

Robotic Simulation of Gases For a Surveillance Task

Wesley Kerr and Diana Spears
Department of Computer Science
University of Wyoming
Laramie, Wyoming, 82071, United States
{wkerr, dspears}@cs.uwyo.edu

Abstract—The task addressed here requires a swarm of mobile robots to monitor a long corridor, i.e., by sweeping through it while avoiding large obstacles such as buildings. In the case of limited sensors and communication, maintaining spatial coverage – especially after passing the obstacles – is a challenging problem. Note that the main objective of this task is coverage. There are two primary methods for agents to achieve coverage: by uniformly increasing the inter-agent distances, and by moving the swarm as a whole. This paper presents a physics-based solution to the task that is based on a kinetic theory approach; our solution achieves both forms of coverage. Furthermore, the paper describes how we transition from our original algorithm to an algorithm utilizing mostly local sensor information, the latter being more realistic for modeling robots. To determine how well our kinetic theory approach performs against a popular alternative controller, experimental comparisons are presented.

Index Terms—swarms, robotics, coverage, surveillance.

I. INTRODUCTION

The surveillance task being addressed is that of sweeping a large group of robots through a very long bounded region (a swath of land, a corridor in a building, a city sector, or an underground passageway/tunnel), to perform a search requiring maximum coverage. The robots are assumed to have limited sensing range for detecting other agents/objects. All robots can sense the global direction to move (e.g., with light sensors), but all other sensing is strictly local. As the robots move, they need to avoid large obstacles (e.g., buildings). This search might be for enemy mines (de-mining), survivors of a collapsed building, or the robots might be acting as sentries by patrolling the area for protection. We assume that the robots need to keep moving because there are not enough of them to view the entire length of the region at once.

The robots begin at one end of a corridor and the task is to move to the opposite end of the corridor (the “goal direction”). This constitutes a “sweep.” Recall that the objective is to maximize coverage, and we are primarily concerned with the success rate for one sweep. While maximizing spatial coverage, the second swarm objective is to maximize temporal coverage by minimizing the sweep time.

Gases are excellent at achieving this task. The following properties of gases convinced us to choose them as a model for our algorithm for this task: they are easily deformed, they are capable of rejoining after parting around an object, and they naturally fill volumes.

II. PREVIOUS WORK

Although swarms of insects and flocks of birds are adequate at solving our task, we prefer a *physicomimetic* over a biomimetic approach. Though biomimetics is effective for swarm design, it produces systems that are difficult to analyze [14]. The biomimetic method with which we compare (see below) is the ant robot approach presented by Koenig and Liu [12], which is guaranteed to yield complete coverage. Information is passed by leaving traces in the environment. Our approach, on the other hand, does not require traces.

Other work in this area deals with creating ad-hoc sensor networks that achieve complete coverage. Batalin and Sukhatme [4] created control algorithms to maximize coverage by increasing inter-agent distances. Each robot moves away from the other robots until coverage is maximized. Megerian et al. [15] provide different approaches guaranteeing coverage by using theoretical graph proofs. While all these approaches are successful at maximizing sensor coverage, they implicitly assume a sufficient number of robots to cover the area. Our approach makes no such assumption, and achieves coverage by sweeping.

Another approach examines different decompositions of the corridor to cover [16]. In these decompositions, space is separated into cells that can be covered by sweeping back and forth. By creating simple cell decompositions these approaches guarantee complete coverage. To accomplish this, the robots must maintain an internal representation of the world in order to determine different decomposition points. This is another constraint we avoid because internal representations require not only more memory but also localization.

Other behavior-based approaches investigate the usefulness of formations while navigating obstacle courses [3], [9]. While this is beneficial for certain tasks, formations provide little benefit for our proposed task. The final work by Bayazit et al. [5] attempts to combine both behavior-based models and global roadmaps. This work successfully navigates the world, but assumes global information known *a priori*.

