Carnegie Mellon

Market-Based Coordination of Recharging Robots

Senior Honors Thesis

Victor Marmol
School of Computer Science, Carnegie Mellon University
vmarmol@andrew.cmu.edu

Advisor:
M. Bernardine Dias, Ph.D.
Robotics Institute, Carnegie Mellon University
mbdias@ri.cmu.edu

Mentor:
Balajee Kannan, Ph.D.
Robotics Institute, Carnegie Mellon University
bkannan@ri.cmu.edu

Carnegie Mellon
SCHOOL OF COMPUTER SCIENCE

THE ROBOTICS INSTITUTE

Abstract

Autonomous recharging is becoming increasingly important to mobile robotics as it has the potential to greatly enhance the operational time and capability of robots. Existing approaches, however, are greedy in nature and have little to no coordination between robots. This leads to less efficient interactions which adversely affect the performance of the team of robots. Therefore, improved coordination can greatly enhance the performance of such a team. This senior thesis has advanced the state of the art in autonomous recharging by developing, implementing, testing, and evaluating a market-based distributed algorithm for effectively coordinating recharging robots. This system is charge-aware and able to autonomously account for current and future battery states as well as current and future tasks. The developed solution has been evaluated on a series of tasks run on worker robots and recharger robots. Simulations were also used to validate the system on larger workloads. Results show that our approach consistently outperforms the state of the art in recharging strategies.

Acknowledgements

I would like to greatly thank my advisor Dr. M. Bernardine Dias for her continued challenges, help, support, assistance, and guidance; without her this thesis would be non-existent and unfocused. I would like to thank her for taking a chance on me when I asked to join her group four years ago as an inexperienced and untested freshman. I would also like to thank my mentor Dr. Balajee Kannan for his patience, assistance, and guidance every step of the way.

I would like to thank my mom and dad for their never-ending hard work, love, support, and dedication; without which I would not be where I am today. All my brothers and my sister for the encouragement and help they have given me through the years. I would also like to thank Samantha Tan for her support and for her patience during my long nights working with the robots.

I would also like to thank Jimmy Bourne, M. Freddie Dias, Nisarg Kothari, and Sairam Yamanoor for their work on the robot hardware and software. I want to thank everyone in the rCommerce Group for developing and maintaining the current system and robots.

Table of Contents

1 | Introduction

The effectiveness of mobile robots has always been affected by the amount of time they can spend in the field and thus how much work they can perform. Inherently, mobile robots can perform a finite amount of work in a single work cycle due to the finite amount of energy contained in their batteries. This makes each action valuable and leads to a heavy emphasis placed on planning the robot's movements, actions, and tasks. The introduction of autonomous recharging brings another dimension to planning. It creates the need for each robot to plan not only their actions but also how these correlate to their battery life and when recharging needs to occur. Autonomous recharging also brings with it the promise of extended runtime. Robots will be able to run continuously for very long periods of time with little to no human interventions. This makes the currently unthinkable runtime of weeks, months, and even years possible. This greatly increases the number of domains in which mobile robots can be effective.

The current generation of mobile robots is equipped with rechargeable batteries which allow the robots to be recharged in the field and thus provide some flexibility for the robot's work cycles. Researchers have also started to incorporate mobile recharging agents into autonomous recharging. These mobile recharging agents are mobile robots with recharging capabilities. These robots are able to provide a non-stationary recharging unit to service worker robots. Both of these features allow flexibility when it comes to where and when to recharge. Unfortunately, with this flexibility comes an even greater need for planning in the part of each robot. It is of importance that this planning occurs in order to increase the total work performed by any given robot. A lack of planning or insufficient planning in the part of each robot can lead to reduced efficiency and work. This comes about due to inadequate choices for recharging time and place.

The motivation of this research stems from the importance of battery life in the amount of work performed by a mobile robot. In the history of mobile robots, the amount of energy used and stored by the robots has been central to how the field has developed. From the first mobile robots that required an external power source, to the modern mobile robots that run on batteries or internal generators; all mobile robots have placed an emphasis on low power consumption and high power storage or generation. With this emphasis on power, it is surprising to find how little planning and coordination is based on power. This motivates us to create a system where all planning is done around power and its availability. Recharging becomes a real part of a robots work cycles.

Another consideration is the increasing trend of multi-robot approaches. It is more and more frequent to find tasks being tackled by groups of mobile robots rather than a single mobile robot. In contrast with single mobile robot systems, these modern systems place an even greater

emphasis on coordination and planning due to the real danger of overlapping work as the number of robots increases. It is very important to create intelligent systems that scale well as we increase the number of robots performing tasks. These robots must coordinate with each other in order to effectively complete their goals and do so in a timely manner.

Autonomous recharging has the potential to greatly increase a robot's effectiveness and to do so with little to no human intervention. This will allow us to build larger and more complex systems incorporating more robots. It is the importance of robotics in the future and how key to that future autonomous recharging is that motivates us to design and build the system described here.

1.1 | Shortcomings of Current Approaches

We now summarize the current approaches to autonomous recharging. These are discussed in greater detail in a later section. Current approaches to autonomous recharging are mostly greedy in nature. Each robot makes recharging decisions based on their current state with little to no regard for past and future state. This type of planning leads to sub-optimal performance and unaccounted edge cases. Most of the time a robot's future tasks are known to at least some extent. This provides an opportunity for better planning and better coordination for each robot.

Existing approaches have little to no coordination for shared recharging resources. In most systems the worker robots will share a static or mobile recharging station. This sharing necessitates coordination between robots in order to avoid deadlock or the loss of a robot due to falling battery levels. The current state of the art systems specify little coordination between robots and the systems that do specify coordination are based on conflict avoidance. These conflict avoiding strategies can lead to cases where robots go uncharged. These strategies also do not scale as the number of robots sharing the resource grows.

1.2 | Our Contribution

This thesis has advanced the state of the art in autonomous recharging by designing, implementing, evaluating, and verifying a charge-aware system wherein robots plan their tasks around their limited supply of power and recharge when necessary. This system incorporates static and mobile recharging stations and provides the necessary coordination in order for multiple worker robots to share a single mobile recharging station.

The primary contribution to the state of the art is the planning done by our charge-aware system. Unlike existing methods, our approach utilizes past, present, and future known

conditions and plans accordingly. Our planner is also optimal in terms of total distance traveled. This planning and coordination allow our system to better scale with the number of tasks as well as with the number of robots. Both of which are contributions to the current state of the art.

Another contribution from our system is the use of a market-based strategy for coordinating recharging. Our approach is the first to utilize a market-based system to schedule worker's tasks and to coordinate a mobile recharger's tasks. This is important due to the many advantages of market-based systems.

1.3 | Market-Based Systems

Market-based approaches use a simulated economy where robots buy and sell tasks according to their estimated cost for completion. In this way, the robots are able to coordinate amongst themselves to allocate the necessary tasks according to the best information they have at the time. The estimated costs of tasks are based on a set pre-defined cost functions shared amongst the agents in the market-based system. These cost functions are used to abstract away the details of a task into common currency which all agents understand. This allows for a system where robots attempt to minimize their individual costs and the cost of the group as a whole which in turn maximizes the work performed by the team of robots.

We chose to use a market-based system due to their maturity and advantages as well as their capability for customization.

Market-based approaches have risen in popularity [14] over the last few years for tackling the problem of multi-robot coordination. Four features that propel the popularity of market-based approaches are their simplicity, their distributed nature, their fault tolerance, and their scalability. Market-based systems are generally considered simple since they only require the definition of cost functions and schedulers in order to completely describe and evaluate any task that the robots will need to perform. The system is distributed since each agent can hold its own auctions and bid on others agent's auctions. There is no central authority where all bids go through, rather the agent holding the auction determined the reserve price and chooses the winner. The system is fault-tolerant since in any given auction if any of the bidding agents fault, the auction can still finish since not all agents need bid. If the auctioning agent is the one that faults then the auction is cancelled and no winner is determined. Market-based systems are inherently scalable due to their abstracted market nature. As the number of robots scales the costs of performing the tasks scale accordingly and the system re-calibrates itself with the new costs. When a new robot is added to a system it can share the cost of existing work by buying

tasks from other robots and thus directly increasing the amount of work done by the group. All this is completed with no changes to the infrastructure of the underlying system.

Market-based systems are highly customizable. Since their building blocks are abstract in nature, a market-based system can be defined around any set of tasks that a group of robots needs to complete. This allows us to easily define the work and recharging tasks necessary for our research.

Finally, we chose a market-based system due to our group's familiarity with such systems. This familiarity allows us to build a better autonomous recharging system by adapting our current system to make it charge-aware.

1.4 | Thesis Outline

The thesis is organized as follows. Section 2 describes the background of our research and the existing strategies for autonomous recharging. This includes a literature review of current algorithms and approaches as well as some information about the system utilized in this research. Section 3 outlines our approach and presents the details of our novel charge-aware system, including the inclusion of mobile recharging agents and the necessary coordination for sharing these mobile recharging agents. In Section 4 we will describe our testing methodology and metrics alongside the results of the experiments we conducted. We then discuss and analyze the work and our results in Section 5. Finally, we conclude with Section 6 which provides our conclusions and a description of future work based on what is outlined here.

2 | Related Work

This section describes existing work in autonomous recharging by examining various approaches along with their limitations. The topic of autonomous recharging has been gaining traction in the literature recently. The existing research on autonomous recharging has focused on four main areas: how to recharge, where to place a recharging station, when the robots should recharge, and exploring the use of mobile rechargers. We will also describe the base case for autonomous recharging: no autonomous recharging at all.

2.1 | No Recharging

Most research mobile robotics systems today give the robots very little to no understanding of its charge state. Rather, they give this information to humans for them to parse and understand. They then interrupt the robot's workflow to introduce the necessary recharging. This is something the robot does not plan for and cannot account for in its planning. This approach is easy to use and is so overwhelmingly chosen due to that ease and the relatively small number of robots in the teams. As the number of robots in the teams increase, it might become necessary to use another strategy as this one becomes cumbersome and unscalable; requiring near constant attention from human operators. Choosing not to implement autonomous recharging, makes it difficult to work with groups of mobile robots and becomes intractable as the number of robots increases.

2.2 | Recharging Hardware

A lot of research has focused on the hardware necessary for static recharging with various different systems having been designed and implemented with that goal. Oh et al. described a system which utilizes a docking-based static recharging station that can be located via long-range infra-red beacons. The robot's short-range maneuvering is done by a grid of specially-aligned laser-visible targets placed on the recharging station [1]. Silverman et al. presents another static docking system, but instead focuses on the short-range docking navigation. The system uses a combination of laser targets and vision targets to hone in and maneuver towards the docking station. Their docking system is also effective at docking from large displacements thus allowing more error in their short-range navigation [2]. Muñoz et al. have developed a very robust recharging dock which uses parallel horizontal plates to make contact with the robot's recharging horizontal plate. The system uses vision and odometry to find and dock with the static recharging station [12].

2.3 | Location of Recharging Stations

Some researchers are interested in where recharging occurs. This is especially important once mobile rechargers are incorporated into a system. The location of such a station can have a great impact on the total work a group of robots can perform. Couture-Beil and Vaughan have considered the location of a recharging station. They have found that when the station is too close to areas of high utilization, the work done by the robot drops as the recharging station serves as a roadblock. It was found that the best location for a recharging station was some distance from the area of highest utilization [9]. This research, however, only considered a continuous task between two locations and did not examine in detail how queuing at the recharging station affected the effectiveness of the group of robots.

2.4 | Condition for Recharging

The next question is when recharging should occur for each robot. The overwhelming answer thus far has relied on a particular threshold, most choices being distance or time. While a simple strategy, these approaches have many edge cases that can lead to unintended behavior.

