A Maintenance-Aware Approach for Sustainable Autonomous Mobile Robot Fleet Management

Syeda Tanjila Atik, Akshar Shravan Chavan, Daniel Grosu, *Senior Member, IEEE* and Marco Brocanelli, *Member, IEEE*

***Abstract*—Autonomous mobile robots (AMRs) are capable of carrying out operations continuously for 24/7, which enables them to optimize tasks, increase throughput, and meet demanding operational requirements. To ensure seamless and uninterrupted operations, an effective coordination of task allocation and charging schedules is crucial while considering the preservation of battery sustainability. Moreover, regular preventive maintenance plays an important role in enhancing the robustness of AMRs against hardware failures and abnormalities during task execution. However, existing works do not consider the influence of properly scheduling AMR maintenance on both task downtime and battery lifespan. In this paper, we propose MTC, a *maintenance-aware task and charging* scheduler designed for fleets of AMR operating continuously in highly automated environments. MTC leverages Linear Programming (LP) to first help decide the best time to schedule maintenance for a given set of AMRs. Subsequently, the Kuhn-Munkres algorithm, a variant of the Hungarian algorithm, is used to finalize task assignments and carry out the charge scheduling to minimize the combined cost of task downtime and battery degradation. Experimental results demonstrate the effectiveness of MTC, reducing the combined total cost up to 3.45 times and providing up to 68% improvement in battery capacity degradation compared to the baselines.**

***Index Terms*—preventive maintenance, autonomous mobile robots, battery degradation, task and charge scheduling**

I. Introduction

Autonomous Mobile Robots (AMRs) can concurrently execute a large variety of *objective tasks* (e.g., intrusion detection, indoor/outdoor delivery) while navigating on the ground of a working environment according to a certain *navigation task*. Given their versatility and ability to operate 24/7, fleets of AMRs are at the core of the Industry 4.0 revolution with an expected market growth from $2B in 2022 to $9B in 2032 [1]. To ensure an *environmentally-sustainable* adoption at scale, always-operating AMR fleets need help to coordinate task allocation, access to charging stations, and scheduling of preventive maintenance periods.

In our previous work, we have demonstrated how to coordinate task and charge scheduling for minimized task downtime and improved battery *quality of life*, i.e., longer lifespan and most of energy used for useful task execution [2]. However, this solution does not account for the frequent preventive maintenance requirements of AMRs, which requires a substantially different approach due to the high complexity of the problem. Traditional computing systems (e.g., data center servers) and fixed-position industrial robots generally have infrequent maintenance requirements (e.g., once a month). However, the mobility feature of AMRs makes them more vulnerable to failure. In fact, previous research has found that AMRs can experience failure every 6 to 20 hours due to various environmental conditions [3], which can be alleviated through daily preventive maintenance, as advised by some real AMR manufacturers [4]. However, as we show in this paper, a sub-optimal choice of maintenance scheduling for AMRs operating 24/7 can highly impact the fleet ability to carry out tasks and their battery degradation.

In this paper, we present MTC, a Maintenance-aware Task and Charge scheduling algorithm that autonomously coordinates the maintenance periods, task allocation, and charging times of always-operating AMR fleets for minimized task downtime and battery degradation. We formulate the MTC problem as a Mixed-Integer Non-Linear Program (MINLP). Given its high complexity, we then break down the MTC problem into two sub-problems. First, we transform the MINLP into a Linear Program (LP) relaxation to help decide the best time to schedule maintenance for a given set of AMRs. Because the obtained LP solution is infeasible in terms of task and charge scheduling, we then leverage the Kuhn-Munkres algorithm [5], a variant of the Hungarian algorithm, to finalize the task assignment and carry out the charge scheduling that minimizes the combined cost of task downtime and battery degradation.

