Towards High-Quality Battery Life for Autonomous Mobile Robot Fleets

Akshar Shravan Chavan
Department of Computer Science
Wayne State University
Detroit, USA
gp6989@wayne.edu

Marco Brocanelli
Department of Computer Science
Wayne State University
Detroit, USA
brok@wayne.edu

***Abstract*—Autonomous Mobile Robots (AMRs) rely on rechargeable batteries to execute several objective tasks during navigation. Previous research has focused on minimizing task downtime by coordinating task allocation and/or charge scheduling across multiple AMRs. However, they do not jointly ensure low task downtime and high-quality battery life.**

In this paper, we present TCM, a Task allocation and Charging Manager for AMR fleets. TCM allocates objective tasks to AMRs and schedules their charging times at the available charging stations for minimized task downtime and maximized AMR batteries' quality of life. We formulate the TCM problem as an MINLP problem and propose a polynomial-time multi-period TCM greedy algorithm that periodically adapts its decisions for high robustness to energy modeling errors. We experimentally show that, compared to the MINLP implementation in Gurobi solver, the designed algorithm provides solutions with a performance ratio of 1.15 at a fraction of the execution time. Furthermore, compared to representative baselines that only focus on task downtime, TCM achieves similar task allocation results while providing much higher battery quality of life.

I. Introduction

Due to their relatively high return of investment and integration with Artificial Intelligence (AI) technologies, fleets of battery-operated Autonomous Mobile Robots (AMRs) are increasingly penetrating society for security patrol [1], [2], precision farming [3], [4], environment exploration [5], and home assistance [6]. Managing an AMR fleet leads to two problems. First, AMRs must have coordinated access to shared charging stations to minimize queuing time, which could lead to task downtime. Second, in line with NSF's objective for sustainable computing [7], the batteries' State of Charge (SOC) must be controlled for high-quality battery life.

Several related studies have proposed task allocation [8]–[11] or charging coordination [12]–[14] for an AMR fleet, but none of them take care of battery quality of life. The problem of controlling the battery lifespan in computing systems has been studied in several scenarios such as data centers and IoT devices [15]–[17]. In general, ensuring a certain lifespan for batteries requires keeping the battery's SOC between two thresholds defined according to the specific battery technology [18]. On the other hand, in this paper we argue that, *for the case of AMRs, ensuring long battery lifespan does not guarantee a high-quality battery life*. While most battery-operated computing devices (e.g., smartphones) can be recharged without energy overheads, an AMR scheduled for recharge needs to interrupt the execution of the *useful* work (e.g., security patrol) and travel a certain distance to reach a charging station. The locomotion, computational, and sensing energy necessary for the AMR to reach the charging station is not spent towards *useful* work, thus we consider it to be *wasted*. To capture how much energy drawn from an AMR battery during its lifetime is actually translated into useful work, in line with the Power Usage Effectiveness (PUE) metric used for data centers [19], we define the *Energy-usage-EFfectiveness (EEF)*. An EEF of 1 and 2 mean that 100% (i.e., ideal) and 50% of energy is spent in useful work, respectively. Thus, for a given set of SOC thresholds used to obtain a certain desired lifespan from the AMR batteries, it is desired to coordinate task allocation and charge scheduling to ensure low task downtime and high-quality battery life, i.e., desired battery lifespan and EEF as close as possible to 1.

In this paper we propose TCM, an energy-aware Task and Charging schedule Manager, that coordinates the allocation of useful tasks to AMRs with the charging schedules to minimize task downtime and provide high-quality battery life. To improve coordination and robustness to energy modeling errors, the TCM problem is defined as a multi-period problem that periodically activates within the working period to (*i*) fetch the current SOC of the AMRs, (*ii*) find a solution for current and future periods, and (*iii*) communicate to the AMRs the allocation/charging decision for the current period. The specific contributions of this paper are as follows:

1) We introduce the concept of Energy Usage Effectiveness (EEF) to measure how much energy drawn from an AMR battery is spent towards useful task execution rather than being wasted on frequent trips to the charging stations. Low battery degradation and near-1 EEF lead to high-quality battery life for AMRs.
2) We formulate the TCM problem as a Mixed Integer Nonlinear Program (MINLP) to trade off between task downtime and battery quality of life based on a weight parameter (chosen by the fleet manager) that gives more or less importance of the each of the two objectives.
3) The defined MINLP has exponential time complexity

This research was supported by the US National Science Foundation under Grant CNS-1948365.

[20]. To find a solution within polynomial time, we design a multi-period TCM algorithm that greedily trades off between task downtime and battery quality of life.

4) We empirically show that our TCM algorithm has a performance ratio of 1.15 compared to the MINLP implementation in Gurobi at a fraction of execution time. Compared to several representative baselines, it achieves similar task allocation performance while ensuring up to 22.36% better EEF and 193% lower battery degradation.

The rest of the paper is organized as follows. Section II describes the related work. Section III formulates the TCM problem as an MINLP problem. Section IV describes our greedy TCM algorithm. Section V shows our experimental results and Section VI concludes the paper.

II. Related Work

Several studies coordinate multiple robots by dynamically allocating tasks without considering the problem of limited energy consumption [21]–[25]. Thus, other solutions focus on energy-efficient path planning for a single AMR [26]–[29], energy-aware task allocation to multiple AMRs [8]–[11], [30], and locomotion/network energy-minimization for robot sensor networks [31]–[33]. Unfortunately, most of the above research work has the following two main problems. First, they mainly focus on the locomotion energy consumption without taking into consideration the sensing and computing energy of objective and navigation tasks, which, as described in our previous work [34], can account for a large portion of the AMR power consumption. Second, they do not coordinate the AMRs for autonomous recharging at charging stations. These issues may result in unexpected downtime for the task execution due to the limited number of charging stations available. In order to handle the autonomous charging problem, some studies decide when to charge a single AMR [35]–[38] while others consider the charging coordination of multiple AMRs [12]–[14] with a limited number of stations. *To the best of our knowledge, no existing solutions have proposed to study how to jointly optimize the task allocation and the charging time of multiple AMRs with the target of minimizing the tasks downtime and ensuring high-quality battery life.*

III. TCM Problem Formulation

A. TCM Problem Overview

For simplicity, in this paper we consider a fleet of homogeneous AMRs. We assume that the end user only needs to define a set of tasks to execute on the AMR fleet over a desired working period (e.g., several hours). Specifically, the task set is composed of *navigation tasks* and *objective tasks*. Each navigation task is designed to follow a path in the user environment and to concurrently execute a set of objective tasks such as detection of hazardous conditions, safety breaches, and so on. Each objective task is characterized by (1) a certain amount of computational demand, (2) the access to a certain number of sensors, and (3) a priority level. Thus, each navigation task, due to the different path and objective tasks, leads to a certain amount of locomotion, computing, and sensing energy consumption. In addition, the AMRs can recharge to any of the charging stations available.

In order to coordinate AMRs, the long working period is divided into several short decision periods (e.g., 10 minutes each). Thus, the TCM problem becomes a *multi-period planning problem*. However, coordinating multiple AMRs for task allocation and recharge requires planning based on mathematical energy consumption models of the locomotion, computing, and sensing hardware. These models may be affected by modeling errors due to approximations and simplifications. Accumulation of such errors over multiple periods can lead to poor performance. TCM addresses this problem by making planning decisions over multiple periods but it only communicates to the AMRs the planning decisions for the current period. Then, at the beginning of each decision period, each AMR reports the current battery SOC and, accordingly, TCM adjusts its decisions for the remaining periods. This strategy, as we will show in Section V-C, makes TCM highly robust to modeling errors, thus minimizing the risk for unexpected task downtime and battery life quality degradation.

