller is optimal at exploration then the controller will achieve a 100% shadow coverage. The results can be seen in Figure 6.12.

KT KT performed very well in the final measurement as can be seen in Figure 6.12. Regardless of the number of obstacles KT appears able to get roughly 80% shadow coverage. This implies that KT has a very good separation between exploitation and exploration. Although KT performs very well at shadow coverage, this area is the most important area to increase performance. Greater exploration will not only affect the shadow coverage but also the total coverage.

As robots are added to the system, the performance of KT also increases. Implied in this result, is that if robots begin to fail then the performance of KT will also

degrade slowly, which is a nice result.

AP AP's performance is about as bad as it can be in shadow coverage. At 40% obstacles AP's performance barely raises above 0% coverage. This occurs for several reasons, but the primary reason was discussed earlier. At every time step all robots controlled with the AP algorithm feel a force pulling them towards the goal. This force allows for at most minimal exploration as demonstrated by the results seen in Figure 6.12.

The results of the second set of experiments shows that even as more robots are added exploration problems still occur for AP.

Ant The Ant controller performs exactly as designed and achieves 100% coverage behind obstacles regardless of the number of robots or the number of obstacles in the environment.

FSM The FSM's performance degrades as more obstacles are added to the corridor. There are several possible explanations for this event happening. The strategy for the FSM learned by the GA may not be adept at searching shadow regions. Another problem could be that the biases we introduced required the FSM to encounter certain sensor states. With limited sensors these states are very rare even at 40% obstacle coverage. Either of these problems could explain the degradation in performance as more obstacles are added.

The results of the second set of experiments shows that as more robots are added, the FSM controller is able to explore more shadow areas. This is expected.

Random Random provides us with a usable baseline. By random search we are able to achieve roughly 20% shadow coverage. Clearly the other algorithms outperform

random search.

Our baseline algorithm's performance is increased when extra robots are added to the environment. This is another expected result.

6.5 Conclusions

Most of the results in this chapter were unsurprising, but there were several important results presented. The experiments show that Ant is superior in the spatial coverage tasks. This superiority comes at a cost though. The cost is a much slower performance time and susceptibility from adversary robots. Other than the Ant algorithm, no other algorithm was able to compete with KT. These results show the applicability of KT to the coverage task. Firstly, KT is able to show excellent coverage *without* leaving traces on the environment. Secondly, if the coverage task requires multiple sweeps KT clearly has the advantage, since temporally it was much faster than Ant. All of the results for KT shown above were also shown to be statistically significant using a Wilcoxon rank sum test with a p-value of 0.05 except for the initial total spatial coverage at 0% and 10% corridor coverage.

While KT performed reliably for all of the performance metrics, AP appeared to suffer at the coverage task. We have shown that the goal force could be responsible for some of the problems associated with AP, but there is another possible problem. The AP controller used in all of the experiments did not use the trilaterative system discussed earlier. The robots in the simulation used 24 sonar sensors mounted all around the robot. For each sensor the robot experienced a repulsive force if the distance was less than a predefined distance. This means that upon encountering a large obstacle that covered i sonar sensors, the robot felt i total forces. This possibly created an abnormally strong repulsive force from any object that covered several

sonar sensors. Although the results were not superior for AP, we can show that they were statistically significant. Again we used a Wilcoxon rank sum test with a p-value of 0.05 and all AP results are significant except temporal coverage comparisons between Ant and AP.

This could be possibly remedied by taking the derivative of the sonar data received in order to determine the starting and ending points of the object blocking the sonar. Once these endpoints have been discovered the AP robot could then feel a repulsive force from the center of the object. This would result in much softer turns, as opposed the the hard turns seen in the simulator.

Chapter 7

Final Thoughts

7.1 Summary

We have successfully created two new controller algorithms for controlling large groups of autonomous robots (Spears, Spears, Heil, Kerr, and Hettiarachchi 2004). Both new algorithms only require limited sensors and communication between robots. This provides a means to generate large numbers of inexpensive robots capable of achieving our coverage task. The algorithms are also fault-tolerant, as robots fail the other robots act independently of each other.

