Virtual Spring Mesh Algorithms for Control of Distributed Robotic Macrosensors

Brian Shucker and John K. Bennett

Department of Computer Science
University of Colorado at Boulder

Technical Report CU-CS-996-05

May 2005

Virtual Spring Mesh Algorithms for Control of Distributed Robotic Macrosensors

Brian Shucker and John K. Bennett
Department of Computer Science, 430 UCB
University of Colorado
Boulder, Colorado 80309
Email: {shucker, jkb}@cs.colorado.edu

***Abstract*— We have developed a novel control mechanism that deploys a large number of inexpensive robots as a distributed remote sensing array, called a Distributed Robotic Macrosensor (DRM). A simple virtual spring mesh abstraction is used to provide fully distributed control that is both flexible and fault-tolerant. We describe and evaluate several algorithms for virtual spring mesh–based control and present simulation results demonstrating the efficacy of the spring mesh approach.**

I. Introduction

Advances in integration, actuator design and power management have resulted in the availability of mass-produced, inexpensive robotic components. It is now feasible to build small, autonomous robots in large numbers at relatively low cost. The resulting potential to deploy robotic sensors on a large scale creates the opportunity to explore a new type of remote sensing, which does not require a pre-deployed sensor network infrastructure. We call such a remote sensing system a Distributed Robotic Macrosensor (DRM). Hundreds, or even thousands, of robots can potentially be deployed to cover and explore an area, record data, and track targets of interest. However, while it is now relatively simple to deploy large numbers of robotic sensors, the coordination and control of the activities of these robots presents a number of challenging problems, including scalability, autonomy, coverage, flexibility of deployment, fault-tolerance, and security.

In addition, there is a need for fully-distributed control algorithms for DRMs. Centralized control mechanisms are subject to compromise and do not scale adequately[1], [2], [3]. In addition, existing distributed control mechanisms have serious practical limitations, especially with respect to complex environments and unknown initial distributions[4], [5], [6].

We have developed a DRM control mechanism based upon a virtual spring mesh. Each robot chooses its actions based only upon local information and a simple physics model, defined in such a way that robots tend to act in a manner that contributes to a common goal. Thus, there is global cooperation without any global control, which makes the system both fault-tolerant and scalable. In our case, the macrosensor itself emerges as a result of the large-scale interactions of individual, nearly stateless components. The spring mesh has a number of desirable properties, described in detail below and in [7]. In this paper, we focus on the algorithms for the creation of spring meshes, and the resulting mathematical and practical properties of those meshes. We also describe some simple extensions of spring mesh control that provide target tracking ability.

II. Related Work

There is a significant body of previous work dealing with coordination of small teams of robots, e.g.[8], [9], [10], [11], [12], [13]. Target tracking has also been addressed, primarily in the context of point targets[14], [15], [16]. More recently, there has been research into behavior-based and virtual-physics based control of large teams of robots[17], [18], [19], [20], [6], [21]. The work most closely related to our own is summarized below.

A. Explicit Coordination

Explicitly coordinated exploration and mapping was examined in the "Cover Me!" [3] project, which defines a coverage metric and then uses an incremental greedy algorithm to deploy robots into locally optimal locations. This approach is not designed to scale to large numbers of robots, as it makes use of global information and only deploys one robot at a time. Similar work by Simmons et al.[4] computes desired deployment locations by attempting to minimize overlap in information gain. Explicit loosely-coupled robot coordination for arbitrary goals (not just exploration) is implemented in the ALLIANCE system[1], [22], which uses behavior-based task selection. RETSINA[2] operates a team of robots through a shared plan, which is communicated and refined over time. DINTA[23] takes a hybrid approach and uses a static sensor network to assign tasks to a set of mobile robots. This approach has many advantages, but requires a pre-deployed infrastructure. None of the explicit coordination schemes is designed to scale to very large groups of robots.

B. Target Tracking

Target tracking has been addressed within some of these systems. Targets are tracked in ALLIANCE[14] through a combination of local virtual forces and high-level behavior-based selection. Jung and Sukhatme[15] also use a multi-layered approach; their system computes a local solution for tracking groups of targets within the same field of view,

operating within a framework that distributes robots into regions according to target density. A different approach has been proposed by Gage[16], who suggests randomized search strategies that use inexpensive systems designed to make detection highly probable.

C. Behavior-based Control

Fully distributed control based upon simple local behaviors has been used in several contexts. Brooks[17] has investigated behavior-based control extensively; Werger[18] later described the design principles of such systems. Balch and Hybinette[24] suggested the use of "attachment sites" that mimic the geometry of crystals; this is used to create formations with large numbers of robots. A variety of projects have made use of "swarm robotics," e.g., [25] and [26], to carry out simple tasks such as light tracking. Gage[19] investigated the use of robot swarms to provide blanket, barrier, or sweep coverage of an area. Several researches have used models based on the interactions of ants within a colony[27], [26], [28]. These approaches generally seek to define simple local behaviors that lead to large-scale properties that are beneficial in a particular application.

D. Virtual Physics

Distributed control based on virtual physics (also called "artificial physics" or "physicomimetics") has also been investigated, although not in the manner described here. Howard, Mataric and Sukhatme[20] model robots as like electric charges in order to cause uniform deployment into an unknown enclosed area. Spears and Gordon[6], [21], [29] use a more sophisticated model analogous to the gravitational force, but make the force repulsive at close range. Both of these models use fully connected graphs, although the latter model cuts off interactions beyond a maximum range. McLurkin[30] used a partially-connected graph with a physics model similar to that of compressed springs to produce uniform deployment within a limited indoor environment.

III. Spring Mesh Control

Virtual spring meshes are an extension of virtual physics-based control. Virtual physics-based robot control is inspired by natural phenomena and has been investigated primarily in the context of swarm robotics[6], [21], [20]. The general idea is that each robot is treated as a particle in a simulated physical system, complete with virtual forces and rules of motion. While the forces exist only in simulation, the robots act in the real world as if the forces were real. The object is to define virtual forces and rules of motion in such a way that the local interactions between robots result in desirable global behavior.

Previous attempts at virtual-physics based systems have generally focused on potential fields of some kind, using force fields analogous to gravity or the electromagnetic force. In contrast, our virtual spring mesh model makes use only of explicit connections between robots. More precisely, if robots are represented as vertices in a graph and force is transmitted through edges, the spring mesh is not a fully connected graph (even locally). Instead, virtual springs are created to transmit force only between self-selected adjacent pairs of robots.

As with real springs, each virtual spring in the mesh has a natural length and a spring constant (that represents the "stiffness" of the spring). These parameters can change over time to suit varying environmental conditions, but are in general the same for every spring at any given time. This restriction is made primarily for convenience, and does not represent a fundamental property of our approach.

We define a spring mesh $\mathbf{M}$ as an undirected graph of M vertices, where the vertices represent robots and the edges represent spring connections between robots. A spring mesh may be *static*, in which the edge set and control constants are fixed, or *dynamic*, where edge set or control "constants," or both, may vary over time.

The control law for each robot is

$$\ddot{\mathbf{x}} = \big[\sum_{i \in S} k_s(l_i - l_0)\hat{\mathbf{u}}_\mathbf{i}\big] - k_d\dot{\mathbf{x}} \tag{1}$$

where $\ddot{\mathbf{x}}$ is the robot's acceleration, $\dot{\mathbf{x}}$ is the robot's velocity, S is the set of springs connected to this robot, l_i is the length of the i'th spring, and $\hat{\mathbf{u}}_\mathbf{i}$ is the unit vector from this robot to the robot on the other end of the i'th spring. Control constants are the natural spring length (l_0), the spring stiffness (k_s), and the damping coefficient (k_d).

