The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

A Distributed Boundary Detection Algorithm for Multi-Robot Systems

James McLurkin and Erik D. Demaine

Abstract— **We describe a distributed boundary detection algorithm suitable for use on multi-robot systems with dynamic network topologies. We assume that each robot has access to its *local network geometry*, which is the combination of a robot's network connectivity and the positions of its neighbors measured relative to itself. Our algorithm uses this information to classify robots as boundary or interior in one communications round, which is fast enough for rapidly changing networks. We use the local boundary classifications to create a robust boundary subgraph, and to determine if the boundary is an interior void or the exterior boundary. A proof of the key property of the boundary detection algorithm is provided, and all the algorithms are extensively tested on a swarm of 25-35 robots in rapidly changing network topologies.**

I. INTRODUCTION

We consider the problem of defining and estimating the boundary of a two-dimensional configuration of robots in a multi-robot system. We desire a boundary that closely matches the shape of the configuration of robots, and that captures concavities, convexities, and interior voids. We require that the boundary form a connected subgraph of the robot network, so that messages can be routed around it and its properties can be estimated. Finally, we require that the algorithm be fully distributed and require only *local network geometry*, which is the combination of a robot's network connectivity and the positions of its neighbors measured relative to itself. The local network geometry sensing model is a practical compromise between a distance-only model, in which each robot knows only distances (or connectivity) to nearby robots, and a global coordinate system. Knowledge of local network geometry is a reasonable assumption for a practical multi-robot system. Figure 1 shows simulation results from our algorithm.

A. Motivation

A distributed algorithm to estimate the boundary of a multi-robot configuration has many practical applications. We can use a boundary as a formal and practical definition of what is inside and outside the network. Knowing the boundary would allow us to estimate the perimeter of the configuration. For a surveillance application, robots on the boundary can specialize into target monitors, and notify the network when a target has entered or left the tracking area.

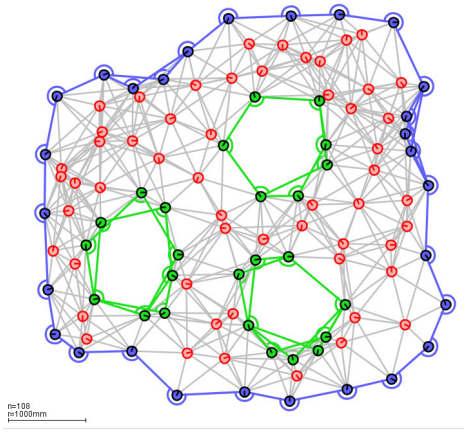


Fig. 1. Simulation results from our boundary algorithm. It detects concave and convex boundaries, interior voids, and produces connected boundary subgraphs. Robots in the interior are drawn in red. Robots and edges on the exterior boundary are blue, those on interior voids are green.

In addition to detecting the exterior boundary, a boundary detection algorithm can find voids in the interior of the configuration. These might simply be voids in the distribution of robots, or might be the footprint of an impassable obstacle in the environment, larger than any one robot could detect individually. These features could be detected, their perimeter estimated, and in the case of voids in the robots' distribution, rectified by moving robots into the empty area.

Boundaries can be used to find *local articulation points*: individual robots whose removal disconnects a local portion of the multi-robot network. These are vulnerable areas in the network, where removing one robot would increase the routing distance between previously nearby robots, or disconnect the network entirely.

B. Related Work

There is much preexisting work on determining boundaries of sets of points. The classic centralized algorithm, the convex hull [12], does not capture concavities in the boundary, requires knowledge of the global coordinates of each point, and assumes that there is an edge between any two points. We assume that our multi-robot network only supports edges of maximum length r, the maximum communications range between

James McLurkin is an assistant professor of Computer Science, Rice University, Houston, TX, 77005, USA jmclurkin@rice.edu
Erik D. Demaine is an associate professor of Electrical Engineering and Computer Science, MIT CSAIL, Cambridge, MA 02139, USA edemaine@MIT.EDU

neighboring robots. Most of these limitations are overcome with the more general definition of boundary defined by the α-shape of Edelsbrunner [1], which we will discuss in detail below.

There are many distributed algorithms for boundary detection. Fekete et al. present two types: one approach takes advantage of the fact that in a dense, uniform distribution of nodes with unit-disk connectivity, nodes on the boundaries will have a smaller degree than those in the interior [4]. This "edge effect" can be used to classify boundary nodes if each node can estimate parameters of the global distribution, but requires very high average node degree, around 100. In other work [3], [6] they develop a deterministic boundary detection algorithm, again for unit-disk networks with only connectivity information. They construct a clever network subgraph called a "flower" that uses connectivity information a fixed number of network hops from the node under consideration. The disadvantages of this approach are the complexity of these neighborhood-based structures and limitations on the minimum size concavity or region they can detect. Wang and Mitchell [14] present a different distributed algorithm that requires only connectivity information, but uses a many rounds of multi-hop communication to produce a result. Ghrist et al. [5] present two algorithms for determining if a node is contained within a given cycle in the network. One algorithm uses network connectivity and angular sorting of neighboring nodes, while another only uses connectivity. Wang, Gao, and Guibas [2] present algorithms to discover and correct routing problems caused by holes in sensor networks. Their definition of interior holes applies directly to our work, but their algorithms are designed for routing, and do not detect all the nodes on the boundaries.

