Measuring the Accuracy of Distributed Algorithms on Multi-Robot Systems with Dynamic Network Topologies

James McLurkin

Abstract Distributed algorithms running on multi-robot systems rely on ad-hoc networks to relay messages throughout the group. The propagation speed of these messages is large, but not infinite, and problems in algorithm execution can arise when the robot speed is a large fraction of the message propagation speed. This implies a robot "speed limit", as any robot moving away from a message source faster than the message speed will never receive new information, and no algorithm can function properly on it. In this work, we focus on measuring the accuracy of multi-robot distributed algorithms. We define the Robot Speed Ratio (RSR) as the ratio of robot speed to message speed. We express it in a form that is platform-independent and captures the relationship between communications usage, robot mobility, and algorithm accuracy, allowing for trade-offs between these quantities at design time. Finally, we present results from experiments with 30 robots that characterize the accuracy of preexisting distributed algorithms. In all cases, accuracy degrades as the RSR increases.

1 Introduction and Related Work

Distributed algorithms running on multi-robot systems require inter-robot communication to share messages. Spanning tree construction is a popular form of broadcast communications in sensor networks [3] and multi-robot systems. This type of communication generates a spanning tree rooted at a distinguished root node (or robot) in the network. This technique is a practical way to propagate a global message throughout the network; it is simple to implement, and messages propagate rapidly, without cycles, and decay in an orderly fashion. Many multi-robot algorithms rely on the resulting trees for communication and navigation [1, 7], but the motion of mobile robots changes the underlying network topology, which neces-

James McLurkin
Massachusetts Institute of Technology, Cambridge, MA, e-mail: jamesm@csail.mit.edu

sitates changes in the spanning tree structure. If the construction process cannot maintain an accurate data structure in a mobile network, algorithms that rely on the tree structure cannot operate properly.

In this paper[1], we empirically measure the accuracy of several multi-robot distributed algorithms that rely on broadcast spanning trees. We test these algorithms under different rates of network topology change. We quantify the rate of network change with a dimensionless measure of robot speed appropriate for multi-robot systems that rely on network communications. We find that the accuracy of these algorithms decrease as the robot speed, and the rate of topology change, increases.

Convergecast is a common technique for accumulating global data onto a single network node [8,9]. In our implementation, robots in the network use the broadcast tree as a routing structure to route messages towards the root. Each robot aggregates messages from its children in the tree, and this result is then propagated towards the root. The aggregation step eliminates the need to route each message individually, which significantly reduces the total communications required from $O(n^2)$ to $O(n)$, where n is the total number of robots. However, it requires a stable routing tree to correctly propagate convergecast messages back towards the root.

There is much work describing navigation algorithms that use the broadcast tree for physical routing [1,7]. These algorithms guide a navigating robot towards the root of the tree in a series of steps, by directing it towards its current parent in the tree. A requirement of these navigation algorithms is a strong correlation between the algorithmic tree structure and the physical tree structure. Our experiments with dynamic networks test the limits of these types of algorithms.

2 Multi-Robot Computational Model

In this section we define a model of a multi-robot system. We limit our discussion to the class of multi-robot 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. This produces a geometric graph, G, in which each robot is a vertex and the communications links between robots are edges. We assume that each robot has a sensor that can estimate the pose of its neighbors relative to its own coordinate frame. We do not assume each robot has access to its position in an external coordinate system. We do not address power utilization or conservation in this work, instead assuming that the robots have sufficient energy to allow unconstrained mobility and communication.

We assume that all the robots announce their public variables to their neighbors periodically with a shared fixed period, τ, but with different individual offsets to prevent message interference. This defines a local *round* of computation; a period of

[1] This work funded by a grant from Boeing Corporation

time in which each robot receives an announcement message from each of its neighbors, processes these messages, and then transmits 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.

We define *algorithm accuracy* as a metric that ranges between $[0,1]$ and quantifies the quality of a configuration with respect to a particular algorithm A. A configuration with an accuracy of 1 is an exemplar of the best possible performance of algorithm A, while an accuracy of 0 is the worst possible performance. The accuracy metric is tailored to each algorithm and application, and must be included in the algorithm specification.