Both controllers provide behavioral assurances for controlling the swarm (Kerr, Spears, Spears, and Thayer 2004; Spears and Kerr 2004). These assurances allow system designers to omit trial and error and directly determine how to set system parameters to achieve a desired effect. The robot controllers complement each other since one is deterministic and the other is stochastic. If robot predictability is required a system designer can choose to use AP; however if stealthy behavior is required they can choose to use KT. KT has the advantage that its behavior is predictable in the *aggregate* as opposed to the individual level.

Neither of the new algorithms are able to guarantee complete coverage, but KT is able to achieve good coverage for our coverage task, confirming our belief that gases are well suited to this task (Kerr and Spears 2005). AP suffers slightly more because

it doesn't explore as much as the KT controller. This provides AP with a faster navigation of the environment, but at the cost of leaving unexplored areas.

7.2 Future Work

Robot Neither of the new algorithms presented have been ported to actual robots. This is the highest priority of future work. Once the algorithms are implemented on our laboratory robots, more experimental results would strengthen the results we were able to achieve. The AP solid algorithm is already run on the UW Distributed Laboratory robots (see Figure 7.1). These robots currently utilize IR sensors in order to detect other robots within their range. They also use a light sensor in order to detect the goal direction. This is all that is needed in order to port the AP gas algorithm to those same robots. Unfortunately the KT algorithm requires some upgrades to our existing robots. KT needs the trilaterative sensing algorithm presented in (Heil 2004). This algorithm allows the robot to determine if what it has sensed is a robot or an obstacle, something very important for the KT algorithm. Another issue delaying the port is that there is currently no means for explicit communication between robots. Because of these issues, we developed the detailed robotic simulator and prepared the algorithms for that simulator.

Behavioral Assurance The theory presented shows that we can predict different aspects of a swarm's behavior. The theory for KT has shown that most of these predictions scale when obstacles are added to corridors. Unfortunately we are unable to provide similar data for AP gas. Not only does this provide an interesting study of the behavior of AP gas, but it also provides insights to the theory and how it can be expanded. This is something that needs to be accomplished to further guarantee the behavior. The KT theory isn't predictive for velocity distribution when obstacles

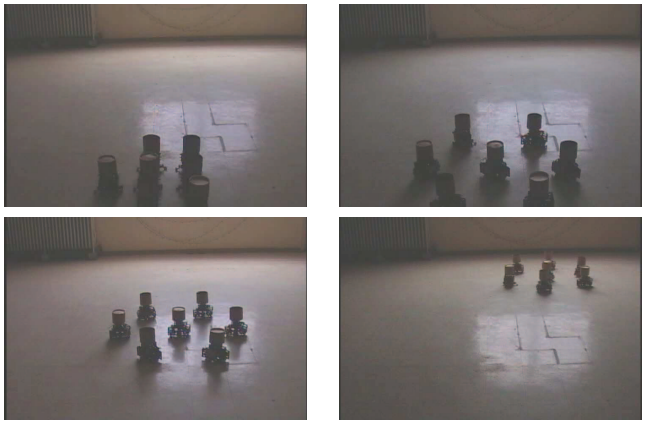


Figure 7.1: Seven robots form a hexagon, and move towards a light source.

are added to the environment, so we need to extend the theory to be more predictive of this behavior.

The theory presented affects the coverage for the swarm, but we are unable to predict the actual coverage. This is another area that needs to be expanded with future work. The Ant algorithm presented is able to guarantee complete coverage and empirically we are able to compete with this Ant algorithm, but theoretical guarantees would be beneficial.

Physics We have shown that the KT algorithm performance models a gas accurately. Unfortunately the same cannot be said for the AP gas model provided. We have provided reasons why this could be occurring, and these warrant further investigation. This also may be accomplished by removing the goal force since it appears to be overbearing. We understand that the KT model accurately conserves momentum in inter-robot collisions, but we haven't fully studied the momentum transfer between wall collisions and how it affects the energy of our system.

Our two new gas algorithms modified different types of fluid flow. The AP algorithm resembled a pressure driven flow and the KT algorithm modeled a Couette

flow. These two types of fluid flow are not encompassing. Future work will look at how different types of flow affect performance. Our KT model resembles a low density gas evacuated through a vacuum tube. Understanding the real particle physics of this representation and the dynamics of mixed gases could lead to greater insight for our KT algorithm and future gas models.