All the previous distributed algorithms are designed for sensor networks and use the connectivity between nodes to infer constraints on the edge lengths of the physical configuration. In order to determine boundary status, they employ complex messaging protocols, which require many rounds of computation, and many messages that travel distances of $O(diam(G))$ through the network. In a multi-robot system with a dynamic network, the configuration could change before these algorithms terminate.

However, it is reasonable to assume that robots in a multi-robot system have access to more information about the positions of their neighbors than sensor networks. This geometric information is required for most configuration control algorithms, which are fundamental for many applications. However, some applications or environments preclude access to a global coordinate system, so we assume instead that the robots have a sensor to measure their local network geometry: the combination of network connectivity and local pose information about neighboring robots. We assume that each robot measures the positions of neighbors in its own coordinate frame. We use this local geometry to compute boundaries in a straightforward way that is robust and performs well in dynamically changing network topologies.

We draw our inspiration from the α-shape algorithm of Edelsbrunner [1]. Given a set of points on the plane and a characteristic dimension α, it is possible to define and construct a well-defined boundary. A point is α-*extreme* if there exists a disc of radius $\frac{1}{\alpha}$ that touches this point and contains no other points. The boundary subgraph can be constructed by taking the union of edges between all pairs of points that a disc of radius $\frac{1}{\alpha}$ can touch and contain no other points. This subgraph is the α-shape. It has a well-defined interior and exterior, and can be interpreted as the boundary of the region containing these points.

The α-shape algorithm requires the user to select a characteristic dimension, α. In a multi-robot configuration, the communication radius of the robots, r, provides a natural characteristic dimension. Setting $\frac{1}{\alpha} = \frac{r}{2}$ is the obvious selection, but this can produce configurations in which *every* robot is classified as a boundary. In particular, the triangular lattice shown in Figure 2 illustrates such a configuration. In order for the interior of such a lattice to be classified as interior, we would need to reduce the distances between neighboring robots, or select $\frac{1}{\alpha} = \frac{r}{\sqrt{3}}$. This α will correctly classify the interior of the lattice, but requires that each robot consider neighbors that are further away than r, which is beyond the range of the local network geometry. There is no entirely satisfactory value of α.

C. Contributions

This paper presents a new boundary detection algorithm called the *cyclic-shape* algorithm. The cyclic-shape algorithm is fully distributed, requiring only the local network geometry available to each robot. In addition, we present two algorithms to characterize global properties of the boundaries. The first gives each boundary a unique name by passing messages between adjacent boundary robots, creating a boundary subgraph . The second allows the robots on each boundary to classify the subgraph as an interior void or the exterior boundary. We provide a proof of correctness of the basic boundary algorithm, and extensive empirical verification of all three algorithms in dynamic network topologies.

II. PROBLEM DEFINITION AND ALGORITHMS

We focus on applications that require highly mobile agents using local inter-robot communication. We define the *state* of an individual robot, a, as the tuple of its unique ID, its global pose, and its private and public variables. We define a *configuration*, C, as the collection of states of all n robots in the network. Each robot can communicate with neighbors within a fixed radius r; we let $r = 1$ for analysis. This produces a geometric

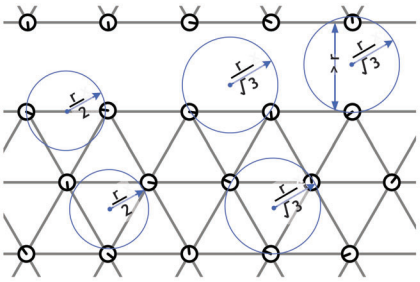


Fig. 2. In a configuration of robots arranged in a triangular lattice, selecting $\frac{1}{\alpha} = \frac{r}{2}$ will properly detect exterior boundaries, but will classify all the interior robots as boundaries as well. Selecting $\frac{1}{\alpha} = \frac{r}{\sqrt{3}}$ will correctly classify interior robots and boundary robots. However, the diameter of this circle is larger than the range of the local network geometry ($\frac{2}{\sqrt{3}}r > r$), so a robot's neighbors would not include all of the robots required to make a correct boundary decision.