3 The Robot Speed Ratio

In the previous section, we assumed that the robots broadcast messages to their neighbors at regular intervals in order to maintain the network as they move. As the robot's speed increases, so will the rate of network topology change, so the frequency of neighbor updates must also increase to maintain accurate connectivity information. Since communications bandwidth is limited, there is a maximum update rate, which correlates to some maximum robot speed. If the robots move faster than this speed, the network can no longer be properly maintained, and the performance of algorithms that rely on network communication will degrade. In this section, we define a metric that captures the relationship between robot speed and communications bandwidth. We want this metric to be dimensionless and able to be estimated *a priori* from basic system parameters, so that it can be used as a design tool for new systems and for algorithm performance comparisons across different hardware platforms.

Our approach is to define the robot's speed as a fraction of the maximum message propagation speed, s_{message}. Given the actual robot speed, s_{robot}, we define the *robot speed ratio*, or RSR, as

$$RSR = \frac{s_{\text{robot}}}{s_{\text{message}}}. \tag{1}$$

The robot speed is measured directly, but the propagation speed of the message through the network must consider buffering delays and channel capacity limits. Any robot moving away from a source of information, such as the root of a broadcast tree, with a RSR > 1 will not be able to receive new information from this source. This effectively disconnects the network, making it impossible for any distributed algorithm to run correctly. This is a conservative limit, as it only considers the speed of a single robot. If the direction of motion is taken into account, motion towards or parallel to the motion of the source might not be subject to this limit. But this requires knowing the particular configuration and velocities for all the robots in the network, making it impossible to compute before the algorithm execution. We consider possible approaches to address this limitation in future work.

To determine the robot speed ratio before execution, we need to estimate s_{robot} and s_{message}. We will assume the actual speed of the robots, s_{robot}, does not deviate far from the commanded speed, so we use the latter for our estimate. We can express the message speed as $s_{\text{message}} = \frac{d_{hop}}{t_{hop}}$, where d_{hop} is the average distance a message travels away from the source per hop, and t_{hop} is the expected latency per hop. We compute d_{hop} by noting that the configuration graph G is an r-disk spanning subgraph of the fully connected graph, G^f, which is formed by connecting all pairs of robots. The *average spanning ratio*, k, is the average stretch of the distance between any two pairs of robots a,b in graph G to the distance between the same two robots in graph G^f. If we let d_{ab} and d^f_{ab} be the distance between robots a and b in graphs G and G^f respectively, we can write $k = \text{mean}\left(\frac{d_{ab}}{d^f_{ab}}\right)$ over all pairs a and b.

Using k, we express d_{hop} as a fraction of the communications radius: $d_{hop} = \frac{r}{k}$. A network with a smaller spanning ratio supports paths that are closer in distance to the Euclidean distance. In the ideal case of no stretch, $k = 1$ and $d_{hop} = r$, but this implies that a message can travel in a straight line, which is only possible with an infinite density of robots. We can estimate k *a priori* if we know the density of robots in the environment and the communication radius, r. For the configurations tested in this work, the average number of neighbors of a robot, $m_{avg} \approx 10$ and $r \approx 1$, yielding $k \approx 1.4$. See the reference by Kleinrock and Silvester [5] for a careful analytic treatment of the calculation of d_{hop} as a function of node degree.

The next step is to compute t_{hop}, the expected latency at each hop. We assume that while each robot uses the same round duration, τ, they all have different offsets, picked uniformly at random from $0 \leq \tau_{\text{offset}} < \tau$. Therefore, the most delay between receipt and retransmit of a message by any robot is τ, and the minimum is σ, where σ is some small processing time. Rounding σ down to 0, we compute the expected time delay at each node as $t_{hop} = \frac{\tau}{2}$. This gives us an expression for message speed of: $s_{\text{message}} = \frac{d_{hop}}{t_{hop}} = \frac{2r}{k\tau}$.

However, the round duration τ is bounded from below by the minimum communication requirements of the algorithm under consideration, A. Given the maximum communication bandwidth of each robot, B, and an assumption on the maximum number of neighbors for each robot, m_{max}, we can compute the minimum round time for algorithm A's communication requirements measured in bits/round, $\mathscr{B}_{\text{A}}$: $\tau_{min} = \frac{\mathscr{B}_{\text{A}}\, m_{max}}{B}$. Note that this assumes that the physical communications medium is shared between neighboring robots and is half-duplex. This assumption is true for most of the radio and optical communications hardware used in a practical multi-robot system.