The simplest approach is recharging when a particular batter threshold is reached [2], [8], [12], [13]. Since this approach is not based on the robot's location, task load, or the location of the recharging station it is very likely that the robot might not actually make it to the recharging station in time. The chosen threshold is critical to the effectiveness of this strategy, particularly when the recharging station is far from the work area. Small thresholds lead to possible loss of battery power before the robot reaches the recharging station. Large thresholds lead to inefficiency in the system as the robots have enough battery to do more work but are asked to recharge prematurely.

Another threshold-based approach uses time as a threshold. Austin et al. have developed such a system. Their robots work for a specified amount of time and then proceed to recharge. This approach produces less edge cases then a battery threshold approach given the right choice of threshold [5]. There is also an inherent difficulty in choosing a threshold. The same problems we saw when choosing a battery level threshold arise here.

A third type of the threshold-based approach opts to recharge when the robot has just enough battery to reach the home recharging station [4], [7]. This approach is more robust that other approaches, but poses a problem if the estimation is done poorly. It is critical for the system to properly determine that the robot has enough battery to reach the recharging station.

Most of the existing works on autonomous recharging deals with a single worker robot. The existing work that deals with multiple workers attempting to recharge simultaneously addresses the coordination issue through conflict resolution rather than coordination between the robots [12]. This greedy approach can lead to less desirable situations, where a group of workers queue up around a charge station waiting for their turn. Under certain conditions this queuing can lead to two adverse outcomes: First, the workers exhaust their batteries while they are waiting to recharge, second when the charge station is completely blocked, effectively rendering it useless.

2.5 | Mobile Rechargers

Research has also focused on mobile rechargers. Mobile recharge stations have increasingly become popular as they offer a lot of flexibility and promise for on-the-go charging for a deployed team of robots. These mobile charging stations also pose a series of challenges, as their location and limited recharging capabilities have to be considered when coordinating the team.

Some researchers have focused on developing the hardware for such systems. Kottas et al. have developed a mobile recharging station based on the Pioneer robotics platform. This mobile recharger is capable of recharging multiple small robots [3]. Our chosen approach has a mobile recharger charging much larger robots and is thus only capable of recharging one at a time. Zebrowski and Vaughan have chosen a similar route with a tanker robot being equipped with a gripper it uses to dock with other robots and power outlets for energy transfer [10]. [11].

Research has also been conducted on how these mobile rechargers interact with the robots they recharge. Litus et al. developed a system which utilizes a tanker recharging robot that rendezvous with other workers and recharges them. Their approach aims to find an optimal set of rendezvous points and orderings in order to maximize the recharger's effectiveness [6]. However, they found their approach to be at least as hard as the travelling salesman problem. Their approach also tries to minimize the travel amount of the recharger and to some extent the workers. This has the side effect of decreasing the amount of work the robots can perform. We believe that the recharger should take a support role as we wish to maximize the amount of work performed by the robots and, in turn, the group as a whole.

2.6 | Challenges & Limitations

Current research into autonomous recharging has laid the foundation necessary with the hardware and concepts necessary. However, current approaches to autonomous recharging have little coordination and do not make full use of the knowledge we may possess about the

past, current, and future state of the robots. Scalability was not a primary goal of current approaches and thus their performance suffers as the number of robots increases. There is currently little work that is able to coordinate robots with a recharging station. The approaches that do handle this coordination do so through conflict avoidance. We aim to advance the state of the art in autonomous recharging by addressing these challenges and limitations. The system we propose is able to schedule recharging tasks as a component of a robot's work cycle. This scheduling is done optimally by using all known information. Recharging stations are also scheduled so as to avoid deadlock and the loss of robots due to battery depletion.

3 | Approach

We now present a system where power is a central component of planning and worker robots incorporate recharging into their work cycle. Towards this goal we have developed a system where each worker robot is charge-aware and is able to plan its schedule of tasks according to which tasks it can and cannot complete given its charge. They also have the capability to insert recharging tasks into their schedules when necessary. We then incorporated mobile rechargers whose goal is to lower the cost of the worker robots' schedules in order to increase the amount of work the workers can perform. Our system uses two types of tasks: A point task which is an abstracted work task comprised of traveling to a specified coordinate. A recharging tasks is a task wherein the robot rechargers its batteries, whether that be via a static recharging station or a mobile recharger.

3.1 | Existing Infrastructure

The existing infrastructure used for this research is the system utilized by the rCommerce group at the Robotics Institute. The system is comprised of a series of robots, an operator interface, and the market-based system that ties them all together.

Figure 1 - 3 Pioneer P3DX robots

The rCommerce group has six Pioneer P3DX robots (Figure 1) with localization and mapping capabilities. Localization is done via wheel-mounted encoders and a centrally located gyroscope. Mapping and environment detection is done through a single laser range scanner

with 180° range of vision. The onboard computing is comprised of single-core 1.2*GHz* x86 processors, 2GB of RAM, and varying levels of solid state storage (Figure 2). All the robots run Ubuntu Linux and communicate via an 802.11n WiFi network. One of the robots is equipped with a recharging arm which it can use to dock with and recharge the other robots (Figure 3). This robot is our mobile recharger. Each robot runs a series of modules which allow it to sense its environment, plan its movements, and coordinate with other robots. The robots are able to autonomously travel to any point and do so while avoiding obstacles.

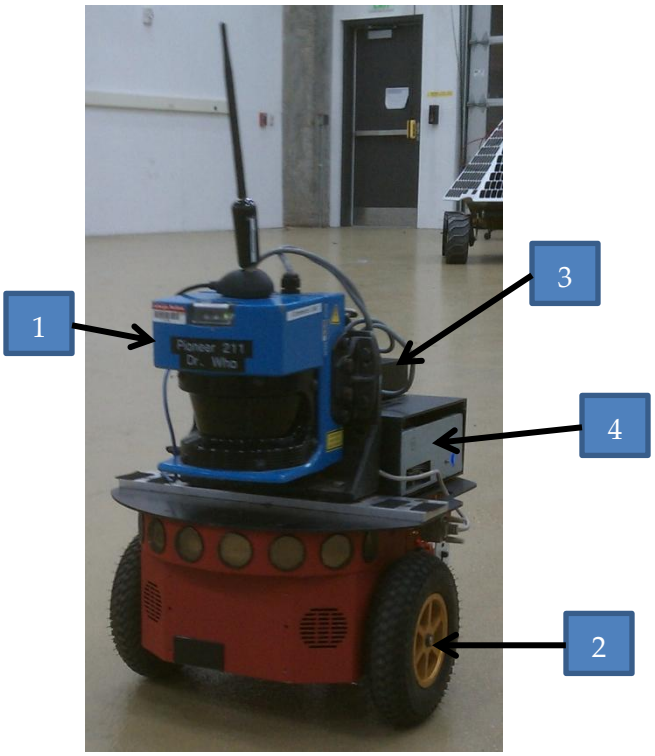

Figure 2 - Pioneer with (1) laser scanner, (2) wheel encoders, (3) gyroscope, and (4) computing.

Figure 3 - Pioneer with recharging arm docked with another Pioneer

The operator interface is comprised of a graphical user interface that merges the data provided by all the robots (Figure 4). It is characterized by a large map with aggregated map information from all robots. The map shows each robot's location, it's currently executing task, and a brief distance trail for the robot. Through the interface one can assign tasks to an individual robot or to a group of robots.

Figure 4 - Graphical user interface showing a robot's location, path, and a map of the known environment. The interface allows for direct control of the robots.

The market-based system utilized by the group is the TraderBots system first developed by Dias et al. [14]. Our current version of the TraderBots library is version 4 and it is developed by Carnegie Mellon's National Robotics Engineering Center [15]. This library implements a fully customizable distributed market-based system. Each robot has a *Task Allocator* module which instantiates the library and is tasked with holding that robot's auctions and bidding for tasks in that robot's name. As a member of the rCommerce group I have taken an active role in the development and advancement of this infrastructure for the past four years.

3.2 | Charge-Aware System

Our charge-aware system runs on each robot and proactively plans in what order the robot should execute its tasks and whether it should recharge at any time. These plans are made based on the robot's battery state and schedule of tasks. The schedule of tasks is comprised of both future and present tasks. In order to give each robot charge-awareness they must first be able to estimate their runtime, to know which tasks can be completed given their available power. Due to the market-based nature of our system the robots must also be able to determine the cost of individual tasks and to schedule accordingly. Finally, the robots must also know what to do when recharging is triggered; that is what actions they actually have to perform in order to recharge.

3.2.1 | Home Base

The robot's start location is an area we call their home base. It is the location where they are first deployed and the location they must return to once deployment is complete. It is in home base where we allow the robots a last-option backup plan for recharging. Since each robot has its own charger at its home base, the robot can always go recharge at home where it is guaranteed to have an available charger for its use. The choice to go home is not optimal for a robot as home can be far from the robot's work area and it is thus somewhat expensive to return home. It would be much lower in cost for the robot to be recharged by a mobile recharger rather than going all the way home. However, mobile rechargers might not always be available or the available ones might be busy recharging themselves or other worker and are unable to recharge the requesting robot. Due to this possibility, we allow home as the recharging backup alternative to assure that a robot's schedule of tasks can always be completed since home is guaranteed to be a viable recharging option.

3.2.2 | State Estimation

Accurately estimating a robot's runtime is important since it is this estimate that the robots use to plan their schedules and chose when to recharge. An overestimated runtime can lead to unexpected recharging where the robot might not have enough time to reach a static or a mobile recharging station. Underestimated runtime won't lead to a loss of a robot due to lack of battery power, but it will lead to reduced total work output since the robot will perform less work per recharging cycle than its batteries allow. The second case, however, is more permissible as it allows the completion and continuation of work by the robot.

In order to provide accurate runtime estimation we must be able to translate a robot's remaining battery power into how much work can be performed. We chose to translate the robot's remaining battery power into a distance to empty calculation. Similar to how modern cars will display how many miles the car can traverse before its tank of gasoline is depleted, we wanted our robots to display the number of meters the robot could traverse given its current battery power.

Towards this end we began by running a series of battery rundown tests. In these states we run the robot continuously until their battery is depleted. We found that the total distance traveled was consistent within $5m$ between runs on the same robot using the same battery pack. There were variations between different robots and between different battery packs on the same robot, but these are differences that can be discerned by knowing what battery pack the robot is using and training it on that pack. With this observation we decided to implement a system that started with this known runtime and could measure how much energy was used during the robot's run. Thus we would be able to have a good estimate of the robot's remaining runtime at any given point in time.

We started designing a system for state estimation by recognizing discrete power states in which the robot operated and proceeded to estimate the amount of power consumed in those states. There are two main states our robots exhibit: motion and idling. Motion is when the robots are actively moving towards a goal and its wheels are providing the propulsion. Idling is when the robot is not actively moving towards a goal but is still receiving sensor data from all sensors. We designed and implemented a model for each of these two states. The models are able to calculate how much runtime is consumed by the robot when it is in that particular state and has been in that state from time t_{start} to time t_{end}. The models take as input the start an end robot state and return how much power was used, in units of meters, during the time period it was in that state.

The motion state model is relatively simple since our runtime is expressed in meters. The model determines what distance the robot has traversed from t_{start} to t_{end} and provides that as the

runtime consumed. This estimation is accurate since it is exactly how many meters of battery life the robot will consume by traversing n meters.

The idle state model is a little more difficult since it requires translating idle time in seconds to a number of meters consumed. Our state model does this by estimating how many meters the robot *could* have traversed had it moved rather than stayed. In other words, if the robot were to move instead of idle for the specified time how much battery would it consume? We do this estimation by conducting an idle rundown test where the robot's battery is drained while the robot is in the idle state. We also conducted further moving rundown test where the robot's battery is drained while the robot is in constant motion. We compare the total time taken to drain the battery while idle to the distance traveled in the moving rundown test. This gives us an estimated number of meters consumed by s seconds idle. If the robot had not stayed idle for the entire idle rundown test it could have traversed the total number of meters traversed in the moving rundown test. Through this correlation we are able to determine how many meters the robot could have traversed had it not stayed idle.