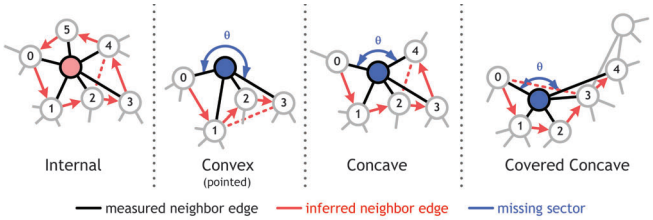


Fig. 3. Illustration of the cyclic-shape algorithm's progress when considering the four main boundary types. Edges to neighboring robots (the local network geometry) are drawn in black, inferred edges that are part of the cycle around the robot are indicated with red arrows, and inferred edges that are not part of the cycle are drawn in red dashed lines. Missing sectors are labeled θ and labeled with a blue arc. The algorithm sorts the neighbors of a robot by their local bearing. It then starts at the first neighbor, and checks for an inferred edge to the next neighbor. If a cycle of inferred edges through all the neighbors of a robot exists, then it is not a boundary.

unit-disk graph, G, in which each robot is a vertex and the communications links between robots are edges. We define the *shape* of G as the union of all the triangles in the graph.

Note that the vertices of the shape of G are not necessarily the vertices of the graph G. This can be seen in the upper-right side of Figure 1, where there are three robots that are covered by triangles from neighboring robots. We cope with this by defining a *maximal polygon subgraph* to be any interior-maximal polygon (with holes allowed) contained in the shape of G whose vertices are vertices of G. Such a shape has several nice properties: its vertices belong to G, it is guaranteed to be contiguous, and it captures all of the holes of G. However, it is not unique: there are many potential maximal subgraphs. We argue uniqueness later in this section.

We assume that each robot has a sensor that can estimate the pose, $p = \langle x, y, \theta \rangle$, of its neighbors relative to its own coordinate frame. We call the collection of these local position measurements and network connectivity the robot's *local network geometry*. We do not assume robots have access to their positions in an external coordinate system. We assume that sensor errors make the local network geometries of neighboring robots different. In order to avoid algorithmic inconsistencies caused by these differences, or the need to reach a consensus between neighboring robots on their shared geometry, each robot performs computations using only its local network geometry. We introduce the notion of *inferred edges* to facilitate this. A robot constructs an inferred edge between any two of its neighbors when its position measurements indicate that they are close enough to communicate. While this eases our algorithm design, it comes at the cost of potentially diverging from the actual network connectivity when the robot makes localization errors, or an obstacle actually prevents communication between two neighboring robots.

We assume that all the robots broadcast their public variables to their neighbors periodically with a shared fixed period, τ, but with different individual offsets. This defines a local *round* of computation; a period of time in which each robot receives messages from each of its neighbors, processes these messages, and then broadcasts its own message. This creates a global synchronizer, which allows us to model group-level algorithm execution as proceeding in a series of discrete global rounds.

A. The Cyclic-Shape Algorithm

The cyclic-shape algorithm works by noting that robots that are within a cycle that passes through all of their neighbors are in the interior of the shape of G. The cycle must pass through all neighbors, even though there might be inferred edges connecting non-adjacent neighbors. Figure 3 shows the algorithm execution on the four main classes of local network geometry, omitting singletons and local articulation points.

We label a robot's neighbors consecutively, starting from the x-axis of its local coordinate system and proceeding counterclockwise. We call a sector between adjacent neighbors *missing* if there is no inferred edge between the two neighbors, or if the angle it subtends is π or greater. If a sector is not missing, we define a *sector triangle* as the triangle formed between the robot and the two adjacent neighbors.

The cyclic-shape algorithm looks for missing sectors around a robot, starting at any neighbor and proceeding counterclockwise through all the neighbors. A robot with no missing sectors is interior. A singleton robot is degenerate, and can trivially be classified as a boundary. A robot with one missing sector with an angle $\theta \geq \pi$ is labeled as a convex boundary. A robot with one missing sector, but with angle $\theta < \pi$, is labeled concave. A robot missing more than one sector is labeled a local articulation point. We define the order of the local articulation point as the number of missing sectors. Intuitively, *local articulation points* are vulnerable sections of the network, where removal

of a single robot disconnects a local portion of the network, but not necessarily the entire network. We defer discussion of local articulation points for future work.

Note that it is possible for a concave robot to have a missing sector, but still be within the shape of G. We refer to this as the *covered concave* case. By labeling covered concave configurations as boundaries, the c-shape algorithm produces a connected subgraph, but at the cost of labeling more robots than needed as boundaries.