Combining the above expressions for d_{hop}, t_{hop}, and τ, we can express the message speed and maximum robot speed ratio as:

$$s_{\text{message}} = \frac{2rB}{k\, m_{max}\, \mathscr{B}_{\text{A}}}, \quad RSR = \frac{s_{\text{robot}}\, k\, m_{max}\, \mathscr{B}_{\text{A}}}{2rB} \tag{2}$$

The maximum RSR makes sense when we vary the parameters in the expression. Increasing r, the communication range, allows the message to travel further with each hop. Increasing B, the bandwidth, decreasing $\mathscr{B}_{\mathrm{A}}$, the algorithm's required bits per round, or reducing m_{max}, the number of neighbors, all allow the round, τ, to be shorter, which increases message speed. A smaller spanning ratio, k, gives the message a straighter, more efficient path. The maximum number of neighbors, m_{max}, can be estimated by knowing the workspace area and the number of robots, and controlled at run-time with dispersion algorithms.

4 Experimental Setup

The SwarmBot [4] robot platform was used to validate algorithm performance. The robots are fully autonomous, using local computation and sensor readings to run the algorithms. Each robot has a 32-bit ARM processor running at 40mhz, a unique ID chip, a bump sensor, and wheel encoders. Large top mounted LEDs on each robot are used to inform the user of the robot's status.

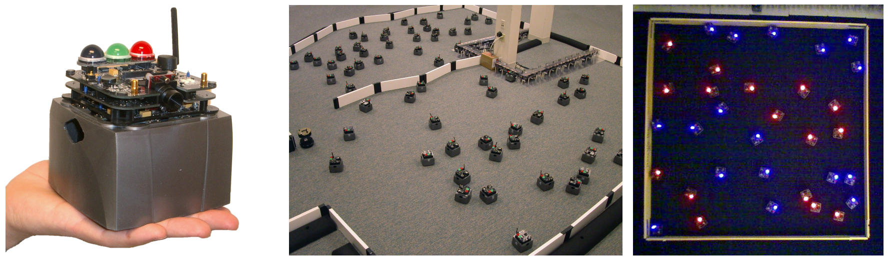

(a) A SwarmBot (b) The Swarm. (c) Hop distribution on robots

Fig. 1 **a.** Each SwarmBot has an infra-red communication and localization system which enables neighboring robots to communicate and determine their pose, $\{x,y,\theta\}$ 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 only for data collection and software downloads. **b.** There are 112 total robots in the Swarm, but a typical experiment uses only 25-35 at a time. **c.** Picture of the robots constructing a broadcast spanning tree in the experimental workspace. Robots that are an even number of hops from the root are flashing their blue light, those located an odd number of hops are flashing their red light. The root is in the lower-left.

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

Ground truth was determined by a vision-based localization system. The system was developed by Newton Labs [6], and tracks the position, $\{x,y\}$, of each robot.

The system uses an IR emitter on the top of each robot that encodes 10 bits of data per second per robot, allowing each robot to be uniquely identified and tracked. Mean positioning error was 15.4 mm, which is adequate to use as ground truth in our experiments. The system has an update rate of 1 hz, so we limited 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 uses the bump sensors to estimate the angle of incidence and "reflect" the robot back into the environment. The behavior works well at keeping robots dispersed throughout the environment, and the uncorrelated random motion changes the robot's neighbors frequently, making it a good way to characterize how sensitive an algorithm's performance is to changing configurations.

We tested each algorithm in a static configuration and over a range of robot speed ratios from 0.005 to 0.640. This range of speeds is best plotted on a logarithmic axis, so we round the RSR of the static configurations 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. This was no less that $10\,diam(G)$ rounds for the two communications algorithms and from 5-10 successful completions of the navigation algorithm.

5 Communication and Navigation Algorithms

In multi-robot systems, broadcast trees [3] (or communication gradients) underpin many other algorithms. Understanding how the performance of trees in dynamic multi-robot networks affect higher-level algorithms is critical to building practical systems. In this section, we measure the performance of three algorithms: one to estimate distance along broadcast tree paths, a convergecast algorithm that uses broadcast trees to aggregate global quantities onto a single robot, and a navigation algorithm that uses the broadcast tree as a physical navigation structure. [1, 2, 11].

