MULTI AGENT PERSISTENT TASK PERFORMANCE

An Undergraduate Research Scholars Thesis

by

JAMES MOTES

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Nancy Amato

May 2018

Major: Computer Engineering

TABLE OF CONTENTS

ABSTRACT

Multi Agent Persistent Task Performance

James Motes
Department of Computer Engineering
Texas A&M University

Research Advisor: Dr. Nancy Amato
Department of Computer Science and Engineering
Texas A&M University

A method to control a system of robots to persistently perform a task while operating under a constraint such as battery life is presented. Persistently performing a task is defined as continuously exectuing the task without a break or stopping due to low battery constraints or lack of capabilites of a particular agent. If an agent is no longer able to execute the task it must be replaced by one that can continue the execution of the task. This is achieved through the utilization of two distinctions of agent roles: workers and helpers. This method is focused on addressing problems that require task handoffs where a second robot physically replaces a robot that has run low on battery. The worker agents are assigned the tasks, and perform the tasks until the constraint prevents further performance. Once a worker agent has reached a low battery threshold a task handoff is performed with a helper agent. This method utilizes a proactive approach in performing these handoffs by predicting the time and place that a worker will reach a low battery threshold and need to perform a handoff. This decreases the time necessary to respond to a low battery in these problems compared to prior developed reactive methods. As a result the total time needed

by the multi agent team to complete a set of tasks is decreased. In this paper, the method is demonstrated utilizing a physics based simulator to model the behavior of the multi agent team. Experiments are run over three standard problems requiring agent task handoffs: sentry, inspection, and coverage. These demonstrate the effectiveness of the method when compared against the existing reactive methods.

ACKNOWLEDGMENTS

I would like to acknowledge the support of the other members of the Parasol Lab at Texas A&M University. In particular, I would like to thank Will Adams and Read Sandstrom for their efforts.

1. INTRODUCTION

A common problem in robotics is the limitation of battery life. This complicates the completion of a variety of tasks for robots. The work a robot can perform is limited by the battery life before the robot must return to a charging location or be refuled in some manner. This extends the amount of time needed for a robot or a team of robots to complete a task or set of tasks. The method presented in this paper aims to reduce the amount of time a multi agent team of robots takes to complete a set of tasks.

The method utilizes a team of robots and solves problems that require robot handoffs to maintain persistent task performance. Persistent task performance is where the task must be performed continuously without stopping, and robots that are no longer capable of performing the task must be replaced by another that can continue the task. A handoff is where the robot performing the tasks is replaced at its physical location with another robot that will take over its tasks so that the original robot can return to a charger. Handoffs can be necessary to compensate for the constraint of limited battery as well as when the capabilities of different robots are needed to complete various stages of a task in which case one robot must handoff the task to another that is able to perform the next stage. Some of these problems that exist in robotics are the sentry, inspection, and coverage problems. In each of these problems there are cases in which it is necessary for one robot to physically replace another in the execution of the tasks. For example, the sentry problem is representative of museum guards patrolling an area. In this scenario it is criticial to the completion of the task that there is persistent performance of the patrol, and that the robots perform a physical swap so that there is no gap in the patrol and thus the security of the museum.

An inspection problem can take the form of video coverage, and a team of robots is assigned to provide a video tour or inspection of an area with a set of points to check.

Once battery constraint is taken into consideration multiple agents will be needed to complete the inspection in any case requireing more than one battery life cycle. For the video footage to be uniterrupted a handoff is required from the robot team.

A coverage problem consists of covering the entire environment. This can require a handoff in the case of searching for mobile query. Once a robot leaves the location it was performing the task, the ensurance of the area already covered is lost as the target query can move back to an area already accounted for in the coverage. A handoff ensures that the coverage performed already is not lost when a robot needs to recharge.

The method presented relies on agent roles to perform handoffs and maintain the necessary persistent task performance needed to solve these problems. It does this by extending prior developed methods that rely on reactive behavior by instead making the behavior proactive. This reduces the downtime that agents spend waiting for a handoff and increases the total amount of productive time across the system.