Any static spring mesh with positive k_d will eventually converge to a stationary state, where all robots have velocity approaching zero. Intuitively, this is because the dynamics of a virtual spring are analogous to those of a real spring, in that virtual springs conserve energy. Since we ensure $k_d > 0$, there is always a damping effect acting against the motion of each robot. This forces a reduction in kinetic energy. Kinetic energy may be gained by converting potential energy stored in springs, but since springs are conservative, the total energy (potential + kinetic) in the mesh cannot increase. Since the existence of kinetic energy (motion) results in a decrease in total energy, and this energy cannot be replenished, kinetic energy must eventually approach zero.

Formally, consider the following energy function:

$$\mathbf{V} = \sum_{r \in R}(\frac{1}{2}\dot{x}_r^T\dot{x}_r) + \sum_{s \in S}\frac{1}{2}k_s(len(s) - l_0)^2 \tag{2}$$

where R is the set of all robots, S is the set of all springs, and $len(s)$ is the length of the spring s. While we omit the derivation (which is nontrivial but fairly straightforward) for brevity, we claim that the derivative of the energy function is the following:

$$\dot{\mathbf{V}} = \sum_{r \in R} -k_d\dot{x}_r^T\dot{x}_r \tag{3}$$

which is obtained by differentiating $\mathbf{V}$ and using Equation 1 to substitute in for $\ddot{x}$. As intended by our choice of control laws, all of the spring potential terms cancel out and leave only the damping terms.

Notice that $\dot{\mathbf{V}}$ is negative definite with respect to $\dot{x}$, but only negative semi-definite with respect to x. Thus, $\mathbf{V}$ is a Lyapunov function for the velocities but not the positions of the robots. It is possible for some potential energy to exist even

in a static spring mesh in its stationary state, as discussed in Section VII. In a dynamic mesh, it is of course possible to add energy by creating a new spring or modifying control constants.

IV. Simulating DRMs

We have developed a simulator in order to test the basic functionality of spring mesh algorithms. The simulator interface allows a user to quickly configure the simulated robots, change control algorithms and parameters, and alter the environment. The simulator environment has simple dynamics: robots may move in any direction at any speed up to a configurable maximum, but are stopped if they attempt to move through an obstacle. A robot may call into the simulator to get its current position and the positions of all other robots that are locally visible, optionally with some position error added. Robots may communicate with adjacent visible robots instantly.

We have also implemented a version of the simulator that supports 3-D environments and more sophisticated hardware and communication models. However, the additional complexity of the 3-D simulator makes it more appropriate for prototyping code for actual robots, and less convenient for investigating differences in high-level algorithms.

V. Candidate Spring Formation Methods

The properties of a spring mesh-based control system depend greatly on the topology of the mesh, which in turn is governed by the algorithm used by the robots to determine with which neighboring robots spring connections are created and maintained. Here we describe several candidate algorithms and the properties of the resulting meshes. We restrict the discussion to fully distributed algorithms for which no central control, global knowledge, or hierarchy among robots is necessary. We also restrict the analysis to stateless algorithms, for which robots need not maintain persistent state information about the spring mesh. The relevant mesh properties supporting these criteria include:

- Computational Complexity—Since mobile robots may have limited computational resources, simpler computations are advantageous.
- Connectivity—In a connected mesh, there exists a path between any arbitrary pair of robots for which every segment is along a spring. A connected mesh ensures that the robots will remain in one group. While describing candidate algorithms, we consider only the case of an unobstructed area of operation, where connections are not broken by obstacles in the environment.
- Symmetry—In a symmetric algorithm, the decision made at robot A concerning a connection to robot B is always the same as the decision made at B concerning a connection to A. This is desirable, because for a spring mesh to correctly implement the physics model, each pair of robots must act in a consistent manner. Thus a spring only exists if the robots at both endpoints agree. With an asymmetric algorithm, potentially expensive communication may be required in order to reach agreement. This is not required with a symmetric algorithm.

Fig. 2. An example initial configuration. The lower-left cluster contains 10 robots.

- Reference Frames—Some algorithms require that all robots share a single reference frame, while others allow each robot to perform computations in its own reference frame. The latter is preferred, since aligning multiple reference frames may be difficult, require extensive communication, and introduce a potential source of error.
- Parameters—Algorithms that take no parameters are preferred over those that require parameters, since it is desirable to minimize the difficulty of mesh configuration. Here, the term "parameter" describes a setting related to the spring formation algorithm, and the term "control constant" describes a setting (such as spring stiffness) required by all spring meshes.
- Stacking/Planarity—Some algorithms cause an effect we refer to as "spring stacking," where many springs running nearly parallel to each other act similarly to a single, ultra-stiff spring. As described below, overly stiff springs can lead to an unstable mesh, so algorithms that avoid stacking are preferred. Note that it is difficult for stacking to occur in planar meshes, where no pair of springs crosses, since planar meshes must have a low edge density.

A variety of candidate spring-formation algorithms are described below, together with examples of those algorithms applied to the initial configuration of three clusters of robots shown in Figure 2.

A. Full Connectivity

Perhaps the simplest algorithm forms a spring connection with every other robot that can be detected. While this approach is computationally simple, it results in $(M*(M-1))/2$ springs when there are M detectable robots. Since the creation of each spring requires some processing, the computational complexity of a fully connected mesh is actually quite high.

A fully connected mesh is symmetric and connected, and does not require a shared reference frame or any parameters. However, the mesh is decidedly non-planar and exhibits significant spring stacking. Stacking effectively increases the spring stiffness as more robots (and thus more springs per robot) are added. We have observed in simulation that fully connected meshes can exhibit severe instability unless they are heavily damped. Further complicating matters is the fact that the required level of damping is dependent on the number of