B. TCM Problem Cost Function.

The TCM problem has two main objectives, i.e., minimize task *downtime* and battery quality of life *degradation*:

$$\min_{X,Z} Downtime(X) + q * Degradation(X,Z) \tag{1}$$

where X and Z are the 4D and 3D arrays of task allocation and charge schedule decision variables, respectively. $x_{k,i,h,j}$ is a binary element of X that is equal to 1 if in period k the AMR i is assigned navigation task h and objective task j, 0 otherwise. $z_{k,i,c}$ is a binary element of Z that is equal to 1 if in period k the AMR i is assigned to charging station c, 0 otherwise. The weight q allows the end user to regulate the importance of task downtime over battery life quality.

Downtime Cost. The downtime cost is defined as follows:

$$Downtime(X) = \sum_{k\in\mathcal{T}} \sum_{h\in\mathcal{H}} \sum_{j\in\mathcal{J}} \left(\gamma_{hj} - \sum_{i\in\mathcal{R}} x_{k,i,h,j} \right) p_j \tag{2}$$

where $\mathcal{T}$, $\mathcal{R}$, $\mathcal{H}$, and $\mathcal{J}$ are the set of decision periods, AMRs, navigation tasks, and objective tasks, respectively. $\gamma_{h,j}$ is a configurable binary element of the matrix Γ that is equal to 1 if the objective task j should be executed during navigation task h, 0 otherwise (note that different navigation tasks may execute the same objective tasks). $p_j \in [0.1, 1]$ is a fractional weight indicating the priority of the objective task j.

Quality of Life Degradation Cost. In general, depending on the specific battery type, the battery lifespan is highly affected by the depth of discharge (DoD) and the maximum energy stored in the battery at any time point [18]. Thus, we assume that the user can leverage existing battery lifespan tools (e.g., [39]) to define a minimum and a maximum threshold of energy that allow the batteries to have a desired lifespan (e.g., 8 years). However, although the lifespan of some batteries is not affected by frequent charges, AMRs can waste a considerable amount of energy to frequently navigate to a charging station. Thus, both desired lifespan and minimal energy waste (i.e., high-quality life) can be obtained by scheduling charging cycles to *start* when the energy level of the AMR is close to the minimum energy threshold (i.e., E^{DOD}) and *stop* close to the selected maximum energy threshold (i.e., E^{max}). The degradation cost can thus be defined as follows:

$$Degradation(X, Z) = \sum_{k \in \mathcal{T}} \sum_{i \in \mathcal{R}} \sum_{c \in \mathcal{C}} D^{start}_{k,i,c} + D^{stop}_{k,i,c} \tag{3}$$

$$\begin{cases} D^{start}_{k,i,c} & = (1 - z_{k-1,i,c})\, z_{k,i,c} \frac{\left|e_{k-1,i} - E^{DOD}\right|}{E^{bat}} \\ D^{stop}_{k,i,c} & = z_{k-1,i,c}\, (1 - z_{k,i,c}) \frac{\left|E^{max} - e_{k-1,i}\right|}{E^{bat}} \end{cases} \tag{4}$$

where $\mathcal{C}$ is the set of charging stations, $e_{k-1,i}$ is an element of the energy array E that indicates the energy level of AMR i at the end of the $k-1$ decision period, and E^{bat} is the maximum energy level that can be stored in the AMR batteries. The start/stop of charging is determined in Equation 4 when the decision variable z for an AMR changes from 0 to 1 (start charging) or from 1 to 0 (stop charging) over two consecutive decision periods (i.e., $k - 1$ vs. k). Thus, the quality of life degradation cost is calculated based on the normalized absolute difference between the AMR energy and the two thresholds when the AMR starts/stops charging.

C. TCM Problem Constraints

The TCM problem includes six types of constraints.

(1) Limited resource capacity. Similar to our previous work [40], the total number of instructions to execute the assigned navigation and objective tasks during each decision period must not exceed the computing capacity of each AMR:

$$\sum_{h \in \mathcal{H}} \left[n_{k,i,h} \cdot r_h + \sum_{j \in \mathcal{J}} x_{k,i,h,j} \cdot r_j \right] \leq M^{max} t \quad \forall\, k, i \tag{5}$$

where $n_{k,i,h} = \max \left[x_{k,i,h,1} \;\; \dots \;\; x_{k,i,h,|\mathcal{J}|} \right]$ indicates which navigation task is allocated to AMR i in period k and is equal to 1 only if at least one of the objective tasks j of navigation task h is assigned to AMR i, i.e., allocating a navigation task without any objective task would not make sense. r_h and r_j are the total number of instructions executed for navigation task h and objective task j during each decision period, respectively. M^{max} is the maximum Instructions Per Second (IPS) for the computing resource of each AMR at highest clock frequency and t is the duration of each decision period. The average number of instructions of the tasks as well as the AMR's maximum IPS can be easily profiled in most computing systems using performance event counters [41].

(2) Energy availability. To plan the task allocation and charge schedule over multiple decision periods we need to have a mathematical model that can estimate the energy available in the battery of each AMR i at the end of each decision period k, i.e., $e_{k,i}$ in Equation 4. Naturally, the energy available is between 0 and the maximum battery capacity E^{bat}:

$$0 \leq e_{k,i} \leq E^{bat} \tag{6}$$

As we have demonstrated in our previous work [34], the power consumption of AMRs are mostly due to computing, sensors, and locomotion mechanism. Thus, $e_{k,i}$ can be estimated as:

$$e_{k,i} = e_{k-1,i} + e^{charged}_{k,i} - e^{comp}_{k,i} - e^{sens}_{k,i} - e^{loc}_{k,i} - e^{change}_{k,i} \tag{7}$$

where, $e_{k-1,i}$ is the energy level at the end of the last decision period, $e^{charged}_{k,i}$ is the amount of energy recharged in AMR i if it is recharged during period k, $e^{comp}_{k,i}$ is the computational energy consumption, $e^{sens}_{k,i}$ is the sensor data acquisition energy consumption, $e^{loc}_{k,i}$ is the locomotion energy consumption, and $e^{change}_{k,i}$ is the amount of energy spent when the AMR changes navigation task, goes to recharge, or goes back to execute tasks after recharge across two consecutive decision periods. Specifically:

- The energy charged in AMR i at any of the charging stations c in period k is simply calculated based on the charging rate ν and decision period length t when the AMR is scheduled for recharge, i.e., $z_{k,i,c} = 1$:

$$e^{charged}_{k,i} = \sum_{c \in \mathcal{C}} z_{k,i,c} \cdot \nu \cdot t \tag{8}$$

- The computational energy consumption can be estimated using well known models [40], [42] based on the task execution time and the third power of clock frequency, i.e., $\alpha^{comp} * Exec\_time * f^3$, where α^{comp} is an estimated parameter and f is the clock frequency. The execution time of a set of tasks, similar to [40], can be easily estimated based on the ratio of the total number of instructions to execute for each allocated task and the maximum CPU's instructions per second M^{max}. Thus:

$$e^{comp}_{k,i} = \alpha^{comp} \frac{\sum_{h \in \mathcal{H}} \left[n_{k,i,h} \cdot r_h + \sum_{j \in \mathcal{J}} x_{k,i,h,j} \cdot r_j \right]}{M^{max}} f^3 \tag{9}$$

  We conservatively assume AMRs always operate at the highest clock frequency to maximize task performance.
- Each AMR has a set $\mathcal{S}$ of sensors (e.g., cameras and lidar). In our previous work [34] we have observed that it takes a non-negligible amount of energy to actually acquire sensor data. Thus, given an average access rate of each navigation and objective tasks to each sensor, we estimate the total sensing energy consumption as follows:

$$e^{sens}_{k,i} = \sum_{l \in \mathcal{S}} \alpha^{sens}_l \sum_{h \in \mathcal{H}} \left[n_{k,i,h} \cdot S_{h,l} + \sum_{j \in \mathcal{J}} S_{j,l} \cdot x_{k,i,h,j} \right] \tag{10}$$

  where α^{sens}_l is an estimated parameter that measures the average energy consumption per data acquisition from sensor $l \in \mathcal{S}$. $S_{h,l}$ and $S_{j,l}$ are the average number of sensor data acquired by each navigation and objective task during the decision period, respectively. Thus, α^{sens}_l is multiplied by the total sensor data accessed by the allocated tasks during each decision period.
- The locomotion energy $e^{loc}_{k,i}$ depends on the assigned navigation path. Given that each path is different in terms

of curves, slopes, and obstacles each navigation task h is characterized by a measurable average locomotion energy consumption α_h^{loc} per period:

$$e_{k,i}^{loc} = \sum_{h \in \mathcal{H}} \alpha_h^{loc} \cdot n_{k,i,h} \tag{11}$$