Specifically, this paper makes the following contributions:

- We formulate the MTC problem as an MINLP to determine the best scheduling of maintenance periods, task allocation, and charge scheduling for minimized task downtime and battery degradation.
- The defined MINLP is infeasible to solve in polynomial time. To find a solution within polynomial time, we design the MTC algorithm, which leverages LP relaxation and the Kuhn-Munkres algorithm to find a near-optimal solution at a fraction of the execution time required to solve the MINLP.
- Experimental results demonstrate the effectiveness our MTC algorithm, reducing the combined total cost up to 3.45 times and providing up to 68% improvement in

Syeda Tanjila Atik, Akshar Shravan Chavan, and Daniel Grosu are with the Computer Science Department at Wayne State University, Detroit, MI 48202. Marco Brocanelli is with the Electrical and Computer Engineering Department at The Ohio State University, Columbus, OH 43210. Emails: {hf1987,aksharchavan,dgrosu}@wayne.edu, brocanelli.1@osu.edu.

Corresponding author: Marco Brocanelli.

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

Manuscript received XX XXXXX XXXX; revised XX XXXXXX XX; accepted XX XXXXXX XXXX. Date of publication XX XXXXXX XXXX; date of current version XX XXXX XXXX.

Digital Object Identifier no. XXXXXXXXXXXXXXXXXXXXX

battery capacity degradation compared to baselines that randomly decide maintenance periods.

The rest of the paper is organized as follows. Section II describes the related work, Section III presents the MTC problem formulation, Section IV explains the polynomial-time MTC algorithm, Section V shows the experimental results, Section VI discusses the limitations of the proposed approach and future work, and Section VII concludes the paper.

II. Related work

Many studies on AMRs focus on navigation and path planning strategies [6]–[14]. However, in our paper, we focus on how to leverage high-level task allocation to improve battery lifespan and schedule preventative maintenance. In this section, we describe the most-related state-of-the-art on how to improve battery lifespan, schedule preventive maintenance, and determine task and charging schedule policies for battery-operated mobile devices.

Battery Lifespan. Recognizing the significance of environmental sustainability in the realm of battery-dependent mobile computing, several studies have offered diverse approaches to maximize the battery lifespan. These studies rely on in-depth analysis of battery behaviour to formulate solutions using analytical techniques. Some solutions implement low-level task scheduling strategies designed to achieve a low average discharge current profile for the battery. Luo and Jha [15] propose two battery-aware static scheduling schemes, one optimizing discharge power profiles to maximize battery capacity utilization, and another for voltage-scalable processing elements, reducing average discharge power consumption and flattening power profiles through efficient slack time re-allocation. Chakrabarti and Chowdhury [16] propose task scheduling with voltage scaling in battery-powered systems to maximize battery performance by introducing an efficient heuristic algorithm based on a charge-derived cost function for task sequencing and slack allocation. Nodoushan et al. [17] propose a battery-aware speed scheduling policy for real-time tasks in a Dynamic Voltage Scaling system, employing Calculus of Variations to analytically establish the upper limit for maximum charge attainable through Dynamic Voltage Scaling. Kwak et al. [18] provide a task scheduling strategy to reduce the battery degradation rate by controlling the battery operating temperature. He et al. [19] provide a custom relaxation-aware charging solution that determines the charging rate of mobile devices based on user's available time. Different from our work, the above solutions focus on improving battery lifespan through either low-level task scheduling (i.e., after assignment) or controlling charging rates, without considering the problem of how to coordinate AMRs in terms of task allocation, charging, and maintenance scheduling for maximized task performance and battery lifespan. These solutions are thus orthogonal and complementary to our work.