5.1 Broadcast Tree Path Distance

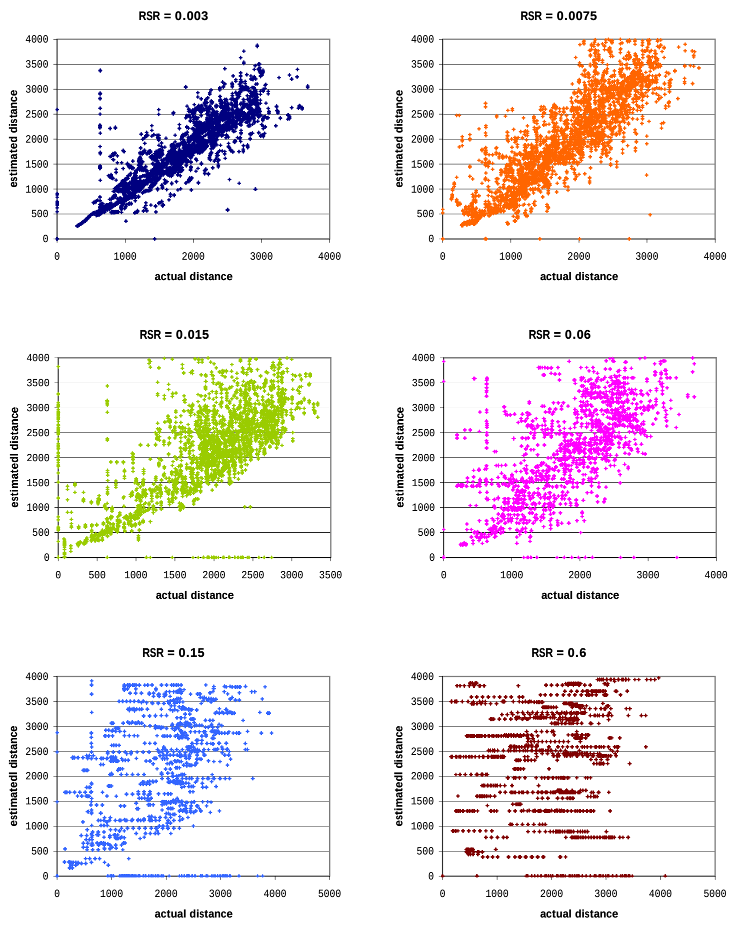
One of the most basic uses of a broadcast tree is to propagate messages from the root outwards. Because we assume each robot has a sensor to detect the positions of its neighbors, it is possible for each robot to estimate the distance to the root by summing the path length a message has traveled to reach it. Under good conditions, the topology of the network will closely match the actual positions of the robots, producing a distance estimate on each robot that is well-correlated to the actual distance of that robot from the root. Figure 1c is a snapshot of the robots constructing a broadcast tree. The root robot is at the lower left-hand corner of the picture, and the parity of the hops from the root is illustrated with a red or blue light on the robot. This static configuration produced (fairly) concentric rings, as the correlation between distance from the root and hops from the root is good.

The algorithm to compute the path distance to the root is as follows: We examine a particular robot a that is not the root of the tree. Robot a has $k \geq 1$ neighbors that are closer to the root than it is. We gather these closer neighbors into the set $\mathbf{P}_a$, and use the position of each robot $p_i \in \mathbf{P}_a$. We add a public variable, *treeDistance*, to the state of each robot. This variable is calculated as: $a.treeDistance = \frac{1}{k}\sum_{i=1}^{k}\left(r_i.treeDistance + \|\overrightarrow{p_i}\|\right)$. The *treeDistance* of the root is 0. The estimate of path distance to the root on robot a is $d = a.treeDistance$.

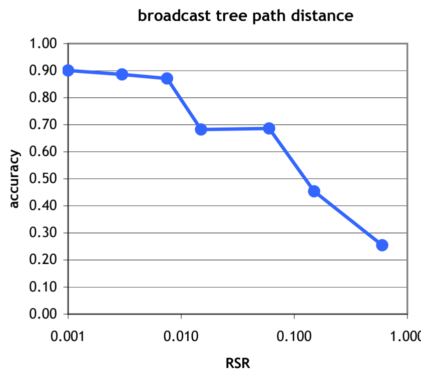
The path distance algorithm runs concurrently with the broadcast tree, and its results are calculated as soon as a robot receives a broadcast tree message. Therefore the total running time is $O(diam(G))$ rounds, and it uses $O(m)$ computation per round. The bits/round transmitted is $O(1)$. The accuracy is the correlation coefficient computed between the actual and estimated path distances: $\mathscr{A}_{\text{rootDistance}} = \text{corrcoef}(d, d_{est})$, which returns a result in the range of $[0,1]$. We compute the algorithm's accuracy without compensating for k, the spanning ratio, as a robot's local measurement of density would introduce errors from edge effects, and a more global estimate would require a more complex algorithm with additional communication. This lack on compensation means that we can never expect to achieve an accuracy of 1 because the spanning ratio in the network makes the measured paths longer than the actual paths. In practice, this path-length error is moderate, but tolerable, usually less than 20%.

