Market-based Coordination of Recharging Robots

Victor Marmol, Balajee Kannan, and M. Bernardine Dias

CMU-RI-TR-12-28

September, 2012

The Robotics Institute

Carnegie-Mellon University

Pittsburgh, Pennsylvania 15213

Abstract

As multi-robot systems gain acceptance for use in functionally-distributed missions that require complex coordination for executing tasks such as planning, coordination, and information sharing in highly dynamic and potentially hazardous operating environments [12, 13, 20, 28-31], the ability of the robots to operate for extended time in the field becomes critical to mission success. Consequently, the problem of 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, leading to inefficient solutions that adversely affect system performance. Effective coordination of robot teams is an ongoing challenge and has been addressed using techniques varying from switched control [38-39], vision-based formation control [40], to market based approaches [27, 42, 43]. In this report, we advance the state of the art in autonomous recharging by developing, implementing, testing, and evaluating a market-based distributed algorithm for effectively coordinating recharging robots. Such a system is "charge-aware" and accounts for battery life when during task allocation process. The developed solution has been evaluated, in simulation and in field tests, on a team of pioneer mobile robots executing a set of transportation tasks in an indoor environment. Results show that our approach consistently outperforms the state of the art in recharging strategies.

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. Inherently, mobile robots can perform a finite amount of work in a single work cycle limited by charge 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. Each robot needs to plan its actions taking into consideration their battery life and when recharging needs to occur. Consequently, this leads to constant human oversight and relatively long down times. . As the number of robots in the teams increase, the system requires near constant attention from human operators. Autonomous recharging brings with it the promise of extended runtime and enhanced system performance with reduced downtime. Robots will be able to run continuously for extended periods of time with little to no human interventions.

Currently, robots operate on low-cost portable batteries that can be recharged in the field using other mobile recharging agents, thereby providing enhanced operational flexibility. The motivation of this research stems from the importance of battery life in the amount of work performed by a mobile robot. 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.

Current approaches to autonomous recharging are mostly greedy and inefficient. Each robot makes recharging decisions based on only their current state with little to no regard for past and future state. This leads to sub-optimal performance and unaccounted edge cases. Additionally, 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. Further, these strategies also do not scale as the number of robots sharing the resource grows.

In this work, we have outlined an autonomous recharging solution 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. Unlike existing methods, our

approach utilizes a robot's task schedule to determine the optimal plan 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. Towards that we use a distributed market-based strategy for coordinated recharging. Market-based approaches [14] use a simulated economy where robots buy and sell tasks according to their estimated cost for completion. The estimated costs of tasks are based on a set pre-defined cost functions shared amongst the agents in the market-based system. This allows for a system where robots attempt to minimize their individual costs and consequently the cost of the group as a whole, which in turn maximizes the work performed by the team of robots. Our approach is the first to use the market-based system to schedule worker's tasks and to coordinate a mobile recharger's tasks.

The rest of the paper is organized as follows. Section 2 describes the background of our research and the existing strategies for autonomous recharging. 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 and provide a description of future work.

2 | Related Work

In this section, we describe the existing work in autonomous recharging solutions for mobile robots. Existing research on autonomous recharging is focused on four main areas: recharging hardware, recharge location, conditions for recharging, and mobile recharging solutions.

2.2 | Recharging Hardware

A lot of research has focused on the hardware necessary for static recharging units. Oh et al. [1] describe a docking-based static recharging station that can be located via long-range infrared beacons. The robot is maneuvered on to the recharging station using a grid of specially aligned laser-visible targets placed on the recharging station. Silverman et al. present an alternate static docking system that 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. The system is effective at docking from large displacements thus allowing more error in their short-range navigation [2]. Muñoz et al. have developed a robust recharging dock that 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]. All of the described systems are implemented for a single robot travelling to a pre-defined location for recharging.