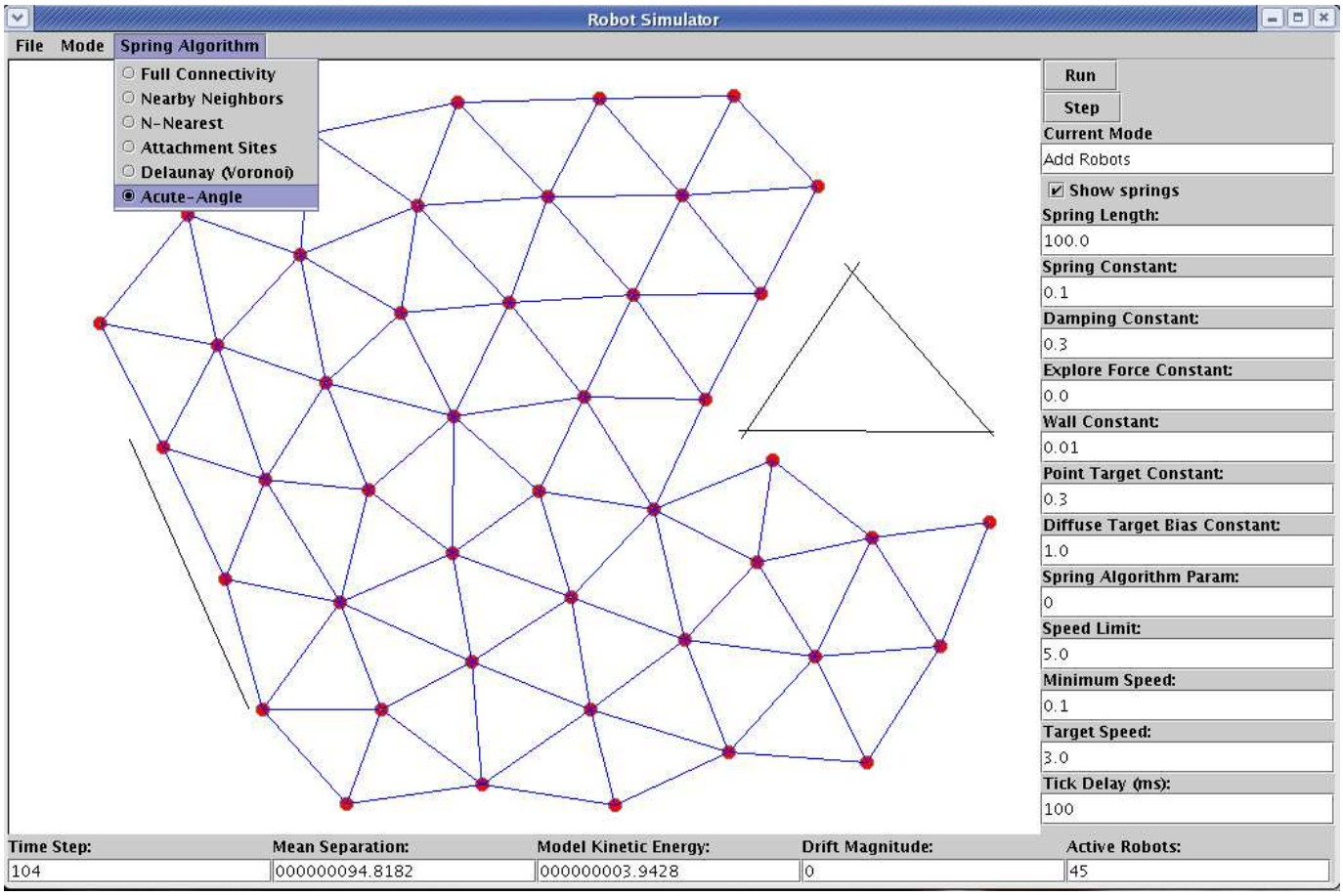


Fig. 1. DRM simulator

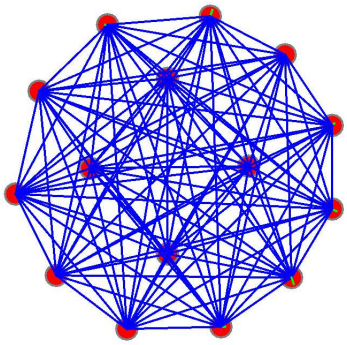


Fig. 3. Result from executing the Full Connectivity algorithm on the Figure 2 initial condition.

robots in the mesh, since the number of springs attached to each robot increases with the total number or robots.

Figure 3 depicts the result of executing the Full Connectivity algorithm, starting from the initial configuration shown in Figure 2.

B. Nearby Neighbors

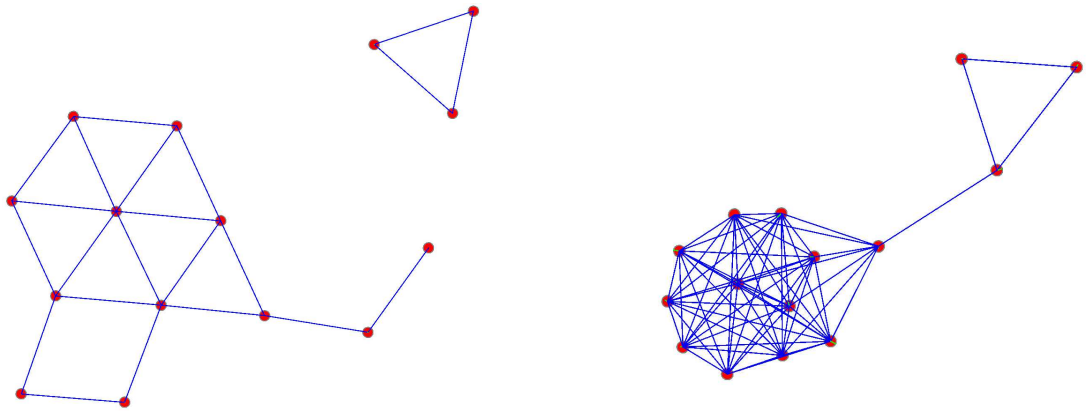
In the Nearby-Neighbors algorithm, a robot A will form a spring with all robots within a given distance (D) of robot A. The nearby-neighbor relation is symmetric and easy to compute. If D is sufficiently large, then Nearby-Neighbors reduces to Full Connectivity. If D is shorter than the natural length of a spring, then the robots will disperse until the mesh is completely disconnected.

Nearby-neighbor meshes are generally not connected or planar, and may exhibit stacking. These properties are controlled by the parameter D. With high D, connectivity is likely but spring stacking increases (and planarity is unlikely). With low D, the opposite is true. Tuning D to get both connectivity and low stacking has proven difficult in simulation. For example, if the robots start in two distinct clusters, D must be very high in order for the two clusters to connect. In that case, there will be full connectivity (and thus significant spring stacking) within the two clusters.

Figure 4 depicts the result of executing the Full Connectivity algorithm, starting from the initial configuration shown in Figure 2, for two values of D.

C. N-Nearest

Like Nearby-Neighbors, the N-Nearest algorithm is a simple restriction of Full Connectivity. In this case, each robot connects to the N nearest robots, where N is a configurable parameter. This is slightly more complex to compute than Nearby Neighbors ($O(MlogM)$, as opposed to $O(M)$, for M robots). Unlike Nearby Neighbors, N-Nearest is not symmetric, so communication between robots is required. This



(a) $D = 1.2 * l_0$ (b) $D = 1.6 * l_0$

Fig. 4. Results from executing Nearby Neighbors algorithm on Figure 2 initial condition, with two different values for D.

asymmetry is evident in Fig. 5(c), where the robots in the right cluster all have 5th–nearest through 8th–nearest neighbors in the left cluster, but robots in the left cluster have all 8 nearest neighbors in their own cluster.

Simulation suggests that with $N < 5$, the N-Nearest mesh is relatively sparse and causes the robots to disperse. For $N = 5$ and $N = 6$, the mesh has little spring stacking and is generally connected, although it is possible to get distinct clusters. With $N > 6$, spring stacking increases, as seen (in a somewhat extreme case) in 5(d). Unfortunately, N must become very large in order to ensure that all clusters of robots connect with each other, which results in more spring stacking and potentially instability.

D. Attachment Sites

The Attachment Sites algorithm was first proposed by Balch[24]. It works by modeling each robot as a K–sided object, with one attachment site on each side. This attachment site will form a spring connection with the nearest robot on its side. Computation is roughly the same complexity as N-Nearest—instead of sorting one list, it involves checking the angle to each robot and then sorting K smaller lists.

A regular hexagonal mesh results in uniform spacing of the robots, so the natural choice for K is 6 ($K = 8$ produces a roughly square mesh). The $K = 6$ mesh is connected and planar, and suffers no stacking, regardless of the initial distribution of robots. However, there are some practical disadvantages to this approach. The algorithm is not symmetric, so additional communication is required. Also, the robots must share a common reference frame in order for the mesh to align into a regular shape. Without a common directional reference, the facing of the robots' sides will likely not be consistent.

Figure 6 depicts the result of executing the Attachment Sites algorithm, starting from the initial configuration shown in Figure 2.

