Communication-Restricted Exploration for Robot Teams

Elizabeth A. Jensen and **Ernesto Nunes** and **Maria Gini**
Department of Computer Science and Engineering, University of Minnesota
4-192 KHKH, 200 Union St SE, Minneapolis, MN 55455
{ejensen, enunes, gini}@cs.umn.edu

Abstract

In the event of an earthquake or fire, search and rescue efforts may be delayed until it is safe for the human rescue team to enter the area. A team of robots could enter in advance to provide maps, images and locations of interest to the human team, allowing them to prepare their approach when they can enter. In a disaster area, communication may be limited, either due to infrastructure being down, or because of environmental interference. We propose an algorithm that makes use of a small number of robots, which spread as far as their communication allows, but which otherwise stay together while they explore the unknown environment. We show that the algorithm will allow the team of robots to fully explore the environment and maintain communication in order to return the information to the waiting search and rescue team. We also show that this can be achieved with multiple methods of communication.

In the event of a fire or earthquake, it is not always possible for a rescue team to enter an area immediately, due to safety concerns. To help speed the rescue process, many have looked into using robots to explore the environment in advance, so that points of interest, such as weak spots in a wall or the location of survivors can be mapped out and relayed back to the rescue team. This approach allows the rescue team to plan their rescue efforts more precisely and prioritize tasks. However, such an approach requires to guarantee that the robots reach every part of the environment.

There are multiple methods for a team of robots to explore an unknown environment. Gage (Gage 1992) proposed three types of coverage. In blanket coverage, the robots cover the entire environment simultaneously. In barrier coverage, the robots set up a perimeter around an area such that nothing can pass into or out of that area without being seen by at least one robot. In sweep coverage, the robots make a pass over the environment and ensure every point has been seen by at least one robot, but don't stay in any one location, instead moving progressively through the environment. Choset (Choset 2001) later presented an extensive overview of coverage path planning algorithms according to those categories. Most coverage algorithms aim to achieve either blanket or barrier coverage. However, both blanket and barrier coverage require enough robots to provide the full coverage, and that number can be prohibitively large.

In contrast, sweep coverage can be done with a small team, down to a single robot, if necessary, when all other robots on the team have failed. Since the environment is unknown, the required number of robots for blanket coverage is also unknown, and, even if known, may well exceed the number of robots available on site. Thus, in our approach, we use an exploration algorithm, in which the team of robots completes a single sweep of the environment to locate points of interest that can be relayed to the search and rescue team.

Our main contribution is a novel algorithm that is fully distributed and which provides full coverage of an unknown environment, while also maintaining communication amongst the robots, even with severe restrictions on the communication type, range, and quality. The primary innovation of this algorithm is that it uses the minimum number of messages, both in size and number of types, making it possible to use a wide range of communication methods to accommodate restrictions from the environment. We provide simulation results and in-depth analysis to show that the algorithm can guarantee full exploration regardless of the number of robots, and that the robots maintain communication.

Related Work

In recent years, multi-robot systems have gained popularity (Arai *et al.* 2002). There are several advantages to the use of a multi-robot system over the use of a single robot, including cost, efficiency and robustness. A single robot can be designed to efficiently complete its task, but it may then be suitable for only a small set of tasks, and added functionality increases the cost, size and energy requirements. In addition, if even a small part fails, the robot may be unable to complete the task. In contrast, a multi-robot system comprised of smaller, individually less-capable robots, with several of each type needed to complete the different parts of the task, can still accomplish their goal even if some of them fail. A multi-robot system has an inherent redundancy that increases the system's robustness (Choset 2001).