- Ideally, we would account for the exact AMR energy consumption due to changes of navigation task and travel to/from each specific charging station across consecutive periods, which is wasted since no useful objective task is being executed. However, doing so would highly increase the problem complexity. Thus, we use the following simplified model that considers average distances between navigation paths and charging stations:

$$e_{k,i}^{change} = \alpha^{change} \left[d^{charge} \cdot f(z_{k,i,c}) + d^{n2n} \cdot f(n_{k,i,c}) \right] \tag{12}$$

  where α^{change} is the average energy per meter required to navigate the AMR during changes in navigation and charge assignment. We assume each AMR uses a default navigation system for such cases that can be profiled offline. d^{charge} and d^{n2n} are the average distance of the navigation paths's centroids to all charging stations and the average distance across the centroids of different navigation paths, respectively. $f(z_{k,i,c})$ and $f(n_{k,i,c})$ are two logic functions designed to be equal to 1 when there is a change in charging status and navigation allocation, respectively, 0 otherwise.

All the parameters α^{comp}, α_l^{sens}, α_h^{loc}, and α^{change} can be estimated on each AMR using low-power energy sensors on each component. The measured energy traces can be periodically uploaded to the base station for periodic automatic re-training, increased accuracy, and ease of modeling for the user. Nonetheless, we handle the remaining energy modeling errors, e.g., due to simplified Equation 12 model, by adapting task allocation and charge schedule decisions at every decision period based on measured energy readings from each AMR.

(3) Charging stations availability. The number of AMRs charging is no more than the number of charging stations $|\mathcal{C}|$:

$$\sum_{i \in \mathcal{R}} \sum_{c \in \mathcal{C}} z_{k,i,c} \le |\mathcal{C}| \quad \forall k \tag{13}$$

(4) State consistency. Each AMR in the fleet can be in three different states: *charging*, *executing*, and *waiting*. In the charging state the AMR is charging at one of the charging stations and cannot execute any tasks. In the executing state the AMR is executing one of the navigation tasks and at least one of the objective tasks. An AMR that is neither charging nor executing any tasks is in the waiting state. As a result:

$$\sum_{c \in \mathcal{C}} z_{k,i,c} + \sum_{h \in \mathcal{H}} n_{k,i,h} \le 1 \ \forall \, k, i \tag{14}$$

(5) Task activity status. The end-user can indicate, for each navigation task in $\mathcal{H}$, which objective tasks in $\mathcal{J}$ to execute by configuring the corresponding elements $\gamma_{h,j}$ of the binary matrix Γ. Each objective task j associated to navigation $h \in \mathcal{H}$

Algorithm 1 TCM_Framework

```
Input: Set of AMRs $\mathcal{R}$, number of decision periods $T$, set of
    navigation tasks $\mathcal{H}$, set of objective tasks $\mathcal{J}$, matrix of navigation
    to objective task assignment $\Gamma$, and set of charging stations $\mathcal{C}$
 1: $\mathcal{T} \leftarrow \{1, 2, ..., T\}$                        ▷ Set of decision periods
 2: $X \leftarrow 0$                ▷ Allocation decision (element $x_{k,i,h,j}$)
 3: $Z \leftarrow 0$                   ▷ Charging decision (element $z_{k,i,c}$)
 4: $E \leftarrow 0$              ▷ Energy level (element $e_{k,i}$, column $e_k$)
 5: $A \leftarrow 1$              ▷ $a_{k,i} = 1$ if AMR available for allocation
 6: $W \leftarrow 0$        ▷ $w_{k,i} = 1$ if AMR waiting for charge scheduling
 7: while $\mathcal{T} \neq \emptyset$ do
 8:     $k^{min} \leftarrow$ Minimum value in $\mathcal{T}$
 9:     $e_{k^{min}-1} \leftarrow$ Read_Energy()       ▷ Read energy from AMRs
10:     TCM()
11:     Transmit($x_{k^{min}}$, $z_{k^{min}}$) ▷ Broadcast $k^{min}$ period decisions
12:     $\mathcal{T} \leftarrow \mathcal{T} \setminus \{k^{min}\}$
13:     Wait()    ▷ Wait for the end of the previous decision period
```

should be allocated to an AMR only if $\gamma_{h,j} = 1$:

$$\sum_{i \in \mathcal{R}} x_{k,i,h,j} \le \gamma_{hj} \quad \forall \, k, h, j \tag{15}$$

(6) Integrity of the decision variables. The elements of decision arrays X and Z can only have binary values:

$$\begin{aligned} x_{k,i,h,j} &= \{0,1\} \quad \forall k, i, h, j \\ z_{k,i,c} &= \{0,1\} \quad \forall k, i, c \end{aligned} \tag{16}$$

D. TCM Problem Implementation: MINLP

The TCM problem defined in Sections III-B and III-C is a Mixed Integer Nonlinear Program (MINLP) due to the integer constraint given by Equation 16 and to the non-linearities in Equation 3. Finding an optimal solution in such problems generally requires exponential time complexity [20]. We implement the TCM problem in Gurobi solver to find a near-optimal solution. However, solvers can still take a long time for some MINLP problem instances. Thus, we design a greedy algorithm that finds a solution in polynomial time.

IV. TCM Framework

In this section, we describe the details of the TCM framework, which periodically fetches the AMR's SOC to adjust the task allocation and charge scheduling decisions using the multi-period polynomial-time TCM algorithm.

The pseudo-code of the proposed TCM_Framework is given in Algorithm 1. The inputs are the set of AMRs $\mathcal{R}$, the total number of decision periods T, the set of navigation tasks $\mathcal{H}$, the set of objective tasks $\mathcal{J}$, the matrix Γ with elements $\gamma_{h,j}$ indicating which objective tasks j to execute during each navigation task h, and the set of charging stations $\mathcal{C}$. First, in Lines 1-6, the framework initializes some important variables such as the set of decision periods $\mathcal{T}$, the task allocation array X with $x_{k,i,h,j}$ variables as elements initialized to 0, the charge scheduling array Z with $z_{k,i,c}$ variables as elements initialized to 0, the matrix of AMR energy level E with elements $e_{k,i}$ all initialized to 0, the matrix A with elements $a_{k,i}$ initialized to 1 (i.e., available) to keep track of each AMR i availability for task allocation during each period k, and the matrix W with elements $w_{k,i}$

Algorithm 2 TCM

```
1: For each AMR charging in $k^{min}-1$ or scheduled for recharge
   in $k^{min}$, set $a_{k,i} = 0\ \forall k \in \mathcal{T}$ and set $w_{k,i} = 1$ from the
   most recent period $k \leq k^{min}$ when the Task_Scheduler set
   this AMR for recharge to the end of the working period. For the
   remaining AMRs, set $a_{k,i} = 1$ and $w_{k,i} = 0\ \forall k \in \mathcal{T}$.
2: Set to 0 the elements of $Z$, $X$, $E$ for all periods in $\mathcal{T}$
3: $\Delta_i \leftarrow 0\ \forall i \in \mathcal{R}$
4: $stop \leftarrow False$
5: Charge_Scheduler()
6: while $stop \neq True$ do
7:     $DT_h \leftarrow$ Downtime for navigation tasks $h \in \mathcal{H}$
8:     Sort downtime cost contribution in non-increasing order
       of $DT_h$. Let $DT_{\psi(1)}, DT_{\psi(2)}, ..., DT_{\psi(|\mathcal{H}|)}$ be the order.
9:     if $\sum_{h\in\mathcal{H}} DT_h > 0$ then
10:        for $h = 1, ..., |\mathcal{H}|$ do
11:            Task_Scheduler($\psi(h)$)
12:        if $\sum_{i\in\mathcal{R}} \Delta_i == |\mathcal{R}|$ then
13:            $stop \leftarrow True$
14:    else
15:        $stop \leftarrow True$
```

initialized to 0 (i.e., not waiting) to keep track of AMRs waiting to complete recharge.

