Sustainable Multi-Robot Patrol of an Open Polyline

Elizabeth Jensen, Michael Franklin, Sara Lahr, and Maria Gini

***Abstract*— We present an algorithm that maintains coverage of an open polyline patrolled by a team of robots. While previous work has focused on the uniformity of patrolling, we focus on ensuring the longevity of the system in the face of robot failures. A central control tower monitors the battery levels of the robots, and recalls them when they are low on power replacing them with fully charged robots. We compare two methods for replacement, both of which aim to keep coverage interruptions to a minimum. We present results obtained through physical experiments and simulations.**

I. Introduction

The purpose of a patrol is to continuously travel in an area to protect or supervise it. Common goals are uniform frequency coverage [1], [2], [3] and detection of adversaries [4], [5]. Frequency-based patrol systems are usually designed for surveillance, but are also useful in tasks such as cleaning, garbage collection, and monitoring. Multi-robot teams are often used for such tasks, as they can improve performance by reducing the area covered by each robot and increasing the frequency of visits of each point [3].

In previous work, algorithms have focused on the different types of routes that need to be covered. With a circular path, any point can be reached from any other point, which makes uniformity of coverage trivial to achieve: simply space the robots evenly and set them to all move along the path in the same direction and at the same velocity. When the path to be patrolled is not a closed tour, but rather an open polyline, providing frequency guarantees becomes more difficult [3].

A detail that is often overlooked is the fact that robots have limited power sources and must be recharged. In the case of a multi-robot patrol system, where we wish to provide a frequency of visit guarantee for the entire patrol route, a robot running out of power would violate those guarantees. We present here algorithms that take this limitation into account, with the primary goal being the longevity of the system in the face of robot failures. We go beyond the frequency-based patrol algorithms that have been previously presented, and extend them to adapt so that a replacement can be sent in when a robot runs low on power. This is made more complex because we focus on open polylines.

The main contributions in this paper are (1) algorithms that successfully maintain a multi-robot polyline patrol system with frequency guarantees, and (2) experimental validation of the algorithms, both with physical experiments and simulations. The remainder of this paper is structured as follows: Section II reviews relevant previous work; Section III describes our long-term multi-robot patrol; we then present results of experiments with real robots and in simulation in Sections IV and V; and we conclude in Section VI.

E. Jensen, M. Franklin, and M. Gini are with the Department of Computer Science and Engineering, University of Minnesota–Twin Cities `ejensen@cs.umn.edu, frank511@umn.edu, gini@cs.umn.edu`

S. Lahr is with the Division of Science and Mathematics, University of Minnesota–Morris `lahrx019@morris.umn.edu`

II. Background and Related Work

Coverage by a team of robots is a common task. Choset provides a detailed summary of different techniques [6]. Coverage generally seeks to visit every point only once (e.g., [7], [8]), while patrol requires that every point be visited as often as possible [9] or repeated times but with some randomness [10]. Some algorithms explore the environment as they cover it, while other create paths based on the communication capabilities of the robots (e.g., [11]). Our robots know how to reach the line they have to patrol and have limited communication channels, so our algorithm assigns each robot a segment to patrol on start-up, but the robots are dynamically reallocated as needed thereafter.

Frequency-based patrolling attempts to achieve uniform frequency of point visits. In [2] this is achieved by generating a closed path that covers the entire area and distributing the robots uniformly along this path. When cyclical behavior is an option, the cyclical path has been shown to be optimal [12]. However, there are tasks for which a cyclical path cannot be created, as shown in [1].

Patrolling a line with uniform frequency requires a more complex patrolling algorithm than patrolling a circular path due to the end segments that cannot be visited with the same frequency. The overlapping patrol algorithm presented in [3] uses an overlap factor, o, which is set to some positive integer (see Alg. 1). Each of the robots patrols across its own segment, and then continues to patrol $o-1$ segments to the right, stopping if it reaches the end of the patrol route. The robots pause at the end of each segment to synchronize before proceeding to the next segment. The overlapping patrol improves the average frequency of point visits in the middle segments, and attenuates the negative impact on uniformity caused by the end segments because there are usually many more middle segments than end segments. Our work extends this algorithm by removing the need to synchronize the robots at the end of each segment and by maintaining coverage when robots leave the segment they are patrolling to be recharged.