We incorporate these models by examining the robot's state at frequent intervals and providing the models with the start and end states. We then deduct the estimated consumed runtime each model reports from the current estimated runtime. This translates roughly to:

$$currentRuntime = lastRuntime - idleModel(start, end) - movingModel(start, end)$$

Where $idleModel$ translates the start and end states into idle time which it then converts to potentially traversed meters. The $movingModel$ provides the distance the robot traversed between the start and end states. The starting runtime for the model is based on the training data we provide. It is the average of the total runtime seen in the calibration runs.

We created the *battery monitor* module which is tasked with carrying out state estimation in each robot. This module receives robot position and state data n times a second from the robot's control module and maintains the robot's current estimated runtime. Other modules can then query the battery module for the most up-to-date estimated runtime. The battery module currently runs the state estimator twice a second.

Another approach that we considered was based on battery voltage. A battery's remaining power is normally determined by the battery's voltage. A fully charged battery will have a certain voltage and as it is depleted this voltage falls until a threshold where the voltage can no longer power the robot. In our robots, full battery voltages range from $13V$ to $12.5V$ and empty

batteries are characterized by voltages around 11.5V. It is well known that battery voltage drops in a non-linear way and is thus difficult to estimate and use effectively. When we ran our rundown tests we logged all voltage data. We then examine the battery voltage curve as a function of time and attempted to model it. What we found was that voltage was very noisy, varied wildly between runs, and wasn't too consistent between runs, robots, and battery packs. These observations can be seen in Figure 5. Another property of the batteries that made it difficult to use for state estimation is the lack of discernable states. Our batteries report voltage up to the tenth of a volt and go from fully charged to deplete in about ten discernible states. Given that the robots can traverse up to 1,300m during this same time, we did not think these readings provided enough resolution.

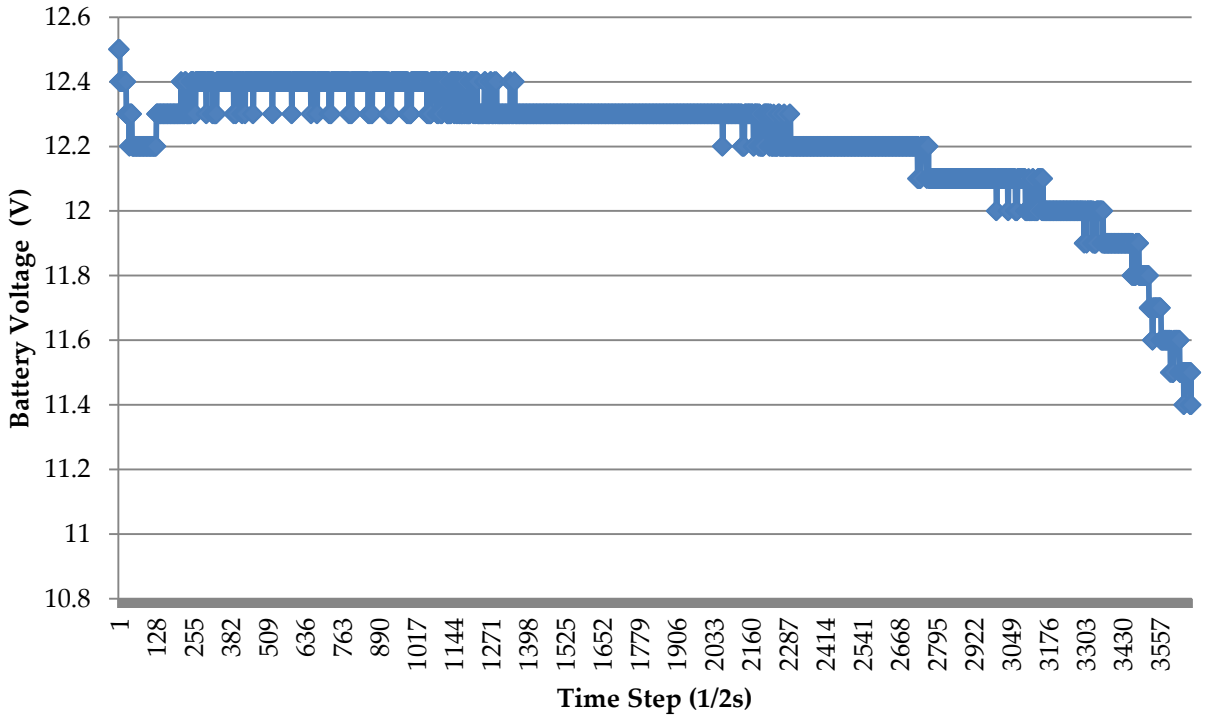

Figure 5- Graph of voltage over time. Note the volatility of readings, especially those between voltage thresholds.

We ran similar tests with multiple robots and multiple batteries packs. Some of the battery packs were new while others were packs the robots had been using for a few years. We found the same fluctuations in voltage in all packs. The main variation was that the runtime was extended for newer battery packs leading to longer robot lifetime.

We did consider other approaches for estimating power usage but chose against those options since they required extra hardware to measure power usage. This would be more costly, complicated, and time consuming. We opted with this approach due to its simplicity and lack of need for any hardware modifications to any component of our system.

3.2.3 | Customizing TraderBots

Market-based systems enforce certain abstractions in order to be able to create a very generic system which can be applied to any set of tasks. TraderBots is particularly well suited for this task as it allows for customization of nearly every aspect of the system [15].

The most basic component of any market-based system is the cost functions. These functions translate the work the robot must perform to complete a task into the *cost* to complete the task. This cost is then used as common currency robots can use to trade tasks and compare workloads. TraderBots' defaults require *pairwise cost functions*; these are a type of cost functions that take two tasks, A and B, as arguments. They then calculate the cost of completing B given the completion of A. We can calculate the cost of a schedule of tasks by running this pairwise cost function through each consecutive pair of tasks and summing the resulting cost. We must also account for the $Null$ task in the front of the schedule. The $Null$ task is representative of the robot's current position; it is used to calculate the cost of performing the first task given the robot's current position. This can be seen in the following example:

$$Cost\big(Schedule(A,B,C,D)\big) = Cost(Null,A) + Cost(A,B) + Cost(B,C) + Cost(C,D)$$

TraderBots also has a *scheduler* which keeps track of a robot's schedule. Schedules are ordered lists of tasks that the robot must performed in sequence. This ordering is placed because of dependency or cost reasons. Dependency reasons arise when one task has to be completed in order for another to be completed. Cost reasons arise when it might be more cost effective to have one task before another (maybe due to their proximity). The schedule does not have any deadlines associated with the tasks. The scheduler is defined as a set of functions applied to the schedule. These functions are outlined in Table 1. The default TraderBots scheduler uses pairwise costing functions and no schedule optimization. We decided to create our own custom scheduler since we wanted to customize how the schedule was optimized and since we also needed to add recharging tasks when necessary in the schedule.

Table 1 - Functions comprising a TraderBots scheduler

Scheduler Function	Function Description
InsertTaskAndCalculateBid	Inserts a task into the end of the schedule and determines the bid for this task by calculating the change in price of the schedule once the specified task is added
EraseTaskAndCalculateReserve	Erases a task in the schedule and determines the reserve cost for this task by calculating the change in price of the schedule once the specified task is removed
OptimizeSchedule	Optimizes the ordering of tasks in the schedule
GetScheduleCost	Returns the cost of the schedule by running the cost functions and adding their results

All other components of the TraderBots system were kept unaltered. Auctions would be triggered and carried out in the default manner. Bids and reserve prices would be calculated in the default manner. Winning bids would also be chosen in the default manner.

3.2.4 | Cost Functions

We wanted to have the cost functions determine the cost of a *balanced* schedule rather than that of the actual schedule. We define a balanced schedule to be a schedule of tasks which can be completed by a given robot given its current battery state and some scheduled recharging tasks. The cost of a schedule is the total distance (in meters) that the robot must travel to complete the tasks. With this in mind, we defined the cost functions to determine when a recharging task needed to be inserted and to account for that insertion in the cost of the schedule.

Since cost functions are evaluated between two tasks, they have to be defined between any two *types* of tasks. Given that we have two types of tasks (point and recharging); we must define four cost functions. We must also remember the *Null* task as a special type of task which is only evaluated at the beginning of the schedule. Given our two types of tasks and given that either one can be the first in our schedule; we have two more cost functions we must implement. This leads to six cost functions in total. These cost functions are summarized in Table 2.

Table 2 - The implemented cost functions.

Cost Function	Description
NullToPoint(NullTask, PoinTask)	The cost of traveling from the robot's current position to a specific location
NullToRecharge(NullTask, RechargeTask)	The cost of recharging at a specific location given the robot's current location
PointToPoint(PointTask1, PointTask2)	The cost of traveling from one specific location to another
RechargeToRecharge(RechargeTask1, RechargeTask2)	The cost of recharging at a specific location after recharging at another location
PointToRecharge(PointTask, RechargeTask)	The cost of recharging once we have traveled to a specific location
RechargeToPoint(RechargeTask, PointTask)	The cost of traveling to a specific location once we have recharged at another location

Generally there are three properties we want to hold in our cost functions. The first property is that we want to be conservative in our creation of balanced schedules and as such are pessimistic in the availability of mobile rechargers. This means that all planning must assume that no mobile rechargers are available and that the robot must instead recharge at the home recharging station. The second is that we want an accurate estimate of the current runtime at any given cost function. This means that when a cost function is being executed, it must have a value for what the estimated runtime will be at that point in time (when the event the function is evaluating will occur). This leads to the third property which is that all cost functions must leave the schedule in a stable state. This means that no cost function can allow the robot to deplete its battery or to put the robot in a situation where it will deplete its battery. This is done by guaranteeing that the robot has enough battery to reach the home recharge station after the completion of the tasks used by the cost function.

For all the cost functions below we determine distance given straight-line Cartesian distance due to its simplicity. There is no reason why we cannot replace this with a more complete path planner. However, that is left for future work. We will now describe the implementation of each cost function.