III. PHYSICS MOTIVATION

In response to the issues described above, Spears and Gordon developed the *physicomimetics* framework [17] for controlling large groups of agents using virtual forces. AP scales well to swarms of agents, and uses only local interactions and no traces [18]. It can model solids, liquids or gases. *Most importantly, multiagent systems designed with physicomimetics*

are easily analyzed using physics-based analysis methods [18]. This enables one to control, predict, and guarantee that the behavior of a swarm will remain within desirable bounds. Our work is an extension of the original AP. Our control algorithm was designed with these analyses, as presented in [11]. Here, a gas state is modeled.

There are two main methods available when designing an algorithm to model a gas: the Eulerian approach, which models the gas from the perspective of a finite volume fixed in space through which the gas flows (which includes computational fluid dynamics), and the Lagrangian approach (which includes *kinetic theory*), in which the frame of reference moves with the gas volume [1]. Because we are constructing a model from the perspective of the agents, we choose the latter. Gas behaviors are amenable to straightforward physics analyses. For example, Jantz and Doty [10] focus on kinetic theory (KT) analyses. They develop simple, elegant formulas that do a good job of predicting robot collision frequency, collision cross section, and the rate of multi-robot effusion through a hole in a wall. Nevertheless, other than our earlier work [11], the research presented here is the first of which we are aware that has explored kinetic theory for swarm *design*.

IV. KINETIC THEORY OF GASES

We created an agent controller based on kinetic theory that is a realistic gas model where robots behave as gas particles [11]. Let us first consider the original decisions for the KT algorithm. This overview borrows from Garcia [6].

The number of particles in a cubic centimeter is huge – 2.687×10^{19} . There is not enough computational power to model a gas deterministically. A typical solution is to employ a stochastic Monte Carlo model that calculates and updates the probabilities of where the particles are and what their velocities are. This is the foundation for KT.

In KT, particles are treated as possessing no potential energy, and particle collisions are modeled as purely elastic collisions that maintain conservation of momentum and energy. One advantage of this model is that it enables us to make stochastic predictions, such as the average behavior of the ensemble as shown in [11] and [6]. The second advantage with real robots is that the probabilistic robot actions avoid individual robot predictability, e.g., for stealth.

For our simulations, we modeled 2D Couette flow, although other flow problems can be modeled as well. The original algorithm was derived from Garcia [6]. Fig. 1 shows a schematic for this one-sided Couette flow, where gas is moving between two walls – one moving with velocity v_{wall} , and the other stationary. Because the gas is Newtonian and has viscosity, there is a linear velocity profile across the system. Gas deformation is due to the shear stress, τ , and 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. We chose a Couette flow so that we can introduce energy into the system and give the particles a direction to move.

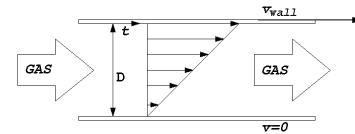


Fig. 1. Schematic for a Couette flow

V. ORIGINAL KT ALGORITHM

We created a 2D simulation world with a pair of corridor walls (considered Couette walls), obstacles, and agents (modeled as gas particles). The gas flow is unsteady and laminar i.e., no turbulence. 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. (Note that wall movement is virtual.) We invented this variant as an effective means of propelling all agents in a desired general direction, i.e., the large-scale gas motion becomes that of the walls.

Particle velocities are initially randomly chosen, and they remain constant unless collisions occur. 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. Each agent is described by a position vector \vec{x} and a velocity vector \vec{v} . At each time step, every agent calculates its new position 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 and from a Gaussian distribution in the y -direction [6]. 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 processed. The number of collisions in any given region is a stochastic function of the number of agents in that region [6]. This process continues until a desired state is achieved.

This algorithm has been implemented and tested in our 2D task simulation. Not only is performance excellent, but the bulk swarm movement has been shown to be highly predictable, using simple KT mathematical formulas [11] for prediction. This demonstrates the enormous advantage of designing multi-robot systems using KT – their behavior is predictable in the aggregate.