In a centralized multi-robot system, an external controller issues instructions to the others and keeps the group coordinated. While this requires less of individual robots, the system is also then susceptible to a complete failure if the controller fails, even if the robots are still running. In ad-

dition, a centralized approach does not scale well, because one machine can control only a limited number of robots at a time. In small environments, however, a centralized approach can be effective. Stump et al. (2008), made a single robot a base station, while the other robots formed a communication bridge as they moved into the unknown region. Similarly, Rekleitis et al. (1997) used one robot as a stationary beacon for another robot, thus reducing odometry error in the robot that was moving. The centralized approach also has the advantage of making it easy to create a global map, which can be used to direct robots (Burgard *et al.* 2005; Stachniss and Burgard 2003; Wurm *et al.* 2008). These approaches require constant monitoring of the robots in order to keep the map consistent and the exploration efficient.

A distributed approach, on the other hand, is inherently more scalable and can also take better advantage of the robustness of having multiple robots. Each robot is responsible for its own movements and data collection, and relies on only local neighbors for coordinating exploration and dispersion. It may seem that the robots are working together on a global scale, but in actuality the decisions are made individually on a local scale. Information can be passed throughout the group, similar to the communication bridge (Stump *et al.* 2008), but using broadcast messages rather than point-to-point messages. A distributed system thus allows an individual to work independently, while also sharing data with neighbors as necessary.

Ma and Yang (2007) show that the most efficient dispersion of mobile nodes is triangular, producing the maximal overall coverage and minimal overlap or gap in the coverage. Liu et al. (2005) have shown that repeated location updates can lead to better coverage over time. Similar approaches by Howard et al. (2002) and Cortes et al. (2004) used potential fields and gradient descent, respectively, to disperse the nodes. A recent trend has been to model distributed multi-robot algorithms on insect behavior. The robots have very little individual ability, but can communicate with local neighbors and arrange themselves according to a desired dispersion pattern. McLurkin and Smith (2004) have developed both a physical robot and several algorithms for dispersion and exploration in indoor environments. These algorithms are similar to those in (Cortes *et al.* 2004; Howard *et al.* 2002; Liu *et al.* 2005), but allow for greater variability in the dispersion pattern, including clusters and perimeter formations. The robots can also perform tasks such as frontier exploration and following-the-leader.

Dirafzoon et al. (2012) provide an overview of many sensor network coverage algorithms which can be applied to multi-robot systems as well. However, many of these rely on individual robots knowing the distance and bearing of other robots around them, which requires more sophisticated sensors. For example, Kurazume and Hirose (2000) developed an algorithm in which the team of robots was split into two groups, one of which remained stationary while the other moved, and then they traded roles. This made for effective movement through an unknown environment, but the robots relied on sophisticated sensors to perform dead reckoning to determine the locations of the stationary robots. On the other hand, research has shown that a team of robots can disperse into an unknown environment using only wireless signal intensity to guide the dispersion (Ludwig and Gini 2006; Jensen and Gini 2013). This method allows the use of simple robots, without the need to carry a heavy payload of sensors, so that the robots can run longer and explore further. Smaller, simpler robots are also less expensive, so more robots can be acquired for a task.

Sweep Exploration Algorithm

The objective of our algorithm is to achieve full exploration of an unknown environment using a team of robots. We do not need blanket or barrier coverage for our scenario, so we can use a small team, even making do with a single robot if needed. We also do not need an exact map, so long as our robots can provide markers to the points of interest, so the robots don't need complex or expensive sensors either. We have chosen a distributed approach to take advantage of the robustness that comes with having multiple robots.

We assume that the robots have a proximity sensor to avoid collisions, the ability to communicate (not restricted to wi-fi, but could be through cameras and LEDs, chemical signals, etc), and the means to carry and drop beacons (such as ZigBee motes or RFID tags). We also assume the specifics of the current environment are unknown, even if information for the pre-disaster environment (such as a map) is available.

The main feature of our algorithm is that the robots use communication as a means to direct their movements, not just to send status updates. This helps to ensure that the robots stay within range of each other, and the algorithm is not dependent on a specific communication method to function correctly, making it more versatile both in the scenarios in which it can be used, and the types of robots which can use it for exploration. This use of communication does require that the robots stay connected at all times, but this restriction also provides a benefit. Because the robots spread out as much as possible while staying connected, each individual robot has few neighbors, keeping local bandwidth requirements low even with larger teams. Lower bandwidth usage makes each message stand out more, and preserves power so that the robots can function for longer stretches.