2. RELATED WORK

2.1 Motion Planning

2.1.1 Configuration Space

A critical part of motion planning is the concept of configuration space. Configuration space is used to determine where a robot can be within an environment without being in collision with any of the obstacles that make up the environment. The configuration of a robot is the exact specification of all the points of the robot system. The configuration space is all of the possible configurations of the robot system in the environment [1]. In configuration space the obstacles are expanded relative the size of the robots. This allows the robots to be represented as a single point in space which makes planning much simpler. This is in contrast to workspace where obstacles remain as they truly are and the robots are represented in their full size as seen in Figure 2.1.

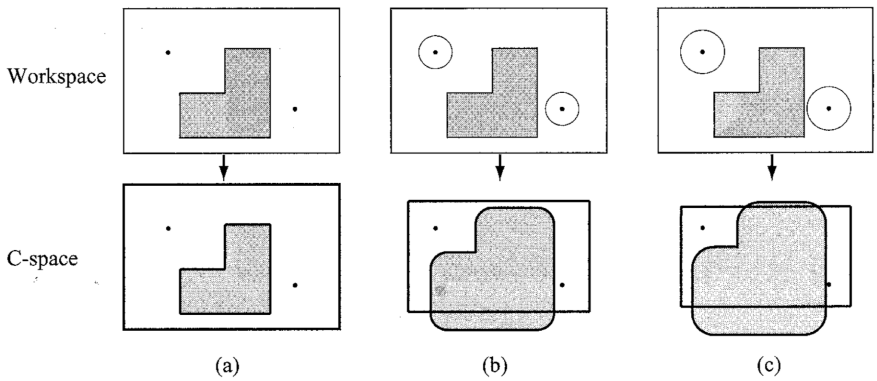

Figure 2.1: Example of Configuration Space compared to Work Space from [1].

2.1.2 Sampling Methods

Calculating an exact path through configuration space is very computationally costly. In order to lower the cost, sampling-based algorithms are used to build up the roadmap that the robots use to navigate the environment. The base method for randomly building a roadmap in an environment is the Probabilistic RoadMap planner. This planner randomly samples the environment to find configurations in free space and attempts to connect them. By randomly sampling in this way, a navigable roadmap of the environment is generated a lot more efficiently, and agents can connect start and end configurations to this roadmap to find paths from one point to another [2].

The path planning utilized to demonstrate this method is the Lazy PRM. Lazy PRM is a method of querying the roadmap to find a path from a start configuration to a goal configuration. It builds this path through configuration space and validates the path each time it is generated. This is opposed to other query methods that validate configurations and edges between configurations while building the roadmap. It is faster than other planners because it does not compute the validity of an edge unless it intends on using that edge in its path [3]. Lazy PRM also allows the planner to account for the adjusted positions of the other robots in a multi agent team.

2.1.3 Holonomic vs. Nonholonomic

An important distinction in robotics is holonomic vs. nonholonomic. Holonomic robots are completely controllable across all their degrees of freedom. Degrees of freedom paramterize a robot's position and orientation. Nonholomic robots are ones in which not all degrees of freedom are controllable. For example, a car cannot move to the side without first turning and orienting itself in that direction. Holonomic planning depends only upon the coordinates and time while nonholonomic planning takes into account the physics of the model. The method here was developed using nonholonomic planning. In

order to demonstrate this, the experiments were run on a physics based simulator. This accounts for the velocity as well as the position of the robot along the path.

2.2 Battery Constraint

There have been methods developed to address the issue of a constraint placed on the agent performing the task that focus on a single robot. These methods develop a coverage plan for a single agent based on pre-calculated limits. One of these methods [4] relies on a Battery-Constrained Sweep which produces a path that obtains coverage by returning back to a refueling/recharging point whenever necessary. This is intended to provide the minimum distance path to obtain coverage given the constraints of a limited battery or fuel. The biggest difference between this and the approach presented here is the use of a single robot instead of a system to allow for continuous coverage. This allows the system to work around the constraint instead of inside of it.

There are also approaches that utilize a team of robots to complete the set of tasks while under constraint such as [5]. This method develops a cost function that determines the amount of work that robot can continue to do before being forced to return to a charging location. This method is intended for problems that do not require handoffs, and the robots in the system can leave their tasks to return to a charging location whenever necessary.