2.3 | Location of Recharging Stations

The location of recharging stations can have a great impact on the total work a group of robots can perform. Couture-Beil and Vaughan [9] 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 at some distance from the area of highest utilization. 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

Currently, the overwhelming solution for when to recharge has relied on a singular threshold, either of distance or time [2], [8], [12], [13]. Since this approach is not based on the robot's location, task load, or the location of the recharging station, there is a relatively lower success rate for robot making it to the recharging station on time. Further, the chosen threshold is critical to the effectiveness of the 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, while larger thresholds lead to inefficiency in the system as the robots have enough battery to do more work but are asked to recharge prematurely. Alternate threshold-based approached uses time as the primary criterion. Austin et al. have developed one such system where the 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]. However, choosing the appropriate threshold is inherently difficult leading to in-efficient solutions. 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. In such cases, it is critical for the system to properly determine when the robot has enough battery to reach the recharging station.

Most existing works on autonomous recharging deals with single worker robot charging. Predominantly, existing work on recharging multiple workers handles coordination through conflict resolution rather than as multi-robot coordination problem [12]. This greedy approach can lead to less desirable situations, where a group of workers queue up around a single 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, and second when the charge station is completely blocked, effectively rendering it useless.

2.5 | Mobile Rechargers

An alternate strategy to recharging is the use of mobile recharging station. Mobile recharge stations have increasingly become popular as they offer a lot of flexibility for robust 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. Kottas et al. [3] have developed a mobile recharging station based on the Pioneer robotics platform. This mobile recharger is capable of recharging multiple small robots 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 that 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 reduces 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.

In summary, current approaches to autonomous recharging have little coordination and do not make full use of the knowledge about the past, current, and future state of the robots. Furthermore, existing solutions are not designed to scale for large teams of robots. In our approach, we 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

Our solution to coordinating team of recharging robots comprises three core components. First, each individual robot is made "charge-aware" and can incorporate recharging as a critical task into its schedule based on schedule history and current charge awareness. Second, we developed a market-based strategy for resource allocation based on individual robot's charge awareness. Finally, we introduced a sub-team of mobile rechargers whose objective is to lower the cost of the worker robots' schedules towards maximizing system performance and extended our market-based solution to incorporate them while determining the best allocation strategy.

3.1 | Charge-Aware System

Our charge-aware system runs on each robot and proactively plans the order in which the robot should execute its tasks, including recharging. These plans are made based on the robot's battery state and schedule of 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.

Home Base

The robot's start location is designated as its home base. It is the location to which the robots must return once deployment is complete. Each robot is guaranteed to have an available charger for its use at the home base. 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 expensive to return home. It would be much lower in cost for the robot to be recharged by a mobile recharger rather than returning. However, depending on availability, or lack thereof, of mobile, 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.

State Estimation

Accurately estimating a robot's runtime is important as it is used 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 given recharging station, static or mobile. Alternately, an underestimated runtime will lead to reduced total throughput as the robot will perform less work per recharging cycle than its batteries allow. 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 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 ran 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 implemented a system that started with the observed 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.

Power Estimation based on Operation State

We identified 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 actively rotating. 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 estimate runtime consumed by the robot in a particular 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 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:

$$currentRuntime = lastRuntime - idleModel(start, end) - movingModel(start, end) \quad (1)$$

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.

Power Estimation based on Battery Voltage

An alternate approach that we considered was based on battery voltage. A fully charged battery will have a certain voltage, which will drop below a threshold during the course of operation. 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 and examined 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 1. 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.