The *NullToPoint* cost function determines the cost of traveling from the robot's current position to a position specified by the point task. We chose to treat this cost function as we would treat a

PointToPoint cost function due to their identical semantics. The implementation simply calls on the PointToPoint function with the robot's current location masqueraded as the first point task. This cost function also initializes the estimated runtime used by the cost functions to the current runtime value queried from the battery monitor module.

```
function NullToPoint(nullTask, pointTask)
      runtime = BatteryModule.currentRuntime()
      newPointTask = nullTask
      return PointToPoint(newPointTask, pointTask)
```

The *NullToRecharge* cost function determines the cost of recharging given the robot's current location. Due the stable property of our cost functions, we know that the robot has enough battery to reach the recharging task. The cost is simply the distance from the robot's current position to the recharging task. We then update the current runtime to the value of a full battery charge as that is what the robot will have once it completes the recharging task.

```
function NullToRecharge(nullTask, rechargeTask)
      runtime = BatteryModule.fullBatteryRuntime()
      return distance(nullTask, rechargeTask)
```

The *PointToPoint* cost function determines the cost of traveling to a specific location given the robot has already traveled to another location. This is by far the most complicated cost function and is where recharging task insertion occurs. The function must determine whether the robot has enough battery to travel to the second point task and then go home to recharge. This is done to maintain the stable property of our cost functions. If this is possible, then the cost of completing the second point task is the distance between both tasks. If this is determined to be impossible, we must insert a recharging task before we complete the second specified task. In this case the cost returned by the function is the distance from the first point task to the home recharging station and then to the second task. In both cases the current runtime is updated by subtracting the distance between the tasks in the first case and by setting the current runtime corresponding to a full battery and then subtracting the distance from home to the second point task. This approach has one edge case however. In the case where there already exists a recharging task in the schedule which is closer than going home but not immediately after the first point task, this function will require a recharging task to be inserted into this schedule unnecessarily. This is circumvented by determining the distance to the next recharging task in the schedule and not inserting a recharging task if that distance is less than the robot's current

runtime. Thus, if it is beneficial to use that recharging task instead of going home, that approach is used.

```
function PointToPoint(pointTask1, pointTask2)
      distanceTaskToTaskToHome = distance(pointTask1, pointTask)
         + distance(pointTask2, home)
      distanceToNextRecharge = getDistanceToNextRechargeTask()

      if (distanceTaskToTaskToHome > runtime AND
            distanceToNextRecharge > runtime)
            // Insert recharging task
            cost = distance(pointTask1, home)
               + distance(home, pointTask2)
            runtime = BatteryModule.fullBatteryRuntime()
               -  distance(home, pointTask2)
      else
            cost = distance(pointTask1, pointTask2)
            runtime -= cost

      return cost
```

The *RechargeToRecharge* cost function determines the cost of recharging given we have just recharged. The cost of this task is simply the distance between both tasks. Current runtime is updated by setting it equal to the runtime corresponding to a full battery.

```
function RechargeToRecharge(rechargeTask1, rechargeTask2)
      runtime = BatteryModule.fullBatteryRuntime()
      return distance(rechargeTask1, rechargeTask2)
```

The *PointToRecharge* cost function determines the cost of recharging given that the robot has already traveled to a specific location. Due to the stable property of our cost functions the robot must have enough battery to reach the recharging location. Thus, the cost of this task is the distance between the tasks. Current runtime is updated by setting it equal to the runtime corresponding to a full battery.

```
function PointToRecharge(pointTask, rechargeTask)
      runtime = BatteryModule.currentRuntime()
      return distance(pointTask, rechargeTask)
```

The *RechargeToPoint* cost function determined the cost of traveling to a specific location given we have just recharged. The cost of this task is simply the distance between both tasks. Current runtime is updated by subtracting the distance traversed during this task.

```
function RechargeToPoint(rechargeTask, pointTask)
      cost = distance(rechargeTask, pointTask)
      runtime = BatteryModule.currentRuntime() - cost
      return cost
```

3.2.5 | Scheduler

The scheduler is defined via a series of pre-defined function. The four main functions are: GetScheduleCost, InsertTaskAndCalculateBid, EraseTaskAndCalculateReserve, and OptimizeSchedule. We now provide descriptions for the implementation of our custom scheduler. These are summarized in Table 3.

Table 3 - Summary of scheduler functions.

Function	Description
GetScheduleCost	Returns the total cost of the current schedule in meters.
InsertTaskAndCalculateBid	Inserts a task into the current schedule and calculates the added cost to the schedule by inserting this task.
EraseTaskAndCalculateReserve	Erases a task from the current schedule and calculates the decrease in cost to the schedule by erasing this task.
OptimizeSchedule	Optimizes the ordering of all tasks in the schedule. Includes optimizing the ordering of recharging tasks.

The *GetScheduleCost* function returns the cost of the current schedule given the current battery state. This is done by running the pairwise costing functions on the schedule as previously described and adding the cost of all these tasks.

The *InsertTaskAndCalculateBid* function calculates the cost of the current schedule and then inserts the specified task into the schedule. It then re-calculates the cost of the schedule to find the difference between the new schedule and the old schedule. This cost difference is what the robot would bid for this task as it is the cost this robot incurs while inserting this task into its

schedule. Optionally, the schedule can be optimized before the cost of the new schedule is calculated. This is option is given so that the schedule will not be optimize multiple times in short succession since optimization is a costly operation (in the case where we are inserting many tasks for example).

The *EraseTaskAndCalculateReserve* function is similar to InsertTaskAndCalculateBid since it finds the cost difference between two schedules; however, in this case the specified task is removed from the schedule. The cost difference represents the cost the robot incurs by having this task in its schedule. This is set as the reserve price; if no robot has a lower cost the current robot should keep the task as it has the lowest cost to complete it. Optionally, the schedule can be optimized before the cost of the new schedule is calculated.

The *OptimizeSchedule* function re-orders the schedule of tasks so that the total schedule cost is minimized. More so, the schedule that is outputted by the function is optimal in terms of distance traversed by the robot. The bulk of our work on the scheduler focused on the schedule optimizer as it proved to be the most complex component. Our optimizer did two types of optimizations: optimize the point tasks and optimize the recharging tasks.

Optimizing the point tasks is comprised of finding the minimal ordering of tasks. This minimal ordering is also optimal. The simplest approach is an exhaustive search which tests all possible permutations. This approach, while correct, produces a runtime of $O(n!)$ which we found to be unacceptable. We then focused on inserting a single task optimally. This is comprised of linearly testing the insertion of the task at each available location and then selecting the one that produces the minimal cost. We extend this to all n tasks by re-inserting the tasks into a new schedule one at a time. Since each task is inserted optimally, we finish this process with an optimal schedule. The runtime of this approach is a much improved$O(n^2)$.

```
function optimizeOneTask(schedule, task)
      forall tasks in schedule
            schedule.insertBefore(currentTask, task)

            if schedule.cost < minCost
                  minCost = schedule.cost
                  minBefore = currentTask

            schedule.erase (task)

      schedule.insertBefore(minBefore, task)
```

```
function optimizeAllTasks(schedule)
      newSchedule = empty schedule
      forall tasks in schedule
            optimizeOneTask(newSchedule, currentTask)

      return newSchedule
```

Optimizing the recharging tasks is significantly more complicated as the location of one recharging task greatly affects the location of other recharging tasks. If we place a recharging task earlier in our schedule, this may move later recharging tasks since it changes the robot's estimated runtime at that point in time. The optimizing function returns a list of locations where recharging tasks need to be inserted. These locations are the optimal placement for recharging tasks given the current schedule.

A naïve exhaustive search has a runtime of $O(2^n)$ as each of the n possible locations can either have or not have a recharging task present. This runtime is simply intractable for more than a handful of tasks. We originally implemented a scheduler in this manner and with a set of 50 tasks, observed the optimizer run for hours.

The next approach we tested was an optimization on the exhaustive search. In this approach we would find the last place where a recharging task could be inserted and still have a balanced schedule. This saved us from testing many solutions that would create unbalanced schedules. We saved this location and tested inserting the recharging task at every location before that saved location. We then optimized the remaining schedule of tasks by recursively applying the same logic to the smaller schedule after the inserted recharging task. The recursion ended when no further recharging tasks where necessary. This was determined by the robot's ability to perform the remaining schedule without recharging. After all of the possible locations were tested, the minimal was chosen and returned. This optimizer also produced optimal schedules and in reduced runtime. However, this reduction is not significant enough and thus does not allow for the optimization of tasks onboard the robots in real-time.

With a goal of significant reduction in runtime we set out to design a better optimizer. The first observation we made was that when we simulated the insertion of a recharging task and then recursed on the remaining schedule, we were in essence optimizing the smaller schedule.

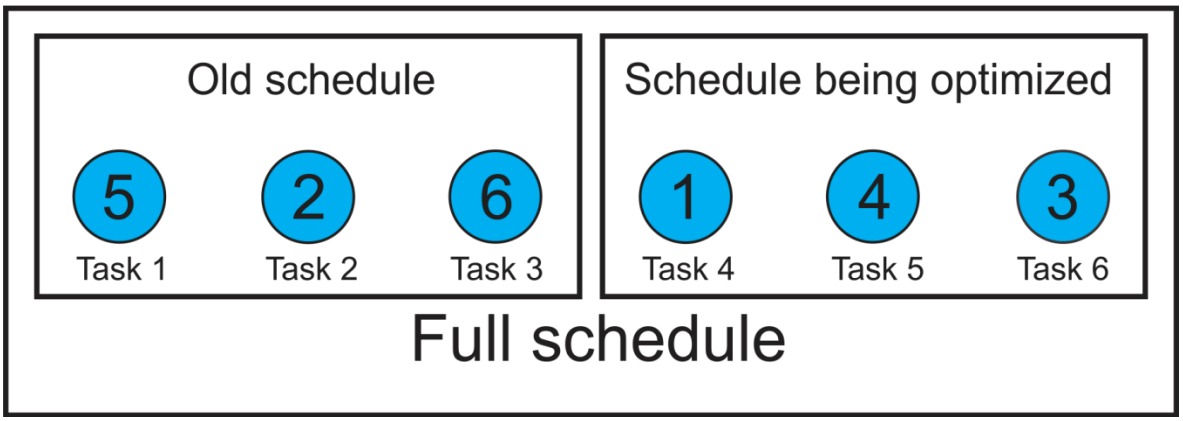

Figure 6 - Relation between the old schedule, the new smaller schedule being optimized, and the full schedule.

The next observation we made was that whenever the optimizer was called with this new smaller schedule to optimize, the answer was never going to change. What was optimal for the smaller scheduler last time we checked will still be optimal this time, even if the *old schedule* has changed. The old schedule is the beginning of the full schedule and the schedule from which we recursed into the new smaller schedule we are optimizing. The old schedule is comprised of the tasks that come directly before the new schedule and are not included in the new schedule. The relationship between the old schedule, new smaller schedule being optimized and the full schedule can be seen in Figure 6.

We used these two observations and applied memoization. When we recurse on a schedule we first check whether we have already calculated this schedule. If we have, we take the previously calculated insertions and return. If we have not then we run the same algorithm as before where we look for the last possible place we can insert a recharging task and check all placements before that insertion place. When we are done computing the result we then store it in a map with the first task in the schedule as the key. This algorithm guarantees that for any given smaller schedule, defined by its starting task in our full schedule, the optimization will only be calculated once. This component optimization takes a linear amount of work. When we do this for all n tasks in our schedule we find the optimal ordering of recharging tasks and do so with greatly reduced runtime. The runtime of this approach is $O(n^2)$ which optimizes our sample 50 task schedule instantly. This proves to be a great improvement over our previous approaches.

```
map memoizedValues.
function optimizeRechargingTasks(startTask, schedule)
      if memoizedValues.find(startTask)
            return memorized insertions

      firstRecharge = find first recharging task

      forall tasks between startTask and firstRecharge
            schedule.insertBefore(currentTask)
            currentInsertions = optimizeRechargingTasks(
               currentTask + 1, schedule)

            if schedule.cost < minCost
                  minCost = schedule.cost
                  minInsertions = currentInsertions

            schedule.eraseBefore(currentTask)

      memoizedValues.insert(startTask minInsertions)

      return minInsertions
```

3.2.6 | Recharging Behavior

Once the robots have created optimal balanced schedules and have inserted recharging tasks into their schedules, they must know what a recharging behavior entails.

The robot's most basic behavior is to go to the home recharging station. When such a task is triggered the robot goes home to its recharging station. It then waits for a human to start recharging the robot by physically plugging the robot into power or by having its batteries replaced with a new set. The batteries can be replaced without disconnecting power to the robot and can be used when a quick turnaround is necessary. The human then signals the robot that recharging is complete and the robot continues. In future prototypes we hope to automate this process by having a docking station for the robots and by having the robots analyze their voltage and determine when their batteries are filled.

Figure 7 - Prototype recharging hardware. Note the recharging robot on the left and the worker robot on the right.

A second type of recharging task is one in which the worker robot rendezvous with a mobile recharger and docks with it to recharge. In this approach the worker robot and the mobile recharger both travel to the rendezvous location and wait for the other robot. When both robots are present the robots dock and once the worker's batteries are filled, undock and continue. The docking and undocking behavior is currently being developed and can be seen in prototype form in Figure 7. Having two robots taking part in the docking maneuver makes it important for both robots to coordinate their actions. Towards this we have decided to have one robot maneuver and dock with the other robot while that robot stays stationary. We ran tests to determine which robot would be best fitted to perform the docking. The tests consisted of variations on the docking angle and variations on which robot, the recharger or the worker, stayed stationary. We found that when the worker was stationary the recharger successfully docked 90%+ of the time when the angle was within $\pm 45°$ from the direction the worker is facing. This is compared to the $\pm 35°$ achieved when the recharger was kept stationary.