Finally, 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.

VI. THE ROBOTS

We decided to modify our KT algorithm to use a more realistic model our robots. They are land-based robots with a

small set of sensors and low computational power. The robot’s current heading is its x -axis. It is also able to determine its y -axis as an orthogonal axis. With this coordinate system, the robot sensors, and our localization system [8], [18], each robot is able to determine the range and bearing to other nearby robots. Each robot has 24 different sonar sensors, spaced evenly every 15° , to detect walls/obstacles, plus the capability of low bandwidth RF communication. Finally, each robot is assumed to have a sensor, such as a light sensor, for detecting the goal direction.

VII. ROBOT-FAITHFUL KT ALGORITHM

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 robot 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. *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 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 every time step. Here, we describe wall collisions first, then inter-robot collisions.

A. 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 in motion. Therefore, the first portion of our algorithm 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 it is in motion. For simplicity, we sometimes call any wall that is parallel to the goal direction (and thus virtually 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.

For the following description of our algorithm, assume that robot A is the current robot doing the processing. 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. The robot processes a wall collision if any of these sensors are 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 it. We chose this approach as opposed to increasing the safe distance for the heading sensor. Fig. 2 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. 2). First, the robot identifies the “essential endpoints,” which allow the robot to determine the corresponding plane 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. To find the second endpoint A moves positively through the sensors beginning at -45° . The essential endpoints are at 45° and -45° in Fig. 2. 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.

Now that the robot has identified two wall endpoints, 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 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.

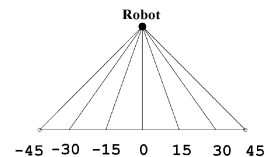


Fig. 2. The sonar information used to determine the orientation of a wall. The sonar at 0° represents the robot’s current heading.

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. 3. 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 . Robot A defines all walls with

$\alpha = 0^\circ + \varepsilon$ or $\alpha = 180^\circ - \varepsilon$ as parallel walls, where ε is a user defined error parameter based on the type of obstacles the robot expects to encounter.

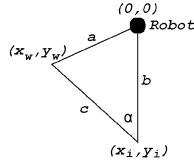


Fig. 3. The information used to determine the correct angle of the wall relative to the goal.

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, which has changed due to collision with a wall. Ideally, in global coordinates, 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, v_{wall} is added to the y -direction.

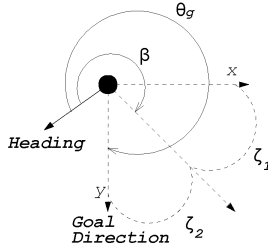


Fig. 4. The information locally used to determine the correct angle to turn when recalculating velocity.

These ideal (global coordinates) velocities need to be converted into real (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, 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 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.

B. 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 communication – to ensure that only one robot processes the collision, which is needed because of the random component inherent in KT.

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 determine if there are any other unprocessed robots left to collide with. One issue is that velocities are reset using a random collision angle; therefore, robots could collide repeatedly. To counter this, collisions can only be processed again for a pair of robots after a predetermined number of time steps have occurred.

Let us consider 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 . 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.

Assume that robot B has not already processed this collision. Robot A requests the bearing from B to A , 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. 5 shows the available information and is a good reference for understanding the equation. We only present one case; the other three are symmetric.

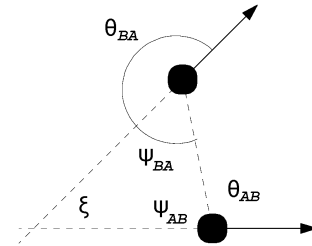


Fig. 5. Case I. $\xi > 0^\circ$ and $\xi < 90^\circ$. Initial case for determining robot B 's velocity in robot A 's coordinate system.

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

$$\xi = 180 + \theta_{AB} - \theta_{BA} \quad (1)$$

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 from communication. 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:

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

Once A has determined what velocities to assign each robot, it transmits the information to robot B . For robot B to understand the new velocity, A first translates the velocity vector into local polar coordinates, 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 (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.