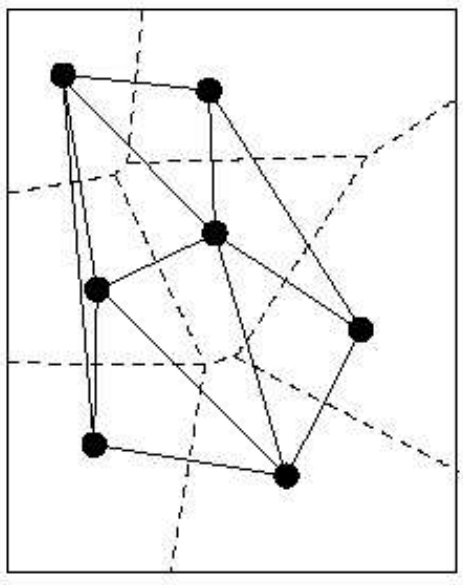
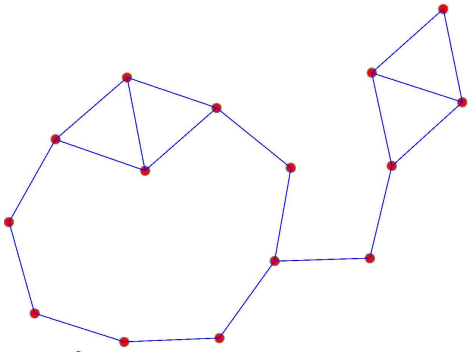


Fig. 7. Example of a Delaunay graph

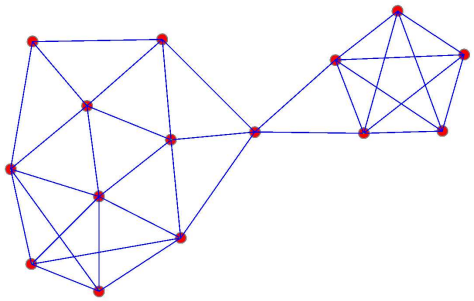
E. Delaunay

The Delaunay graph is related to the more well-known Voronoi diagram[31]. In a Voronoi diagram, a plane that contains some number of point-like sites is divided into regions, with each region centered on one site. The region boundaries are drawn so that for a site A, every point inside site A's region is closer to site A than to any other site. In a Delaunay graph, the sites are the vertices and an edge exists between any two sites whose Voronoi regions share a boundary. Figure 7 depicts this relationship; the Voronoi boundaries are shown in dotted lines and the edges of the Delaunay graph as solid lines. McLurkin[30] suggested using Delaunay graphs in a control application similar to spring meshes, although he apparently did not implement this approach.

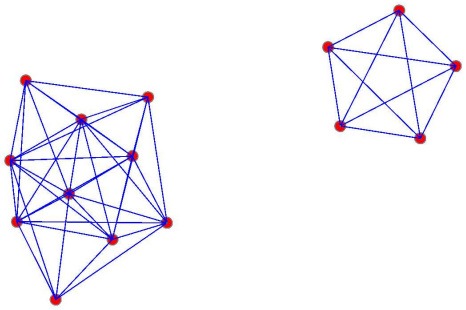
A minor drawback of the Delaunay graph is computational complexity; a straightforward implementation runs on each



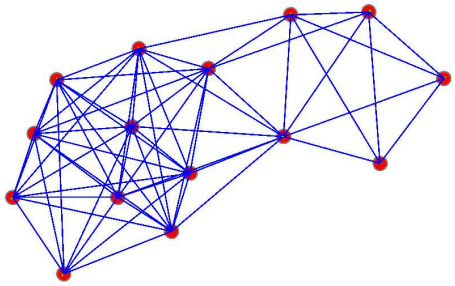
(a) $N = 3$



(b) $N = 5$

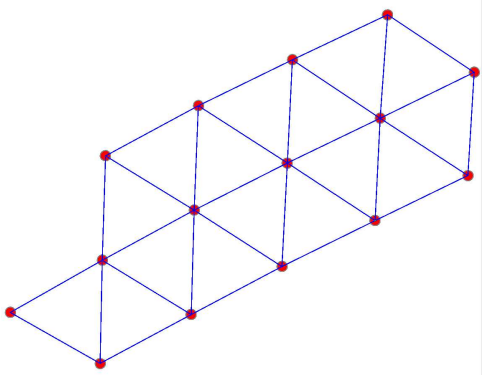


(c) $N = 8$

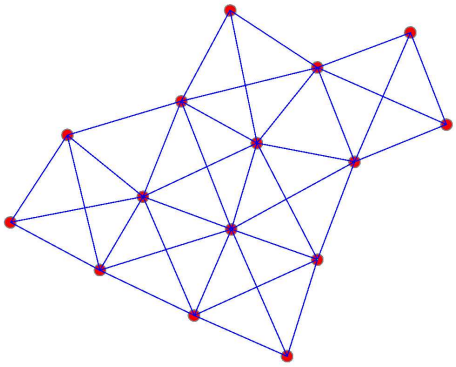


(d) $N = 10$

Fig. 5. Results from executing N-Nearest algorithm on Figure 2 initial condition, with various values of N.



(a) $K = 6$



(b) $K = 8$

Fig. 6. Results from executing Attachment Sites algorithm on Figure 2 initial condition, with two different values for K.

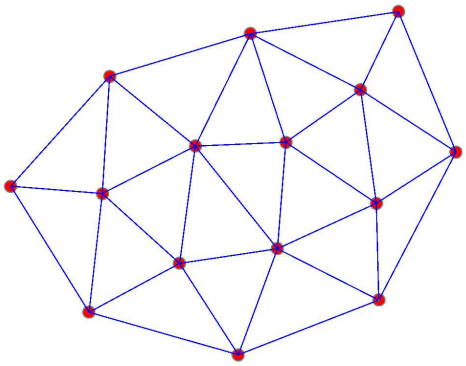


Fig. 8. Result from executing the Delaunay algorithm on Figure 2 initial condition.

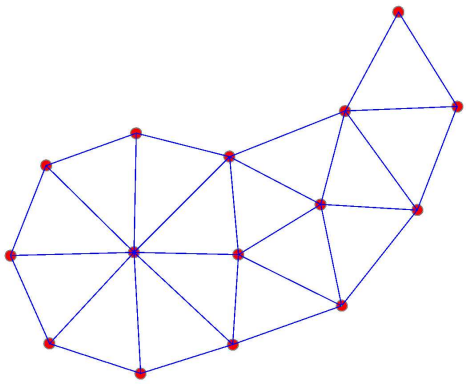


Fig. 10. Result from executing the Acute Angle algorithm on Figure 2 initial condition.

robot in $O(M^2)$ time, where M is the number of visible neighbor robots (it is possible, but not straightforward, to compute the graph in $O(MlogM)$ time, as in [31]). However, this algorithm is parameter–free and symmetric, it does not require a common reference frame, and Delaunay graphs are provably connected and planar. The resulting meshes are highly stable, and a roughly uniform mesh forms regardless of the initial distribution of robots.

One unique feature of the Delaunay graph is that edges must exist along the convex hull of the mesh. This forces the overall mesh into a convex shape. This constraint represents a potential disadvantage for a DRM, particularly when tracking targets that may not occupy a convex area.

Figure 8 depicts the result of executing the Delaunay algorithm, starting from the initial configuration shown in Figure 2.

F. Acute-Angle Test

We have developed a new algorithm that is based upon an *acute-angle test*[7]. Under the Acute-Angle Test algorithm, a given robot A will form a spring to a neighboring robot B if and only if for all other neighbors C, the interior angle $\angle ACB$ is acute. This creates a mesh of acute triangles. Note that the acute-angle test is equivalent to a test for the presence of any robot C inside the circle with diameter $\overline{AB}$, which is a more efficient test to compute.