A different approach to using a system or team of robots to address the issue of constraints while trying to achieve coverage or performing a task is one that has the team adjust its coverage plan as robots reach a low resources point and need to return to a refueling/recharging point [6]. This method has the various agents aware of their need to eventually aquire more resources and plan in such a way that as this need becomes more pressing, the agents position themselves closer to a refueling/recharging point. Additionally, the system of agents adjusts to the need of an individual agent to drop out to aquire

more resources so that as a system they continue to efficiently achieve the coverage desired [6]. This differs from the approach presented here in that agents adjust their plans to another agent's need to refuel/recharge rather than having a fully charged/fueled agent come replace the low resources agent in the system.

There have also been methods developed to decide when a robot needs to recharge while it is executing a task. These rely on thresholds based on different parameters to find the optimal point to return to a charging location. We integrate the idea of a threshold into our method by using the threshold to determine when a robot should call for assistance in completing its task, and thus handing off the task before returning to recharge. This is a static threshold with the level being such that a robot can return to a charging location from any point in the environment upon reaching the threshold. An adaptive threshold such as the one presented in [7] is likely to further reduce the amount of time the robot system spends dealing with battery constraints rather than performing the assigned tasks. This is outside the scope of this method and is included in the future work.

2.3 Agent Roles

A solution to the handoff problem is presented in [8]. This method assigns the roles of worker and helper to the agents in the group. Worker agents are agents that are actively performing the tasks assigned to the group. These agents work on completing the tasks until they reach a low battery threshold at which point they call for help. Helper agents are agents that remain on standby to replace worker agents when they reach a low battery threshold. When a worker agent places a call for help, a helper agent is assigned to the worker agent, and plans a path to the low battery worker agent.

Once the helper agent reaches the worker agent, the helper agent executes a handoff and takes over the task the worker agent was previously performing before reaching the low battery threshold. The helper agent takes on the worker agent role at this point, and the

original worker agent switches roles to a helper agent, and plans a path back to a charging location. Once reaching the charging location, the new helper agent replenishes its battery and is on standby for a worker agent to call for help. This method allows for persistent task performance while accounting for the battery constraint. The limitation of the system is its reactive nature. The method presented in this paper is an extension of the method in [8] that converts this reactive behavior into proactive behavior that reduces the downtime of the agents.

3. METHOD DESCRIPTION

3.1 Battery Breaks

The objective of the method presented in this paper is to extend the method presented in [8] to reduce the downtime agents spend waiting to switch roles due to the reactive nature of the method. In order to develop proactive swapping of agent roles, the system first needs to know when and where a swap will need to occur. This is done through the calculation of battery breaks along worker agent paths.

A battery break is the time and place along a path at which an agent will reach a low battery threshold and will need to return to a charging location. This is done by stepping through the local plans between configurations in the path to find the point at which the low battery threshold is reached. This time and configuration in the environment is saved as a battery break and communicated to the system.

Algorithm 1 Algorithm for Calculating a Battery Break.

```
1: function FINDBATTERYBREAK(worker w, timestep t, depletionRate d, currentTime
   c)
2:     l ← w.batteryLevel
3:     for all step ∈ w.path do                ▷ each step is a configuration for the robot
4:         l ← l − t ∗ d
5:         c ← c + t
6:         if l <= w.threshold then
7:             b ← BatteryBreak(step, c)
8:             return b
9:     return null
```

The system stores all of the battery breaks calculated by worker agents currently performing tasks. The system then assigns these battery breaks to the helper agents as they

become available throughout the system. Initially of course all helpers are available, but as the scenario progresses some helpers may be returning to a charging location or replenishing their battery levels to a point at which they are able to perform a handoff and continue working. Each availble worker continuously queries the system for unassigned battery breaks to initiate the proactive exchange.

3.2 Proactive Swapping

In the reactive method of [8], helper agents waited until a worker agent places call for help before being assigned. In this method, once a helper agent is assigned to a battery break it plans a path to the configuration of the battery break. The helper agent then calculates the time the path will take and subtracts that from the expected time of the battery break to determine its departure time.