The algorithm then executes Lines 7-13 at the beginning of each decision period of the working period. Specifically, the framework reads the energy levels of AMRs at the end of the last decision period $k^{min}-1$ and updates the column $e_{k^{min}-1}$ (Lines 8-9). Then, it calls the TCM algorithm, which we will explain in the next sections, to find a multi-period charging and task allocation solution (Line 10). Finally, it transmits the solution for the current period $x_{k^{min}}$ and $z_{k^{min}}$ to the AMRs, removes the current period from the set $\mathcal{T}$ and waits for the beginning of the next decision period (Lines 11-13).

A. *Multi-period Polynomial-time TCM Algorithm*

At every invocation, the TCM algorithm first coordinates the charging schedule and then allocates tasks to the AMRs over multiple periods. Algorithm 2 shows the pseudo-code of the TCM algorithm. Modeling errors in the energy models may lead each AMR to be at an energy level different from the one estimated by the previous TCM invocation. Thus, in Lines 1-4, the TCM algorithm resets some key variables as follows. For AMRs that where charging in the last period $k^{min}-1$ or scheduled to recharge starting the current period k^{min}, TCM assumes they are not yet available for allocation and are still waiting for complete their recharge, i.e., $a_{k,i} = 0$ for all remaining periods and $w_{k,i} = 1$ from the first period $k \leq k^{min}$ the AMR was set to recharge to the end of the working period, respectively. The other AMRs are assumed to be ready for allocation, i.e., $a_{k,i} = 1$, and not waiting for recharge, i.e., $w_{k,i} = 0$, from the current period to the end of the working period (Line 1). All the other variables (e.g., Z, X, E) are set to 0 for the remaining periods (Line 2). Finally, it sets the variables Δ_i to 0 and $stop$ to $false$, which indicate when each AMR cannot be allocated any more tasks and when the allocation loop in Lines 6-15 must be interrupted, respectively.

Algorithm 3 Charge_Scheduler

```
1: Sort AMRs in non-increasing order of $\sum_{k\in 1}^{T} w_{k,i}$.
   Let $\sigma(1), ..., \sigma(|\mathcal{R}|)$ be the order.
2: for $i = 1, ..., |\mathcal{R}|$ do
3:     for $c \in \mathcal{C}$ do
4:         for $k \in \mathcal{T}$ do
5:             if AMR $\sigma(i)$ is waiting to schedule recharge then
6:                 if station $c$ is available for AMR $\sigma(i)$ then
7:                     if $Charge\_Cost\_Comp(k, \sigma(i))$ then
8:                         Set $z_{k,\sigma(i),c} = 1$ for AMR$_{\sigma(i)}$
9:                         Update $e_{k,\sigma(i)}$ using Equation 7
10:                    else
11:                        Set $a_{\hat{k},\sigma(i)} = 1\ \forall \hat{k} \in [k, |\mathcal{T}|]$
12:                        Set $w_{\hat{k},\sigma(i)} = 0\ \forall \hat{k} \in [1, T]$
13:                        Jump to Line 2
14:                else
15:                    if $\exists$ another charging station available then
16:                        Jump to Line 3
```

In order to properly coordinate all AMRs for task allocation, TCM first determines the charging schedule for the AMRs waiting to complete a recharge by executing the Charge_Scheduler algorithm (Line 5), which will be described in details in the next section. When the Charge_Scheduler terminates, all AMRs are set to be available for task allocation starting a specific period in the remaining working period. Then, TCM coordinates task allocation across AMRs (Lines 6-15). Specifically, it calculates the current contribution of each navigation task h to the downtime cost DT_h using Equation 2. Then, it sorts the navigation tasks in non-increasing order of downtime contribution (Lines 7-8). If there are tasks to be allocated (Line 9), the Task_Scheduler algorithm is executed to allocate the objective tasks of each navigation task h, starting from the one that has the highest contribution to the downtime cost (Lines 10-11). The TCM algorithm stops when all AMRs cannot be allocated anymore tasks (Lines 12-13) or when there are no more tasks to be allocated (Lines 14-15).

B. *Charge Scheduler Algorithm*

The pseudo-code of Charge_Scheduler is given in Algorithm 3. It executes to coordinate the charge schedule of the waiting AMRs, i.e., $w_{k^{min},i} = 1$, with the target of making them available for task allocation at some point in the future.

The algorithm first sorts the AMRs in non-increasing order of cumulative wait time (Line 1) to prioritize the AMRs waiting the longest time to complete a recharge. Then, starting from the AMR with the highest charging priority, it finds an available charging station. If the selected AMR was already charging before k^{min}, the scheduler selects the same charging station. Otherwise, it finds the first period $k \in \mathcal{T}$ when a charging station is available (Lines 3-6 and 14-16). The core of the scheduler is in Lines 7-13. When a charging station is selected for an AMR, the scheduler uses the *Charge_Cost_Comp()* function in Line 7 to compare what would be the total cost calculated using Equation 1 if (a) the AMR would be recharged vs (b) the AMR would be made available for task allocation. For simplicity, this comparison

Algorithm 4 Task_Scheduler

```
Input: selected navigation task $h$
1: for $i \in \mathcal{R}$ do
2:     for $k \in \mathcal{T}$ do
3:         if $a_{k,i} = 1$ then
4:             $\mathcal{O} \leftarrow Allocation\_Cost\_Comp(k, i, h)$
5:             if $\mathcal{O} \neq \emptyset$ then
6:                 Allocate temporary variables $\bar{x}_{k,i,h,j} = 1$,
                   $\bar{a}_{k,i} = 0$ for all $j \in \mathcal{O}$
7:             else
8:                 Set $\bar{a}_{\hat{k},i} = 0\ \forall \hat{k} \in \mathcal{T}$
9:                 Set $\bar{w}_{\hat{k},i} = 1\ \forall \hat{k} \in [k, |\mathcal{T}|]$
10:                Jump to Line 11
11:    $\tilde{k} \leftarrow \text{argmax}_{k \in \mathcal{T}}\, e_{k,i}$
12:    $Slack_i \leftarrow \frac{1}{e_{\tilde{k},i}} \sum_{k \in \mathcal{T}} a_{k,i}$
13:    $Q_i \leftarrow f(\bar{x}, z)$ potential number of periods AMR $i$ must wait
       for an available charging station
14:    $U \leftarrow f(a, x)$ potential number of periods with downtime if
       navigation $h$ would be allocated to other available AMRs
15:    if $U > Q_i$ then
16:        $Q_i \leftarrow 0$    ▷ leaving task $h$ to other AMRs may lead to
                                  higher downtime despite some queue at
                                  charging stations.
17: Sort AMRs in non-increasing order of Slack, then sort them in
    non-decreasing order of $Q$. Let $\sigma(1), ..., \sigma(|\mathcal{R}|)$ be the order.
18: $DT_h^{temp} \leftarrow$ Current downtime of navigation task $h$
19: for $i = 1, ..., |\mathcal{R}|$ do
20:     Assign the temporary decision made in Lines 1-10 for
        AMR $\sigma(i)$ to $X$, $A$, $W$
21:     $\Delta_{\sigma(i)} \leftarrow 1$
22:     $DT_h \leftarrow$ New downtime of navigation task $h$
23:     if $DT_h^{temp} \neq DT_h$ then
24:         break
```

considers only the selected period k and the unallocated tasks of the navigation with the highest downtime contribution. If it is more convenient to recharge, the function returns a *true* value (Line 7), the AMR is set for recharge (Line 8), and the AMRs' energy level at the end of period k is updated according to Equations 7 to 12 (Line 9).