Preventive Maintenance. Maintenance is essential for AMRs to ensure higher performance, lower downtime, and reduced repair costs. Early work on preventive maintenance does not focus on AMRs. For example, Schouten and Vanneste [20] provide a preventive maintenance policy for a deteriorating installation with a buffer, aiming to minimize interruptions in a production system considering both installation age and buffer content. They characterize the form of optimal policy determined through a semi-Markov decision process and solved it using a policy-iteration technique. Meller and Kim [21] provide a solution considering a system with two production operations connected by a buffer, aiming to identify the optimal buffer inventory level for triggering preventive maintenance on the first operation. It utilizes a cost model that encompasses preventive maintenance, unscheduled repairs, and inventory management. More recent studies focus on maintenance for robot arms rather than AMRs and do not cover the joint problem of task and charge scheduling. For example, Chen et al. [22] explore knowledge-driven techniques for smart equipment management in Mixed Model Assembly (MMA). They propose a knowledge sharing-enabled multi-robot collaboration strategy for preventive maintenance, featuring ontology-based modeling, task-specific actions, and knowledge exchange. Their objective is to reduce unexpected downtime, achieve robot workload equilibrium, and avert equipment degradation within the MMA context.

Task and Charging Schedule. Several related studies propose coordinated policies for maximizing the performance of AMR fleets with limited resources by focusing either on task allocation [23]–[26] or charging schedule [27]–[29]. For example, Lee et al. [23] propose a resource-oriented, decentralized auction algorithm for multirobot task allocation considering the limited robot communication range. Sempé et al. [29] focus instead on the design of a group of self-sufficient mobile robots so that they can remain in operation and efficiently share a charging station. Considering the joint problem, Chen and Xie [30] propose a joint task allocation, routing, and charging problem to simultaneously minimize the total energy consumption, total energy charged, and total service time. However, they overlook the potential downtime caused by maintenance activities and do not consider the effect on the battery lifespan. In our previous study [2], we focus on joint task allocation and charging schedule for AMRs to achieve high-quality battery life and low task downtime. We have formulated the problem as an MINLP problem and proposed a polynomial-time multi-period greedy algorithm. However, including maintenance scheduling makes the problem too complex and requires a substantially different approach.

To the best of our knowledge, this is the first paper that proposes an approach to jointly schedule maintenance periods, allocate task, and schedule charging for multiple AMRs with the objective of minimizing task downtime and battery degradation.

III. MTC Problem Formulation

In this paper, we focus on a simplified scenario involving a fleet of identical AMRs operating 24/7 in the working environment. We assume the fleet manager specifies a set of tasks to be carried out by the AMR fleet, during a specified long working period (e.g., 24 hours). The fleet manager can also select which AMRs in the fleet need preventive maintenance during the working period. Daily maintenance

for AMRs involve a standard set of operations, such as hardware inspection, checking for leakage, cleaning chips and sensors, and testing batteries [31], [32] to be carried out by specialized maintenance personnel. Thus, we consider that the average time necessary to carry out such operations is similar across AMRs. However, our framework can be easily adapted to support heterogeneous maintenance durations (see Section III-C).

The task set is made up of *objective tasks* and *navigation tasks*. Each navigation task allows an AMR to travel a path in the environment while also carrying out a number of associated objective tasks, such as the detection of dangerous situations and safety violations. The navigation tasks can either generate or be given waypoints to reach a certain location. Between waypoints, the task usually periodically generates and updates a local path to follow the waypoints. The navigation can be based on camera frames and/or Lidar data analysis through standard techniques or appropriate machine learning models. Each objective task has three characteristics: a specific level of priority, a given quantity of computing demand, as well as access to a particular number of sensors. Additionally, the AMRs have the ability to recharge at any of the available charging stations.

The working period is split up into multiple decision periods (e.g., ten minutes each) to coordinate AMRs and handle runtime variations. As our paper targets to provide a long-term plan to coordinate the scheduling of maintenance, task execution, and charging of multiple AMRs, we assume that the task load and environment is generally static. However, our approach can also be easily adapted to *react* to unexpected variations on energy usage, AMR failure, addition of AMRs, or new tasks to execute, by collecting at every decision period status information from the AMRs, which can be used to update the subsequent schedule according to the algorithm described in Section IV. We remind to Section VI for an in-depth discussion on how to address various low-level dynamic factors, including energy variations and communication delays.