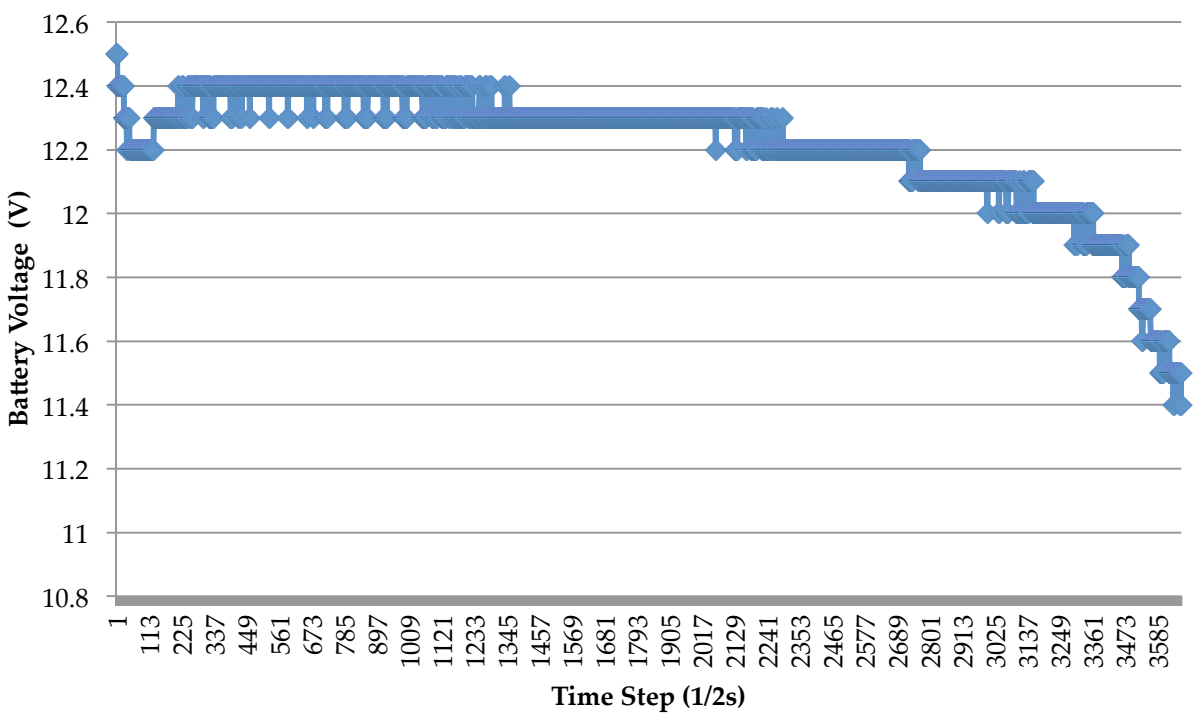

Figure 1- Graph of voltage over time. Note the volatility of readings, especially those between voltage thresholds.

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.

3.2 | Market-based Strategy for Coordinating Recharging Robots

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 key component of any market-based system is the cost function. 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 cost paradigm involves *pairwise cost functions*; cost functions that take two tasks, A and B, as arguments and 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. Further, the default TraderBots *scheduler* actively keeps track of a robot's schedule. Schedules are ordered lists of tasks that the robot must performed in sequence. This ordering is placed when one task has to be completed in order for another to be completed or when it might be more cost effective to have one task before another (maybe due to their proximity)..

3.2.1 | Cost Functions

The cost functions need to 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.

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). Finally, 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.

As part of our costing solution, we define 6 key costing functions. These include:

(1) *NullToPoint* cost function determines the cost of traveling from the robot's current position to a position specified by the point task.

(2) *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.

(3) *PointToPoint* cost function determines the cost of traveling to a specific location given the robot has already traveled to an alternate location. 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. 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. We over come this 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.

(4) *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.

(5) *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.

(6) 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.

3.2.2 | Scheduler

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.