The algorithm then continues by evaluating the next decision periods until *Charge_Cost_Comp()* returns *false*, i.e., it is more convenient to allocate tasks. At this point, the charge scheduler sets this AMR available for task allocation in the remaining periods, stops the recharge (Lines 11-12), selects the next AMR in order of priority, and repeats the above steps.

C. Task Scheduler Algorithm

The pseudo-code of the Task_Scheduler is given in Algorithm 4. It takes as input the navigation task h selected in Lines 7-11 of Algorithm 2. The Task_Scheduler finds not only which AMR should be allocated the objective tasks of h, but also for how many decision periods this AMR should execute those tasks for minimized task downtime and maximized battery life quality. Specifically, similar to the *Charge_Cost_Comp()* function of Algorithm 3, we define an *Allocation_Cost_Comp()* function that compares what would be the total cost of Equation 1 for the considered period if (a) the AMR would be allocated objective tasks in order of priority p_j vs (b) the AMR would be recharged. Note, this function considers the energy wasted to travel the average distance d^{charge} from/to the charging station through Equations 7 and 12. For simplicity, this function only looks at cost one decision period and one AMR at a time, without evaluating the potential downtime caused by queuing at charging stations.

In order to minimize downtime and queuing at charging stations, the Task_Scheduler first temporarily allocates the objective tasks to each AMR (Lines 1-10). Specifically, for each available AMR i and starting from the current period (Lines 2-3), the algorithm uses the function *Allocation_Cost_Comp()* (Line 4) to decide the temporary allocation of the unallocated objective tasks $j \in \mathcal{J}$. If allocating tasks to this AMR is cost effective in this period, the function stores the indices of the unallocated objective tasks that can be assigned to AMR i in the set $\mathcal{O}$. This temporary allocation is stored by setting the temporary variables $\bar{x}_{k,i,h,j} = 1$ and $\bar{a}_{k,i} = 0$ for all $j \in \mathcal{O}$ (Lines 5-6). When it becomes more convenient to recharge the AMR than allocating tasks, the function returns an empty set $\mathcal{O}$ and the AMR is set for recharge (Lines 7-10).

The algorithm then needs to select the best AMR that minimizes downtime and queuing time by finalizing the best temporary allocation. To do so, it first sorts the AMRs in non-increasing order of *Slack*. Then, maintaining the slack order, it sorts the AMRs in non-decreasing order of queuing Q_i (Line 17). The *Slack* is calculated in Lines 11-12 for each AMR as the ratio of the total AMR availability over the energy $e_{\tilde{k},i}$, which is the energy level of AMR i in the first period $\tilde{k}$ when the AMR can be allocated tasks. Sorting in non-increasing order of *Slack* prioritizes the AMR with higher availability for task allocation but also with lower energy level. Doing so allows to maximize the probability that, when the low-energy AMR reaches the charging point, a higher-energy AMR has enough energy to take over the task execution without any downtime. The value Q_i is calculated in Line 13 as the number of potential periods AMR i waits to get recharged due to queuing at charging stations. The algorithm then calculates the variable U in Line 14 to estimate the potential number of periods with downtime if navigation h and its objective tasks would be allocated to other available AMRs. If allocating the tasks to other AMRs in the selected periods would lead to a higher number of periods with downtime than the queuing time introduced by AMR i (Line 15), Q_i is set to zero to prioritize AMR i for final allocation (Line 16).

To finalize the allocation decision the algorithm starts from the highest priority AMR, stores the current downtime contribution of task h (Line 18), and finalizes the temporary allocation of the selected AMR (Line 20). Then, it flags the AMR as unavailable for further allocations of other navigation tasks by setting its variable Δ_i to 1 (Line 21). If the new downtime contribution of task h is different from the one calculated in Line 18, i.e., tasks of h have been allocated to the AMR, then the algorithm breaks the loop and returns to Algorithm 2 (Lines 22-24). Otherwise, i.e., the selected AMR has been set for recharge without any allocation, it repeats Lines 19-24 for the next AMR in the sorted order.

D. TCM Complexity Analysis

We analyze TCM's computational complexity considering R as the number of AMRs, T as the number of decision periods, H as the number of navigation tasks, J as the number of objective tasks, C as the number of charging stations, and S as the number of sensors on each AMR. Algorithm 4 executes $O(R^2TJ)$ times to allocate the objective tasks of the selected navigation task. Algorithm 3 executes $O(RCTS)$ times for charge scheduling. Algorithm 2 executes $TR(HJ + C)$ times to re-initialize the fundamental variables and call Algorithm 3 once. It performs task allocation H times. Therefore, the total time complexity of TCM is $O(R^2THJ)$. Algorithm 1 calls Algorithm 2 T times. Therefore, the computation complexity of the TCM_Framework is $O(T^2R^2HJ)$.

V. Experimental analysis

A. Experimental Setup

We use our HydraOne autonomous mobile robot prototype [43] to train the parameters of the models described in Section III and ensure realistic results. It has an NVIDIA Jetson AGX, Dual 250W hub motors, 156Wh Li-ion battery pack, 2 Intel RealSense cameras, a Slamtec RPLidar S1 lidar, and an Arduino Mega 2560 to read the current sensors data.

We profiled the instruction count of a representative navigation task based on HydraNet[1] and a representative objective task for object detection based on MobileNet-SSD [34]. To account for the variability of instruction count across different tasks, we then generate problem instances using two uniform distributions that have mean values equal to the instruction count measured with MobileNet-SSD for objective tasks (e.g., $0.3 * 10^6$ measured, range $3 * 10^3$ to $0.5 * 10^6$) and HydraNet for navigation tasks (e.g., $0.12*10^6$ measured, range $0.01*10^4$ to $0.3 * 10^6$).

We generate different problem instances with 1-8 AMRs, 1-6 charging stations, 4-10hrs working period with 10min decision periods, 1-7 navigation tasks, and 5-7 objective tasks per navigation task. We repeat each problem instance multiple times with a distinct initial SOC and random objective task priorities p_j. The simulations are executed on an Intel Core i7-8750H CPU at 2.20GHz and 24GB memory.

Performance Metrics. *Task Allocation* (TA) is the percentage of allocated objective tasks in the working period, which is directly related to the downtime cost in Equation 1.

In order to measure the battery quality of life degradation, we define two metrics: *Energy-usage-EFfectiveness (EEF)* and *State of Charge Violation* (SOC$_V$). EEF measures the quality of battery utilization and is calculated as follows:

$$EEF = \frac{\sum\limits_{k\in\mathcal{T}} \sum\limits_{i\in\mathcal{R}} E^{useful}_{k,i} + E^{wasted}_{k,i}}{\sum\limits_{k\in\mathcal{T}} \sum\limits_{i\in\mathcal{R}} E^{useful}_{k,i}} \tag{17}$$

where $E^{useful}_{k,i}$ is the energy utilized to execute a given navigation and objective tasks by AMR i in time period k and $E^{wasted}_{k,i}$ is the energy utilized to go back/forth to the charging station. In line with the PUE definition for data centers efficiency [19], EEF is ≥ 1, with 1 being the best, i.e., utilizing 100% of battery charge to perform useful tasks. SOC$_V$ is the sum of violations of the battery SOC thresholds in % by an AMR during the battery charge cycle. A violation of these thresholds results in degrading the battery lifespan:

$$SOC_V = \frac{100}{E^{bat}} \sum_{k\in\mathcal{T}} \sum_{i\in\mathcal{R}} \big[max((e_{k,i} - E^{max}), 0) + max((E^{DOD} - e_{k,i}), 0)\big] \tag{18}$$

where $e_{k,i}$ is the energy level of AMR i in period k, E^{bat} is the max energy capacity, E^{max} and E^{DOD} are the maximum and minimum energy thresholds input by the user. In line with Lithium-ion batteries [18], lifespan degrades below the user desired duration when the energy level exceeds the indicated thresholds. Thus, accordingly, SOC$_V$ increases only when the energy level drops below E^{DOD} or increases above E^{max}. The closer SOC$_V$ is to zero, the closer the lifespan is to the desired one. Here we do not consider other factors such as temperature but we leave it as future work. Ideal battery quality of life is achieved with $EEF = 1$ and SOC$_V = 0$.