Our algorithm uses the intensity of the communication to disperse the robots, and beacons to mark locations, but the innovation in our approach lies in how the robots coordinate and explore beyond the bounds of their initial dispersion. Not allowing the team to split up gives our approach two advantages. First, the robots are less likely to get lost, since they will have a trail to follow to get back to the entrance. Second, the robots will clear each room and corridor in a methodical manner, similar to the pattern used in law enforcement, thus reducing the likelihood of missing an area. The most important feature and innovation, however, is that our algorithm will achieve full exploration independent of the type of communication in use, including radio, line-of-sight, and chemical methods.

Algorithm Details

The Sweep Exploration Algorithm (SEA) is based on our previous work, the Rolling Dispersion Algorithm

(RDA), which is shown in an abbreviated form in Algorithm 1 (Jensen and Gini 2013).

Algorithm 1 Rolling Dispersion Algorithm

```
loop
    Update and share connectivity graph
    if I am too close to an obstacle then
        set behavior to Avoid Collisions
    else if I am disconnected then
        set behavior to Seek Connection
    else if I am in a dead end then
        drop a beacon set to Repel
        set behavior to Retract
    else if I am on the frontier then
        set behavior to Guard
        if my only neighbor is my sentry then
            request additional explorers
    else if I have received a request then
        if I am an explorer then
            drop a beacon set to unexplored
            set behavior to Follow Path
        else if I am a sentry w/a beacon parent then
            drop a beacon set to unexplored
            set behavior to Follow Path
        else
            set behavior to Guard
            pass on the request
    else if I have reached the requesting robot then
        set behavior to Disperse
    else if if I am an explorer then
        set behavior to Disperse
    else
        set behavior to Guard
    Apply chosen behavior
```

In our new approach, each robot still uses information about connectivity with its neighbors and nearby obstacles to determine its next move. Each robot is constantly working to avoid obstacles and maintain connectivity, but from there we diverge from the previous algorithm. We are primarily concerned about reducing the communication load, so rather than delineate only behaviors, we have instead focused on what messages are essential to complete the exploration. We have narrowed it to the following seven messages, which also correspond to robot and beacon states, shown in Table 1. Each message is comprised only of the state and no other information, and can thus be encoded as a color or 3-bit message, depending on the communication method, making the message size extremely small.

Figure 1 shows the process of how each robot and beacon uses information from their local environment, including state messages from neighboring agents, to determine their own state, which then informs their next movement behavior. The beacons are not mobile, and so do not change their behavior at all, but their state can change as a means to pass information along to other agents, for instance requesting additional robots to explore a frontier or to fill a gap from an agent failure. Robots decide between the following five movement behaviors (reduced from the six in the RDA).

AVOID COLLISIONS: The robot uses its proximity sensors to avoid colliding with walls, objects and other robots.

DISPERSE: The robot moves away from neighbors and explored areas using the communication signal intensity.

GUARD: The agent stays in place to act as a sentry for the other agents. This is the only behavior that beacons use.

RETRACT: The robot moves towards the nearest agent on the retract path.

SEEK CONNECTION: The robot seeks to reconnect with the group if the connection is lost.

As the robots initially move into the environment, they establish a path of robots and beacons between the entrance and the current frontier. This path can be used to guide a robot to the edge of the explored area. As robots reach the edge of their communication range, they will call for more explorers, a message which will propagate back to any robots not yet exploring or guarding an area. The robots will eventually reach the maximum dispersion they can achieve without losing communication. As in the initial stage, the robot on the edge of the frontier will send out a call message, but this time all robots will be busy. Instead the message will propagate back to the beginning of the robot path. The robot closest to the starting point of the path will drop a beacon to mark the path and then switch states to explorer and follow the call path to the frontier.