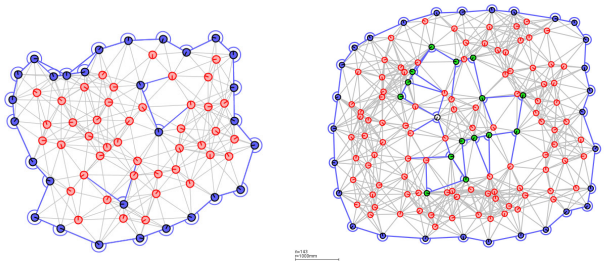
The core of the cyclic-shape algorithm is the FINDEMPTYSECTORS function which returns a list of empty sectors around a given robot:

Algorithm 1 FINDEMPTYSECTORS(*neighbors*)

```
1: N ← SORTBYBEARING(neighbors)
2: emptySectors[] ← ∅
3: for all n_i in N do
4:     n_j ← NEXTCOUNTERCLOCKWISENBR(N, n_i)
5:     θ ← COUNTERCLOCKWISEANGLE(n_i, n_j)
6:     if θ ≥ π then
7:         ADD(emptySectors, {n_i, n_j, θ})
8:     else if (INFERREDEDGE(n_i, n_j) = FALSE) then
9:         ADD(emptySectors, {n_i, n_j, θ})
10:    end if
11: end for
12: return emptySectors
```

The FINDEMPTYSECTORS algorithm first sorts the neighbors by their bearing in the coordinate frame of the robot running the algorithm. Starting from an arbitrary neighbor, the algorithm looks for an inferred edge to the next cyclicly adjacent neighbor. If there is not an inferred edge, the sector in between the two neighbors is stored in the list of empty sectors. The function terminates when it has checked for inferred edges between all the neighbors. The algorithm completes in one round, and the running time per robot per round is $O(\Delta \log \Delta)$, where Δ is the number of neighbors of the robot. The communications messages transmitted per each round of computation is $O(1)$ bits/robot/round, as the robots only need to announce themselves to their neighbors. For local boundary classification, there is no need to propagate messages further than one hop away from any robot, making this algorithm useful for networks with rapidly changing topologies. Our stronger assumptions about sensing, *i.e.* that each robot can measure the local network geometry, allows the C-Shape algorithm to determine local boundary status in $O(1)$ rounds of computation using one-hop communications. This is in contrast to previous work which makes weaker assumptions on sensing, but requires $O(diam(G))$ rounds of computation to determine local boundary classification.

Assuming error-free local network geometry measurements, we prove that if the c-shape algorithm labels a robot as interior, then that robot is contained within the shape of G:



(a) Local classification errors (b) Cycle of errors

Fig. 4. **a.** Because the c-shape algorithm only uses local network geometry, it is possible for robots that are on the interior of the network to be classified as boundaries. **b.** It is even possible to construct a cycle of erroneously classified boundary robots, making this kind of error more difficult to detect.

Lemma 2.1: If a cycle in the local network geometry can be created through inferred edges between neighbors sorted in cyclic order, then the robot is in the interior of the global polygon.

Proof: If there is a cycle through the ordered neighbors in cyclic order, then there is an edge between each adjacent neighbor. Since each neighbor is part of two triangles, there can be no space between adjacent triangles, and the union of these triangles will leave no empty space around the robot. This robot will be within the interior of this local polygon. Since the global polygon is the union of all these triangles from all the robots, the robot will also be in the interior of the global polygon. ■

The converse is not always true: the c-shape algorithm can label robots as boundaries when they are within the shape of G. Figure 4 shows two configurations with such errors. We can detect these errors by noting that the true boundary (exterior or interior) must form a cycle, and that the neighbors adjacent to each robot's missing sectors must also be boundary robots. Therefore, any robot that classifies itself as a boundary, but has an empty sector adjacent to an interior neighbor, must be on the interior. We call such a robot a *boundary error*, and can use this definition to remove single-robot errors. Errors that form chains can be corrected recursively by finding all single-robot errors, suppressing their boundary status and labeling them interior robots, then repeating this process until there are no more single-robot errors. A robot that has its erroneous boundary state suppressed is called a *suppressed error*.

We conjecture that the remaining set of boundary robots, those that are not suppressed errors, form the *true boundary*: the interior-minimal polygon (with holes allowed) contained in the shape of G whose vertices are boundary robots. Note that we know from our definition above that such a boundary actually exists. We

also conjecture that it is unique. We have implemented an algorithm to remove propagated boundary errors in simulation. It uses $O(1)$ bits/robot/round of communication for robots to announce their error status. The algorithm has a running time of $O(\Delta)$ per robot per round, and requires up to $O(n)$ rounds of computation in the worst-case: a configuration composed entirely of errors. Our implementation is fully distributed, but requires some sort of global synchronization to periodically reset the suppressed error status on the robots. We did not implement this algorithm on the robots, because the observed rate of single-robot errors in simulation is low, as they require a careful and unlikely configuration of robots. Also, the robot's mobility and changing network topology cause much larger rates of error, which we explore carefully in the experimental results section.

It should also be possible to achieve some error reduction through configuration control. One approach could be to maintain a minimum density of robots, or require some additional conditions on local network geometry, such as a Yao graph, or a Gabriel graph, or a NET graph [7], [11], [13]. Maintaining a minimum density should be readily achievable in practice, but it might be difficult to maintain more complex geometric constraints at run time. We leave these ideas to future work.