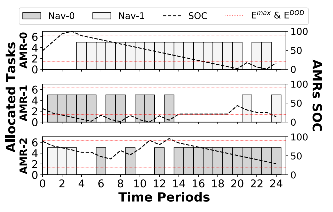
Baselines. We compare TCM with three representative baselines (1) Minimum Downtime (Min-DT), (2) Minimum Downtime-Constrained (Min-DTC), and (3) MINLP. Min-DT is a simple optimization model for task allocation and charging schedule that maximizes only TA. We implement this baseline in Gurobi solver by simply setting weight value $q = 0$ in the model proposed in Section III-A. Min-DTC is an optimization model that considers maximizing the task allocation while ensuring that the energy level remains within the minimum (E^{DOD}) and maximum (E^{max}) battery threshold. To implement this baseline we modify Min-DT by adding such constrain in Gurobi. (3) MINLP is the TCM problem implementation in Gurobi as described in Section III-D.

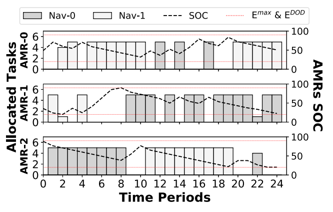
B. TCM Performance Results

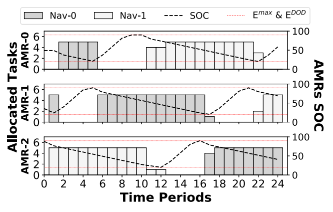
Here, we compare TCM with the baselines mainly using a high value of battery life quality weight q. Section V-B4 shows the results for other q values. Note, Min-DT and Min-DTC only minimize downtime and are not influenced by q.

1) Detailed Performance Comparison: Figure 1 shows the performance of the baselines and TCM on a problem instance with 3 AMRs, 2 navigation tasks each one with 5 objective tasks, 24 decision periods of 10min each, and 1 charging station. Figure 1a shows the SOC variation and task allocation of AMRs with Min-DT. It charges the AMRs to minimize downtime without considering the effect of frequent charging and energy threshold violations on the quality of battery life. Same as Min-DTC and MINLP, it provides 100% TA but has an EEF of 1.46, i.e., only 68% of the total energy drawn from the battery is useful. In addition, it has a high SOC$_V$ of 259%, which degrades the battery lifespan. For example, AMR-1 spends most of the time at an SOC below the minimum threshold and frequently travels to charging stations. Figure 1b shows the results for Min-DTC. It constrains the SOC between

[1]A Convolutional Neural Network (CNN) that gets as input a camera frame and outputs the next linear and angular speed of the AMR.


(a) Min-DT


(b) Min-DTC


(c) MINLP

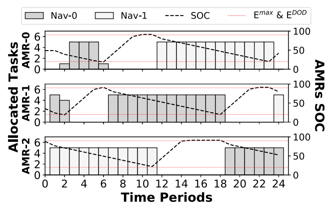
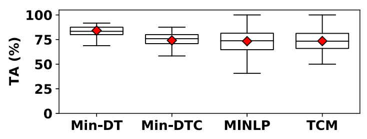
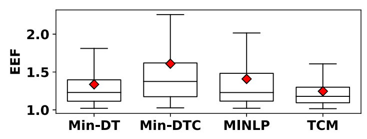

(d) TCM

Fig. 1. Example comparison between the baselines (a) Min-DT, (b) Min-DTC, (c) MINLP, and (d) our proposed TCM algorithm in terms of battery SOC and task allocation for a case study of 3 AMRs, 2 navigation tasks, 5 objective tasks, 1 charging station, and SOC threshold of 20% and 90% of battery capacity.


(a)


(b)

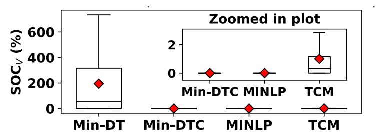
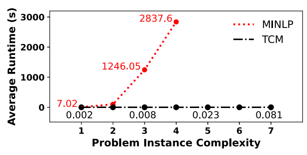

(c)

Fig. 2. (a) Task Allocation, (b) Energy Usage Effectiveness, and (c) per-run SoC threshold violation comparison for Min-DT, Min-DTC, MINLP, and TCM.


(a)

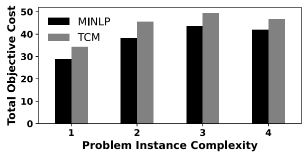

(b)

Fig. 3. MINLP vs. TCM: (a) Average runtime and (b) total objective cost for various problem instances.

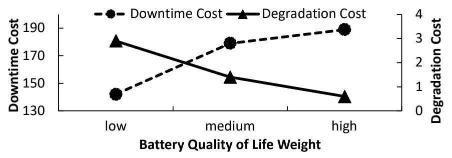


Fig. 4. Effect of the weight q in Equation 1 on *downtime* cost and battery quality of life *degradation* cost using the proposed TCM algorithm. Case study of 8 AMRs, 6 charging stations, 7 navigation tasks, each with 5 objective tasks.

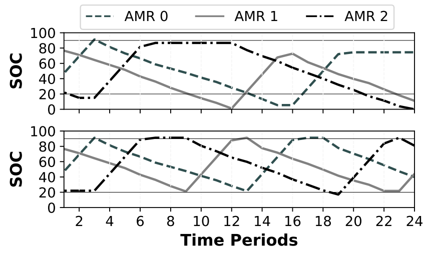
the thresholds to preserve lifespan, but has a slightly higher EEF of 1.5 due to the smaller SOC range, i.e., only 66% of the total energy is consumed on performing useful tasks. Figures 1c and 1d show the results of the proposed MINLP and TCM. Both perform task allocation based on the trade-off between task downtime and quality of battery life. Thus, they both use the available energy to preserve lifespan while starting/ending charging cycles near the SOC thresholds for improved EEF. MINLP and TCM result in 100% and 98.33% TA, respectively, and 1.19 and 1.16 EEF, respectively. This means that MINLP and TCM improve the EEF by 20.67% and 22.66% compared to Min-DTC, respectively, while ensuring similar TA performance.

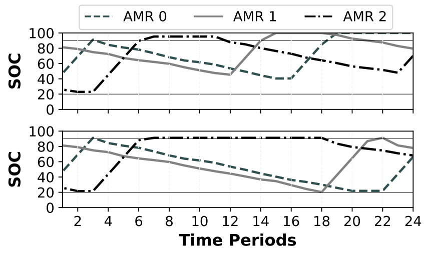
Compared to TCM, MINLP finds a slightly better solution in terms of task allocation at the cost of a slightly higher EEF. The main reason for TCM to have a 1.67% lower TA than MINLP is due to its greedy approach. For example, MINLP schedules a recharge for AMR-1 in period 1 before its battery SOC reaches E^{DOD}, as shown in Figure 1c, causing the higher EEF. However, this early recharge eliminates the wait time in period 5, therefore reducing task downtime. TCM greedily selects AMRs by sorting them first based on non-increasing order of $Slack$ and then in non-increasing order of queuing time Q (see Algorithm 4). Thus, it first allocates navigation task 0 to AMR-1 due to its higher *Slack*. For navigation task 1, selecting AMR-0, which has the second-highest $Slack$, would result in a downtime of two periods because of queuing at the charging station. Thus, TCM allocates navigation task 1 to AMR-2 and avoids queuing-related downtime, as shown in Figure 1d. The greedy approach of TCM leads sometimes to a small downtime such as in period 6, where it allocates only 1 out of 5 objective tasks to AMR-0. On the other hand, MINLP takes 91s to find the solution in each period, while TCM only takes 0.25s, i.e., 365 times faster than MINLP.