We also argued that gas behaviors were desired for the task, but we didn't present any empirical results for liquid behaviors. The primary reason for this is that not as much is known about liquids as gases and therefore finding a good model of a liquid is a challenging task.

KT is able to avoid box canyons. This emergent behavior was not witnessed on the AP controllers. AP needs this behavior in order to increase the performance of the algorithm. There are several different methods to accomplish this (Lee and Arkin 2001; Balch and Arkin 1993).

Environment All of our experiments were run in corridors much longer than wide and the controllers were only allowed to make one sweep. Extensions to this work could show that for a perimeter task a higher temporal coverage is desirable, since it requires multiple sweeps. This surveillance task is similar to the task presented in (Spears, Zarzhitsky, Hettiarachchi, and Kerr 2005). The goal for the robots is to detect as many of the targets as possible in a given time. Rather than using a square environment, the robots would be presented with a long corridor.

All of the obstacles presented have been rectangles. Rectangular obstacles have desirable computational properties, i.e. determining shadow coverage, but the real world is full of obstacles of many different shapes and sizes. These shouldn't present any large problems for the robot control algorithms, but empirical results have not been presented. In addition to random obstacles we need to investigate randomly

shaped and sized corridors. Emperical results need to be presented since straight corridors tend to be rare.

Software Development Throughout the course of the project, different simulators were generated to try out ideas and run experiments. Greater understanding and preparation would have led to better design decisions and possibly less simulators. From this a simulation framework could have been developed that allows both particle (point-mass) simulations as well as robotic simulations. By providing both of these abilities researchers would be able to transition from pure physics-based controllers to robotic controllers much more easily.

BIBLIOGRAPHY

Anderson, J. (1995). *Computational Fluid Dynamics*. McGraw-Hill.

Arkin, R. (1989). Motor schema-based mobile robot navigation. *International Journal of Robotics Research 8*(4), 92–111.

Balch, T. and R. Arkin (1993). Avoiding the past: A simple, but effective strategy for reactive navigation. In *International Conference on Robotics and Automation*, pp. 678–685.

Balch, T. and R. Arkin (1998). Behavior-based formation control for multi-robot teams. *IEEE Trans. on Robotics and Autom. 14*(6), 926–939.

Balch, T. and M. Hybinette (2000). Social potentials for scalable multi-robot formations. In *IEEE Int. Conf. on Robotics and Automation*, Volume 1, pp. 73–80.

Batalin, M. and G. Sukhatme (2002). Spreading out: A local approach to multi-robot coverage. In *Proceedings of the 6th International Symposium on Distributed Autonomous Robotics Systems*, pp. 373–382.

Bayazit, O., J. Lien, and N. Amato (2002). Better group behaviors in complex environments with global roadmaps. In *Proceedings Int. Conf. on the Simulation and Synthesis of Living Systems*, pp. 362–370.

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation 2*(1), 14–23.

Bruemmer, D., D. Dudenhoeffer, M. McKay, and M. Anderson (2002). A robotic swarm for spill finding and perimeter formation. In *Spectrum 2002*.

Butler, Z. (2000). *Distributed Coverage of Rectilinear Environments*. Ph. D. thesis, Carnegie Mellon University.

Butler, Z., A. Rizzi, and R. Hollis (1999). Cooperative coverage of rectilinear environments. In *Proc of IEEE Int'l Symposium on Intelligent Control*.

Choset, H. (2000). Coverage of known spaces: The boustrophedon cellular decomposition. *Autonomous Robots 9*(1), 247–253.

Choset, H., E. Acar, A. Rizzi, and J. Luntz (2000). Exact cellular decompositions in terms of critical points of morse functions. In *IEEE International Conference on Robotics and Automation*.

Choset, H. and P. Pignon (1997). Coverage path planning: The boustrophedon decomposition. In *Int. Conf on Field and Service Robotics*.