III. Boundary Subgraph Construction

Once robots have determined their local boundary classification using one-hop communication, we can run additional algorithms to construct a global boundary subgraph and estimate its properties. The algorithms presented in this section and the next measure global properties of the boundary, and as such they require $O(diam(G))$ rounds of computation. Intuitively, this would make them more sensitive to dynamic network topologies, an assertion we will explore in the experimental results section.

The first algorithm constructs a boundary subgraph amongst boundary robots. Each boundary subgraph has a single distinguished robot, the root. This root uses its unique ID to name the boundary. The subgraph construction algorithm is straightforward. After each robot has determined its local boundary classification, all boundary robots use a broadcast-tree leader-election algorithm [10], but with the scope of the messages limited to only boundary robots. [1] The robot in the boundary subgraph with the lowest ID becomes the root, and this ID is transmitted to all the other robots in the boundary.

The subgraph broadcast tree is constructed in $O(diam(G))$ rounds. Computation per round consists of evaluating broadcast messages from neighbors, which requires $O(\Delta)$ time per robot per round. Each boundary root sends one boundary subgraph message per round. Normal boundary robots only relay one message, but local articulation points can handle as many as five, as that is the maximum number of missing sectors a robot can have. Therefore, communication complexity is $O(1)$ bits/robot/round. If an algorithm requires that each robot know of the existence of all the boundaries in the network, *e.g.* to count the number of voids, then the communication complexity would grow as $O(B)$, where B is the total number of boundaries in the network.

[1]Boundary "routing helpers" also participate in the relaying of boundary messages. We introduce this class of robot in the experimental results section.

IV. Global Boundary Classification

We wish to be able to classify boundaries as exterior or interior to allow robots to perform different tasks depending on their global boundary type. For example, robots on an interior boundary (a void) might try to eliminate the void by moving inwards. If their motion is blocked by an impassible obstacle, the robots can quantify the object's perimeter. Robots on the exterior boundary can estimate the perimeter, or run configuration control algorithms to help keep the group connected.

Determining the global boundary classification is accomplished by computing the exterior angle of the shape the boundary defines. The exterior angle of the outside of a polygon will sum to 2π, while the angle of an interior void will sum to -2π. Each robot uses its local geometry to compute its *turning angle*, and uses a convergecast to aggregate this quantity towards the subgraph root [8]. The subgraph root computes the sum of the turning angles, determines the global boundary classification, then rebroadcasts the results back down the subgraph. All three of these operations are pipelined, with the broadcast tree messages, convergecast partial sums, and re-broadcast totals all propagating through the network simultaneously.

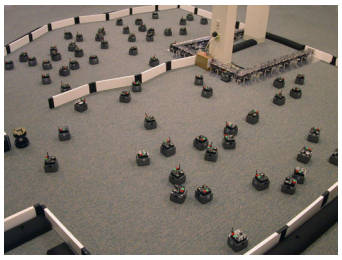
The algorithm requires a broadcast tree on the boundary subgraph to operate. We add three public variables, *parentID*, *turningAngleSum*, and *globalClassification*, to each robot. We examine a particular robot a. Robot a has $k \geq 0$ children in the broadcast tree. We select one of these children at random, say child b. Robot a computes its partial sum in each round by adding its turning angle, $\theta_a - \pi$, to the partial sum of the selected child:

$$a.turningAngleSum = (\theta_a - \pi) + b.turningAngleSum.$$

After $diam(G)$ rounds, the root robot will have the total sum of the turning angles, and can classify the global structure of the boundary subgraph as exterior if $root.turningAngleSum \geq 0$, or as interior otherwise. The root then broadcasts the global boundary classification to all the robots in the subgraph, using the broadcast tree as a routing structure.

Broadcast tree construction, convergecast, and rebroadcast are pipelined and run concurrently. The sub-

(a) A SwarmBot



(b) The Swarm.

Fig. 5. **a.** Each SwarmBot has an infra-red communication and localization system which enables neighboring robots to communicate and determine their pose relative to each other. The three lights on top are the main user interface, and let a human determine the state of the robot from a distance. The radio is used for data collection and software downloads. **b.** There are 112 total robots in the Swarm, but a typical experiment uses only 30-50 at a time.

graph broadcast tree is built in $depth(T)$ rounds, correct turning angle sums reach the root robot after another $depth(T)$ rounds, and the rebroadcast requires another $depth(T)$ to reach the rest of the boundary subgraph. Therefore, the algorithm has a total running time of $3depth(T)$ rounds, or $O(diam(G))$ rounds. Computing the sums and relaying them requires $O(\Delta)$ computation per robot per round, and the communication complexity is $O(1)$ bits/robot/round.

V. EXPERIMENTAL RESULTS

The SwarmBot robot platform was used to validate algorithm performance. The robots are fully autonomous, using only local computation and sensor readings to run the algorithms. Each robot has a 32-bit ARM processor running at 40mhz, a unique ID, and bump sensors. Large top mounted LEDs are used to inform the user of the robot's boundary status; red = interior, blue = exterior boundary, green = interior boundary, white = local articulation point.