2) General Performance Comparison: Figure 2 shows the performance comparison of the baselines and TCM for all the problem instances described in Section V-A. Min-DT utilizes all the available battery capacity to schedule tasks giving an average TA of 84.12%, which is 13.52%, 14.73%, and 14.5% higher than Min-DTC, MINLP, and TCM, respectively. However, utilizing the available battery capacity increases the average SOC_V, which is 194% compared to 0% for Min-DTC and MINLP, and 1% for TCM, as shown in Figure 2c. Min-DT has also an average EEF of 1.34 as shown in Figure 2b, which is 16.77% lower than Min-DTC due to the larger energy range. For a similar reason, MIN-DT shows 4.28% lower EEF than MINLP, which, however, has a 0% SOC_V. TCM has the lowest EEF, which is 10.71% lower than MINLP, 13% lower than MIN-DT and 22.36% lower than MIN-DTC.

These results show that MINLP and TCM perform a balanced trade-off between TA, EEF, and SOC_V, thus ensuring high performance and high-quality battery life at same time.

3) MINLP vs. TCM Comparison Analysis: We study the performance of MINLP and TCM by comparing their execution time and total objective cost for 7 different problem instances ordered by complexity. As Figure 3a shows, when the problem complexity increases, the average execution time of MINLP increases exponentially reaching about 45min for problem instance 4 (3 AMRs, 2 charging stations, 2 navigation tasks, 5 objective tasks, and a working period of 4 hours). We do not show the results for instances 5-7 due to their extremely long execution times. Conversely, TCM takes polynomial time to solve all the problem instances. For example, it only takes 0.015s for problem instance 4 while causing only an average


(a) Under estimated error


(b) Over estimated error

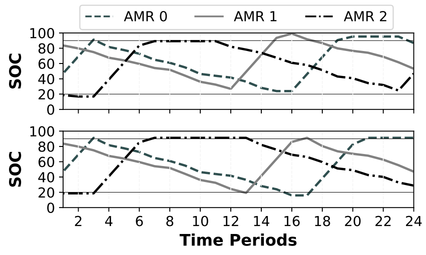
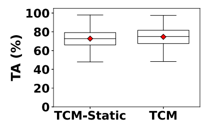
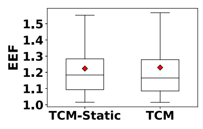

(c) Fluctuating error

Fig. 5. Robustness to energy usage estimation errors of TCM-Static (top figures) and TCM (bottom figures).


(a)


(b)

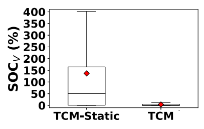

(c)

Fig. 6. (a) Task Allocation, (b) Energy Usage Effectiveness, and (c) per-run SoC threshold violation comparison for TCM-Static and TCM.

of 15% higher total objective cost than MINLP (Figure 3b), which translates in an empirical performance ratio of 1.15. These experiments prove the effectiveness of the designed TCM greedy algorithm.

4) Task Allocation vs. Battery Quality of Life: Here, we show the results for different values of weight q, which allows the end-user to regulate the importance of task downtime over battery life quality. For space reasons, we only show the results for a large problem instance of 8 AMRs, 6 charging stations, 7 navigation tasks, each with 5 objective tasks. As Figure 4 shows, for a low q value the downtime cost (Equation 2) is 142 with a high degradation cost of 2.9 (Equation 3). The downtime cost increases to 189 while the degradation cost decreases to 0.6 for a high q value, which is mainly due to a variation in SOC_V from 83% to 5%. These results show that TCM can flexibly provide solutions that give more or less importance to minimizing task downtime compared to preserving battery quality of life based on the weight q.

C. *TCM Robustness to Modeling Errors*

In this section, we evaluate the robustness of TCM to the uncertainties of the real-world environment by comparing the results with TCM-Static baseline, a version of TCM that only finds the solution for the entire working period once at the beginning of the working period without adapting to energy modeling errors. We measure the robustness of TCM using three general cases: *Under estimated error*, i.e., measured energy *higher* than estimated; *Over estimated error*, i.e., measured energy *lower* than estimated; *Fluctuating error*, i.e., measured energy fluctuates between underestimated and overestimated error across decision periods. We apply a random error in each period in the range [0, 60]% for the first two cases and [-60, 60]% for the third case. Figure 5 shows the SoC comparison of AMRs for a test case of 3 AMRs and 1 charging station in each of these three different case studies using TCM-Static (top figures) and TCM (bottom figures).

Even after planning the task and charging schedules considering both objectives, as the top figures show, the AMRs for TCM-Static violate the battery capacity thresholds providing weak performance under uncertainties. The bottom figures show the results of AMR SOC using TCM, which reads the actual energy level from each AMR at the end of every decision period and updates the decisions accordingly. This strategy limits the effect of the error only to one period, hence making TCM more robust to errors. In Figure 6, we compare the results of TCM-Static and TCM over various problem instances. The TA for TCM is 2.63% higher than TCM-Static because TCM-Static may occasionally lead to downtime due to uncertainties. For example, AMR-1 reaches 0% SOC in period 15 in the top Figure 5a, leading to unexpected downtime. This results in an average SOC_V of 136.66% for TCM-Static, which is 3422.16 times higher than TCM. The EEF has less than 1% difference between TCM and TCM-Static as TCM-Static follows the recharge schedule.

These results show that TCM is robust to energy modeling errors, which are inevitable in real-life deployments.

VI. Conclusions

In this paper, we have presented TCM, an energy-aware Task allocation and Charging schedule Manager for AMR fleets that can flexibly tradeoff between task downtime and battery quality of life according to end-user preferences. We have experimentally showed that it has a performance ratio of 1.15 at a fraction of the execution time compared to its MINLP implementation in Gurobi. We have also compared TCM with several representative baselines and showed that it is robust to energy modeling errors, achieves similar task allocation performance, and ensures up to 22.36% and 193% lower EEF and battery lifespan degradation, respectively.

The current TCM model is a centralized approach. In the future, we plan to implement a decentralized TCM model to handle a large swarm of AMRs performing tasks over a larger area with enroute charging stations.

Acknowledgement

We thank Dr. Murat Yildirim and his PhD student Farnaz Fallahi for their help in implementing MINLP in Gurobi.

References

[1] Knightscope, "Autonomous security," https://www.knightscope.com, 2019.
[2] SMP Robotics, "Robot guard," https://smprobotics.com/application_autonomus_mobile_robots/robot-guard/, 2019.
[3] Kelvin Chan, "Robots in the field: farms embracing autonomous technology," https://phys.org/news/2018-11-robots-field-farms-embracing-autonomous.html, 2019.
[4] Khasha Ghaffarzadeh, "Agricultural robots and drones 2018-2038: Technologies, markets and players," https://www.idtechex.com/en/research-report/agricultural-robots-and-drones-2018-2038-technologies-markets-and-players/578, 2018.

[5] W. Dai, H. Lu, J. Xiao, Z. Zeng, and Z. Zheng, "Multi-robot dynamic task allocation for exploration and destruction," *Journal of Intelligent & Robotic Systems*, vol. 98, no. 2, pp. 455 – 479, 2020. [Online]. Available: https://link.springer.com/article/10.1007/s10846-019-01081-3

[6] Aeolus Robotics, "Robot," https://aeolusbot.com/our-robots/, 2018.

[7] National Science Foundation, "Design for Sustainability in Computing," https://beta.nsf.gov/funding/opportunities/design-sustainability-computing, 2022.