Algorithm 2 Algorithm for Proactive Exchange

```
1: function ProactiveExchange(Helper h, BatteryBreaks B, Worker w, ChargingLocation l)
2:     if B.empty() then
3:         return
4:     b ← B.front()
5:     h.path ← GeneratePath(h, b.configuration)
6:     waitTime ← h.path.time
7:     while waitTime > currentTime do
8:         PauseAgent(h)
9:     FollowPath(h)
10:    h.task ← w.task
11:    w.path ← GeneratePath(w, l)
```

This allows the helper agent to immediately resume the task that the worker agent had been performing before reaching the low battery threshold whereas in the prior method the task would effectively be put on pause while the worker agent waited for the helper

agent to come and switch roles to take over the task. Now the helper agent should arrive at the same time that the worker agent reaches the low battery threshold and immediately continue the task significantly reducing the amount of downtime across the system.

The assignment of battery breaks to helper agents follows a chronological priority system. The first battery break in the system is assigned to the first available helper agent. This is a simplistic approach, and there are several cases in which this could lead to inefficiencies in the assignment of helper agents, but not enough to significantly detract from the reduction in downtime across the system from the prior method.

3.3 Corrective Behavior

As a result of the dynamic nature of the robots, sometimes the battery levels of a worker agent reaches the threshold before the worker agent has reached the calculated battery break. This can be a result of any unplanned step or action the robot had to make along its path. When this scenario occurs the helper agent is at the predicted battery break and not in handoff proximity when the worker agent reaches the low battery threshold.

To compensate for the gap, the helper agent responds to the workers call for help as it would in a reactive system. Unless the amount of unplanned battery expenditure is large, the distance between the predicted and actual battery break is minimal, so the resulting time the helper agent has to cover is as well since it is already close. This allows for persistent task performance to continue in the event of the actual battery break occurring outside handoff proximity of the predicted battery break rather than leaving the worker agent stranded.

3.4 Planning Primatives

3.4.1 Planning

All of the plans generated in the method utilize Lazy PRM. This validates only the edges along the path, which reduces the amount of time spent planning. It also helps

implement the collision avoidance method utilized in the system.

3.4.2 Collision Avoidance

An issue that often occurred in the experiments was collision avoidance. When a worker agent is operating towards the edges of the environment, it is likely that a switch out there results in both agents needing to move back towards the rest of the environment. To account for this, a priority based system was utilized that based priority on the current role of the agent. When a potential collision arose, the lower priority agent would halt, while the higher priority agent would generate a new plan with Lazy PRM to avoid the halted agent agent and continue towards its goal. When the higher priority agent moved outside of the collision proximity, the lower priority agent would then plan a new path to its goal as well and resume progress. In the case of more than two agents being within collision proximity, the lower priority agents remain halted until the highest clears the group, and then the next highest plans and leaves, until only one agent remains and plans its own path.

4. EXPERIMENTS AND RESULTS

4.1 Experiments

4.1.1 Problems Tested

There are three problems that the method was tested on that require agent handoffs and persistent task performance. The first of these is the sentry problem. In this problem the objective is to patrol a hallway continuously. Two points at either end of the hallway are designated as goals, and the task is to go back and forth between them. The patrol duratino is long enough for each agent to be replaced once in the performance of the task.

The second problem is the inspection problem. In this problem several points next to obstacles in the environment are set as goals, and the task of the multi agent team is to go inspect each of these obstacles. The set of inspection points is again long enough for each agent to be replaced in performing the task.

The third problem is the coverage problem. In this problem, the task is to obtain coverage of the environment. Goals are set such that the multi agent team covers the extent of the environment. This requires multiple role switches for the team to complete the task.

4.1.2 System Used

All methods were implemented in a C++ motion planning library at Parasol Lab at Texas A&M University. It used a distributed graph data structure from the Standard Template Adaptive Parallel Library (STAPL) [9], a C++ library for parallel computing developed at Parasol Lab.

All experiments were performed on a desktop, at Parasol Lab, running CentOS 7 with Intel® Core™ i7-3770 at 3.40 GHz, 16 GB of RAM, and the GNU gcc compiler version

4.8.5.