Figure 9 shows two examples of the acute-angle test. In Figure 9(a), a spring forms between A and B, since all interior angles $\angle ACB$ are acute (with $C1$, $C2$, and $C3$ as robot C in each of three tests). In Figure 9(b), the spring does not form because the acute-angle test fails with robot $C4$. The circle with diameter $\overline{AB}$ is also shown; it is equivalent to say that the spring does not form because $C4$ is inside the circle.

Like the Delaunay algorithm, the acute-angle algorithm runs in $O(M^2)$ time, but since there is less computation at each step, in practice it runs about 10 times faster.

Figure 10 depicts the result of executing the Acute-Angle Test algorithm, starting from the initial configuration shown in Figure 2.

As we will describe below, the acute-angle test is symmetric and results in a provably planar and connected graph, regardless of the initial distribution of robots. It is parameter–free, does not require a global reference frame, and does not put any constraints on the global shape of the mesh. For these reasons, the Acute-Angle Test algorithm appears to be the best all-around candidate. Sections VI and VII further explore the properties of the Acute-Angle Test algorithm.

VI. Properties of Acute-Angle Meshes

This section presents proofs that characterize several of the properties of the Acute-Angle Test algorithm. To simplify the analysis, an unobstructed environment with unlimited visibility and perfect knowledge of location has been assumed. A discussion of real-world considerations follows in Section VIII.

Definition 6.1: $A \bowtie B$ is a relation on robots A and B. $A \bowtie B$ iff $\forall$ robots C distinct from A and B, the interior angle $\angle ACB$ is acute. $A \bowtie B$ indicates that a spring exists between A and B.

Definition 6.2: $A \neg B$ is defined as (**not** $A \bowtie B$).

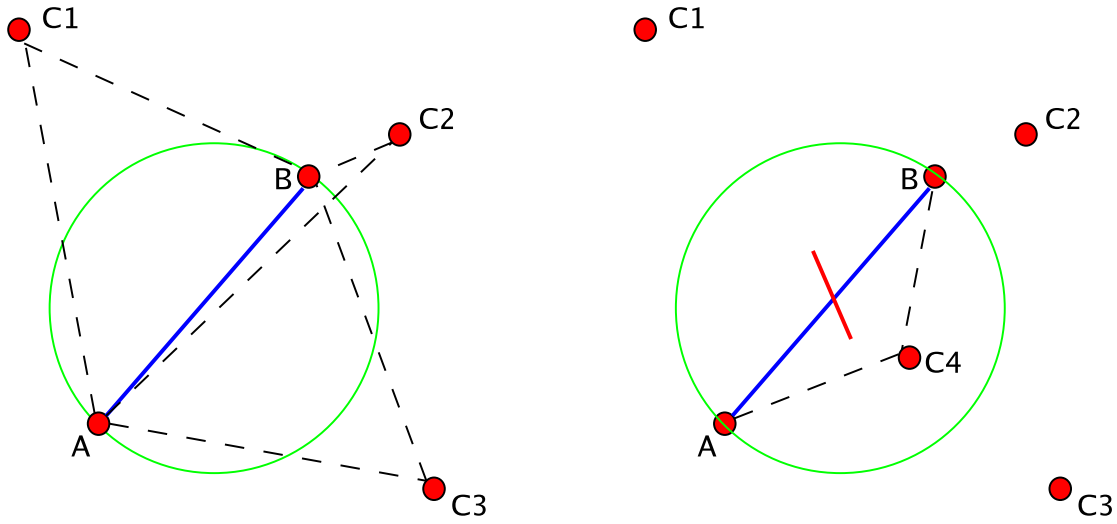
Definition 6.3: $dist(X, Y)$ represents the distance between robots X and Y.

Lemma 6.4: All robots have a spring connection to the nearest neighboring robot.

Proof: Suppose $\exists$ at least 2 robots. Pick any robot A. Then some robot B must be closest to A; that is, $\exists$ robot B, $B \neq A$, such that $\forall$ robots C distinct from A and B, $dist(A, B) \leq dist(A, C)$. We want to show $A \bowtie B$. Let b represent interior angle $\angle ABC$ and let c represent interior angle $\angle ACB$. Since $dist(A, B) \leq dist(A, C)$, we know $c \leq b$. Thus c must be acute, since $c+b < 180$ and c is the smaller of the two. This is true for any choice of robot C, which is exactly the condition that defines $A \bowtie B$. ■

Lemma 6.5: For robots A and B, if $A \neg B$ then $\exists$ robot C distinct from A and B such that $dist(A, C) < dist(A, B)$ and $dist(B, C) < dist(A, B)$. That is, if A and B are not connected then some C is closer to A and to B than they are to each other.

Proof: By definition, if $A \neg B$ then $\exists$ robot C distinct from A and B such that the interior angle $\angle ACB \geq 90$ (this is the contrapositive of the definition). With this choice of C, segment $\overline{AB}$ is the longest side of triangle ABC, since it is opposite the largest angle. Thus, $dist(A, C) < dist(A, B)$ and $dist(B, C) < dist(A, B)$. ■



(a) Satisfied acute-angle test

(b) Unsatisfied acute-angle test

Fig. 9. Illustration of acute-angle test

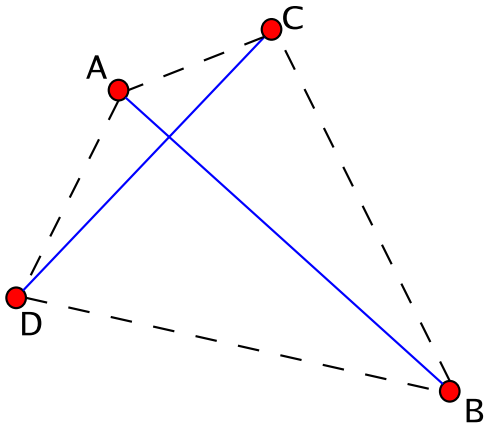


Fig. 11. Illustration of Theorem 6.6.

Theorem 6.6: Any acute-angle spring mesh is planar.

Proof: By contradiction (see Figure 11): Suppose an intersection exists. Specifically, suppose $A \bowtie B$ and $C \bowtie D$ for distinct robots A, B, C, D and $\overline{AB}$ intersects $\overline{CD}$. Consider quadrilateral $ACBD$. Some angle in any quadrilateral must be at least 90 deg. Without loss of generality, let $\angle DAC \geq 90$. Then by definition, $C \neg D$ since it fails the acute-angle test with A. This contradicts $C \bowtie D$. Since any intersection leads to a contradiction, the mesh must be planar. ■

Theorem 6.7: Any acute-angle spring mesh is connected.

Proof: Consider a spring mesh $\mathbf{M}$ partitioned into two parts, $\mathbf{M_1}$ and $\mathbf{M_2}$, so that every robot is in either $\mathbf{M_1}$ or $\mathbf{M_2}$ and there is at least one robot each in $\mathbf{M_1}$ and $\mathbf{M_2}$. It is sufficient to show that there exists a spring between some robot in $\mathbf{M_1}$ and some robot in $\mathbf{M_2}$ for any such partitioning.[1]

Pick robots $A \in \mathbf{M_1}$ and $B \in \mathbf{M_2}$ such that for any $A' \in \mathbf{M_1}, B' \in \mathbf{M_2}$, $dist(A, B) \leq dist(A', B')$. That is, pick the robots A and B with the smallest distance between them. Now we show $A \bowtie B$ by contradiction.