III. Long-Term Open Polyline Patrol

High-frequency coverage is unsustainable if the robots cannot be replaced to allow for recharging/refueling. Our

Algorithm 1 Frequency-based Overlapping Patrol Algorithm [3]

```
1: repeat
2:     Patrol $o$ segments, or until edge segment is reached
3:     Turn and synchronize with other robots
4:     if located in right-most segment then
5:         Wait for left neighbor to move over one segment
6:     end if
7:     Turn and synchronize with other robots
8: until Stop command issued
```

main objective in this research is to keep interruptions in coverage to a minimum by taking advantage of the robustness inherent in multi-robot systems. In this section, we specify the problem and our approach. We then present the algorithms we developed, and discuss performance criteria.

We start with a team of r robots and a line split into s segments of equal length l, where $s \leq r$. At the center of the line, there is a control tower where extra robots are queued until they are needed (see Fig. 1). We assume that the length and number of segments is commensurate with the amount of power needed for the robots to move.

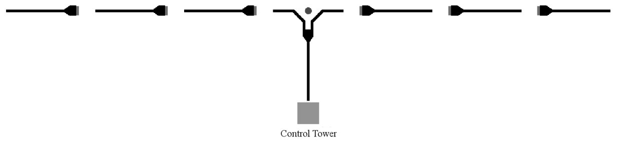

Fig. 1. The layout, with $s/2$ segments on each side of the control tower.

We define a lap as the patrol of a segment in both directions (forward and back) and $2T$ is the time it takes the slowest robot to complete a lap (worst-case lap completion time). Lap times remain consistent until the battery power drops below a certain level, at which point the robot is recalled. That threshold, b, is set to ensure that the robots are able to return to the control tower under their own power. Though all robots start at the same battery level, we have found that, in practice, it is rare for even two robots to fall below this threshold at the same time.

A. Algorithms.

We have based our algorithms on Alg. 1, using an overlap factor of $o = 1$, so each robot patrols only its assigned segment. We removed the need for robots to synchronize amongst themselves at the end of each segment, as the original algorithm required. By allowing our robots to patrol their own segment independently of other robots, we reduced communication costs, which are often one of the largest drains on battery power, and the idle time caused by variability in the robots' movements. We also include in our algorithms the initial deployment of the robots. We present here three algorithms: the base algorithm, the proactive recall algorithm, and the leap-frog algorithm.

Algorithm 2 Base Algorithm

```
1: Assign and deploy a robot to each patrol segment
2: repeat
3:     if Battery level is greater than threshold then
4:         Complete one patrol lap
5:     else
6:         Send low battery message
7:         Leave segment
8:         Locate and return to tower using camera
9:     end if
10:    Check messages from other robots
11:    if a segment is empty then
12:        for all active robots $r$ do
13:            if $r$ is patrolling a segment between empty segment and tower then
14:                Move $r$ out one segment
15:                Adjust segment number of $r$
16:            end if
17:        end for
18:        Deploy reinforcement robot from idle queue at central control tower
19:    end if
20: until All robots have failed
```

On system start-up, all robots are located at the control tower's queue, and the tower initiates communication with each robot. Robots also start communication channels with each other, so that they can reorganize as needed (see Alg. 2). Once communications are initialized, robots are deployed to each segment in the polyline until all segments are covered. Any remaining robots wait in the queue until needed.

In the *base algorithm*, once deployed each robot autonomously patrols its assigned segment until it either runs low on power or it receives a message that indicates it needs to move to a different location. Once a robot reports that it is low on power, it leaves its assigned segment and returns to the control tower. Each robot patrolling between the tower's queue and the now empty segment will jump to the next segment out, filling in all but the segment closest to the central tower. That last segment will be filled by a robot deployed from the queue. We assume that there is a supply of charged batteries at the control tower, and that the time to replace batteries in a robot when it returns to base is negligible in comparison to the time taken to deplete them.