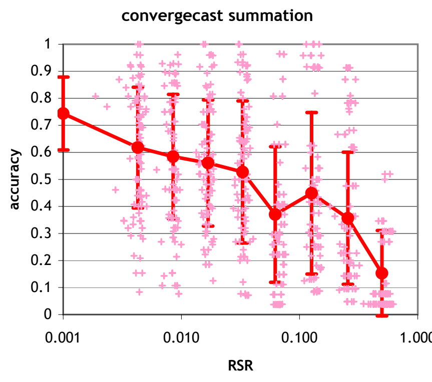
Voids in the network will cause a larger stretch in the measured path length, as the messages must route around such obstacles. In a convoluted, maze-like environment, the type of local communication and pose estimation system employed will introduce more serious errors. We imagine two types of communication systems, one that uses line-of-sight communication, and one that is able to communicate through obstructions. Assuming the line-of-sight path between any two robots is also navigable, this system will produce a distance estimate that reflects the path the broadcast messages traveled, which is also the path a robot should follow to navigate to the root. However, the geometric error between the navigable path and the actual distance to the root can be large if the path is convoluted. Therefore, using this distance estimate to produce a coordinate system [11] would produce poor results. On the other hand, if robots can communicate and estimate local network geometry without interference from obstacles, then the distance estimate might be close to the Euclidean distance to the root, but the path might not be navigable.

Figure 2a shows plots of the root distance estimate vs. the Euclidean distance to the root for a variety of RSRs. Note that these quantities are correlated at low RSRs, and become increasingly uncorrelated at high RSRs. Figure 2b plots the accuracy of the path distance estimate over increasing RSRs. Because the estimate is uncorrected with knowledge of the spanning ratio, this algorithm can never achieve an accuracy of 1, as any path through the network will always be somewhat crooked. As the RSR increases, the correlation between the robot's distance estimate and the actual distance decreases. The intuition is that robots moving at a high RSR can travel a large distance in each round of computation, carrying their previous hop



(a) Broadcast tree path distance data for various RSRs.





(b) Broadcast tree path distance accuracy vs. RSR

(c) Convergecast summation accuracy vs. RSR

Fig. 2 **a:** The distance estimate is strongly correlated with the Euclidean distance in a static configuration. The vertical lines in the plots were caused by a robot with faulty communications.**b:** The accuracy of the distance estimate decreases as the RSR increases. **c:** Convergecast summation requires a stable network to route messages back to the root, and its accuracy also declines rapidly at high RSRs.

counts and distance estimates with them. In turn, this affects neighbors in the next round, ultimately producing an inaccurate estimate.

5.2 Convergecast Summation

A convergecast aggregates a global quantity q onto the root robot by using the tree as a routing structure to propagate partial sums to the root. Convergecast requires the tree to be stable while messages route back to the root, and we would expect its performance to be sensitive to changing network topologies.

The algorithm adds a public variable, *partialSum*, to the state of each robot. We examine a particular robot a with $k \geq 0$ children, $c_1 \dots c_k$, in the broadcast tree. Robot a computes its partial sum in each round by adding the value of its quantity, q_a, to the partial sum of each of its children: $a.partialSum = q_a + \sum_{i=0}^{k}(c_i.partialSum)$. The partial sum computed on robot a is stored in its public variable, $a.partialSum$, making it accessible to all of its neighbors, in particular, its parent. By this process, each robot in the tree computes the partial sum of q over the subtree rooted at itself. The subtree rooted at the root robot is the entire tree, and the partial sum computed by the root robot is the total sum of q over the entire network. In our experiment, we choose to compute the total number of robots in the configuration, so each robot sets their value of $q = 1$.