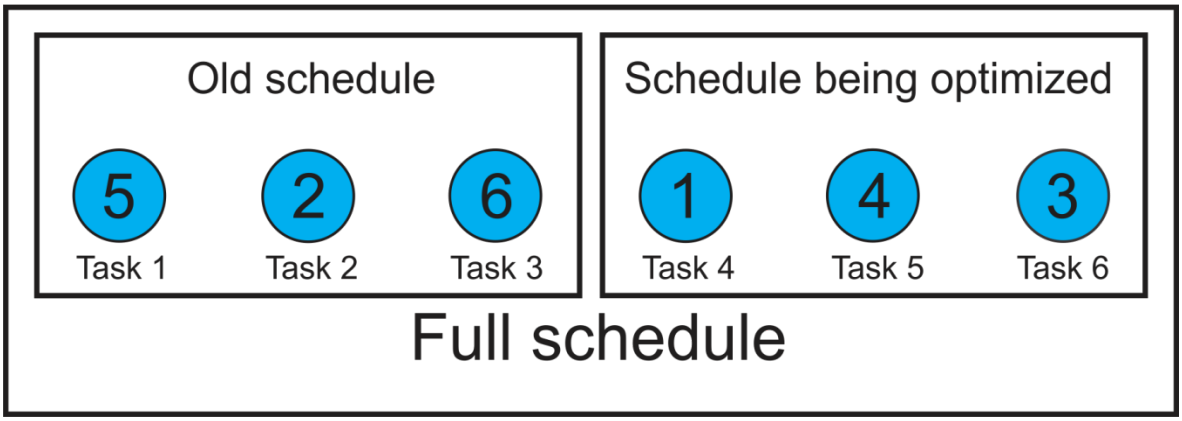

Figure 2 - Relation between the old schedule, the new smaller schedule being optimized, and the full schedule.

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 2.

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
```

3.2.3 | 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 3 - 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 3. 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.

3.2.4 | 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.

3.2.5 | 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 1 and will be described in detail.

Table 1 - 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 **Error! Reference source not found.**. 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
```

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 4.

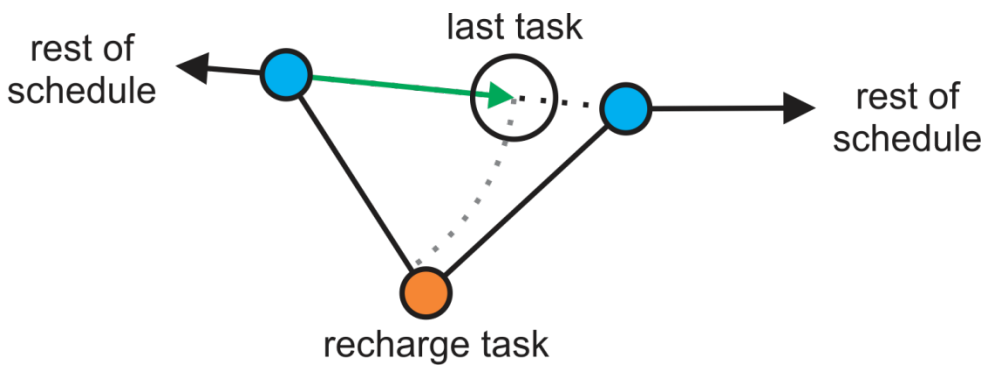

Figure 4 - The creation of the last task in the recharge task's partial schedule.

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.

3.2.6 | 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.

Experimental Platform

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 5 - 3 Pioneer P3DX robots

The rCommerce group has six Pioneer P3DX robots (Figure 5) 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. 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 6). 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 6 - Pioneer with recharging arm docked with another Pioneer

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. 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.

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 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. In our robots, full battery voltages range from $13V$ to $12.5V$ and empty batteries are characterized by voltages around $11.5V$

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 2.

Table 2 - 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 7).

Figure 7 - 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 3 and Table 4. We have also included the number of recharging tasks as another means of comparison.

Table 3 - 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 4 - 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 5 and Table 6 summarize the gains made by our approaches when compared to existing strategies.

Table 5 - 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 6 - 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 7 and Table 8. We have also included the base runtime for reference.

Table 7 - 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 8 - 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 9 and Table 10.

Table 9 - 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 10 - 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 8.