In summary, *the MTC problem aims at jointly finding the best time to schedule AMR maintenance, determining task allocation to AMRs, and scheduling charging times for minimized task downtime and battery life quality degradation.* In this section, we describe the cost function and the necessary constraints to formulate the MTC problem. Note that part of the cost function and constraints are similar to our previous paper [2]. We focus on the new constraints related to scheduling maintenance of the AMRs in Section III-C.

A. Cost Function of the MTC Problem

We have formulated our cost function based on the two desired objectives. The first objective is to minimize the task downtime and the other one is to minimize the degradation of battery quality of life [2], i.e., desired lifespan and most of the energy spent for execution of useful objective tasks rather than wasted in frequently driving the AMR to charging stations. Thus, the total cost function is defined as follows:

$$\min_{X,Z,U} Downtime(X) + q \cdot Bat\_Degradation(X,Z) \quad (1)$$

Here, X and Z are the four dimensional and three dimensional arrays used to store the decisions for task assignment and charge scheduling respectively. $x_{k,i,h,j}$ is a binary element of X, which is equal to 1 if in period k AMR i is assigned to navigation task h and associated objective task j, and, 0 otherwise. $z_{k,i,c}$ is a binary element of array Z, which is equal to 1 if in decision period k the AMR i is scheduled for recharge at the charging station c, and, 0 otherwise. U is a 2D matrix that specifies the start of the maintenance of a particular AMR. $u_{k,i}$ is a binary element of U, which is 1 if AMR i is scheduled to start maintenance at time period k, and 0 otherwise. Note that the maintenance variables $u_{k,i}$ affect the cost function through a set of constraints (see Section III-B). q is considered as a weight factor that enables the end user to give more importance to battery quality of life over task downtime.

Cost of Downtime. We define the downtime cost as follows:

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

where $\mathcal{R}$, $\mathcal{T}$, $\mathcal{H}$, and $\mathcal{J}$ are the set of AMRs, decision periods, navigation, and objective tasks, respectively. $\gamma_{h,j}$ is a binary element of the matrix Γ, which can be adjusted according to configuration. Its value is set to 1 if the objective task j needs to be performed during the navigation task h, and 0 otherwise. Additionally, the fractional weight denoted as p_j, ranging from 0.1 to 1, represents the priority assigned to the objective task j.

Battery Quality of Life Degradation Cost. Depending on the specific type of battery, the overall lifespan of batteries is significantly influenced by two key factors: the depth of discharge (DoD) and the maximum energy stored within the battery at any given moment [26]. Therefore, it is assumed that users can utilize existing tools designed for assessing battery lifespan (such as [33]) to establish maximum and minimum thresholds of energy that align with their desired battery lifespan, such as 8 years. However, while the lifespan of certain batteries remains unaffected by frequent charging, a substantial amount of energy can be wasted by the AMR when it frequently navigates to charging stations. Consequently, achieving both the desired lifespan and minimal energy waste (referred to as high-quality battery life [2]) can be accomplished by scheduling charging cycles to commence when the AMR's energy level is near the minimum threshold (e^{DoD}) and cease when it approaches the selected maximum threshold (e^{max}). The battery degradation cost is thus defined as:

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

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

where $e_{k-1,i}$ denotes the energy level of AMR i at the end of the previous decision period $(k-1)$, e^{bat} is the maximum energy level that can be stored in the AMR batteries and

$\mathcal{C}$ is the set of charging stations. The decision to start or stop charging is determined by Equation (4), which triggers when the decision variable z for an AMR goes from 0 to 1 (indicating the start of charging) or from 1 to 0 (indicating the stop of charging) over two consecutive decision periods ($k-1$ to k).

B. General Constraints of the MTC Problem

Limited resource capacity. The total amount of instructions to carry out the designated navigation and objective tasks during each decision period must not exceed the computational capacity of each AMR, similar to our earlier work [34]:

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

where $n_{k,i,h}=\max\left[x_{k,i,h,1}\ \ \ldots\ \ x_{k,i,h,|\mathcal{J}|}\right]$ indicates which navigation task is allocated to AMR i in period k. It 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. As also demonstrated by previous studies [35]–[39], 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.