3.3 | Mobile Recharging Agents

The next component of this research is the incorporating of mobile recharging agents into the already implemented charge-aware system. Our goal with mobile rechargers is to lower the cost of the worker robot's schedules by providing a recharging station closer than the home recharging station. Towards this, we have developed auctions, a scheduler, and cost functions that are unique to the mobile recharger and aim to have the mobile rechargers assist the worker robots as much as possible. This leads to an increase in total work performed by the group of robots as a whole.

Similar to the charge-aware implementation present in the worker robots, we must create a custom scheduler and cost functions in order to implement the mobile recharger's logic. As before, all other components of the market-based system are kept as the TraderBots defaults.

3.3.1 | Recharging Auctions

In order to create this coordination between workers and mobile rechargers we will have the workers auction recharging tasks. These tasks will specify the worker's schedule and ask to be recharged somewhere along its tour. In turn, each recharger will bid their cost to recharger the worker at a rendezvous point of their choosing. The recharger with the lowest cost will win the task.

The recharging tasks for mobile rechargers are different from the recharging tasks on worker robots. When a worker auctions a recharging task he must specify a range of tasks in its schedule through which he is available to be recharge. This necessitates that a mobile recharger's recharging task include a copy of this the worker's schedule.

When a worker has inserted a recharging task into its schedule it must also examine the possibility of having that task carried out by a mobile recharging agent. To do so the worker creates a recharging task that it will auction. The worker examines its schedule and determines that he must be recharged somewhere between the last recharging task and the recharging task currently being auctioned. Thus, the worker includes this segment of the schedule in the recharging task to be auctioned. The robot also includes the task right after the recharging task currently being auctioned. This will allow the recharger the option to rendezvous along the worker's path, away from the home recharging task, but still inside of the worker's range given its battery state. This task is then auction through the TraderBots library.

When a mobile recharger receives an auctioned task it must determine its bid for the task. It does so by first determining a rendezvous point in which recharging will occur, and then calculating the cost of traveling to that rendezvous point after it has completed its current schedule of tasks.

3.3.2 | Cost Functions

The cost functions for a mobile recharger must determine the best rendezvous point with the worker given the mobile recharger's existing schedule. The best rendezvous point is defined to be the closest one. This way we minimize the amount of distance the mobile recharger has to travel while still placing a greater emphasis on reducing the cost of the worker's schedule. Since

the mobile recharger only handles a single type of task, recharging tasks, it only needs to implement two cost functions. These are summarized in Table 4 and will be described in detail.

Table 4 - Mobile recharger cost functions

Cost Function	Description
NullToRecharge(NullTask, RechargeTask)	Determines the cost of recharging the specified worker given its current location.
RechargeToRecharge(RechargeTask1, RechargeTask2)	Determines the cost of recharging the specified worker given it has finished recharger another worker.

The *NullToRecharge* cost function determines the cost of recharging the specified worker given its current location. Since this cost function has the same semantics as the RechargeToRecharge cost function, it is implemented by simply calling the RechargeToRecharge cost function with the current location masqueraded as a completed recharging task.

```
function NullToRecharge(nullTask, rechargeTask)
      newRechargeTask = nullTask
      return RechargeToRecharge(newRechargeTask, rechargeTask)
```

The *RechargeToRecharge* cost function determines the cost of recharging the specified worker given it has already recharged another worker. All that is important about the completed recharging task (the first recharging task) is the location at which it occurred since it if from this location that the robot must travel to recharge the second worker. As such, this cost function must find the shortest distance between the first recharging task's rendezvous point and the schedule of the robot in the second recharging task.

We must keep in mind that the shortest distance might not be to one of the task's location but to a location in between two tasks. To do this we analyze each consecutive pair of tasks as a line segment and find the shortest distance between the point and the line segment. This shortest distance will be between some point on the line, what we call the *shortest point*, and our original point. If this shortest point is within the line segment then we calculate the distance to that point. If the shortest point is outside of the line segment, we select the closes endpoint of the line segment and calculate the distance to that point. An example of this is shown in Figure 8. We then find the minimal distance between the point and any line segment on the tour. This

final shortest point is what the mobile recharger will use as the rendezvous point. The distance to this point is the cost returned by the cost function.

```
function RechargeToRecharge(rechargeTask1, rechargeTask2)
      originalPoint = rechargeTask1.rendezvousPoint

      taskLines = consecutive tasks in rechargeTask2.schedule

      forall lines in taskLines
            shortestPoint = shortestPoint(currentLine,
               originalPoint)

            if distance(shortestPoint, originalPoint)<= minCost
                  minCost = distance(shortestPoint, originalPoint)

      return minCost
```

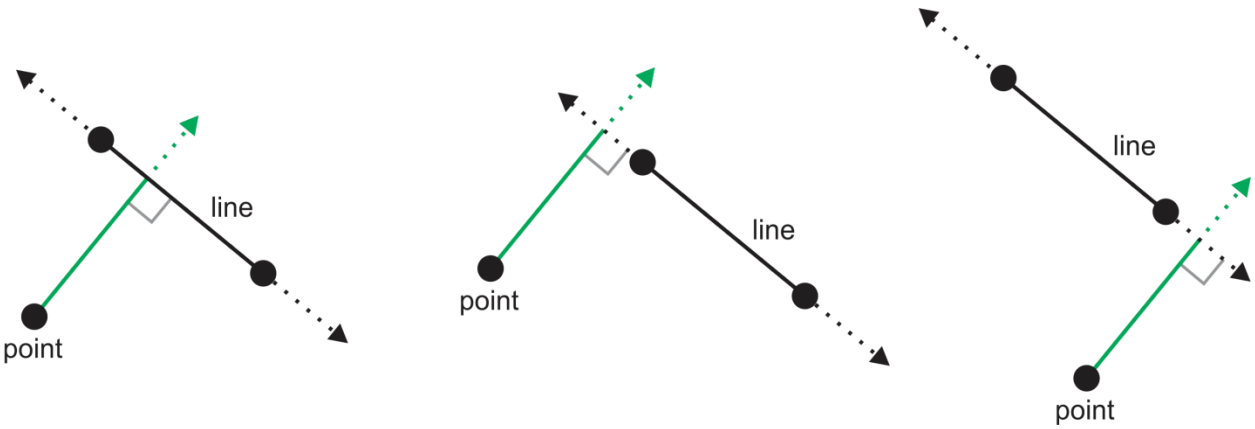

Figure 8 - The three posible locations of the shortest point. On the line segment (left), to the left of the line segment (center), and to the right of the line segment (right).

The shortest distance between a line and a point will always lie on a point within the line. The line formed by the shortest point and the original point intersect perpendicularly to the original line. To implement this we find the slope of the intersecting line and using that slope and the original point we now have the equation of the intersecting line. From here we simply find the intersection between both lines.

```
function shortestPoint(line, point)
      intersectLineSlope = -1 / line.slope
      insersectLine = make line from point and slope
      intersection = intersect of line and insersectLine

      if intersection is inside line
            return intersection
      else if distance(intersection, line.startPoint) <
         distance(instersection, line.endPoint)
            return line.startPoint
      else
            return line.endPoint
```

The only special case in this cost function is the final task. Recall that the schedule used by the mobile recharger to bid on the recharging task has the worker's go-home-and-recharge task as the second to last task in the schedule. The last task in the schedule is the task immediately after the go-home-and-recharge task. Since we want to intercept the worker somewhere on its tour we may need to do so in the last leg of the tour: between the last point task and the go-home-and-recharge task. To minimize travel by the worker we want to instead intercept him between the last point task and the point task after the go-home-and-recharge task (from now on referred to as the *first point task*). Thus, circumventing the original go-home-and-recharge task since the mobile recharger will be doing the recharging anyways. We must keep in mind that the worker is only guaranteed to have enough battery to reach the home recharging station. As such, we take the distance between the last point task and the go-home-and-recharge task and project it on the line between the last point task and the first point task. We then use that point as the final point in the schedule instead of the actual go-home-and-recharge task; unless the first point task is closer, in which case that task is used instead. An example of this is shown in Figure 9.

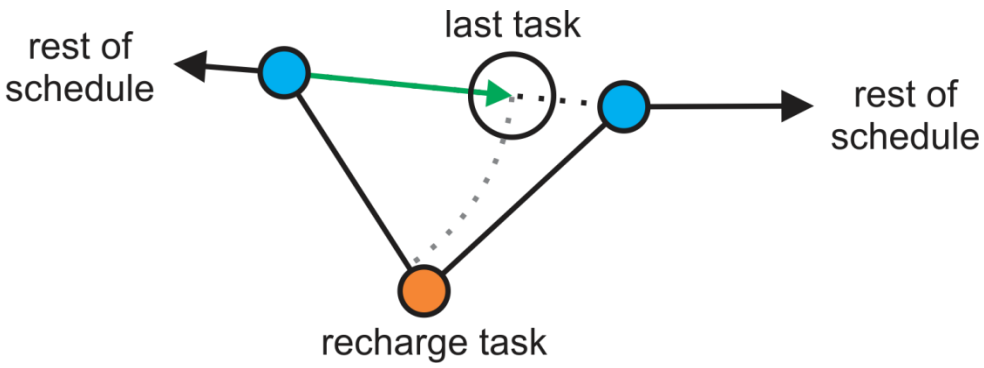

Figure 9 - The creation of the last task in the recharge task's partial schedule.

To implement this we calculate the distance between the last point task and the go-home-and-recharge task as well as the distance between the last point task and the first point task. We also calculate the ratio of their distances. We then scale the vector from the last point task to the first point task by the ratio of their distances. This gives us the desired last task.

```
function createLastTask(lastPoint, rechargeTask, nextPoint)
      distancePointToPoint = distance(lastPoint, nextPoint)
      distancePointToRecharge = distance(lastPoint,
         rechargeTask)

      if distancePointToPoint <= distancePointToRecharge
            return lastPoint

      scaleFactor = distancePointToRecharge /
         distancePointToPoint

      return (nextPoint - lastPoint) * scaleFactor
```

One property of the rendezvous point chosen by these cost functions is that since it is on the path of the worker's schedule, it is guaranteed to decrease the cost of the worker's schedule no matter where the rendezvous point is chosen.

3.3.3 | Scheduler

The recharger's scheduler must implement the same set of functions described in the charge-aware section. The mobile recharger has a different scheduler due to its different goal: to minimize the schedule of the worker robots. The recharger's scheduler is simpler since it is a first-in-first-out scheduler. The requests for recharging are serviced as a queue, thus the first worker to ask for recharging will be recharged before the second worker to ask for recharging. This is done to establish fairness in the process and to not cause the starvation of requests by favoring some type of request (whether that is a closer request or a request with a particular robot).

The *GetScheduleCost* cost function simply runs the pairwise cost functions along the schedule and returns the sum of the individual costs.

The *InsertTaskAndCalculateBid* cost function appends a task to the current schedule and calculates the bid from adding that task. No optimization is done.

The *EraseTaskAndCalculateBid* cost function removes a task from the current schedule and calculates the reserve cost from removing that task. No optimization is done.

The *OptimizeSchedule* cost function does not perform any action on the schedule. While it is possible to lower the cost of the mobile recharger's schedule this possible optimization is not done for the sake of focusing instead on decreasing the cost of the worker's schedules. It is also important to note that once a rendezvous point is chosen it is similar to a contract with the worker and as such must be kept intact. Optimizing the mobile recharger's schedule might involve changing this rendezvous point which will have adverse effects to the worker if he is not appropriately contacted and consulted.