The *proactive recall algorithm* (see Alg. 3) aims to reduce both the communication requirements and the number of robots returning at any given moment. This algorithm makes use of a group leader for the team of robots, which we have set as the control tower in our examples and implementation, though it could be any of the robots instead. We use two threshold levels in this algorithm, $b_l = b$, the threshold at which a robot must be recalled; and $b_u = b_i + e$, where e is some small constant (0.1 volts in our experiments) that acts as a warning level threshold. Every ten laps, the robots send the leader a message with their battery level. The leader maintains a sorted list of robot battery levels, which it uses

Algorithm 3 Proactive Recall Algorithm

```
 1: Sort robots by increasing value of battery levels
 2: Let $b_1, b_2, b_3$ be respectively the battery levels of the first
    three robots
 3: if $b_1$ is below threshold $b_l$ then
 4:     Recall first robot
 5: else if $b_1$ is below threshold $b_u$ then
 6:     Check battery level $b_2$ of next robot
 7:     if $b_2 < b_1 + e/2$ then
 8:         if $b_3 < b_2 + e/2$ then
 9:             Recall second robot
10:         else
11:             Recall the first robot
12:         end if
13:     end if
14: end if
```

to determine when a robot needs to be recalled.

The *leap-frog algorithm* improves upon the deployment of replacement robots. In our base algorithm, the robot with the most battery power remaining always ends up nearest the control tower, while the robots with less power end up further away. We hypothesized that this would be sub-optimal, as those robots would then have to go further to get back to the control tower, so they would need to be recalled sooner than if they had stayed in their original places. In the leap-frog algorithm, the replacement robot makes its way to the segment that was vacated, rather than simply filling in at the segment closest to the control tower. To do this, we relaxed the constraint that only one robot may be on a segment at a time, which we accomplished by adding a second segment parallel to each existing segment to act as a passing lane.

B. Performance Criteria.

We use the performance criteria proposed in [3]. *Uniformity* refers to the frequency with which each point on the route is visited. We would like to reduce the variance in this frequency. *Maximal average* refers to the average frequency of visits to each point. We want to maximize this, so that each point is visited as often as possible, on average. The main goal here is to make sure that no point is left unvisited for too long. *Maximal minimum frequency* is the minimal frequency of visit for any point, which we want also to maximize. This is accomplished trivially by setting the overlap factor $o = 1$.

When no robots are low on power, our patrol system has the same properties as that of [3]. As in the original algorithm, the frequency of visits is uniform only at the center of each segment, and the difference in visit frequencies grows larger as points are closer to the end of the segment. The endpoints will be visited twice in a row, so the time interval between visits is 0, but it will take an entire lap before the endpoint is visited again, which will average to T between visits. At the center point of a segment, the robot will move to the end and then back to the center, which takes T time to complete. For all other points the two intervals vary, but the average time interval between visits is still T.

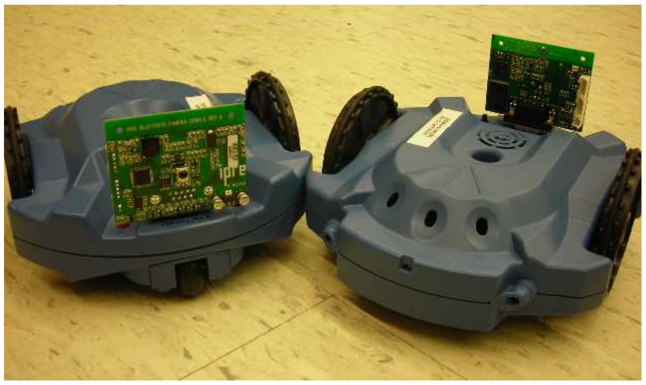

Fig. 2. Two Scribbler robots with attached Flukes.

Determining the maximal average of visits is more difficult when a robot returns to base and other robots fill in for it. For a short time, the interval between visits to each point is increased, because one segment is not being covered. When the robot, r_0, patrolling segment s_0, runs low on power, the robot to its left, r_1 must fill in. In the best case, r_1 will be just arriving at the point between s_1 and s_0 and can simply jump over to s_0, in which case it will take only T to return to the normal visit frequency for segment s_0 (r_1 must reach the far end of s_0 for this to happen, which will take T). In the worst case, r_1 just turned back to return to the opposite end of s_1, a process which we do not wish to disrupt for two reasons. First, we do not want to leave any point uncovered for too long, and completing the lap ensures that we can make guarantees about the frequency of visit; and second, because of odometry errors r_1 might get lost, which would be detrimental to the system's overall performance. Allowing r_1 to complete its current lap leads to a $3T$ delay before s_0 will be patrolled with an average interval of T again. This process must be repeated for every other segment between the vacated segment and the control tower, however robots can be jumping simultaneously, and the new robot is deployed as soon as the near segment is cleared, so it will still only take $3T$ to return to normal.