When a robot reaches a dead-end in the area it is exploring, it will send a retract message and return along the retract path. A dead-end is defined as a location where the explorer is surrounded by walls, repel beacons, and robots on the retract path, such that there is nowhere for the explorer robot to move that is not towards a wall or an explored area. As the robot retracts, it will drop a repel beacon to mark the explored area and reduce the likelihood of repeat explorations of the area. The robot will return to the last branching point, and then become an explorer again and move along another branch to explore that frontier. The branching points are determined using the robots' proximity sensors. In our method, each branch is explored sequentially, so we do not need to worry about a call message going down multiple branches at once and resulting in multiple responses.

Table 1: Messages.

Name	**Robot**	**Beacon**	**Behavior/Purpose**
Explorer	Yes	No	Explore frontier or follow call or failure paths
Branch	Yes	Yes	Mark a branch point, stay in place and relay messages
Call Path	Yes	Yes	Mark path to frontier
Retractor	Yes	No	Return to last branch point or entry
Retract Path	Yes	Yes	Mark return path for retractors
Repel	No	Yes	Mark explored areas to prevent repeated coverage
Failure Path	Yes	Yes	Mark path to failed robot

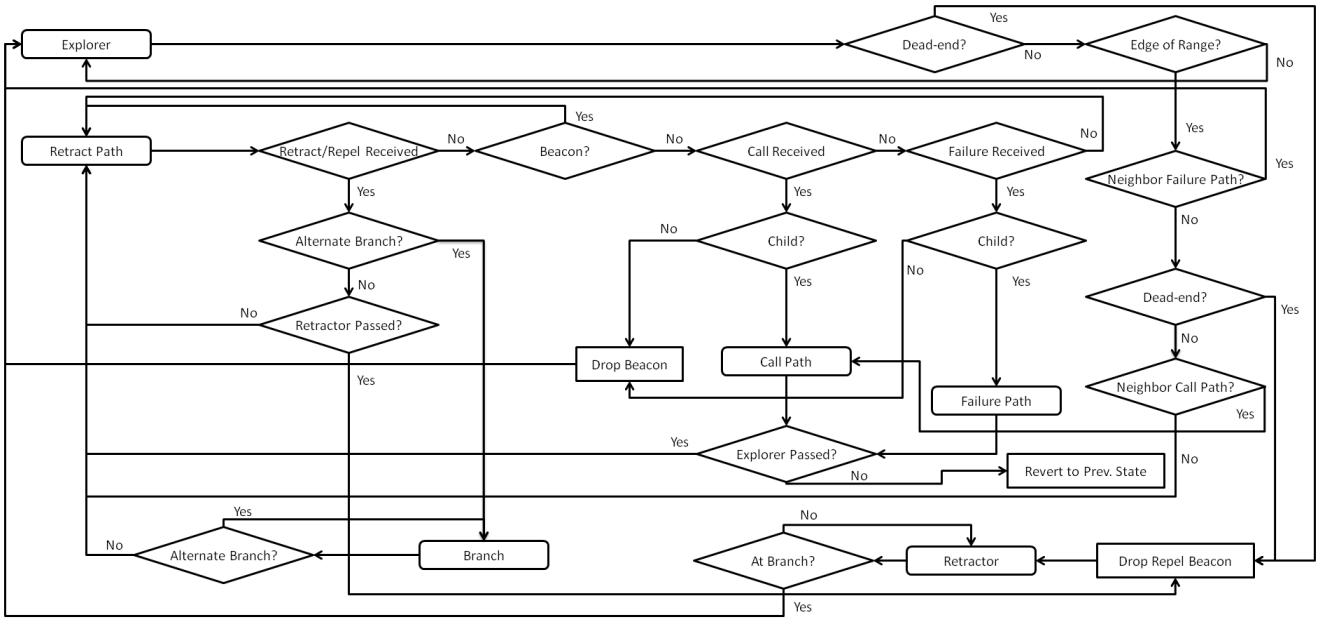

Figure 1: A flowchart showing how the Sweep Exploration Algorithm is executed on each robot at each iteration. Round-cornered rectangles represent states, diamonds represent decisions, and rectangles represent actions.