Cortes, J., S. Martinez, T. Karatas, and B. Francesco (2002). Coverage control for mobile sensing networks. In *IEEE International Conference on Robots and Automation*, pp. 1327–1332.

de Carvalho, R., H. Vidal, P. Vieira, and M. Ribeiro (1997). Complete coverage path planning and guidance for cleaning robots. In *Industrial Electronics*, Volume 2, pp. 677–682.

Decuyper, J. and D. Keymeulen (1991). A reactive robot navigation system based on a fluid dynamics metaphor. In H.-P. Schwefel and R. Männer (Eds.), *Parallel Problem Solving from Nature (PPSN I)*, Volume 496 of *Lecture Notes in Computer Science*, pp. 356–362. Springer-Verlag.

Dudenhoeffer, D., D. Bruemmer, M. Anderson, and M. McKay (2001). Development and implementation of large-scale micro- robotic forces using formation behaviors. In *Proceedings of SPIE, Unmanned Ground Vehicle Technology*.

Feynman, R., R. Leighton, and M. Sands (1963). *The Feynman Lectures on Physics*. Addison-Wesley Publishing Company.

Fredslund, J. and M. Matarić (2002). A general algorithm for robot formations using local sensing and minimal communication. *IEEE Transactions on Robotics and Automation 18*(5), 837–846.

Gage, D. (1992). Command control for many-robot systems. *Unmanned Systems 10*(4), 28–34.

Gage, D. (1993). Randomized search strategies with imperfect sensors. In *Proceedings SPIE Mobile Robots VIII*, pp. 270–279.

Garcia, A. (2000). *Numerical Methods for Physics* (Second ed.). Prentice Hall.

Goldberg, D. and M. Matarić (2002). Design and evaluation of robust behavior-based controllers. In *Robot Teams: From Diversity to Polymorphism*. Nature Publishing Group.

Gordon-Spears, D. and W. Spears (2002). Analysis of a phase transition in a physics-based multiagent system. In *Proceedings of FAABS II*.

Heil, R. (2004). A trilaterative localization system for small robots in swarms. Master's thesis, University of Wyoming.

Hettiarachchi, S. and W. Spears (2005). Moving swarm formations through obstacle fields. In *International Conference on Artificial Intelligence*.

Horling, B., R. Vincent, R. Mailer, and J. Shen (2001). Distributed sensor network for real time tracking. In *Agents'01*, pp. 417–424.

Howard, A., M. Matarić, and G. Sukhatme (2002). An incremental self–deployment algorithm for mobile sensor networks. *Autonomous Robots 13*(2), 113–126.

Jantz, S. (1998). Toward a synthetic ecosystem: Group behavior of autonomous mobile robots. Master's thesis, University of Florida.

Jantz, S. and K. Doty (1997). Kinetics of robotics: The development of universal metrics in robotic swarms. Technical report, Dept of Electrical Engineering, University of Florida.

Kadrovach, B. and G. Lamont (2002). A particle swarm model for swarm-based networked sensor systems. In *Proceedings of the 2002 ACM symposium on Applied computing*, pp. 918–924.

Kerr, W. and D. Spears (2005). Robotic simulation of gases for a surveillance task. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'05)*.

Kerr, W., D. Spears, W. Spears, and D. Thayer (2004). Two formal fluids models for multiagent sweeping and obstacle avoidance. *Lecture Notes in Artificial Intelligence 3228*.

Koenig, S. and Y. Liu (2001). Terrain coverage with ant robots: A simulation study. In *Agents'01*, pp. 600–607.

Koenig, S., B. Szymanski, and Y. Liu (2001). Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence 31*, 41–76.

Latombe, J. (1991). *Robot Motion Planning*. Kluwer.

Lee, J. and R. Arkin (2001). Learning momentum: Integration and experimentation. In *IEEE International Conference on Robotics and Automation*, pp. 1975–1980.

Matarić, M. (1995). Designing and understanding adaptive group behavior. *Adaptive Behavior 4*(1), 51–80.

Megerian, S., F. Koushanfar, M. Potkonjak, and M. Srivastava (2005). Worst and best-case coverage in sensor networks. *IEEE Transactions on Mobile Compting 4*(1), 84–92.