Each robot has an infra-red communication and localization system that allows nearby robots to communicate with each other and determine the pose $p = \{x, y, \theta\}$ of their neighbors [10] . The system was run at it's lowest power setting, which has a range of about 1.0 meters. This produces multi-hop networks within the confines of our experimental workspace, which is an 2.43 m $\times$ 2.43 m (8′ $\times$ 8′) square.

Ground truth positions were measured with a vision-based localization system that tracks the position, $\{x, y\}$, of each robot. Each robot uses an infrared emitter blinking with a unique ID. This lets the camera tracking system uniquely identify each robot. Mean positioning error was 15.4 mm, which is small compared to the linear dimension of the workspace, and we use these position measurements as ground truth in our experiments. The system has an update rate of 1 hz, so we limit the maximum speed of the robots in all experiments to 80 mm/s.

Each algorithm was run on 25-35 robots moving randomly around the environment. The motion behavior moves the robots in a straight lines until they contact an obstacle, then use the bump sensors to estimate the angle of incidence and "reflect" the robot back into the environment. The behavior is good at keeping robots dispersed throughout the environment, and the constant mobility changes the robot's neighbors frequently, making it an effective test of an algorithm's performance in changing configurations.

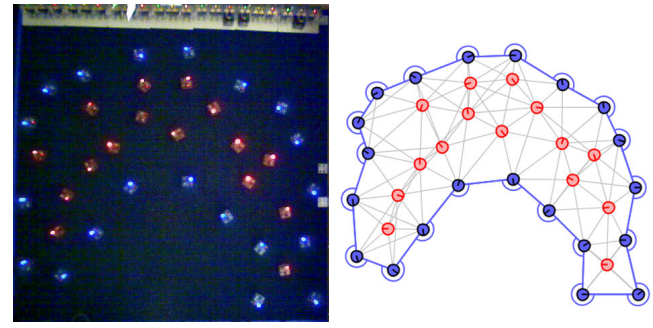
Each experiment tests the algorithm over a wide range of robot speeds, from static configurations to networks that are too dynamic for the algorithms to function properly. The speed and communication rate are combined to calculate the robot speed ratio (RSR) [9], which is the ratio of the robot's speed to the message propagation speed. We tested each algorithm in a static configuration and with RSRs from 0.005 to 0.640. This range of speeds is most visible when plotted on logarithmic axis, so we round the RSR of a static configuration up to 0.001 to plot on a logarithmic scale with the rest of the results. Each algorithm was tested at each speed long enough for multi-hop communications to complete and for performance to stabilize.

A. Local Boundary Classification

Figure 6 shows pictures of robots in two static configurations, and simulation images that show the network and boundaries under ideal conditions. We define the physical accuracy of the local boundary classification algorithm as the ratio of correctly identified robots to the total number of robots. The cyclic-shape local boundary classification algorithm performs well in static configurations, with an accuracy of 0.86. The data from dynamic networks is shown by the black line in Figure 9, which plots the accuracy vs the RSR. The accuracy decreases as the RSR increases, approaching 0.5 at high RSRs. A high RSR means the robots are moving much faster than they can communicate with their neighbors, and their measurements of their neighbor's positions become increasingly uncorrelated with the actual positions. At the limit, we would expect a ratio of boundary robots to interior robots approximately equal to the ratio of the perimeter to the area of the environment. Our experimental environment is a square with sides of length 2.43 m. This produces a perimeter to area ratio of 0.5, supporting our measurements.

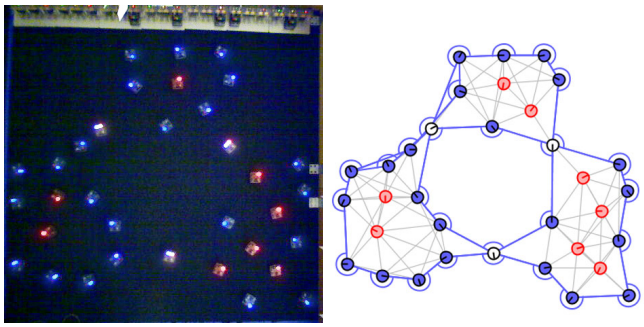
B. Boundary Subgraph Construction

To compute the accuracy of subgraph construction, we compare the boundary subgraphs the robots produce to the results of a centralized version of the subgraph algorithm that uses the robot's ground-truth positions. We define the accuracy of the boundary subgraph algorithm as the ratio of robots who have correctly identified their subgraph to the total number of robots. Because the test environment is small, there



(a) Photo of robots detecting boundaries

(b) Simulation



(c) Photo of robots detecting local articulation points

(d) Simulation

Fig. 6. The cyclic-shape local boundary classification algorithm running on two static configurations. Interior robots, boundary robots, and local articulation points display red, blue and white lights, respectively **a, b.** A configuration with convex and concave boundaries. There is an incorrectly classified robot on the left-hand side. **c, d.** A configuration with three local articulation points. There are two erroneously classified robots in the picture, on in the upper middle, and one on the lower right. The classification mistakes are caused by errors in sensing the positions of neighboring robots.

is only room for one boundary. At high RSRs, any robot that classifies itself as a boundary (which can be up to half of them) is likely to receive a boundary subgraph message with the correct root (because there is only one). We attempt to remove this source of error by tracing the path from each robot back to the root using a combination of logged network data and ground-truth positions, but when the network is changing very rapidly, the probability that a robot has correct data for the wrong reasons increases, and this is reflected in the accuracy data.