In summary, the Sweep Exploration Algorithm works by first having the robots disperse. This movement is primarily directed by the communication signal intensity between the robots, as in previous works (Ludwig and Gini 2006; Jensen and Gini 2013), with influence from both the robots' proximity sensors and the messages sent between robots. The robots will extend along a single branch until they have fully explored it, then return to the previous branch, leaving behind repel beacons to prevent repeat exploration, and continue exploration from the branch point. When necessary, a robot from early in the path will be replaced by a beacon, and the robot will then move to the frontier to continue the exploration. When the exploration is complete, the robots all retract to the entrance, leaving behind a trail of beacons.

Algorithm Correctness

The primary goal of the algorithm is to achieve full coverage of the environment by having each point in the environment viewed at least once by a robot, with the constraint that the robots maintain communication with each other throughout the exploration. The following analysis shows that the algorithm guarantees complete exploration, so long as a single robot remains functional, and that the robots will return to the entrance at the end. We also show that we can use no fewer than the seven messages/states previously described.

Preventing Infinite Loops Our first concern in completing the exploration is that of infinite loops, where the robots end up circling an obstacle or repeatedly cycling through a set of rooms that all connect. Our algorithm avoids this through the use of both the retract path agents and repel beacons, which mark explored areas. In Figure 2, the circle represents an explorer robot, and the squares represent retract path agents (either robot or beacon, the form does not matter). In its current location in Figure 2a, the explorer robot has walls on two sides, and retract path agents on the other two sides, so it detects that it has reached a dead-end. It has been moving down the gradient of signal intensity from the agent it just passed (which is below it in the figure), but on finding the dead-end, it will reverse direction and move back up the gradient between a third and halfway to the retract path agent before dropping a repel beacon marking the location as explored, and begin retracting (Figure 2b). In the worst case, the explorer robot will have actually gone towards the other retract path agent (to the left in Figure 2a), which will result in the robot exploring the loop twice, but on the second pass, the explorer robot will reach the repel

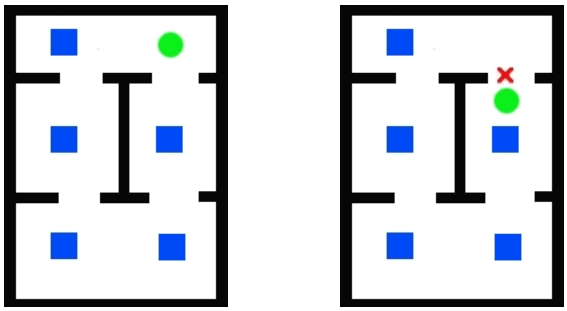

Figure 2: The robot and beacon placements before the cycle has been detected (a), and after blocking the cycle to prevent infinite looping (b).

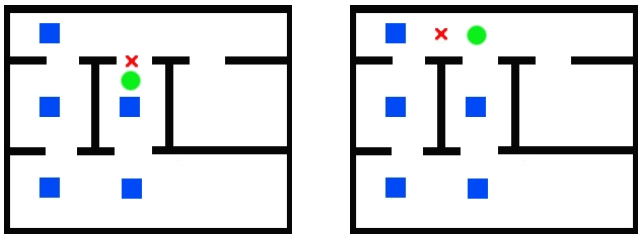

Figure 3: A complex loop example (two solutions shown).

beacon and retract back to the entrance.

If, instead of the simple loop described above, we have a larger environment with a loop, such as in Figure 3, when the explorer robot reaches the intersection shown, it has a wall on one side and retract path agents on two sides, but the remaining side is open. Even with very simple forms of communication, we assume that the robots can determine the number of neighbors they have, even if they cannot differentiate each other (no IDs). Thus, the explorer robot will take note that it is in an intersection, where it can determine that there are three branches and retract path agents in two of those branches. It will retract towards one of the retract path agents, drop a repel beacon as in the case above, and retract back to the previous branch. Depending on the direction in which it retracts, the explorer robot may end up most of the loop path again, but the repel beacon will prevent it from attempting another cycle, so again, the worst case scenario leaves us with the majority of the cycle being traversed at most twice. Note that by dropping the beacon within range of one of the two retract path agents, the explorer robot leaves a path open to complete the exploration no matter which location it chooses.