4.2 Results

4.2.1 Sentry Problem

In the sentry problem, when comparing the proactive behavior to the prior developed reactive behavior it is easy to see in Figure 4.1 the improvement made in the amount of down time the system experiences while the worker agent waits for the helper agent to come perform the task handoff. A majority of the handoffs in the proactive behavior implenetation are effectively instantaneous while the shortest wait for the reactive behavior was still over 8 seconds. On average, the down time in the sentry problem accounted for 1.76% of the total running compared to the 10.1% of the total running time in the reactive implementation. In the proactive implementation, the productive time accounted for 85.7% of the running time, and the reactive implementation had an average productive time accounting for 76.1% of the total running time. The total running time of the proactive implementation was 5.02% faster than the reactive implementation.

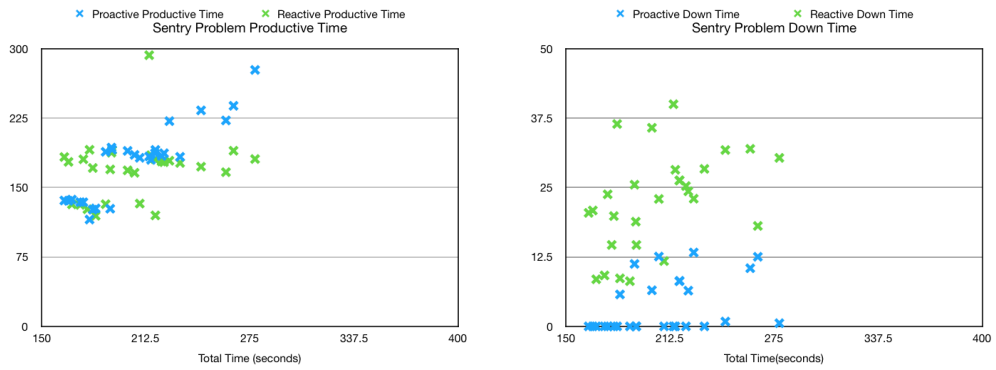

Figure 4.1: Results of Sentry Problem.

4.2.2 *Inspection Problem*

As can be seen in the results for the inspection problem in Figure 4.2, the down time is not as minimal as it is the sentry problem, however, it still only accounts for 4.29% of the total time in the proactive behavior implementation compared to the 6.65% of the total time that the down time accounts for in the reactive behavior implementation. The productive time in the proactive behavior implementation accounts for 45.6% of the total time, and the reactive behavior implentation production time accounts for 33.0% of the total time.

The rest of the time is taken by handoffs and planning. The nature of the inspection problem leads to extra collision avoidance being necessary as the goals take the agents into harder to reach areas of the environment with a higher frequency which leads to more complex handoffs where the agents may have to construct several plans to avoid each other immediately before and after a handoff while moving out of the more crowded areas of the environment. The total time spent on the problem by the proactive behavior takes on average 31.8% less time than the reactive behavior implementation.

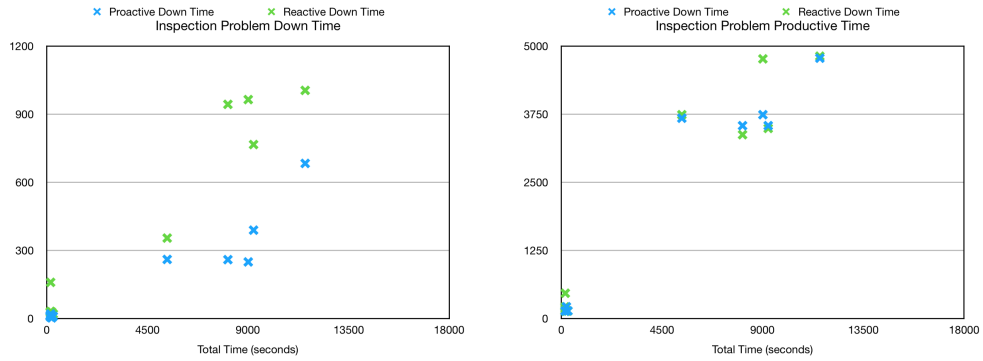

Figure 4.2: Results of Inspection Problem.