In a practical application, the boundary subgraph is a fragile structure, as removal of any robot or disruption of a single communication link will disconnect the subgraph. To mitigate this, we introduce *routing helpers*: robots who are not classified as boundary robots, but are near a boundary and participate in the routing of boundary subgraph messages. We classify a robot as a routing helper if it is within a fixed range d of a boundary robot. We empirically determined that $d = \frac{r}{2}$ produces good results. Figure 7 shows an experiment where a boundary with an error has been patched by a routing helper.

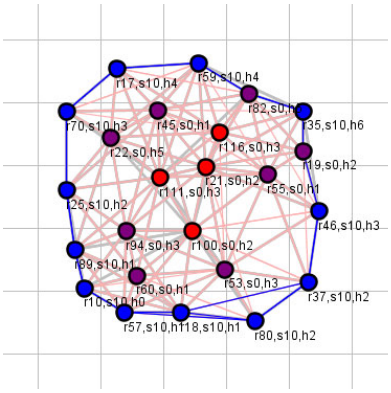


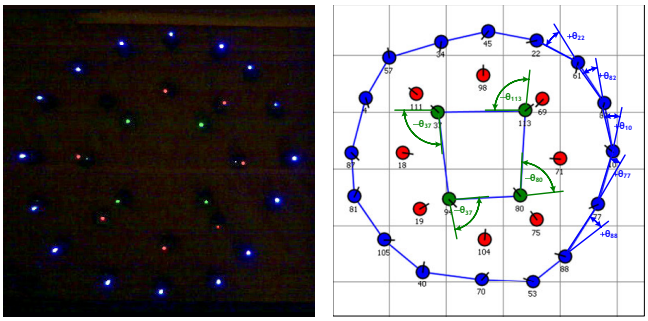
Fig. 7. Boundary routing helpers add redundancy to the boundary subgraph, which can increase the stability considerably. This illustration is from a log file from an experimental run. Boundary subgraph routing helpers are drawn in purple, regular boundary robots in blue, and interior robots in red. Note that the helpers have patched a break in the boundary near the top of the configuration.

Results from the boundary subgraph experiments are plotted in the blue line in Figure 9. The algorithm accuracy decreases as the robot speed ratio increases. There is a slight upward trend at the highest RSRs tested, which should not be possible, as the network is becoming less correlated with the robot's physical positions. In spite of our strict definition of accuracy, the metric can still produce false positive results. At the highest RSR tested, 0.640, each robot can travel 0.640 m between neighbor updates. In these experiments, the communications radius was 1.0 m, and the workspace is 2.43 m on a side, so two robots can't get very far from each other, reducing the effectiveness of this accuracy metric. A larger environment would reduce these measurement errors. The dip at the RSR of 0.01 was repeatable and is difficult to explain, but could possibly be caused by electrical noise from the robot's drive motors interfering with the communications system.

C. Global Boundary Classification

Achieving a configuration with an interior boundary was difficult with our experimental setup. The overhead camera's field of view was too small to position enough robots in the workspace to construct a robust configuration with two boundaries. Reducing the allowable range for neighbors in software allowed more robot into the field of view, but this does not change the physical communication range of the system, so the added network traffic increased the errors in the neighbor position estimates. The few examples we were able to build were fragile, as any connection between the outer and inner boundaries would create a "short circuit", merging the two into a single boundary. Figure 8 shows such an example. Because of this difficulty, we measure the accuracy of classifying the exterior boundary, but not of the interior boundaries.

We define the physical accuracy of the global boundary classification algorithm as the ratio of correctly



(a) Image of robots classifying an interior and exterior boundary.

(b) Screen capture from the same run with turning angle annotations.

Fig. 8. **a.** Picture of robots running the global boundary classification algorithm. The robots flashing their blue lights have classified their boundary as exterior, and the green robots have classified theirs as interior. Robots flashing red lights are not boundaries. The lights are blinking with a sinusoidal pattern. In the picture, brightness variations are evident, and the missing red robot in the upper left has been caught with its lights down. (Which is very embarrassing for a robot) **b.** An annotated data frame from the same experiment, showing the robot's boundary status and some of the turning angles.