Robot and Beacon Failures The next issue is to deal with the possibility of robot or beacon failures. Given the disaster scenario we are considering, where the environment is too dangerous for a human to enter, there is a high likelihood that a robot or beacon may be destroyed during the exploration. If the failure occurs at the edge of the explored region, then a failed beacon means re-exploring a small area. A failed robot would be on the frontier only if it was waiting for more explorers, so the loss will have little impact, as a new explorer would already be on the way and would simply take the place of the failed robot. A failed explorer would need to be replaced, which would be taken care of when the agent closest to the frontier loses the connection with the explorer and sends a call message for a new explorer. If a beacon fails in the middle of a path, it may not be noticed until the robots are retracting, but at that point they will treat the empty area as an unexplored region and will be buffered by other beacons, so it won't be a large gap for them to re-explore as they find their way back to the entrance. If a robot between a beacon and a robot in the middle of the path fails, then it will be treated as if it was a beacon, and will be replaced when the robots retract. However, if a robot between two other robots fails in the middle of a path, that is the most difficult failure to resolve. One of the robots will notice the failed robot's absence and call for a replacement. This causes an issue because the new explorer will find itself between two retract path agents, and probably with walls on the other sides, which would normally mean a loop and lead to the explorer dropping an explored beacon and retreating. We considered a buddy system, where every retract path agent is made up of two agents (two robots or a robot and a beacon), but determined that the best solution was to use a message for a failed robot, such that when the replacement robot reached the gap, it would become a retract path agent without dropping an explored beacon.

If more than one robot or beacon fails, they will be filled in one at a time in the manner described above. Should it not be possible to reconnect the path, an alternate path will be sought (through reopening explored areas to search for cycles), and should that fail, then the robots would use the break in the path as the new entry point, in the hopes that when the humans can get in, they will be able to pass the obstruction blocking the robots, and can then pick up the information from the waiting robots (with the beacon trail completed beyond the obstruction). This method will complete the exploration so long as a single robot survives to the end, though in the worst case exploration of an area may be repeated each time the agent in that area fails, but never more times than there are failed agents.

Complete Exploration To achieve complete exploration, we first need to ensure that there are no infinite loops and that the algorithm will continue so long as a single robot remains active. In addition, we need to show that the robots will not miss an area. In the section on infinite loops, we showed that preventing loops will not cut off an unexplored area by accident. Robots will continue into an area until they reach a dead-end. At the dead-end, the robots then work backwards and leave behind repel beacons to block off the area from repeated exploration. Each robot will only leave a single repel beacon along its retract path. If it was the explorer that found the dead-end, then it leaves the repel beacon there. If it was a retract path agent, then it waits until all retractor robots have passed it before it drops its own repel beacon and retracts as well. In this way, the path remains until all robots beyond that point have retracted, and only then does the last agent on the path block it off. This continues back to the most recent branching point along the path.

A branching point is identified by the agent in the intersection. The robot proximity sensors can detect obstacles up to the width of an average corridor, so a robot can detect when there are multiple branches. It does not matter which branch an explorer robot chooses to explore first. When the robots complete exploration of a branch, a repel beacon will be placed on the edge of the intersection for that branch, blocking it off without obstructing the other branches. The repel beacons are treated as walls. The agent at the branch point will decrement its branch count when a repel beacon is placed. Eventually, this leads to the retraction path pushing back to a previous branching point. When there are no branching points left, the retraction path extends to the entrance. When all robots have reached the entrance again, with all branches explored the exploration is complete.