IV. Physical Robot Experiments

In the physical experiments, we were most interested in showing that our algorithm worked properly in a real world implementation. We wanted to ensure the algorithm would work with the physical robots first, and use the results of the experiments to build our simulation, so that the simulation would be an accurate representation of the real world.

A. Experimental Setup.

We used the Scribbler robots augmented with the Fluke (see Fig. 2), which has a camera and Bluetooth capability. We used the line-following IR sensors located underneath the Scribbler, which provide a binary response distinguishing light from dark, and the camera, which provides blob following. The Scribblers have limited capabilities, so we had to adjust our base algorithm to work within those

TABLE I

RESULTS FROM A SIX ROBOT, FOUR SEGMENT EXPERIMENT. TIMES ARE MEASURED IN SECONDS.

Name	**Anders**	**Caprica**	**Hera**	**Leoben**	**Sharon**	**Tory**	**Overall**
Avg. Lap Time	16.11	20.89	25.21	16.28	28.15	26.29	19.60
Lap Time SD	3.02	3.60	2.52	2.99	2.06	3.92	4.80
Avg. Return Time	3.25	3.43	0	3.83	0	3.75	3.48
Idle Time	66.72	1533.75	0	792.77	152.19	2598.39	5143.82
% Idle	1.64%	37.57%	0.00%	19.58%	21.28%	68.71%	24.82%

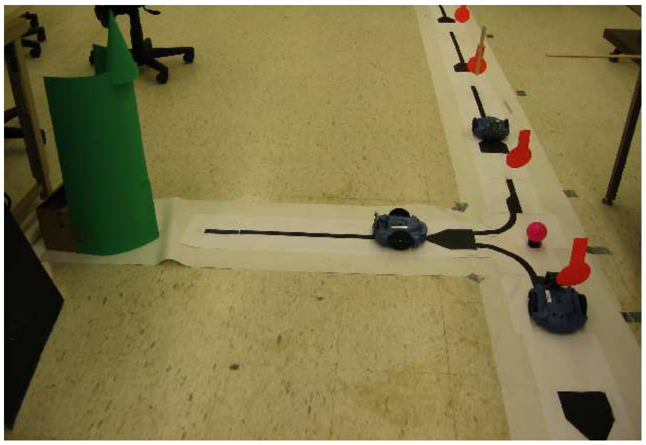
Fig. 3. The physical experimental setup. Two Scribblers are already patrolling segments, while a third stands by in the queue.

limitations. For example, the availability of only Bluetooth communication, forced us to implement our algorithm in a more centralized manner, with all messages going through the control tower.

The physical environment setup can be seen in Fig. 3. It consists of a straight line divided into six equal-length segments. The tower is located in the middle with a queue leading to the main patrol line. Each segment is a black, one inch wide and 33 inch long strip, with an eight-inch approach region on the end nearest to the control tower. All segments are separated by a nine-inch white space to prevent collisions between robots on adjacent segments. The following accommodations were made to attenuate the limitations of these particular robots. The approach region is wider than the rest of the segment to allow for error when the robots jump between segments. The Scribblers follow the segments using the two IR sensors underneath them. Orange targets hung above the path mark the beginning of the segments, and were used for additional alignment functionality using the Fluke's camera. The camera is also used for locating and returning to the control tower, marked by a large green target. Obviously, for a real deployment we would use more sophisticated robots and different methods for tracking the segments to be patrolled.

B. *Experiments and Results.*

We tested the longevity of the system using six robots and four segments, with two robots idle in the queue. The robots were run for one 75 minute experiment, including 1.5 minutes to start-up (tower connecting with robots) and 3.5 minutes of initial deployment time, giving 70 minutes of patrol time. The robots all started at the control tower, and were deployed one by one. The results of this experiment can be seen in Table I. The average lap time was 19.60s and the average time spent idle was 24.82% of the total run time. We also measured the rate of decrease in battery level in a separate, 40 minute experiment. The battery level decreased by 0.33 volts every hour, with an average lap time of 18.12s.