Suppose $A \neg B$. Then by Lemma 6.5, there is a robot C such that $dist(A, C) < dist(A, B)$ and $dist(B, C) < dist(A, B)$. Robot C must be in either $\mathbf{M_1}$ or $\mathbf{M_2}$. If $C \in \mathbf{M_1}$ then $dist(A, B) \leq dist(C, B)$ because of how we selected A and B (we are using C as A' and B as B'). However, we know $dist(C, B) = dist(B, C) < dist(A, B)$, which is a contradiction. If $C \in \mathbf{M_2}$ there is a similar contradiction, so $A \bowtie B$.

This proof is valid when there are at least 3 robots. The 2–robot case is covered by Lemma 6.4. The 1–robot case is meaningless. ■

VII. Stability

Even when the spring–formation algorithm takes no parameters, any spring mesh still requires three control constants. These are the natural spring length (l_0), the spring stiffness (k_s), and the damping coefficient (k_d). The robots will deploy to an average spacing very close to l_0. The constants k_s and k_d govern the stability of the mesh.

There are actually three distinct stability modes for a spring mesh: full stability, "quasi-stability," and instability. Each of these is described below, as are the conditions under which each mode occurs.

[1]This is true because if some group of robots $\mathbf{M}' \subset \mathbf{M}$ is not connected to the others, then one can set $\mathbf{M_1} = \mathbf{M}'$ and $\mathbf{M_2} = \mathbf{M} \setminus \mathbf{M}'$. This immediately leads to a contradiction, since there must be at least one spring between $\mathbf{M_1}$ and $\mathbf{M_2}$.

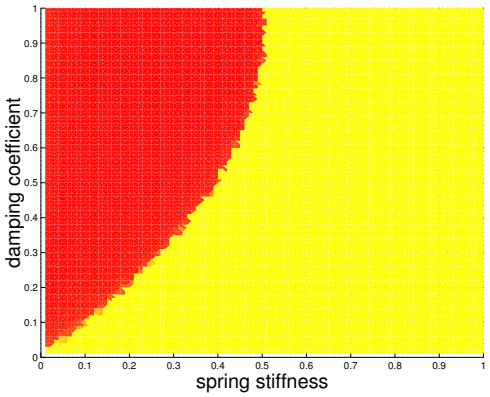


Fig. 13. Full stability boundary for simple 7-robot test case

A. Full Stability

We define a fully stable mesh as one in which the speed of every robot is arbitrarily close to zero when the environment is static. In other words, if there are no targets or new obstacles with which to react, all of the robots will come to a complete stop.

Full stability depends on the spring stiffness k_s, and the damping coefficient k_d. The effect of varying these two factors is shown in Figure 12. In this experiment, seven robots deploy from a tight cluster into an open space, with no targets or obstacles present. This situation was repeated 10,000 times, with various combinations of k_s and k_d. The simulator output the number of time steps until all robots came to a complete stop. If the simulation ever reached 1,000 time steps, it assumed that the robots would never stop moving and output 1,000.

As shown in Figure 12, there is a clear boundary beyond which the mesh is not fully stable (the "wall" between values as k_s approaches 0.5. Figure 13 shows a top-down view of the same data, so this boundary can be more easily seen. While standard analytical techniques for predicting the location of such boundaries do not directly apply (since the springs are dynamic), these simulation results show that the boundary can be determined.

Also notable in Figure 12 is the lip on the left side of the figure; convergence time increases when spring stiffness is low and the damping coefficient is high. This result is intuitive, as the robots move more slowly, and thus take longer to reach their final positions, under those conditions.

B. Quasi-stability

Some spring meshes that do not meet the definition of fully stable are nevertheless stable in some sense. A "quasi-stable" mesh is one in which, when the environment is static, no springs are created or destroyed and the mean velocity of every robot is near zero. Note that the mean speed of the robots may be high even though the mean velocity is low; this is the case when the robots move in tight circles or "wobble" back and forth. By this definition, any fully stable mesh is also quasi-stable.

In the experiments described in the previous section, every mesh was at least quasi-stable for all the tested combinations of k_s and k_d. In the large regions where the meshes were not fully stable, they were always quasi-stable. In general, the robots in a quasi-stable mesh travel in tight circles around a fixed point, much like one would expect with oscillating springs in two dimensions.

While quasi-stable meshes may not be optimal (particularly because real robots will consume more power than necessary), a DRM can still function effectively in a quasi-stable state. Also, quasi-stable states are easy to detect. We are currently investigating the possibility of automatically adapting control constants to move a spring mesh from a quasi-stable to a fully stable state, without the need for direct intervention.

1) Instability: It is possible for a spring mesh to be in an unstable state, i.e., any state that is not quasi-stable. This occurs when the robots are allowed to move large distances (relative to the length of a spring) in a single time step. That effect is not unexpected, and can be avoided by using a shorter time step or limiting the speed of the robots. The exact values that ensure some form of stability will vary according to the physical limitations of the robots comprising the DRM.

Most of the spring formation algorithms, including Acute-Angle Test, may exhibit instability when the spring mesh is under compression (that is, when obstacles contain the mesh in a small area). Under such circumstances, it is possible for some robots to reach equilibrium near a point where their spring connections change. This results in unstable behavior. However, we have overcome this limitation by adding "free" springs between robots separated by less than l_0; this minor change to the spring formation algorithm eliminates the unstable cases with little other effect.

VIII. Practical Considerations

A number of real-world considerations arise in the design of DRMs intended to be deployed. This section describes practical considerations relating to the limitations of real hardware, uncertainty in location estimation, and other effects. Unless otherwise noted, the specific results in this section are based on a simulation of seven robots running the Acute-Angle Test algorithm and deploying from a tight cluster, with $l_0 = 100$, $k_s = 0.1$, and $k_d = 0.3$. However, the specific results presented demonstrate the broader issues involved.

A. Location Uncertainty

Real robots have physical extent and control uncertainty because of mechanical and electrical differences. Thus, location measurements are always likely to be uncertain. In order to examine the robustness of the spring-mesh control scheme in the presence of such uncertainty, we have introduced a simple uncertainty model in the simulator. Each robot has a fixed bias plus some random noise in the determination of its position. The fixed bias is an added term that is always in the same direction and of the same magnitude; it is different for each robot, but does not change over time. The noise changes every time step and is zero-mean. In addition, each robot has a fixed bias and some noise added to its measurement of its neighbors' positions.