Oh, J. S., Y. H. Choi, J. B. Park, and Y. Zheng (2004). Complete coverage navigation of cleaning robots using triangular-cell-based map. *IEEE Transactions on Industrial Electronics 51*(3), 718–726.

Preparata, F. and M. Shamos (1985). *Computational Geometry: An Introduction*. Springer-Verlag.

Ranganathan, A. and S. Koenig (2003). A reactive robot architecture with planning on demand. In *IEEE International Conference on Intelligent Robotics and Systems*, Volume 2, pp. 1462–1468.

Reif, F. (1965). *Fundamentals of Statistical and Thermal Physics*. McGraw-Hill.

Reif, J. and H. Wang (1999). Social potential fields: A distributed behavioral control for autonomous robots. *Robotics and Autonomous Systems 27*(3), 171–194.

Rekleitis, I., V. Lee-Shue, A. P. New, and H. Choset (2004). Limited communication, multi-robot team coverage. In *2004 IEEE International Conference on Robotics and Automation*.

Spears, D. and W. Kerr (2004). Fluid-like swarms with predictable macroscopic behavior. *To Appear*.

Spears, W. and D. Gordon (1999). Using artificial physics to control agents. In *IEEE International Conference on Information, Intelligence, and Systems*, pp. 281–288.

Spears, W. and D. Gordon-Spears (2002). *Theory and Applications of Evolutionary Computation: Recent Trends*, Chapter Evolution of Strategies for Resource Protection Problems, pp. 367–392. Springer-Verlag.

Spears, W., D. Gordon-Spears, J. Hamann, and R. Heil (2004, August). Distributed, physics-based control of swarms of vehicles. *Autonomous Robots 17*, 137–162.

Spears, W., D. Spears, and R. Heil (2004). A formal analysis of potential energy in a multiagent system. In *Proceedings of FAABS III*.

Spears, W., D. Spears, R. Heil, W. Kerr, and S. Hettiarachchi (2004). An overview of physicomimetics. *Lecture Notes in Computer Science, State-of-the-Art Series 3342*.

Spears, W., D. Zarzhitsky, S. Hettiarachchi, and W. Kerr (2005). Strategies for multi-agent surveillance. In *IEEE Networking, Sensing and Control*.

Stark, H. and J. Woods (1986). *Probability, Random Processes, and Estimation Theory for Engineers*. Prentice-Hall.

Svennebring, J. and S. Koenig (2002). Towards building terrain-covering ant robots. *Lecture Notes in Computer Science 2463*, 202–215.

Svennebring, J. and S. Koenig (2003). Trail-laying robots for robust terrain coverage. In *Proceedings of International Conference on Robotics and Automation*, pp. in print.

Vaughan, R., K. Stoey, G. Sukhatme, and M. Matarć (2000). Whistling in the dark: Cooperative trail following in uncertain localization space. In *Proceedings of the International Conference on Autonomous Agents*, pp. 187–194.

Wagner, I. and A. Bruckstein (1995). Cooperative cleaners: a study in ant-robotics. Technical Report CIS-9512, Center for Intelligent Systems.

Wagner, I., M. Lindenbaum, and A. Bruckstein (1996). Cooperative covering by ant-robots using evaporating traces. Technical Report CIS-9610, Center for Intelligent Systems.

Wagner, I., M. Lindenbaum, and A. Bruckstein (1998). Effeciently searching a graph by a smell-oriented vertex process. *Annals of Mathematics and Artificial Intelligence 24*, 211–223.

Wagner, I., M. Lindenbaum, and A. Bruckstein (1999, October). Distributed covering by ant–robots using evaporating traces. *IEEE Transactions on Robotics and Automation 15*(5), 918–933.

Wong, S. and B. MacDonald (2003). A topological coverage algorithm for mobile robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'03)*, pp. 1685–1690.

Wong, S. and B. MacDonald (2004). Complete coverage by mobile robots using slice decomposition based on natural landmarks. *Lecture Notes in Computer Science 3157*, 683–692.

Zelinsky, A., R. Jarvis, J. Byrne, and S. Yuta (1993). Planning paths of complete coverage of an unstructured environment by a mobile robot. In *Proceedings of International Conference on Advanced Robotics*.