C. Finishing

Executing the new velocity requires turning and then moving. Robots use \tan^{-1} of their new velocity vector to determine the direction of 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.

VIII. ANT ALGORITHM

To experimentally compare our algorithm we duplicated an algorithm designed in [13]. 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.

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 repeating the process. This update value is based on Node Counting as described in [12].

IX. EXPERIMENTAL DESIGN

To better understand the performance of our KT algorithm, we ran a set of combinatorial experiments, varying the obstacle percentage. First, we formalize the problem. Assume:

- 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 0-30% coverage.

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). The shadow region is hardest to cover.

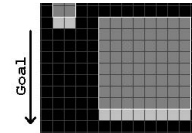


Fig. 6. Measurements taken by the simulation. Lighter gray squares measure c_s . Obstacles are colored darker gray. Free space is colored black.

Further experimental specifics are the following. We generated 5 corridors for each obstacle percentage. Each experiment consisted of 10 runs through each of the 5 corridors with random initial locations and orientations of the robots. Each corridor was run 10 times because the algorithms are stochastic. Mean performance is graphed. Robot speeds were approximately the same for all algorithms.

X. EXPERIMENTAL RESULTS

All of the same experiments were run for the KT controller, Ant controller, and a controller called “Random” that performs a random walk. *It is important to note that Ant has an inherent advantage over KT, i.e., Ant robots have superior sensorimotor capabilities (they can leave and read marked trails).* Because Random is essentially a version of Ant with these extra capabilities ablated, it is our baseline. We include comparisons with Ant for two reasons: Ant is “state-of-the-art” for coverage tasks, and we wish to see whether KT can perform almost as well as Ant, despite lacking the extra platform capabilities assumed by Ant.

Quite surprisingly, the first graph (Fig. 7) shows that KT finishes in fewer time steps and achieves better temporal coverage than *both* the Ant and Random controllers. This is because, unlike KT, the Ant and Random controllers waste time covering the same proximal areas repeatedly.

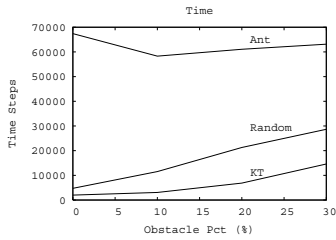


Fig. 7. Experimental results for sweep time, w , for swarms.

The next graph shows the total spatial coverage, c_c , of the two controllers. Note that KT's performance is very close to that of Ant's, and much better than Random's performance (Fig. 8). In other words, despite being hindered by a sensorimotor handicap with respect to Ant, it is capable of nearly approaching Ant's level of performance – by algorithmically mimicking the highly effective motion of a fluid, rather than by any extra platform capabilities.

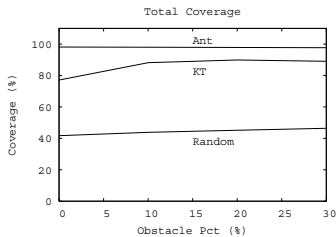


Fig. 8. Experimental results for total coverage, c_c .

In the graph of shadow coverage, c_s , (Fig. 9), of the corridor versus the percentage of obstacles, the results are similar to those of Fig. 8.

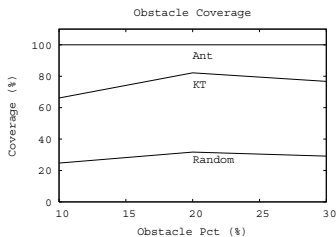


Fig. 9. Experimental results for “shadow” coverage, c_s .

It is noteworthy that the KT controller achieves nearly the performance of Ant, which takes advantage of added robot platform capabilities, in terms of spatial coverage, and it even exceeds the performance of Ant in terms of temporal coverage. Temporal coverage is important because faster sweeps imply that an intruder could be spotted sooner, especially in the case when there are few agents and a long corridor. In conclusion, KT is an excellent alternative to Ants that does not require environmental marking. This is significant for either low-cost applications or those that require stealthy surveillance with no detectable markers.