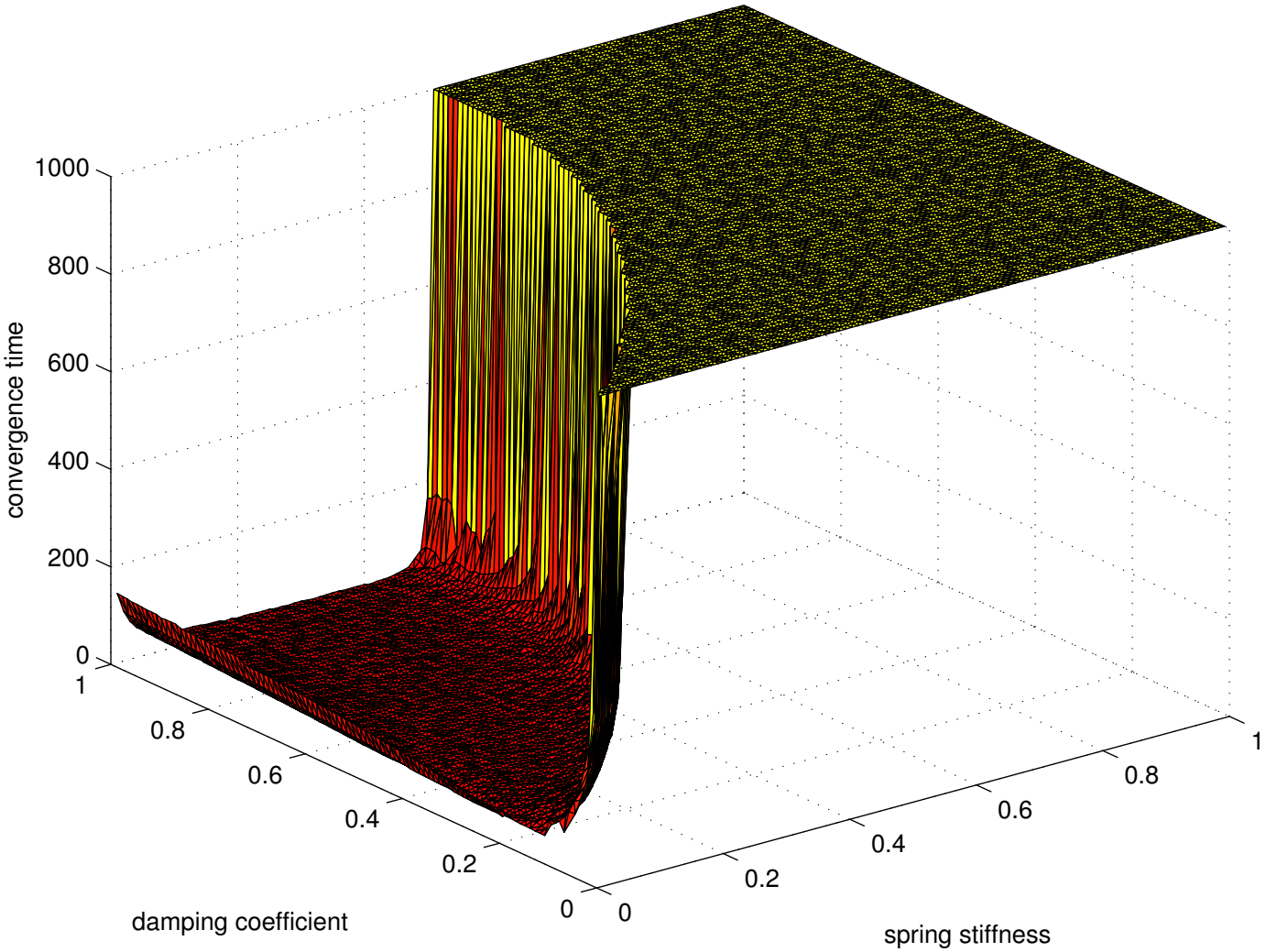


Fig. 12. Convergence time for simple 7-robot test case

Since the Acute-Angle Test algorithm is reference-frame independent, it is not necessary for a robot to know its own position in order to execute the algorithm correctly. Thus, uncertainty in measuring a robot's own position is irrelevant to the control system. Simulation has verified this fact; no amount of uncertainty in self-locating had any effect.

Figures 14(a) and 14(b) show simulation results for the maximum location error as a function of the neighbor-measurement noise and bias levels. For these experiments, the noise and bias levels are expressed as a percentage of the natural spring length (l_0) corresponding to the largest allowed measurement error. The actual errors are distributed randomly inside this limit. For the bias experiment, each robot is assigned a bias randomly, but that bias stays fixed for the entire length of the experiment. The maximum error shown in these figures is expressed as a percentage of l_0, and is the worst–case error; specifically, it refers to the largest error in position for any robot at any time during the experiment. Since we employ spring-like dynamics, it is not surprising that the position error is linear with respect to location uncertainty.

Qualitatively, the random noise in the measurement of a neighbor's location creates a small amount of "wobble," but averages out to zero net effect. Bias potentially has more far-reaching consequences. Suppose two robots, A and B, are connected by a spring and separated by exactly l_0. If A (incorrectly) measures that B is too close, it will correct by moving away from B. If B simultaneously makes the opposite error and measures that A is too far away, it will move towards A. Note that in this case, both robots are moving in the same direction, so the distance between them does not change. If both robots consistently make the same erroneous measurements, they can continue moving indefinitely. When this effect is extended to a large mesh, significant measurement bias can cause an overall translation or rotation of the mesh.

Note that in this case, the robot's position errors are still small relative to each other, even though the errors can result in significant global motion. This kind of effect is intrinsic to a control scheme that makes use of neighbor–relative location information only. Since it is not necessary to control the robots to any particular location relative to the environment, the overall mesh may in fact move; however, such movement will in general not degrade the performance of the DRM. The actual global position and motion of the mesh will be determined by other means, such as tracking targets or

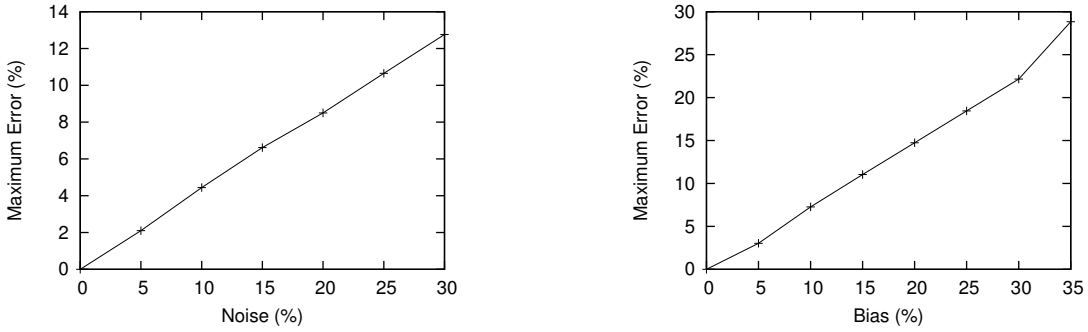


Fig. 14. Maximum error as a function of noise and bias levels

obstacles in the environment. Since any target information reported by individual robots will be normalized with respect to this position and motion, bias-induced translation is not a control concern.

B. Scalability

Virtual spring mesh-based control is inherently scalable. The complexity of the algorithm running on each robot increases with the number of locally visible robots and targets (that is, the number of neighboring robots and targets that are currently in sight), but significantly, *complexity is independent of the total number of robots*. In fact, robots are not explicitly aware of the existence of any other robot that is not locally visible. For this reason, there is no particular limit on the number of robots that may be members of a single macrosensor. Additionally, the total area covered by the macrosensor increases linearly with the number of member robots, which makes the macrosensor highly extensible.

There is no fundamental communication overhead associated with spring mesh control. Since the acute-angle spring-formation algorithm and force computations are symmetric, it is not necessary for robots to communicate with each other in order to form springs and execute the virtual physics model.

C. Fault Tolerance

A significant advantage of the virtual spring mesh approach is that fault tolerance and attack resistance are inherent properties of the macrosensor. Since there is no hierarchy or centralized control, there are no single points of failure or obvious points of attack. Additionally, individual robots are nearly stateless, so recovery from robot failures is simple and rapid. Fault tolerance is discussed in more detail in [7].

D. Other Practical Considerations

Even with damping, springs can oscillate indefinitely with continually decreasing amplitude. Rather than let the robots reproduce this behavior, we introduce a minimum speed v_{min}. Any speed below v_{min} is considered to be zero, and the robot stops moving.