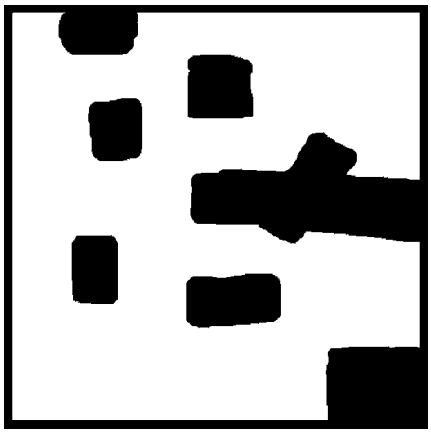

Figure 4: A cave-like environment with large obstacles and lots of open space.

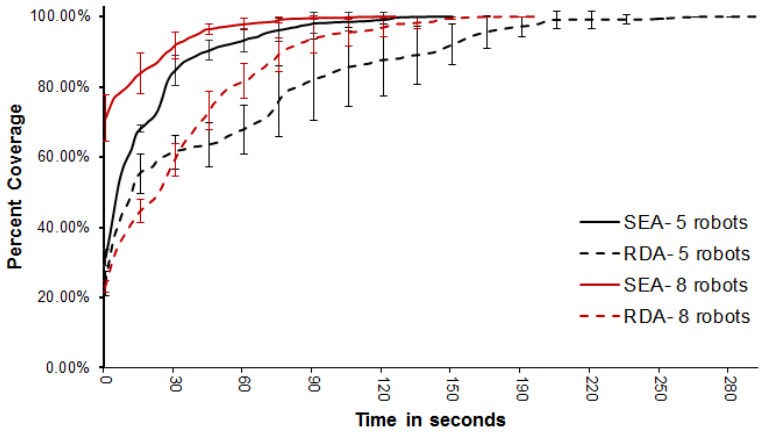

Figure 5: Percentage of environment covered over time for the SEA and RDA algorithms in the cave environment.

Minimum Message Types The last property we wish to demonstrate about our algorithm is that it would not complete the exploration with fewer than the seven previously described messages/states. Consider starting with only two messages/states. The explorer and retract path are the most straightforward, and both are needed in order to create the backbone of paths to the frontier and back to the entrance. But there is no way to request additional robots to the frontier when the last explorer reaches the edge of the communication range with its nearest neighbor. So we need a third state, the call path, which both makes the request and leads the new explorer to the frontier. But then the explorer hits a dead-end, and without the repel state, the robots might explore the same area over and over, or go into an infinite loop. This brings us to four states. Without the retractor state, the agents on the retract path wouldn't know when to either change to the repel state (for beacons) or retract themselves (for robots). Without the branch state, each retractor robot would go all the way back to the beginning, and a branch might be cut off from further exploration because the retraction protocol requires that the robots leave repel beacons along the way. We have already explained the need for the failure path state, making seven messages/states.

Experiments and Results

We used simulation to test the algorithm's viability, and ran our experiments using ROS/Stage. We use the cave environment and the same number of robots and maximum robot speed as used in previous experiments with the RDA for comparison. The environment, shown in Figure 4, is very open, and has large obstacles at irregular intervals and non-uniform shapes, with wide open areas and potential loops. All the robots start in a cluster in the area between the three obstacles in the upper left corner.

We ran sets of experiments with five and eight robots. The reported percentage coverage is averaged over 10 experiments for each set. In both cases, SEA achieved full exploration faster than RDA. With five robots, our algorithm was 1.59 times faster, with an average of 120.67 seconds to complete the exploration, compared to 192 seconds using RDA. With eight robots, our algorithm was 1.48 times faster, with an average for 108.83 seconds compared to 161 seconds. With SEA, our experiments started with a higher percentage of coverage than RDA, due to the way the robots are clustered at the start to prevent collisions on start-up (since SEA uses a different initial dispersion method), but facing in different directions, so they have a large field of view from the starting point, but even accounting for that (by adding the 10 seconds it took RDA to go from its starting coverage to SEA's starting coverage) our algorithm would average 118.83 seconds, which is still 1.35 times faster. The average time to complete the exploration and the percentage covered at a given time are shown in Figure 5 for both algorithms. We believe that SEA reaches full coverage faster than RDA because robots use fewer messages and spend less time processing incoming messages when deciding their next move.