The algorithm has a running time of $O(diam(G))$ rounds. The convergecast algorithm runs concurrently with the broadcast tree construction, but correct results only reach the root robot after $2\,depth(T)$ rounds after startup or any population change. There is $O(m)$ computation per round, and the bits/round transmitted is $O(1)$. The accuracy metric is defined in terms of the error in the value of q computed at the root: $\mathscr{A}_{\text{convergecast}} = 1 - \frac{|q_{\text{measured}} - q_{\text{actual}}|}{|q_{\text{actual}}|}$, bound to $[0, 1]$.

The accuracy of the convergecast summation algorithm is shown in Figure 2c. The accuracy degrades quickly as the RSR increases. This is expected, because the algorithm relies on a stable broadcast tree to relay partial sums through its neighbors back towards the root robot.

Our accuracy metric does not take into account whether or not messages from a particular convergecast were accumulated in the summation, *i.e.*, did the messages converge onto the root "on time". The broadcast and convergecast messages are overlaid and pipelined and the algorithm does not preserve individual message routes, as this would require transmitting $O(n)$ bits/round from each robot. Therefore, the root may be receiving messages from other convergecasts that are "early" or "late", depending on how the network is reconfigured by the robot's motion. A way to eliminate this error would be to tag each message with its initial round, and maintain several parallel convergecast summations over multiple rounds. However, we do not expect that this change would affect overall results appreciably.

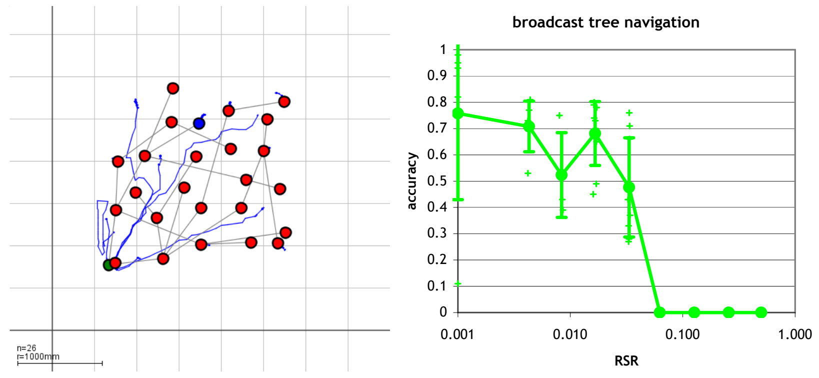
5.3 Broadcast Tree Navigation

The broadcast tree can be used to guide any robot in the configuration to the root by using the other robots in the network as navigational aids. We assume that the path between any two neighboring robots (robots that can communicate) is navigable. Any robot can navigate to the root of the tree in a series of steps, by moving towards its current parent in the tree. In the process of moving towards its current parent, it will move to a new part of the network, and will select a new parent that is closer to the root than its previous one. The cycle repeats until the navigating robot reaches the root of the tree. See reference [10] for an empirical evaluation of this algorithm and reference [7] for a more theoretical treatment.

In our implementation, the navigating robot a collects all k closer neighbors to the root into the set $\mathbf{P}_a$. The navigating robot computes the midpoints between all the *cyclically adjacent* neighbors from $\mathbf{P}_a$. Cyclically adjacent neighbors are two robots that are adjacent in the angular ordering of neighbors. The navigating robot selects the closest midpoint and moves towards it. Moving towards the midpoints between neighbors helps to reduce collisions. The motion of the navigating robot will cause its neighbors to change, and the set of neighbors in $\mathbf{P}_a$ is updated each round. If $|\mathbf{P}_a| = 1$, then the robot has only one closer neighbor (its parent), and moves directly towards it. In this case, we rely on the low-level bump-sensing obstacle avoidance behavior to guide the robot around its parent, or any other obstacle.

The key requirement for success is that the neighbors in $\mathbf{P}_a$ are actually closer to the root along the navigable path. Figure 2b shows that the correlation between the path distance estimate and the Euclidean distance degrades at high RSRs, so we would expect the accuracy of this navigation algorithm to suffer as well. The accuracy of this navigation algorithm is best captured by the *path efficiency*: the ratio of the path actually traveled by the navigating robot to the shortest navigable path possible. For a given start and goal location in the external coordinate frame, the physical accuracy of the broadcast tree navigation algorithm is: $\mathscr{A}_{\text{navigation}} \equiv \frac{s_{\text{optimal}}}{s_{\text{robot}}}$. Paths closer to the optimal are more efficient and judged to be more accurate. Paths that are of infinite length, *i.e.* the navigating robot does not get to the goal position, have an accuracy of 0.