Finally, there is inevitably a constraint on the maximum speed of real robots. Fortunately, clamping a robot's speed to some maximum value can only remove energy from the virtual physical system; the robots deploy more slowly, but enforcing a top speed does not result in instability.

IX. Target Tracking

Spring-mesh DRMs are capable of tracking targets of both a discrete and a diffuse nature. Discrete targets include people, vehicles, and other targets that have a well-defined position. Diffuse targets include chemical plumes, radiation, fires, and other targets that can be represented as an intensity map.

Discrete targets can be easily addressed within the spring mesh framework by adding an attractive force that draws robots toward these targets. The spring formation algorithm is used to determine which robot will track the target. Since intercepting the targets is likely to be appropriate in many applications, the current implementation of the point target force is designed to match the robot's velocity with a vector that will intercept the target.

Diffuse target tracking is somewhat more complex, as these targets do not have a well-defined location. Here, in addition to identifying the location of the target, it is desirable to know both the extent of the target substance and its density gradient. It also may be desirable to increase the density of sensor coverage in the vicinity of a diffuse target. For example, when tracking a chemical plume, one may want to precisely map the plume extent in areas of significant concentration, while sacrificing detailed information about areas of lower concentration or about areas outside the plume. The spring mesh model is well suited to adaptive robot deployment density, since each robot can control its own spring parameters. Robots that detect the desired diffuse target simply shorten their springs, thus drawing in their neighbors to areas of higher concentration. Target tracking is described in more detail in [32].

X. Conclusion

We have developed a control mechanism for distributed robotic macrosensors using a virtual spring mesh. Using simulation, we have implemented and evaluated a number of algorithms for spring mesh formation. We have introduced a novel algorithm based upon an acute-angle test, which has several advantages over other candidate algorithms.

Using simulation, we have verified the basic functionality of virtual spring mesh control, and we have analyzed the impact of various control parameters. The virtual spring mesh approach was demonstrated to satisfy our goals of distributed, fault tolerant control in a DRM. We are currently constructing a distributed robotic macrosensor prototype in hardware, which we will use to further evaluate virtual spring mesh-based control of DRMs.

REFERENCES

[1] L. E. Parker, "An architecture for fault-tolerant, cooperative control of heterogeneous mobile robots," in *Proceedings of IEEE/RSJ/GI International Conference on Intelligent Robots and Systems (IROS)*, pp. 776–783, September 1994.

[2] J. A. Giampapa and K. P. Sycara, "Team-oriented agent coordination in the retsina multi-agent system," in *Autonomous Agents and Multi-Agent Systems*, July 2002.

[3] A. Howard and M. J. Mataric, "Cover me! a self-deployment algorithm for mobile sensor networks," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2002.

[4] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrum, and H. Younes, "Coordination for multi-robot exploration and mapping," in *Seventeenth National Conference on Artificial Intelligence*, 2000.

[5] M. Gini and R. Morlok, "Dispersing robots in an unknown environment," in *7th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, June 2004.

[6] W. M. Spears and D. F. Gordon, "Using artificial physics to control agents," in *Proceedings of IEEE International Conference on Information, Intelligence, and Systems*, 1999.

[7] B. Shucker and J. K. Bennett, "Scalable control of distributed robotic macrosensors," in *7th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, June 2004.

[8] T. Balch and L. E. Parker, *Robot Teams: From Diversity to Polymorphism*. A K Peters Ltd, 2002.

[9] T. Balch and R. C. Arkin, "Behavior-based formation control for multirobot teams," *IEEE Transactions on Robotics and Automation*, vol. 14, pp. 926–939, 1998.

[10] T. Balch and R. C. Arkin, "Motor schema-based formation control for multiagent robot teams," in *First International Conference on Multi-Agen Systems (ICMAS)*, 1995.

[11] M. Roth, D. Vail, and M. Veloso, "A world model for multi-robot teams with communication," in *IROS-2003*, 2003. (under submission).

[12] W. Burgard, D. Fox, M. Moors, R. Simmons, and S. Thrun, "Collaborative multi-robot exploration," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[13] B. P. Gerkey, R. T. Vaughan, K. Stoy, A. Howard, G. S. Sukhatme, and M. J. Mataric, "Most valuable player: A robot device server for distributed control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2001.

[14] L. E. Parker and B. A. Emmons, "Cooperative multi-robot observation of multiple moving targets," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2082–2089, 1997.

[15] B. Jung and G. S. Sukhatme, "Tracking targets using multiple robots: The effect of environment occlusion," *Autonomous Robots*, vol. 13 (3), pp. 191–205, 2002.

[16] D. W. Gage, "Randomized search strategies with imperfect sensors," in *Proceedings of SPIE Mobile Robots VIII*, vol. 2058, pp. 270–279, 1993.

[17] R. A. Brooks, "Integrated systems based on behaviors," *SIGART Bull.*, vol. 2, no. 4, pp. 46–50, 1991.

[18] B. B. Werger, "Cooperation without deliberation: A minimal behavior-based approach to multi-robot teams," *Artificial Intelligence*, vol. 110, pp. 293–320, 1999.

[19] D. W. Gage, "Command control for many–robot systems," in *Proceedings of Nineteenth Annual AUVS Technical Symposium*, June 1992.

[20] A. Howard, M. J. Mataric, and G. S. Sukhatme, "Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem," in *6th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, June 2002.

[21] D. F. Gordon, W. M. Spears, O. Sokolsky, and I. Lee, "Distributed spatial control, global monitoring and steering of mobile agents," in *Proceedings of IEEE International Conference on Information, Intelligence, and Systems*, 1999.

[22] L. E. Parker, "On the design of behavior-based multi-robot teams," *Advanced Robotics*, vol. 10 (6), pp. 547–578, 1996.

[23] M. A. Batalin and G. S. Sukhatme, "Sensor network-based multi-robot task allocation," in *International Conference on Intelligent Robots and Systems (IROS)*, 2003.

[24] T. Balch and M. Hybinette, "Behavior-based coordination of large-scale robot formations," in *Proceedings of the Fourth International Conference on Multiagent Systems (ICMAS*, pp. 363–364, July 2000.

[25] G. Baldassarre, S. Nolfi, and D. Parisi, "Evolving mobile robots able to display collective behaviors," in *Proceedings of the International Workshop on Self-Organization and Evolution of Social Behaviors*, pp. 11–22, September 2002.

[26] E. Şahin and N. Franks, "Measurement of space: From ants to robots," in *Proceedings of WGW 2002: EPSRC/BBSRC International Workshop Biologically-Inspired Robotics: The Legacy of W. Grey Walter*, (Bristol, UK), pp. 241–247, Aug. 14-16, 2002.

[27] S. Koenig and Y. Liu, "Terrain coverage with ant robots: A simulation study," in *Proceedings of the International Conference on Autonomous Agents*, pp. 600–607, 2001.

[28] B. B. Werger and M. J. Mataric, "From insect to internet: Situated control for networked robot teams," in *Annals of Mathematics and Aritificial Intelligence*, pp. 173–197, 2001.

[29] D. F. Gordon-Spears and W. M. Spears, "Analysis of a phase transition in a physics-based multiagent system," in *Proceedings of NASA–Goddard/IEEE Workshop on Formal Approaches to Agent-Based Systems*, 2002.

[30] J. McLurkin and J. Smith, "Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots," in *7th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, June 2004.

[31] S. Fortune, "A sweepline algorithm for voronoi diagrams," *Algorithmica*, vol. 2, pp. 153–174, 1987.

[32] B. Shucker and J. K. Bennett, "Target tracking with distributed robotic macrosensors," *Submitted to MILCOM 2005*.