Conclusions

We have developed a distributed algorithm which allows a team of robots to completely explore an unknown environment while staying connected at all times. The algorithm requires fewer robots than would be needed for blanket coverage, but still provides the necessary information about the environment. We have shown that the algorithm achieves full exploration so long as at least one robot remains active until the end, and that upon completion, the remaining active robots will return to the exit, leaving behind a trail of beacons that can be used by the rescue team at a later time. Our simulation experiments demonstrate that the algorithm works and is faster than our previous algorithm. In future work, we will test the algorithm in larger and more complex environments and in physical experiments.

Acknowledgment: Partial support is gratefully acknowledged from NSF grant IIS-1208413.

References

T. Arai, E. Pagello, and L. E. Parker. Guest editorial advances in multirobot systems. *Robotics and Automation, IEEE Transactions on*, 18(5):655–661, October 2002.

W. Burgard, M. Moors, C. Stachniss, and F.E. Schneider. Coordinated multi-robot exploration. *Robotics, IEEE Transactions on*, 21(3):376–386, June 2005.

Howie Choset. Coverage for robotics – a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31:113–126, 2001.

J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *Robotics and Automation, IEEE Transactions on*, 20(2):243–255, April 2004.

A. Dirafzoon, S. Emrani, S. M. Amin Salehizadeh, and M. B. Menhaj. Coverage control in unknown environments using neural networks. *Artificial Intelligence Review*, pages 237–255, 2012.

D. W. Gage. Command control for many-robot systems. In *19th Annual AUVS Technical Symposium*, pages 22–24, 1992.

Andrew Howard, Maja J. Mataric, and Gaurav S. Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems*, pages 299–308, 2002.

E. A. Jensen and M. Gini. Rolling dispersion for robot teams. In *Proc. Int'l Joint Conference on Artificial Intelligence (IJCAI)*, pages 2473–2479, 2013.

R. Kurazume and S. Hirose. An experimental study of a cooperative positioning system. *Autonomous Robots*, pages 43–52, 2000.

Benyuan Liu, Peter Brass, Olivier Dousse, Philippe Nain, and Don Towsley. Mobility improves coverage of sensor networks. In *MobiHoc '05: Proc. 6th ACM Int'l Symposium on Mobile ad hoc Networking and Computing*, pages 300–308, New York, NY, USA, 2005. ACM.

L. Ludwig and M. Gini. Robotic swarm dispersion using wireless intensity signals. In *Proc. Int'l Symposium on Distributed Autonomous Robotic Systems (DARS)*, pages 135–144, 2006.

Ming Ma and Yuanyuan Yang. Adaptive triangular deployment algorithm for unattended mobile sensor networks. *Computers, IEEE Transactions on*, 56(7):946–847, July 2007.

J. McLurkin and J. Smith. Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots. In *Proc. Int'l Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2004.

Ioannis Rekleitis, Gregory Dudek, and Evangelos Milios. Multi-robot exploration of an unknown environment, efficiently reducing the odometry error. In *Proc. Int'l Joint Conference on Artificial Intelligence (IJCAI)*, volume 2, pages 1340–1345, Nagoya, Japan, August 1997. Morgan Kaufmann Publishers, Inc.

C. Stachniss and W. Burgard. Exploring unknown environments with mobile robots using coverage maps. In *Proc. Int'l Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

E. Stump, A. Jadbabaie, and V. Kumar. Connectivity management in mobile robot teams. In *Proc. IEEE Int'l Conference on Robotics and Automation (ICRA)*, pages 1525–1530, May 2008.

K. M. Wurm, C. Stachniss, and W. Burgard. Coordinated multi-robot exploration using a segmentation of the environment. In *Proc. IEEE/RSJ Int'l Conference on Intelligent Robots and Systems*, pages 1160–1165, Sept. 2008.