Several example paths from experiments with RSR = 0 are shown in Figure 3a, and the accuracy of the algorithm is shown in Figure 3b. The algorithm performs well at low RSRs, but then fails completely at a RSR of 0.08 or above. This is consistent with our intuition that navigation accuracy relies on the broadcast root vector distance accuracy. When this requirement is not met, navigating robots cannot determine the correct direction to drive in, and move around randomly because the neighbors in $\mathbf{P}_a$ are not actually closer to the root.



(a) Navigation Paths (b) Navigation accuracy vs. RSR

Fig. 3 **a.** Example paths from the broadcast tree navigation algorithm at a RSR of 0. The broadcast tree network is shown with grey edges. Note that it is not the minimal spanning tree, nor is it planar. Using this tree directly for navigation would produce poor results. **b.** Broadcast tree navigation accuracy vs. RSR. The algorithm did not function at a RSR of 0.08 or higher.

6 Conclusions and Future work

The accuracy of all of the algorithms decreased as the robot speed ratio increased. In our experiments, a RSR of 0.005 allows good accuracy in all algorithms, a RSR of 0.02 allows reasonable accuracy in simple algorithms, and all algorithms tested are essentially useless at a RSR of 0.10 or higher. Intuitively, the network stability required for each of these algorithms differs, with convergecast and navigation requiring a more stable network than the broadcast tree path distance. Roughly speaking, accuracy for algorithms with higher network stability requirements degrade more rapidly as the robot speed ratio increases. However, quantitative comparisons between accuracies is limited, because each accuracy metric is user-designed and application-dependent. The accuracy metrics used in this work were all designed towards quantifying the relationship between the broadcast tree computational data structure and the broadcast tree network geometry. A different application using the same algorithms might have different accuracy metrics.

While the RSR is a convenient way to estimate the relationship between mobility and communications, it does not actually measure the changes in topology which cause the accuracy degradation. There are many other metrics to capture topology changes, including "network half life"; the amount of time until half of the population has changed [12], and "network churn"; for example, taking into account the the amount of time a node is in the network. Future work could develop a notion of "neighbor half-life" which could directly capture the rate of topology change for a given configuration control algorithm. However, it is not clear how to generalize this to avoid having to calculate different rates of change for different algorithms. Additionally, the RSR metric will only be proven useful after evaluation on different hardware platforms.

The data confirms the intuition that communications bandwidth, robot mobility, and algorithm accuracy are interrelated. Given an accuracy vs. RSR plot for a particular algorithm, the designer can use the robot speed ratio in equation 2 to trade-off between these three quantities at design time. This can be a useful tool for the designer of multi-robot systems.

References

1. Maxim Batalin, Gaurav S. Sukhatme, and Myron Hattig. Mobile robot navigation using a sensor network. pages 636–642, New Orleans, Louisiana, April 2004.
2. Qing Fang, Jie Gao, L.J. Guibas, V. de Silva, and Li Zhang. Glider: gradient landmark-based distributed routing for sensor networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 1, pages 339–350 vol. 1, 2005.
3. C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks.
4. iRobot. Swarmbot. In *www.irobot.com*, 2002.
5. L. Kleinrock and J. Silvester. Optimum transmission radii for packet radio networks or why six is a magic number. *Proceedings of the IEEE National Telecommunications Conference*, 4:14.3, 1978.
6. Newton Labs. Model 9000 vision system. In *www.newtonlabs.com/9000.htm*, 2001.
7. Qun Li and Daniela Rus. Navigation protocols in sensor networks. *ACM Trans. Sen. Netw.*, 1:3–35, 2005.
8. S. Madden, J. Hellerstein, and W. Hong. Tinydb: In-network query processing in tinyos. *Intel Research, IRB-TR-02-014, October*, 2002.
9. S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems (TODS)*, 30:122–173, 2005.
10. James McLurkin. *Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots*. PhD thesis, Massachusetts Institute of Technology, 2004.
11. R. Nagpal, H. Shrobe, and J. Bachrach. Organizing a global coordinate system from local information on an ad hoc sensor network. *Proc. of Information Processing in Sensor Networks (IPSN)*, 2003.
12. Ting Yan, Tian He, and John A. Stankovic. Differentiated surveillance for sensor networks. pages 51–62, Los Angeles, California, USA, 2003. ACM.