4.2.3 Coverage Problem

Again as can be seen in Figure 4.3, a majority of the handoffs in the proactive behavior implenetation are effectively instantaneous while the shortest wait for the reactive behavior was still over 8 seconds. On average, the down time in the coverage problem accounted for 2.08% of the total running compared to the 15.2% of the total running time in the reactive implementation. In the proactive implementation, the productive time accounted for 88.0% of the running time, and the reactive implementation had an average productive time accounting for 72.4% of the total running time. The total running time of the proactive implementation was 11.3% faster than the reactive implementation.

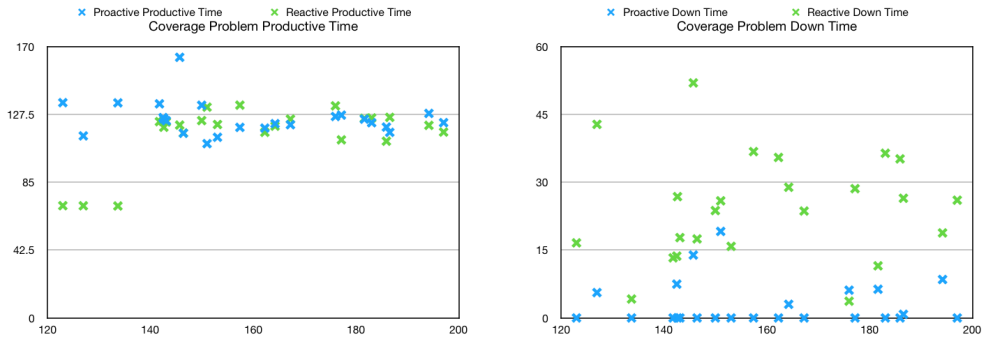

Figure 4.3: Results of Coverage Problem.

4.3 Observations

The method calculates a battery break whenever a worker agent plans a path from the task or goal it just completed to its next destination. If the worker agent can complete this path without reaching a battery break then it does not store a battery break prediction in the system for a helper agent to respond to. This leads to scenarios occurring where

the worker agent completes a goal/task with a battery level slightly above the threshold. When it generates its next plan the battery break is almost immediate, and the reduction in time spent waiting by proactively dispatching the helper agent to perform the handoff is negated.

This can be compensated by setting a battery buffer such that if a worker agent is going to complete a goal/task with its remaining functional battery below the buffer, a battery break is recorded in the system at the time and place of the completion of the goal/task. This allows the helper agent to reach the worker agent when it reaches the low battery threshold rather than effectively resorting to reactive behavior. The nonholonomic nature of the robots this method was tested with often results in the progress made by the worker agents with the small amount of battery remaining consisting of primarily orienting themselves for the next path which the helper agent will have to perform for itself after switching roles. If the worker agent does manage to make enough progress to leave handoff proximity of the battery break, then the corrective behavior will compensate and the time spent waiting to switch is still reduced.

5. CONCLUSION

The proactive nature of the method presented here allows the helper agents to take over the tasks of the worker agents in significantly less time. As a result the total task time is decreased, and a greater precentage of the time is spent performing the task. The percentage of time spent on down time is decreased by up to 13% and the total running time is decreased by up to 33%.

5.1 Discussion

The extra cost of calculating the battery break and the departure time is the tradeoff for the reduced downtime the system has waiting for agents to switch roles. This, however, can be hidden if the calculation to find the battery break and the departure time are done in parallel to the movement of the agents. When the calculations are performed this way, the effect on the total solution is negligible.

The experiments were run using a pair of agents to focus on the reduction of downtime during an agent handoff. When the number of agents is increased the ration of worker agents to helper agents will affect the effectiveness of the proactive approach. If the number of worker agents becomes significantly larger than the number of helper agents then the worker agents will end up being forced to wait for helpers to return and charge before being dispatched to execute a handoff. This will result in similar performance to the reactive behavior as the proactive nature is effectively negated.