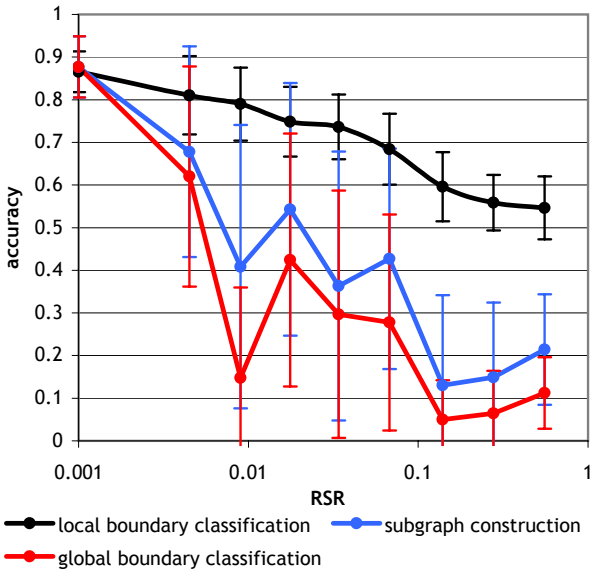


Fig. 9. Accuracy results for boundary detection algorithms vs. RSR. The black line is the accuracy of the cyclic-shape local boundary classification algorithm, the blue line is for boundary subgraph construction, and the red line is global boundary classification.

classified robots on the subgraph to the total number of robots on the subgraph, using a centralized version of the algorithm as ground truth classification. The accuracy of the global boundary classification is shown by the red line in Figure 9. As with the other algorithms, the accuracy of global boundary classification starts high, but quickly declines as the RSR increases. The errors from the subgraph accuracy metric also affect this data, and produce the slight upward trend at high RSRs in these results as well.

VI. CONCLUSIONS AND FUTURE WORK

We have presented a distributed boundary detection algorithm suitable for multi-robot systems with dynamic network topologies. A proof of the key property of the algorithm was presented, and the performance has been evaluated in extensive empirical tests. To the best of our knowledge, this is the first boundary detection technique designed specifically for mobile robots. The cyclic-shape local boundary classification algorithm uses local network geometry, and is more accurate in rapidly changing network topologies than algorithms that use global communication floods.

VII. ACKNOWLEDGMENTS

We thank Sándor Fekete, Sébastien Collette, Martin Demaine, and Stefan Langerman for helpful initial discussions. McLurkin is supported by grants from Boeing Corporation and DARPA's ASSIST program (contract number NBCH-C-05-0137).

REFERENCES

[1] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *Information Theory, IEEE Transactions on*, 29(4):551–559, July 1983.

[2] Q. Fang, J. Gao, and L. J. Guibas. Locating and bypassing holes in sensor networks. *Mobile Networks and Applications*, 11(2):187–200, 2006.

[3] S. P. Fekete, M. Kaufmann, A. Kröeller, and K. Lehmann. A new approach for boundary recognition in geometric sensor networks. *Arxiv preprint cs.DS/0508006*, 2005.

[4] S. P. Fekete, A. Kröeller, D. Pfisterer, S. Fischer, and C. Buschmann. Neighborhood-based topology recognition in sensor networks. *Algorithmic Aspects of Wireless Sensor Networks: First International Workshop (ALGOSENSOR)*, pages 123–136, 2004.

[5] Robert Ghrist, David Lipsky, Sameera Poduri, and Gaurav S. Sukhatme. Surrounding nodes in coordinate-free networks. In *Workshop on the Algorithmic Foundations of Robotics*, 2006.

[6] A. Kröeller, S. P. Fekete, D. Pfisterer, and S. Fischer. Deterministic boundary recognition and topology extraction for large sensor networks. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1000–1009, 2006.

[7] Xiang-Yang Li, Peng-Jun Wan, and Yu Wang. Power efficient and sparse spanner for wireless ad hoc networks. In *Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on*, pages 564–567, 2001.

[8] S. Madden, J. Hellerstein, and W. Hong. TinyDB: In-Network query processing in TinyOS. *Intel Research, IRB-TR-02-014, October*, 2002.

[9] J. McLurkin. Measuring the accuracy of distributed algorithms on Multi-Robot systems with dynamic network topologies. *9th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2008.

[10] James McLurkin. *Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots*. S.M. thesis, Massachusetts Institute of Technology, 2004.

[11] Sameera Poduri, Sundeep Pattem, Bhaskar Krishnamachari, and Gaurav S. Sukhatme. A unifying framework for tunable topology control in sensor networks. Technical report, 2005.

[12] Franco P. Preparata and Michael I. Shamos. *Computational geometry: an introduction*. Springer-Verlag New York, Inc., 1985.

[13] Yu Wang and Xiang-Yang Li. Distributed spanner with bounded degree for wireless ad hoc networks. In *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, pages 194–201, 2002.

[14] Yue Wang, Jie Gao, and Joseph S.B. Mitchell. Boundary recognition in sensor networks by topological methods. In *Proceedings of the 12th annual international conference on Mobile computing and networking*, pages 122–133, Los Angeles, CA, USA, 2006. ACM.