C. *Discussion.*

From these experiments, we observed that the lap times were fairly consistent for each individual robot, but were not uniform across robots. We also saw an increase in lap time between the four robot and six robot experiments, which was caused by two robots being much slower than the other four.

We realized that the amount of idle time and active time depends greatly on the ratio of robots to segments, and is very important to the overall longevity of the system. One reason for this is that too many robots in the queue results in more time spent idling, which is especially problematic if the robots are also expending power during the idle time. We have seen a robot run out of power and fail while idling in the queue. Since we wanted the system as a whole to be as autonomous as possible, we did not want to have to build in an interval in which we would manually switch on a robot only when it was needed, so the idle robots always had to be ready. With r robots and s segments to patrol, where $r > s$, the fraction of the time the robots are idle on average is $(r - s)/r$. While we saw an idle time of only 24% of the total time in the second experiment, the reported times are misleading, because one robot, Sharon, failed upon returning to base, and so 3319 seconds of idle time are missing from the totals. When this is accounted for, the percentage of idle time is 35%, which is closer to what we expected with 1/3 of the robots in the idle queue at all times.

Our base algorithm successfully maintained patrol of every segment. However, there was no way to expand our experiments to a longer line or multiple towers due to space limitations in our lab. Time limitations were also a concern because we wanted to scale the system up, but even the small experiments we conducted required constant supervision, making larger scale experiments impractical. For these reasons, we decided that a simulation, using the data collected from the physical experiments as a model, would be beneficial in further testing of our algorithm. A simulation

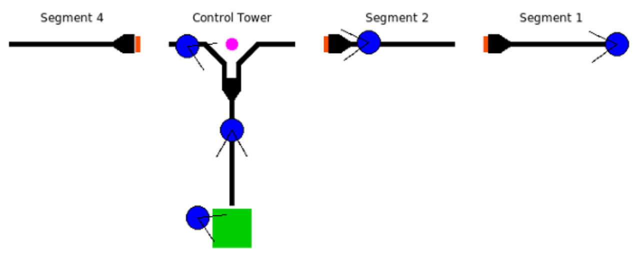

Fig. 4. Partial view of the simulation environment. The control tower is designated as segment 3. The robot on the left is in the process of returning to base. The "whiskers" on the round robots denote the field of view of the camera, which is used to track the tower and to avoid obstacles.

would allow us to test larger and more complicated scenarios, and as well as allow us to do many more experimental runs.

V. Simulations

Our goal for the simulations was to run longer and more complex experiments involving more robots. The main problem we encountered with the physical experiments was the sensor noise inherent in the environment. The line sensors operated in the IR spectrum and finding a material that would absorb IR and return the necessary values proved to be difficult. Even with a suitable medium, inconsistencies such as dirt, changes in lighting, and scuff marks from the robots' wheels caused errors in the sensor readings. In our simulation we wanted to increase the reliability of the sensor data, while retaining the system's non-deterministic properties.

We implemented our own simulation environment (see Fig. 4). Within the simulation environment, we assumed that the robots have perfect sensor readings, but we added noise–a random variable from a Gaussian distribution with a mean of 0 and a standard deviation of 0.125–into the robots' linear and rotational velocities at each time step to simulate environmental and hardware variances. This was combined with the simulated sensors into a proxy that made use of the same API used by the physical robots, allowing us to switch between real robots and simulations with no changes to the underlying patrol algorithms. We ran all three algorithms in the simulations, continuing with the centralized version in order to obtain meaningful comparisons between the simulations and physical experiment results.

A. Experiments and Results.

As a test of the comparability of the simulation to the physical experiments, we measured deployment time for 6 robots over 4 segments, just as we did in the physical environment and found them to be very close (physical took 3.5 minutes, simulation took 3.25 minutes).

Since one of our main reasons for using simulation was to increase the size of our experiments, we then ran a set of simulations using eight robots patrolling six segments, with the same setup and procedure as the physical experiments. The simulation data was collected from multiple runs totaling over 5 hours of patrol time. With 2 robots always idle, we expected to see idle times around 25%. In the runs using the base algorithm, the average lap time was 16.04s, and the

TABLE II
Lap times (in seconds) for each experiment and algorithm.