4 | Experiments & Results

In order to evaluate the effectiveness of our approach and compare it to current approaches we must evaluate its components through a series of tests. We have divided our evaluation into two main components for evaluation: The state estimation used to determine remaining runtime. Second, the evaluation of the entire system and how it compares to existing systems.

4.1 | State Estimation

The first component we chose to test was the accuracy of our state estimation. We designed a series of tests to gauge the effectiveness of our state estimation in various states of robot activity.

We selected tasks that would have the robot run continuously in an open space and varied the frequency of the tasks. This way we also varied the percentage of time the robot was active. The chosen periods of activity were 100%, 75% 50%, and 25%. We took some initial training data with a run of 100% and 0% activity. These two correlate to our moving and idle rundown tests.

At the end of each run we compared the actual remaining runtime against the runtime estimated by our model. The error is defined as the actual remaining runtime minus the remaining runtime calculated by our state estimation. The actual remaining runtime is always zero since the battery has been depleted. A positive error correlates to underestimating the remaining runtime. This means that we estimated the battery would run out before it actually did. A negative error correlates to overestimating the remaining runtime. This means that the battery was expected to run out later than it actually did. The results of the tests can be seen in Table 5.

Table 5 - Summary of results for state estimation.

Percent Activity	Error
100%	3.7
75%	2.44
50%	1.52
25%	-12.03

4.2 | System Evaluation

The second component we chose to test is the effectiveness of our approach, how much total work is done by the robots as a whole. We also wanted to compare the performance of our system against that of current systems. Towards that we designed a series of tests from which we would run all approaches and compare their performance. We ran tests on actual hardware to verify our results on a real system and in simulation for testing larger scenarios.

We have chosen to run our tests on a single worker robot. This was done to create simpler and more manageable test cases while still showcasing the effectiveness of each recharging strategy. The only exception to this is in the cases where we test a system with a mobile recharger. In those cases we have a second robot which functions as the mobile recharger.

There are five recharging strategies that will be testing during our evaluation:

1. *Infinite battery*: This strategy assumes that the robot has an infinite amount of charge and thus never requires recharging. This strategy is used as an unreachable lower-bound on the performance of other strategies; no recharge strategy can perform better than the strategy that does not require recharging.
2. *Battery threshold*: This strategy determines when recharging should occur based on the current battery voltage. A robot is instructed to recharge when its battery voltage drops below a certain percentage of the full battery voltage.
3. *Distance threshold*: This strategy instructs robots to recharge when they have just enough battery to reach the recharging station. At the same time, this strategy wishes to maximize the number of tasks completed. Thus, if a robot has enough power to perform a task and then recharge, this strategy will chose that ordering.
4. *Charge-aware*: This strategy is our charge-aware system described previously. It determines the optimal ordering of recharging tasks and utilizes that ordering.
5. *Charge-aware with mobile recharger*: This strategy is identical to the charge-aware strategy but with the inclusion of a mobile recharger. This mobile recharger is in a support role and thus is tasked with maximizing the amount of work done by the worker robot. From now on we will refer to this strategy as *mobile recharger* for convenience.

The first series of tests we ran are designed to test the effectiveness of each strategy on a set of tasks that would require recharging multiple times. The tests were conducted with one of our Pioneer robots in the Carnegie Mellon University, Gates and Hillman Center high bay. We tested on an open area $10m$ by $5m$ with no obstacles in the robot's path (Figure 10).

Figure 10 - The test area at the Gates and Hillman Center high bay. Shown is a robot during one of the test runs.

4.2.1 | Real System Tests: Distance Metric

We created two schedules, each comprised of 50 point tasks placed randomly within the testing area. The sum of the total distance between the tasks was about $200m$ for both schedules. These tasks were given to the robot to execute. The robots performed no optimization on the order of the given tasks. The tasks were performed in the order given. To make the runtime of the tests tractable we artificially limited the robot's battery to $50m$ of runtime.

We continuously logged time, position, and battery. All strategies were run twice on each schedule and the presented results are the average values between both runs. The metric used for evaluation is total distance traveled in meters.

All presented schedules were computed in real-time except that of the mobile recharger strategy. Due to network-based constraints in TraderBots we were unable to trade the tasks in real-time. As such the schedules were manually pre-computed before the run. We do not believe that this pre-computation affected the results significantly due to our distance based metric and the speed of TraderBots' trading infrastructure.

The threshold used for the battery threshold strategy was 20% for the first schedule and 28% for the second schedule. These are percentages of total battery capacity. These values were chosen

since any smaller value caused the robot to deplete its battery during the run before it reached the recharging station.

The results of both schedules are presented below in Table 6 and Table 7. We have also included the number of recharging tasks as another means of comparison.

Table 6 - Schedule 1 results for all strategies

Strategy	Total Distance Covered (m)	Number of Recharging Tasks
Infinite battery	220.95	0
Battery threshold	242.58	5
Distance threshold	251.05	5
Charge-aware	234.75	5
Mobile recharger	221.31	7

Table 7 - Schedule 2 results for all strategies

Strategy	Total Distance Covered (m)	Number of Recharging Tasks
Infinite battery	169.67	0
Battery threshold	186.93	4
Distance threshold	186.80	3
Charge-aware	171.04	6
Mobile recharger	170.63	5

The results above show that our system exhibits a substantial reduction in distance traveled when compared to existing threshold approaches. The mobile recharger approach is even significantly close to the lower bound established by the infinite battery strategy. Table 8 and Table 9 summarize the gains made by our approaches when compared to existing strategies.

Table 8 - Summary of gains from charge-aware and mobile recharger strategies for Schedule 1. The center column is for charge-aware and the right column is for mobile recharger.

Strategy	Distance Gains (m) Charge-Aware	Distance Gains (m) Mobile Recharger
Infinite battery	-13.8 (-5.88%)	-0.36 (-0.16%)
Battery threshold	7.83 (3.33%)	21.27 (9.61%)
Distance threshold	16.30 (6.94%)	29.74 (13.44%)
Charge-aware	-	13.44 (6.07%)
Mobile recharger	-13.44 (-5.73%)	-

Table 9 - Summary of gains from charge-aware and mobile recharger strategies for Schedule 2. The center column is for charge-aware and the right column is for mobile recharger.

Strategy	Distance Gains (m) Charge-Aware	Distance Gains (m) Mobile Recharger
Infinite battery	-1.37 (-0.08%)	-0.96 (-0.56%)
Battery threshold	15.89 (9.29%)	16.30 (9.55%)
Distance threshold	15.76 (9.21%)	16.17 (9.48%)
Charge-aware	-	0.41 (0.24%)
Mobile recharger	-0.41 (-0.24%)	-

4.2.2 | Real System Tests: Time Metric

We would also like to present the results of these tests if they were to be evaluated under a time-based metric. In order to provide these results we must estimate the time required for recharging. We will estimate using two different methods.

The first method is equivalent to changing batteries rather than recharging them. Our robots are equipped with a set of batteries that can be changed on-line without turning off the robot's power. We have many sets of spare batteries on fast chargers just for this type of use. This method will estimate a constant time required for recharging. In order to calculate the total runtime we use the following equation:

$$time = tes\ runtime + number\ of\ recharging\ tasks * constant\ recharging\ time$$

The second method estimates the time it would take to recharge the battery to full capacity given its current state. We assume a linear relation between battery power and recharging time. Since we utilized a smaller simulated battery we will equally scale the known recharging times. This will give us an estimated recharging time that increases linearly as the battery is more depleted. In order to calculate the total runtime we run the following function:

```
function getTime(taskList, testRuntime)
      totalTime = testRuntime
      forall tasks in taskList
            totalTime += (1 – currentTask.batteryPercentage) *
               fullRechargeTime

      return totalTime
```

Using these equations and our existing data we get the results shown in Table 10 and Table 11. We have also included the base runtime for reference.

Table 10 - Schedule 1 results for all strategies under the two time metrics

Strategy	Base Runtime (*s*)	Method #1 (*s*)	Method #2 (*s*)
Infinite battery	960	960.00	960.00
Battery threshold	1042	1065.10	2934.90
Distance threshold	1092	1115.10	2810.70
Charge-aware	990	1013.10	2801.20
Mobile recharger	965	997.31	2555.40

Table 11 - Schedule 2 results for all strategies under the two time metrics

Strategy	Base Runtime (*s*)	Method #1 (*s*)	Method #2 (*s*)
Infinite battery	841	841.00	841.00
Battery threshold	893.5	911.96	2253.7
Distance threshold	934	947.85	2117.6
Charge-aware	901	928.69	1935.7
Mobile recharger	879.5	902.58	1904.8

We see that our approaches perform better for the most part except for the case of battery threshold and charge-aware in schedule 2. However, we are not as close to the time lower bound as we are to the distance lower bound. We summarize the gains for each method on Table 12 and Table 13.

Table 12 - Summary of gains from charge-aware and mobile recharger strategies for Schedule 1 using the two time metrics. The center two columns are for charge-aware and the right two columns are for mobile recharger.

Strategy	Method #1	Method #2	Method #1	Method #2
Infinite battery	-53.10 (-5.24%)	-1841.2 (-65.73%)	-37.31 (-3.74%)	-1595.4 (-62.43%)
Battery threshold	52.00 (5.13%)	133.70 (4.77%)	67.79 (6.80%)	379.50 (14.49%)
Distance threshold	102 (10.07%)	9.50 (0.34%)	117.79 (11.81%)	255.30 (9.99%)
Charge-aware	-	-	15.79 (1.59%)	245.80 (9.62%)
Mobile recharger	-15.79 (-1.56%)	-245.80 (-8.77%)	-	-

Table 13 - Summary of gains from charge-aware and mobile recharger strategies for Schedule 2 using the two time metrics. The center two columns are for charge-aware and the right two columns are for mobile recharger.

Strategy	Method #1	Method #2	Method #1	Method #2
Infinite battery	-87.69 (-9.44%)	-1094.7 (-56.55%)	-61.58 (-6.82%)	-1063.8 (-55.85%)
Battery threshold	-16.73 (-1.80%)	318.00 (16.43%)	9.38 (1.04%)	348.90 (18.32%)
Distance threshold	19.16 (2.06%)	181.90 (9.40%)	45.27 (5.02%)	212.8 (11.17%)
Charge-aware	-	-	26.11 (2.89%)	30.9 (1.62%)
Mobile recharger	-26.11 (-2.81%)	-30.9 (-1.60%)	-	-

4.2.3 | Scaling Number of Tasks

We then ran a series of tests on our simulator in order to test with larger data sets. Our simulator is comprised of the TraderBots scheduler. The scheduler is able to give an estimated cost for the entire schedule given the list of tasks. We use this to estimate the total distance the robot will travel as both cost and distance are measure in meters in our case. We have found the simulator to be within about 1% of the value found by experimentation. Thus, we believe the simulator's results are a good estimate of the behavior of the system.

The first behavior we wanted to observe was how each strategy scaled; in other words, how each strategy performed as we increased the number of tasks. In order to test this scenario, we generated a random schedule of 150 tasks and ran experiments from 50 to 150 tasks in increments of 10 tasks. Each strategy was run under each incremental schedule. The results of this test can be seen in Figure 11.