[8] D.-H. Lee, "Resource-based task allocation for multi-robot systems," *Robotics and Autonomous Systems*, vol. 103, pp. 151 – 161, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092188901730310X

[9] D. Lee, S. A. Zaheer, and J. Kim, "A resource-oriented, decentralized auction algorithm for multirobot task allocation," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 4, pp. 1469–1481, Oct 2015.

[10] T. F. Roos and M. R. Emami, "A framework for autonomous heterogeneous robot teams," in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, Nov 2018, pp. 868–874.

[11] K. J. O'Hara, R. Nathuji, H. Raj, K. Schwan, and T. Balch, "Autopower: toward energy-aware software systems for distributed mobile robots," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, May 2006, pp. 2757–2762.

[12] M. Rappaport and C. Bettstetter, "Coordinated recharging of mobile robots during exploration," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 6809–6816.

[13] F. Michaud and E. Robichaud, "Sharing charging stations for long-term activity of autonomous robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, Sep. 2002, pp. 2746–2751 vol.3.

[14] F. Sempé, A. Muñoz, and A. Drogoul, "Autonomous robots sharing a charging station with no communication: a case study," in *Distributed Autonomous Robotic Systems 5*, H. Asama, T. Arai, T. Fukuda, and T. Hasegawa, Eds. Tokyo: Springer Japan, 2002, pp. 91–100.

[15] L. Liu, H. Sun, C. Li, T. Li, J. Xin, and N. Zheng, "Managing battery aging for high energy availability in green datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3521–3536, 2017.

[16] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar, "Benefits and limitations of tapping into stored energy for datacenters," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, 2011, pp. 341–351.

[17] J. Ma, S. Shi, S. Gu, N. Zhang, and X. Gu, "Age-optimal mobile elements scheduling for recharging and data collection in green iot," *IEEE Access*, vol. 8, pp. 81 765–81 775, 2020.

[18] Battery University, "How to prolong lithium-based batteries," https://batteryuniversity.com/learn/article/how_to_prolong_lithium_based_batteries, 2019.

[19] L. A. Barroso, J. Clidaras, and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition*. Morgan & Claypool Publishers, 2013. [Online]. Available: http://dx.doi.org/10.2200/S00516ED2V01Y201306CAC024

[20] Y. Pochet and L. A. Wolsey, *Production planning by mixed integer programming*. Springer Science & Business Media, 2006.

[21] M. Elango, S. Nachiappan, and M. K. Tiwari, "Balancing task allocation in multi-robot systems using k-means clustering and auction based mechanisms," *Expert Systems with Applications*, vol. 38, no. 6, pp. 6486 – 6491, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417410013357

[22] A. Jevtic, A. Gutierrez, D. Andina, and M. Jamshidi, "Distributed bees algorithm for task allocation in swarm of robots," *IEEE Systems Journal*, vol. 6, no. 2, pp. 296–304, June 2012.

[23] M. J. Matarić, G. S. Sukhatme, and E. H. Østergaard, "Multi-robot task allocation in uncertain environments," *Autonomous Robots*, vol. 14, no. 2, pp. 255–263, Mar 2003. [Online]. Available: https://doi.org/10.1023/A:1022291921717

[24] L. Vacariu, B. P. Csaba, I. A. Letia, G. Fodor, and O. Cret, "A multiagent cooperative mobile robotics approach for search and rescue missions," *IFAC Proceedings Volumes*, vol. 37, no. 8, pp. 962 – 967, 2004, iFAC/EURON Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal, 5-7 July 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667017321055

[25] G. Ferri, J. Bates, P. Stinco, A. Tesei, and K. LePage, "Autonomous underwater surveillance networks: A task allocation framework to manage cooperation," in *2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO)*, May 2018, pp. 1–10.

[26] S. Dogru and L. Marques, "Energy efficient coverage path planning for autonomous mobile robots on 3d terrain," in *2015 IEEE International Conference on Autonomous Robot Systems and Competitions*, April 2015, pp. 118–123.

[27] Yongguo Mei, Yung-Hsiang Lu, Y. C. Hu, and C. S. G. Lee, "Energy-efficient motion planning for mobile robots," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 5, April 2004, pp. 4344–4349 Vol.5.

[28] J. A. Broderick, D. M. Tilbury, and E. M. Atkins, "Optimal coverage trajectories for a ugv with tradeoffs for energy and time," *Autonomous Robots*, vol. 36, no. 3, pp. 257–271, Mar 2014. [Online]. Available: https://doi.org/10.1007/s10514-013-9348-x

[29] C. Henkel, A. Bubeck, and W. Xu, "Energy efficient dynamic window approach for local path planning in mobile service robotics**this work was conducted at the university of auckland, auckland, new zealand," *IFAC-PapersOnLine*, vol. 49, no. 15, pp. 32 – 37, 2016, 9th IFAC Symposium on Intelligent Autonomous Vehicles IAV 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405896316308813

[30] A. Amory, T. Tosik, and E. Maehle, "A load balancing behavior for underwater robot swarms to increase mission time and fault tolerance," in *2014 IEEE International Parallel Distributed Processing Symposium Workshops*, 2014, pp. 1306–1313.

[31] S. Yu and C. S. G. Lee, "Lifetime maximization in mobile sensor networks with energy harvesting," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 5911–5916.

[32] C. Anagnostopoulos, S. Hadjiefthymiades, and K. Kolomvatsos, "Accurate, dynamic, and distributed localization of phenomena for mobile sensor networks," *ACM Trans. Sen. Netw.*, vol. 12, no. 2, pp. 9:1–9:59, Apr. 2016. [Online]. Available: http://doi.acm.org/10.1145/2882966

[33] Y. Wang, W. Peng, and Y. Tseng, "Energy-balanced dispatch of mobile sensors in a hybrid wireless sensor network," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 12, pp. 1836–1850, Dec 2010.

[34] L. Liu, J. Chen, M. Brocanelli, and W. Shi, "E2M: An energy-efficient middleware for computer vision applications on autonomous mobile robots," in *The 4th ACM/IEEE Symposium on Edge Computing (Accepted in SEC 2019)*, 2019.

[35] V. Berenz and K. Suzuki, "Risk and gain battery management for self-docking mobile robots," in *2011 IEEE International Conference on Robotics and Biomimetics*, Dec 2011, pp. 1766–1771.

[36] F. de Lucca Siqueira, P. Della Mea Plentz, and E. R. De Pieri, "A fuzzy approach to the autonomous recharging problem for mobile robots," in *12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, Aug 2016, pp. 1065–1070.

[37] B. Kannan, V. Marmol, J. Bourne, and M. B. Dias, "The autonomous recharging problem: Formulation and a market-based solution," in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 3503–3510.

[38] M. Tomy, B. Lacerda, N. Hawes, and J. L. Wyatt, "Battery charge scheduling in long-life autonomous mobile robots," in *2019 European Conference on Mobile Robots (ECMR)*, 2019, pp. 1–6.

[39] Argonne National Laboratory, "Battery Life Estimator," https://www.anl.gov/partnerships/battery-life-estimator, 2021.

[40] T. Bahreini, M. Brocanelli, and D. Grosu, "Vecman: A framework for energy-aware resource management in vehicular edge computing systems," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[41] M. Brocanelli and X. Wang, "Hang doctor: Runtime detection and diagnosis of soft hangs for smartphone apps," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18. New York, NY, USA: ACM, 2018, pp. 6:1–6:15. [Online]. Available: http://doi.acm.org/10.1145/3190508.3190525

[42] S. Wang, Z. Qian, J. Yuan, and I. You, "A dvfs based energy-efficient tasks scheduling in a data center," *IEEE Access*, vol. 5, pp. 13 090–13 102, 2017.

[43] Y. Wang, L. Liu, X. Zhang, and W. Shi, "Hydraone: An indoor experimental research and education platform for cavs," in *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.