Environment	Algorithm	Average	St. Dev.
Physical	Base Algorithm	19.60	4.84
Simulation	Base Algorithm	19.22	1.27
Simulation	Proactive Recall	18.90	3.53
Simulation	Leap-Frog	18.65	1.44
Simulation	Leap-Frog + Proactive	19.44	3.05

percentage of time spent idle was 34.18%. The simulations using the proactive recall algorithm had an average lap time of 18.90s and only 11.26% idle time. We determined that this significant difference of idle time stemmed from the way we had structured communication between the robots and the tower in the proactive recall algorithm. We restructured the communication for the base algorithm and found that we now had 19.22s average lap times and 12.21% idle time, similar to the proactive recall results.

In our final set of experiments, we ran the simulations with 10 robots patrolling six segments, and so expected at most 40% idle time. We ran three simulations for comparison, one using the proactive recall algorithm, one using the leap-frog algorithm, and one using a combination of the two algorithms. For each setup, we collected 24 hours of data. Again, the average lap times are similar to those in both the original physical experiments and in our previous simulations (see Table II for complete data). There was no significant difference between any of the three algorithms (see Fig. 5). We also recorded the time between visits to one of the endpoints on each line segment (see Figure 6) for each of the algorithms. We see that there are some fluctuations in point visit intervals depending on the line segment, which we believe is due to how often a segment is traversed on the way to a further out segment. However, we see that the leap-frog method has shorter intervals, because there can be two robots on a segment for a short time, so the patrol is not interrupted as much as in the base and proactive algorithms.

B. Discussion.

In comparing simulations to the physical experiments, using just the base algorithm, there were some minor differences, but overall the results were comparable. The average lap time, in particular, was more consistent than we had expected, though the perfect sensor data reduced the variability of the lap times in the simulation.

We saw greater improvement than we had anticipated after implementing the proactive recall algorithm. Further analysis showed that most of the performance improvement came from the manner in which we implemented the communication between the control tower and the robots. The proactive recall algorithm is still necessary for maintaining maximal average frequency of point visits. In the physical experiments, there were long intervals between returning robots, so it was not apparent that the proactive recall algorithm would be useful, let alone necessary. However, in

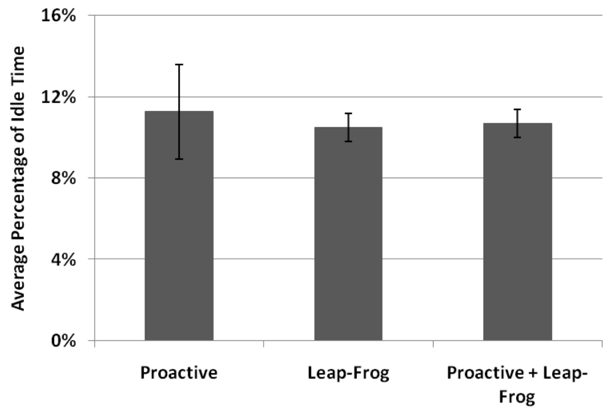

Fig. 5. Average percentage of time spent idle in simulations by algorithm: proactive recall, leap-frog, and leap-frog with proactive recall. Error bars show 1 standard deviation.

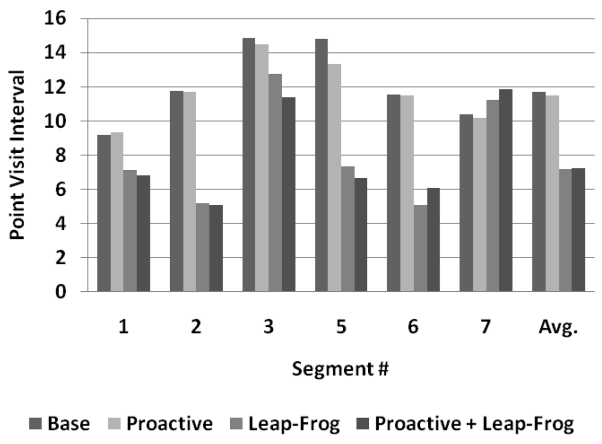

Fig. 6. Time between visits (in seconds) to a specific point on each line segment in a 10 robot, 6 segment simulation.

the simulation we saw all the robots return to the tower at the same time. Not only did this increase the percentage of time spent idle for the entire system, but it also greatly reduced the average frequency with which the segments were patrolled, which is a significant issue when our objective is to maintain a high frequency of coverage. The proactive recall algorithm limits the number of robots that are returning at any given time to maintain the maximal average frequency at the level of our original algorithm.