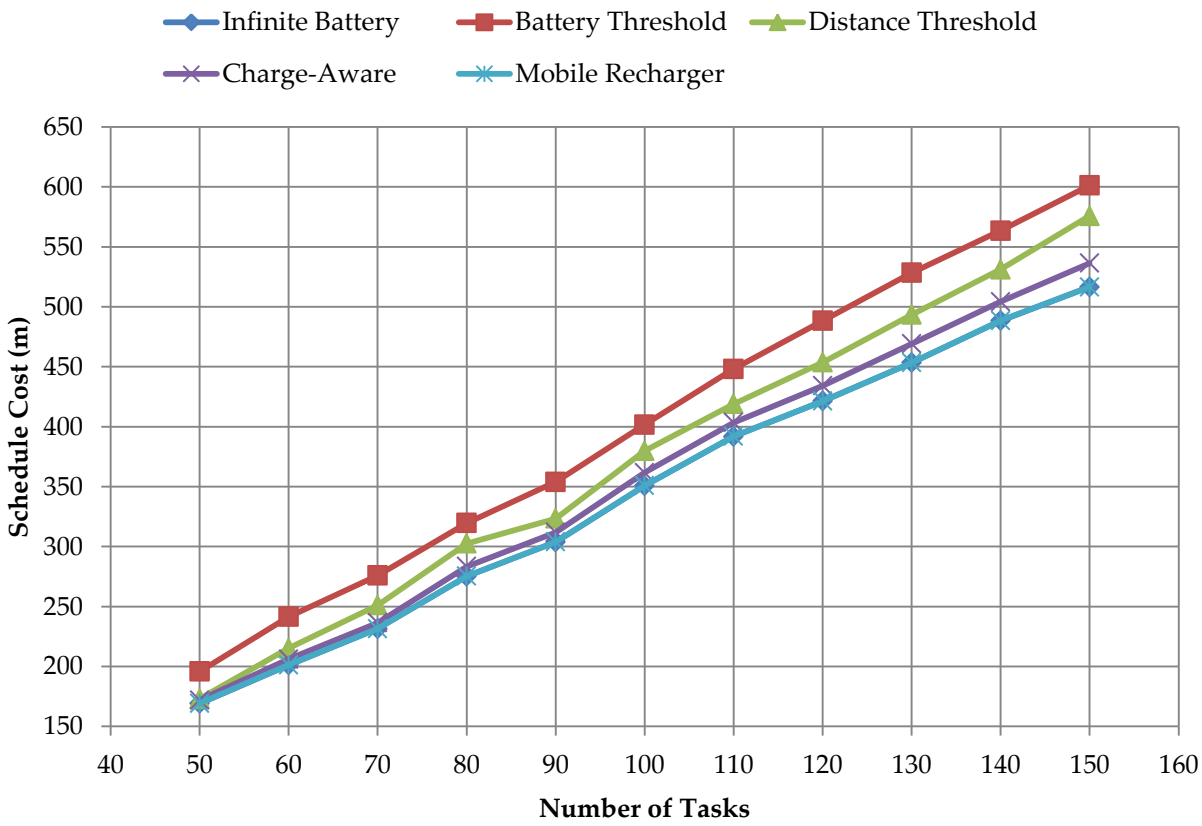

Figure 11 - Graph of schedule cost over number of tasks. Note that mobile recharger and infinite battery are near-identical.

From the above test we can see that our strategies consistently to outperform existing strategies.

4.2.4 | Effects of Battery Threshold

We also wanted to observe how the chosen threshold for the battery threshold strategy affected the performance of the strategy. We ran schedule 1 with varying thresholds from 20% to 36% in 2% intervals. The results of this test can be seen in Figure 12.

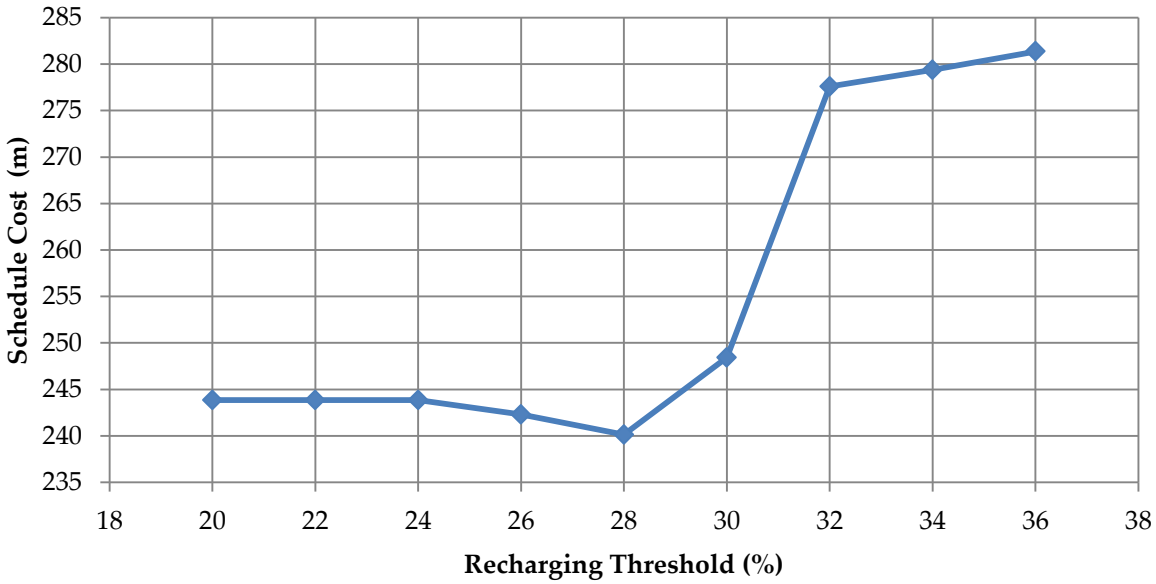

Figure 12 - Graph of schedule cost over different battery thresholds

These results show that there isn't a significant change in performance for the small thresholds we chose. However, performance quickly degrades with higher thresholds.

5 | Discussion & Analysis

We will now discuss some of the limitations and assumptions made during the implementation of our system and how these affect the performance of the system as a whole. We will also discuss the advantages of different components of our approach, as well as introduce some of the areas that can and will be met with improvements.

5.1 | Accuracy of State Estimation

We found that our state estimator gave us a good idea of remaining runtime for runs when the robot was mostly in motion for the duration of the run. Overall for these runs when it is accurate the state estimator always underestimated the runtime when compared to the actual runtime. This is a good sign in that our estimates are conservative and will not lead to the misplacement of a robot due to their battery losing charge (Table 5).

The only run that resulted in bad estimation was for the 25% moving run. We believe that this is an acceptable error, since the robot will be moving most of the time. This is due to the fact that we want the robot to be doing useful work as much as possible and spending large portions of time idle equates to lost productivity. In other words, we do not expect to have the robot perform runs where it will be mostly idle, and thus our current state estimation should be sufficient for our purposes.

The error in the 25% moving run can be attributed to a greater than expected power consumption by the robot while in idle mode. To mitigate this we could add a scaling factor to the idle power consumption; but this will serve to lower the accuracy of our other estimates. Since we plan to have the robot moving most of the time, we chose not to have this tradeoff. It may also be beneficial to examine a model where idle power consumption depends on what percentage of the robot's total lifetime has been spent idle. This, however, is considered future work. We discuss other possible methodologies for state estimation in the future work section.

5.2 | Advantages of Market-Based Systems

We chose to implement the above system as a market-based system due to our familiarity with such systems and because of their advantages. Our system exhibits many of these advantages in its implementation. There are three advantages in particular that contribute greatly to the potential of our system.

Market-based systems are very generic. These systems are not designed around one specific type of task and as such have the potential to be customized for any type of task needed by the users of the system. This generic nature is also seen in our system. We have been able to create a generic autonomous recharging solution that can be used with any type of work task. All that is required to create a new work task is to define its cost functions which are used by the system to understand the amount of work involved. The details of the execution of the work task are abstracted away from the recharging component allowing for its use without modifications.

A second advantage of market-based systems is their scalability. Other systems need to explicitly design for scalability with significant amount of interaction. However, a market-based system is self-adapting. The cost of tasks and of each robot's schedule is dynamic and adapts to the number of robots, the number of tasks, and the complexity to the environment. This is the reason why this type of system is commonly utilized in dynamic environments. Our system is directly impacted by this advantage in that we designed the system without any explicit decisions in the name of scalability; rather it is something we received for free. Rather than explicitly state the behavior of the recharger once there are more workers than it can service; this is solves via the auction mechanism and works regardless of the number of workers or rechargers. This is a limitation we found with many existing systems as their conflict-resolution strategies quickly broke down with the number of robots.

Similar to market-based system's scalability is their ability to run in a distributed fashion and the inherent fault tolerance. This system has no central point of failure and no synchronization between all robots. If a worker goes down, other workers can pick up its tasks and the recharger's schedules are automatically adjusted for this change. If a recharger goes down, worker robots are able to re-schedule their tasks into their original recharger-less schedules.

5.3 | Effectiveness of Our Strategies

Through our testing, we found that our system continuously outperformed existing systems in both metrics of distance and time. We now discuss both metrics in detail along with some limitations and considerations.

Our system outperformed existing threshold strategies under the metric of total distance traveled. This was the metric for which we designed our system and the metric for which we optimize. We chose this as a good estimate of power usage and work. We also believed that there would be some correlation between total distance and time. Our charge-aware strategy performed between 3% to 9% better than existing approaches with an average of about 7.2% decrease in total distance. This translated to about $13m$ less distance on our schedules (Table 8

and Table 9). Our mobile recharger approach performed even better with a decrease in distance between 9% and 13% with an average of 10.5%. This represents a decrease of over $20m$ in the worker's schedule (Table 8 and Table 9). The mobile recharger approach closely trails the lower bound of total distance, represented by the infinite battery approach. This is mainly due to the choice of recharging tasks along the worker's tour. This provides verification that our choice of utilizing the mobile recharger in a support role greatly increased the amount of work produced by the workers and in turn the group of robots as a whole.

Our approaches did tend to insert more recharging tasks than previous approaches, but these were inserted when it was opportune to do so. Of special note is the second schedule which had several non-recharging tasks on the home recharging station. Our approaches chose to take advantage of this fact and insert a recharging point close to those tasks. Thus, we gained recharging with little to no added distance. However, we can conceive of some repercussions for the added number of recharging tasks, further discussed below.

While our system was designed to optimize for time, we believe that for completeness the results should also be examined under the metric to total time. Our systems still managed to outperform existing strategies under this metric for the most part, but did so with varying improvements. Our assumption of the correlation between distance and time seems to have held given our results. The charge-aware strategy saw a gain of 2% to 10% using the first recharging method. There was one case where the charge-aware strategy performed worse than the battery threshold strategy. This is mostly due to the larger number of recharging tasks inserted by the charge-aware strategy. Under the second method we saw gains of 1% to 16% (Table 12 and Table 13). The mobile-recharger strategy again exhibited even greater gains than the charge-aware strategy. Under the first method we saw a reduction of 1% to 11% with an average of one minute saved. Under the second method we saw gains of 9% to 18% which equates to a gain of 4 to 6 minutes (Table 12 and Table 13). This approach does not reach the lower bound for time, as it necessitates some recharging, unless the recharging time is significantly shortened. Both of our recharging strategies show a significant savings in total time. This can be easily translated to potential for more work performed by the group.

Some would argue that our strategies are less efficient due to their use of more recharging tasks. However, since more tasks are inserted at convenient times, less distance is traveled. At the same time, since we are recharging with a more full battery than other strategies, our recharging times are less. This can be clearly seen by the results, especially how our gains tend to be better when the second method is used. A fixed time penalizes our strategies for coming in with non-empty batteries while a capacity-based time rewards the strategy.

Our reasoning for the two recharging methods is their relevance. The main problem we face in autonomous recharging is the total time necessary for recharging. This time can be large in comparison to the amount of time the robot can spend working. Thus, one solution is similar to method 1 which involved the exchanging of batteries. This proves to be efficient use of time but significantly more complicated than simple recharging. On the other hand there is method 2 which involved the recharging of the batteries. This method is simpler and can be achieved by almost all modern robotics platforms. It does, however, take a much longer amount of time. We believe that as battery and recharging technology improves we will see recharging time become less and less of a problem. These are the same problems faced by the automobile industry today in their drive for electric car. Both recharging and battery exchanging are being actively pursued by that industry as well.

While outside of the scope of this thesis, we do hope to optimize our schedules for time and will further discuss this in our future work section.