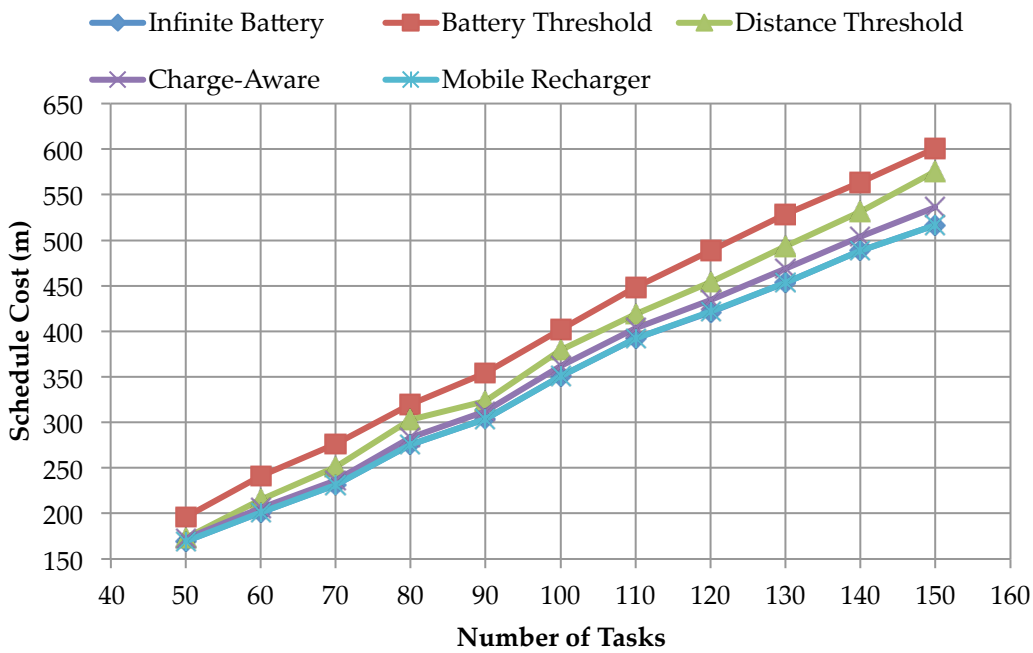

Figure 8 - Graph of schedule cost over number of tasks. Note that mobile recharger and infinite battery are near-identical.

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 9.

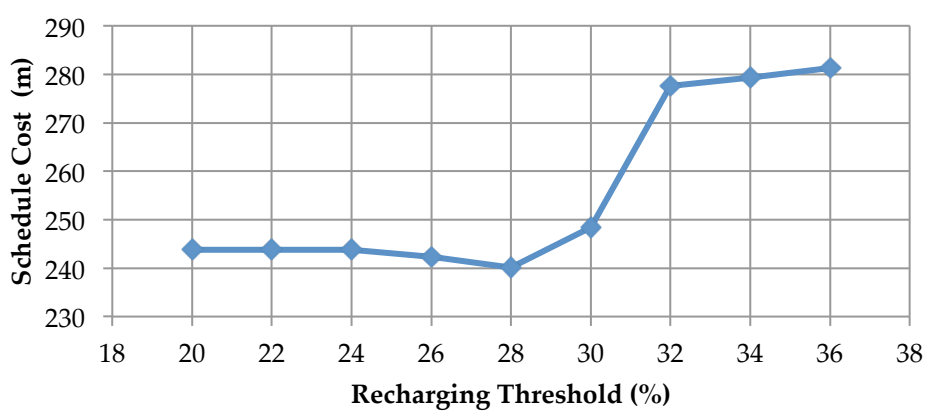

Figure 9 - Graph of schedule cost over different battery thresholds

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 2).

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 ~5% better than existing approaches with an average of about 7% decrease in total distance. This translated to about $13m$ less distance on our schedules (Table 5 and Table 6). 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 5 and Table 6). 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 9 and Table 10). 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 9 and Table 10). 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 8). 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.

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 9). However, we did see that there is an optimal threshold for any given set of tasks. 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 on multiple fronts. We would like to consider optimization of 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.

Acknowledgements

This work is sponsored in part by The Boeing Company. The content of this paper does not necessarily reflect the position or policy of the sponsors and no official endorsement should be inferred. The authors would like to thank Jimmy Bourne, M. Freddie Dias, Nisarg Kothari, and Sairam Yamanoor as well as other members of the rCommerce Laboratory at Carnegie Mellon University for their valuable contribution during development and testing.

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