The average return time with the leap-frog method is shorter than in our original recall method (47s and 65s, respectively), due to the fact that the robots that are returning are closer to the control tower. This is because we are no longer pushing the robots with lower battery power out to the end (since the first robot to return is almost always an end robot), but are instead getting the fresh robot out there. As an added benefit we have more time to recharge a robot after it returns, which will reduce the number of backup robots necessary to maintain the patrol. We expect that this improvement would be even more noticeable if there were more than 3 segments on each side of the tower.

VI. Conclusions and Future Work

We extended a previous algorithm for frequency-based patrol on an open line, with a focus on maintaining the patrol over the long-term. We achieved this by keeping some robots in reserve, while the other robots patrol assigned segments. A robot is removed from the line and replaced once its power level drops below a threshold. We explored different approaches to the recall and replacement actions. We performed both physical robot and simulation experiments.

We have begun work on creating a multi-tower scenario, in which there are multiple control towers, each with their own set of segments and robots. This extension paves the way for a large scale implementation, with the added bonus that neighboring towers will be able to lend or borrow robots if necessary, making the system more robust to robot failures. We are also exploring the best allocation of resources, so that as few robots as possible are idle in the queue. One method is to continually move robots with lower power reserves towards the tower to minimize replacement time. Our third line of inquiry is into paths composed of complex polylines, with curves and unequal segment lengths, in order to simulate changes in terrain.

References

[1] A. Almeida, G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyre, “Recent advances on multi-agent patrolling,” in *Advances in Artificial Intelligence (SBIA 2004)*, April 2004, pp. 474–483.

[2] Y. Elmaliach, N. Agmon, and G. Kaminka, “Multi-robot area patrol under frequency constraints,” in *Proc. IEEE Int’l Conf. on Robotics and Automation*, April 2007, pp. 385–390.

[3] Y. Elmaliach, A. Shiloni, and G. A. Kaminka, “A realistic model of frequency-based multi-robot polyline patrolling,” in *Proc. Int’l Conf. on Autonomous Agents and Multi-Agent Systems*, 2008, pp. 63–70.

[4] N. Agmon, S. Kraus, and G. Kaminka, “Multi-robot perimeter patrol in adversarial settings,” in *Proc. IEEE Int’l Conf. on Robotics and Automation*, May 2008, pp. 2339–2345.

[5] N. Basilico, N. Gatti, and F. Amigoni, “Leader-follower strategies for robotic patrolling in environments with arbitrary topologies,” in *Proc. Int’l Conf. on Autonomous Agents and Multi-Agent Systems*, 2009, pp. 57–64.

[6] H. Choset, “Coverage for robotics – a survey of recent results,” *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1-4, pp. 113–126, 2001.

[7] K. Williams and J. Burdick, “Multi-robot boundary coverage with plan revision,” in *Proc. IEEE Int’l Conf. on Robotics and Automation*, May. 2006, pp. 1716 –1723.

[8] P. Amstutz, N. Correll, and A. Martinoli, “Distributed boundary coverage with a team of networked miniature robots using a robust market-based algorithm,” *Annals of Mathematics and Artificial Intelligence*, no. 2-4, pp. 307–333, 2009.

[9] A. Machado, G. Ramalho, J.-D. Zucker, and A. Drogoul, “Multi-agent patrolling: An empirical analysis of alternative architectures,” in *Multi-Agent-Based Simulation, Third International Workshop*, 2002, pp. 155–170.

[10] A. Marino, L. E. Parker, G. Antonelli, and F. Caccavale, “Behavioral control for multi-robot perimeter patrol: A finite state automata approach,” in *Proc. IEEE Int’l Conf. on Robotics and Automation*, 2009.

[11] I. Rekleitis, A. P. New, E. S. Rankin, and H. Choset, “Efficient boustrophedon multi-robot coverage: an algorithmic approach,” *Annals of Mathematics and Artificial Intelligence*, vol. 52, no. 2-4, pp. 109–142, 2008.

[12] Y. Chevaleyre, F. Sempe, and G. Ramalho, “A theoretical analysis of multi-agent patrolling strategies,” in *Proc. Int’l Conf. on Autonomous Agents and Multi-Agent Systems*, vol. 3, 2004, pp. 1524–1525.