We were very interested in how the approaches scaled with larger and larger schedules. After running our tests we saw that our strategies outperformed current threshold approaches at every size schedule we tested (Figure 11). We can see a widening gap between our charge-aware approach and the threshold based approaches as the number of tasks increase.

One assumption made by our tests was that the entire schedule was known ahead of time and thus used for optimization. It is very common to have a very good idea of a robot's schedule of tasks, while this might not be complete knowledge; it is frequently to have good knowledge of the next few tasks. Our strategies are particularly well suited for exploiting this future knowledge and are able to plan around it. However, in the case that this knowledge is not available, it is important to note that our strategies degrade gracefully. The charge-aware strategy will match the performance of the distance threshold strategy as it will seek to recharge when there is just enough power to reach the recharging station. The mobile recharger strategy should still outperform other strategies, even by a small margin; due to its use of mobile rechargers that rendezvous worker's along their paths.

We did not test with an increasing number of worker robots but we expect the result to be similar to those obtained by our experiments. This is due to the fact that each individual robot's schedule can be seen separate from the others and any of these schedules will be better optimized by our strategies. Thus, the savings at each robot's schedule will be multiplied by the number of robots present in the group.

We briefly examined the effect of the chosen threshold on the performance of the battery threshold strategy. From our experiments, we saw that there was no significant difference between the performances of the smaller working thresholds (Figure 12). However, we did see

that there is an optimal threshold for any given set of tasks. The fact that this threshold changes depending on the particular tasks makes it very difficult to be chosen. We believe that this is a detriment of that strategy as the choice of threshold can significantly affect the performance if the threshold is too large or too small. When we tested smaller thresholds, in our case less than 20%, we found that the robot was too greedy and ran out of battery before it could recharge. Larger thresholds caused the robot to recharge too early and greatly decreased performance.

5.4 | Effectiveness of Mobile Recharging Agents

From our testing we saw the substantial decrease in both distance and time afforded by the inclusion of mobile rechargers. During our experiments we considered the case of one worker and one recharger where the worker had exclusive use of the recharger. As more workers are utilized, the mobile recharger will no longer be able to service all of the worker's requests. As this occurs we will see some of the worker's auctions finishing without bids and the workers being forced to go to the home recharging station. However, since the workers still utilize the charge-aware strategy their performance will not degrade to that of threshold based strategies. Rather, their performance will be somewhere between that of the charge-aware strategy and that of the mobile recharger strategy.

As we increase the number of rechargers, we can again have full coverage of the worker's recharging needs. It is interesting to see what the ratio of rechargers to workers should be for equilibrium to be reached and how that varies with tasks.

Since we do not optimize our tasks in terms of time, we do not place much emphasis or reducing the number of recharging tasks in the schedules that use the mobile recharger strategy. However, as we turn towards time-based metrics this becomes increasingly important and we must place some emphasis on minimizing the number of recharging tasks added to a schedule. Since these will again be on the worker's path, the distance change for the worker is negligible.

We have chosen to treat the mobile recharger as a support unit and place little emphasis on the cost of their schedules. However, we might be introducing inefficiency in our system, whereas we might be able to increase the work output of the group as a whole by asking the workers to go slightly out of their way. This approach is very complex and has already been examined by Litus et al. They have found such solutions to be at least as difficult as the NP-complete traveling salesman problem. This might make such solutions intractable for larger sets of tasks.

One final simplification we have made in our system is the state of the recharger. We currently do not account for the fact that the recharger must itself recharge and how that might affect its choice in tasks and locations. We believe that such work can be added as an extension to our

current system mainly through the cost functions. In essence, we can also view the mobile recharger as being charge-aware and planning its schedule accordingly. The only difference would be that the mobile recharger's work tasks are not point tasks but recharging tasks.

6 | Conclusion & Future Work

We began our work into autonomous recharging motivated by the importance of this topic to the field of mobile robotics. An analysis of current approaches led us to the conclusion that current systems were too greedy in nature, too near-sighted, exhibited too many edge cases that exhibit poor performance, and coordination was always seen as an after-thought.

We then set out to design a system that utilized all known information to create better schedules for all robots. We aimed to have coordination be a real and present part of our whole system with scalability as a close goal.

In general, we believe that our system exhibits a clear advancement in the state of the art for autonomous recharging. We are now better able to utilize past, present, and future knowledge of the robot's state and schedule of tasks. Utilizing this knowledge, we are able to create optimal recharging schedules which greatly reduce the distance, time, and power used by the robots to perform their work. This leads to an increase in the total amount of work performed by the group of robots as a whole.

We also see our implementation as sufficiently generic to be used for any type of task necessary for the completion of a group's goals. The choice of a market-based approach has also given us the inherent advantage of ease of scalability. Our system is able to handle a large number of workers and mobile rechargers in its current form with no changes and no sacrifice of robustness in coordination.

We see great room for improvement as the work presented here is just the beginning of more detailed and complete work. Autonomous recharging is an important and underdeveloped topic in robotics. It is a topic that holds much promise for the future and it is currently found in its infancy. Our goal is to develop a more complete, more robust, and less limited system.

With that in mind, we see this research continuing in multiple fronts, as listed below.

One of the first things we would like to consider is to optimize the robot's tasks for time. Motivated by the results of our distance-optimized schedules, we believe that there is room for improvement in terms of time. Such an approach must start with some cost applied to recharging time. This will emphasize the need to minimize this time in order to lower the total time. This will create systems that are more efficient in their use of power and will only recharge the necessary amount for completing scheduled tasks.

As the mobile recharger's role increases in importance, it is crucial for our simplifications to be removed. Mobile rechargers must also be charge-aware and schedule their own tasks around

the very real nature of their finite batteries. We believe that these changes can be reached through changes in our existing cost functions. These changes will see the worker's and recharger's cost functions becoming increasingly similar.

We found our state estimator to accurate but with room for improvement. We believe that excellent state estimation is essential for the optimization of the placement of recharging tasks. It is through this estimation that we are able to predict robot conditions and schedule accordingly. With the goal of better power estimation in mind we consider the previously-discarded approach of direct power estimation. We can directly measure the power usage of each of the robot's components. We can then separate a robot's work tasks into how much energy is used per component. Knowing the power capacity of the battery alongside the power usage of all components will allow us to better estimate remaining runtime. This approach holds promise as it has been used to measure and reduce power consumption of cellular phones [16]. We can use a similar approach coupled with marking and automatic recognition of batteries. This will allow each robot to store battery usage history and better predict the power available with any given battery it uses.

In the current implementation all distance measurement is done via straight line distances. While this works for the simple environments we use for testing, it is an assumption that will start to break down as we test on more complex environments. In order to mitigate this, we wish to move towards a better distance measurement heuristic such as D* Lite. D* Lite is an advanced heuristics-based path planner that is able to provide the shortest paths in environments with known obstacles and unknown areas. Utilizing D* Lite, will allow us to use our system in any environment currently traversed by mobile robots.

An important issue not discussed in this research is robot low power states. A robot may find itself idle at many times while either waiting for work or waiting for another robot or event. Modern computers have sophisticated power management frameworks that allow the system to consume little to no power while idle. We believe that such approaches can be extended to robotics in order to create robot low power states. In these states, robots will be able to switch off computing or sensors as deemed appropriate for the expected idle time. The robots can use a Wake-on-LAN mechanism to receiving a signal to leave the low power state. In this way, a robot queued for recharging could wait a nearly unbounded amount of time until it could recharge.

Another topic that must be considered by mobile rechargers is the importance of the tasks executed by the worker robots. Some robots might be performing mission-critical tasks with much higher priority than any other tasks. Mobile robots must be able to recognize and

prioritize cases like these. Other issues to consider include the state of the battery of the robot, giving preference to robots less likely to make it to the home recharging station.

One final item for future work is the role of a mobile recharger with multiple capabilities. In many systems, including our own, the mobile recharger has capabilities beyond just recharging. Our rechargers are also equipped with mapping capabilities. Thus, it might be beneficial at times to use these other capabilities while the recharger is mostly idle. Careful consideration and costing must be performed in order for this to succeed. This strategy will move a recharger from a support role into a worker role. We must take care not to have these roles conflict and cause less work to be completed by the group as a whole.

7 | References

[1] Seungjun Oh, A. Z. & K. Taylor (2000). Autonomous battery recharging for indoor mobile robots, in the proceedings of Australian Conference on Robotics and Automation (ACRA2000).

[2] Silverman, M.C ; Nies, D ; Jung, B & Sukhatme, G.S (2002). Staying alive: A docking station for autonomous robot recharging, in IEEE Intl. Conf. on Robotics and Automation, 2002

[3] Kottas, A., Drenner, A., and Papanikolopoulos, N. 2009. Intelligent power management: promoting power-consciousness in teams of mobile robots. In Proceedings of the 2009 IEEE international Conference on Robotics and Automation (Kobe, Japan, May 12 - 17, 2009). IEEE Press, Piscataway, NJ, 2459-2464.

[4] J. Wawerla and R. T. Vaughan. Optimal robot recharging strategies for time discounted labour. In Proc. of the 11th Int. Conf. on the Simulation and Synthesis of Living Systems, 2008.

[5] D. J. Austin, L. Fletcher, and A. Zelinsky, .Mobile robotics in the long term,. in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct. 2001.

[6] Litus, Y., Vaughan, R. T., and Zebrowski, P. (2007). The frugal feeding problem: Energy-efficient, multi-robot, multi-place rendezvous. In Proceedings of the IEEE International Conference on Robotics and Automation.

[7] Wawerla, J. and Vaughan, R. T. (2007). Near-optimal mobile robot recharging with the rate-maximizing forager. In Proceedings of the European Conference on Artificial Life.

[8] M. Silverman, B. Jung, D. Nies, G. Sukhatme. "Staying Alive Longer: Autonomous Robot Recharging Put to the Test." Center for Robotics and Embedded Systems (CRES) Technical Report CRES-03-015. University of Southern California, 2003.

[9] Alex Couture-Beil and Richard T. Vaughan. Adaptive mobile charging stations for multi-robot systems. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS'09).

[10] St. Loius, MO, October 2009.Zebrowski, P ; Vaughan, R (2005). Recharging Robot Teams: A Tanker Approach, International Conference on Advanced Robotics (ICAR'05), Seattle, Washington, July 18th-20th, 2005.

[11] Yaroslav Litus, Pawel Zebrowski, and Richard T. Vaughan. A distributed heuristic for energy-efficient multi-robot multi-place rendezvous. IEEE Transactions on Robotics, 25(1):130-135, 2009.

[12] Munoz A., Sempe F., and Drogoul A. (2002). Sharing a Charging Station in Collective Robotics.

[13] Sempé F., Muñoz A., Drogoul A. "Autonomous Robots Sharing a Charging Station with no Communication: a Case Study." Proceedings of the 6th International Symposium on Distributed Autonomous Robotic Systems (DARS'02). June 2002.

[14] M. B. Dias, "Traderbots: A new paradigm for robust and efficient multirobot coordination in dynamic environments," Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, January 2004.

[15] TraderBots User's Guide. Carnegie Mellon University, National Robotics Engineering Center. August 1, 2008.

[16] Flinn, J., Satyanarayanan, M. Energy-aware Adaptation for Mobile Applications. In Proceedings of the 17th ACM Symposium on Operating Systems and Principles. Kiawah Island, SC, December, 1999.

[17] McFarland, D. & Spier, E. (1997). Basic cycles, utility and opportunism in self-sufficient robots. Robotics and Autonomous Systems, 20, 179-90.

[18] Birk A. (1997) Autonomous Recharging of Mobile Robots. In: Proceedings of the 30th International Sysposium on Automative Technology and Automation. Isata Press

[19] Ngo, T. D., Raposo, H., Schioler, H., Being Sociable: Multirobots with Self-sustained Energy, Proceedings of the 15th IEEE Mediterranean Conference on Control and Automation, Athens, Greece, 27-29 July, 2007