The location of charging locations can also have a significant effect on the impact of the proactive behavior. An ideal placement of charging locations will decrease the time for a helper to reach a battery break regardless of proactive or reactive behavior, however, in a small environment the time to reach a battery break can be reduced to a point where the leaving early only saves a small amount of time. This effect is reduced in large

environments and the impact of the proactive behavior again becomes more prevalent.

6. FUTURE WORK

Further work to seek to reduce downtime across the system is finding ideal switching points that may occur before the battery breaks along a worker path as well as finding more ideal assignments of switching points to helper agents. Currently the method sends a helper agent to meet a worker agent at the point where the worker agents reaches its low battery threshold, but there are cases where it would be more efficient to switch roles before the worker agent reaches this point.

This could be done by generating the cost of each potential switching point, and determining the total lowest combination of switching costs across the system. This cost function would include the cost of sending helpers from various locations, so that it would also address the issue of assigning helper agents to more ideal switching points rather than simply the first available.

This cost function could be parameterized in various ways to allow for the focus of the function to shift. Reduction in downtime, smallest total time spent achieving tasks, less time with agents with full batteries sitting at charging locations are just a few possible focuses of the cost function.

Another method that integrating is likely to improve the efficiency of the system is a dynamic battery threshold like the one in [10]. This potentially allows the worker agents to perform the task a lot longer as opposed to setting the static battery threshold that is dependent upon the furthest location from a charging location in the environment. Taking this into account in the planning of the task assignment and the switch cost function will allow the system to further refine the switching points of workers and helpers.

The collision avoidance method utilized here can impact the accuracy of the battery breaks if the energy cost of waiting is high. To improve this, a method of utilizing dynamic

obstacle planning [11] can be utilized to generate plans that account for other agents in the proximity. Instead of waiting for all higher priority agents to clear the collision proximity, once an agent is the highest priority agent in the collision proximity without a plan, it plans its path while treating the moving agents around it as dynamic obstacles with known paths, as the system knows the when and where those agents will be at each time step. This enables agents to navigate around each other without losing time waiting on other agents to clear the collision proximity.

The method is currently tailored towards performing handoffs once a worker reaches a low battery threshold, however, it the same method solves the problem of performing a handoff when a task needs to be passed to an agent with different capabilites. Testing this task handoff and improvement the proactive method makes in the performance of the task is additional future work.

REFERENCES

[1] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*.

[2] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. on Robotics and Automation*, pp. 566–580, 1996.

[3] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," *IEEE International Conference on Robotics and Automation*, 2015.

[4] G. P. Strimel and M. M. Veloso, "Coverage planning with finite resources," *2014 IEEE International Conference on Intelligent Robots and Systems*, pp. 2950–2956, 2014.

[5] D. Mitchell, M. Corah, N. Chakraborty, K. Sycara, and N. Michael, "Multi-robot long-term persistent coverage with fuel constrained robots," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 1093–1099, IEEE, 2015.

[6] J. Derenick, N. Michael, and V. Kumar, "Energy-aware coverage control with docking for robot teams," *2011 IEEE International Conference on Intelligent Robots and Sytems*, pp. 3667–3672, 2011.

[7] J. Wawerla and R. Vaughan, "Near-optimal mobile robot recharging with the rate-maximizing forager," *Advances in Artificial Life, ser. Lecture Notes in Computer Science, F. Almeida e Costa, L. Rocha, E. Costa, I. Harvey, and A. Coutinho, Eds. Springer Berlin Heidel- berg*, vol. 4648, pp. 776–785, 2007.

[8] S. Mishra, S. Rodriguez, M. Morales, and N. M. Amato, "Battery-constrained coverage," *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 695–700, 2016.

[9] P. An, A. Jula, S. Rus, S. Saunders, T. Smith, G. Tanase, N. Thomas, N. Amato, and L. Rauchwerger, "STAPL: A standard template adaptive parallel C++ library," Jul 2001.

[10] T. Wang, B. Wang, H. Wei, Y. Cao, M. Wang, and Z. Shao, "Staying-alive and energy-efficient path planning for mobile robots," *American Control Conference*, 2008.

[11] M. Philips and M. Likhachev, "Sipp: Safety interval path planning for dynamic environments," *2011 IEEE International Conference on Robotics and Automation*, pp. 5628–5635, 2011.