XI. CONCLUSIONS AND FUTURE WORK

Our new KT algorithm is capable of gas-like behavior, using only local information. This controller provides excellent coverage of a long corridor, even when the corridor contains large obstacles. The algorithm is decentralized, fault-tolerant, and easily scales to both large numbers of robots as well as just a few robots. The KT controller is relatively simple and requires few sensors in order to function properly. Because of this, it is cost effective to implement this control algorithm.

The next step will be to port the algorithm to the laboratory robots. We also plan to explore the possibility of using KT as one layer in a behavior-based architecture, such as that of Goldberg and Matarić [7] or Balch and Arkin [2].

REFERENCES

- [1] J. Anderson. *Computational Fluid Dynamics*. McGraw-Hill, 1995.
- [2] T. Balch and R. Arkin. Behavior-based formation control for multi-robot teams. *IEEE Trans. on Robotics and Autom.*, 14(6):1–15, 1998.
- [3] T. Balch and M. Hybinette. Social potentials for scalable multi-robot formations. In *IEEE Int. Conf. on Robotics and Automation*, volume 1, pages 73–80, 2000.
- [4] M. Batalin and G. Sukhatme. Spreading out: A local approach to multi-robot coverage. In *Proceedings of the 6th International Symposium on Distributed Autonomous Robotics Systems*, pages 373–382, 2002.
- [5] O. Bayazit, J. Lien, and N. Amato. Better group behaviors in complex environments with global roadmaps. In *Proceedings Int. Conf. on the Simulation and Synthesis of Living Systems*, pages 362–370, 2002.
- [6] A. Garcia. *Numerical Methods for Physics*. Prentice Hall, second edition, 2000.
- [7] D. Goldberg and M. Matarić. Design and evaluation of robust behavior-based controllers. In *Robot Teams: From Diversity to Polymorphism*. Nature Publishing Group, 2002.
- [8] R. Heil. A trilateral localization system for small robots in swarms. Master's thesis, University of Wyoming, 2004.
- [9] S. Hettiarachchi and W. Spears. Moving swarm formations through obstacle fields. In *International Conference on Artificial Intelligence*, 2005.
- [10] S. Jantz and K. Doty. Kinetics of robotics: The development of universal metrics in robotic swarms. Technical report, Dept of Electrical Engineering, University of Florida, 1997.
- [11] W. Kerr, D. Spears, W. Spears, and D. Thayer. Two formal fluids models for multiagent sweeping and obstacle avoidance. *Lecture Notes in Artificial Intelligence*, 3228, 2004.
- [12] S. Koenig and Y. Liu. Terrain coverage with ant robots: A simulation study. In *Agents'01*, pages 600–607, 2001.
- [13] S. Koenig, B. Szymanski, and Y. Liu. Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence*, 31:41–76, 2001.
- [14] Y. Liu, K. Passino, and M. Polycarpou. Stability analysis of m-dimensional asynchronous swarms with a fixed communication topology. In *IEEE Transactions on Automatic Control*, volume 48, pages 76–95, 2003.
- [15] S. Megerian, F. Koushanfar, M. Potkonjak, and M. Srivastava. Worst and best-case coverage in sensor networks. *IEEE Transactions on Mobile Computing*, 4(1):84–92, 2005.
- [16] I. Rekleitis, V. Lee-Shue, A. P. New, and H. Choset. Limited communication, multi-robot team coverage. In *2004 IEEE International Conference on Robotics and Automation*, 2004.
- [17] W. Spears and D. Gordon. Using artificial physics to control agents. In *IEEE International Conference on Information, Intelligence, and Systems*, pages 281–288, 1999.
- [18] W. Spears, D. Gordon-Spears, J. Hamann, and R. Heil. Distributed, physics-based control of swarms of vehicles. *Autonomous Robots*, 17:137–162, August 2004.