Energy Availability. The energy remaining in each AMR's battery at the end of each decision period, denoted by $e_{k,i}$ in Equation (4), must be estimated by a mathematical model in order to plan the task assignment and charging schedule over several decision periods. Naturally, the available energy must be in between 0 and the maximum capacity of the battery e^{bat}:

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

The power consumption of AMRs is mostly caused by computation, sensors, and locomotion procedures, as we have shown in our prior work [40]. Therefore, we can estimate the energy level of AMR i in period k as:

$$e_{k,i}=e_{k-1,i}+e_{k,i}^{charged}-e_{k,i}^{compt}-e_{k,i}^{sensor}-e_{k,i}^{locom}-e_{k,i}^{change} \tag{7}$$

where $e_{k,i}^{charged}$ denotes how much energy is recharged in AMR i if it is set to recharge during period k, $e_{k-1,i}$ denotes the level of energy of AMR i at the end of the previous decision period, $e_{k,i}^{compt}$ denotes the energy consumption due to computation, $e_{k,i}^{sensor}$ denotes the energy consumption due to acquisition of sensor data, $e_{k,i}^{locom}$ denotes the energy consumption due to locomotion, and $e_{k,i}^{change}$ denotes the amount of energy that is required for the AMRs when changing the navigation task, going to the charging station for recharge, or going back to execute tasks after recharging between consecutive decision periods. Specifically:

- The amount of energy that is charged in AMR i in period k at any of the charging stations c is simply calculated as follows when the AMR's recharge is scheduled by setting $z_{k,i,c}=1$:

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

  Here, t is the length of the decision period and ν is the rate of charging.
- The estimation of computational energy consumption can be done utilizing well known models [34], [41] based on the time for executing task and the third power of clock frequency, i.e., $\alpha^{comp}\cdot Exec\_time\cdot f^3$, where f is the clock frequency and α^{comp} is an estimated parameter. Similar to [34], the execution time of a set of tasks, can be estimated easily based on the ratio of the total number of instructions required to execute for each task that is allocated and the maximum CPU's instructions per second M^{max}. Thus:

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

  We conservatively assume that in order to achieve the maximum task performance, AMRs always run at the highest clock frequency.
- Each AMR has a set of sensors $\mathcal{S}$ (e.g., cameras and lidar). In our earlier research [40], we have found that collecting sensor data requires an amount of energy that is non-negligible. Hence, given an average access rate of each navigation and objective tasks to each sensor, the total sensing energy consumption is estimated as follows:

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

  where α_l^{sens} 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_l^{sens} is multiplied by the total sensor data accessed by the allocated tasks during each decision period.
- The locomotion energy $e_{k,i}^{loc}$ is dependent on the navigation path that is assigned. Given that each path is different in terms of slopes, curves, and obstacles each navigation task h is characterized by a measurable average locomotion energy consumption α_h^{loc} per period:

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

- In an ideal scenario, we would take into consideration the precise AMR energy consumption due to variations in navigation task and journey to/from each particular charging station over time, which is wasted as no useful objective task is being carried out. However, this would significantly increase the complexity of the problem.

Hence, we use a simplified model that takes into account the average distances between navigation paths and charging stations as follows:

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

where α^{change} represents the average energy per meter thar is required to navigate the AMR during changes in charge and navigational assignment. We assume each AMR uses a default navigation system and 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 indicator functions designed to be equal to 1 when there is a change in charging status and navigation allocation, respectively, and, 0 otherwise.

Availability of Charging Stations. The charging AMRs cannot exceed the number of charging stations $|\mathcal{C}|$:

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

Objective Task Allocation. The fleet manager decides the objective tasks that should be executed on top of each navigation task through the elements $\gamma_{h,j}$ of Γ. In addition, each objective task associated with a specific navigation task $h \in \mathcal{H}$ should be allocated to only one AMR:

$$\sum_{i\in\mathcal{R}} x_{k,i,h,j} \leq \gamma_{hj} \qquad \forall\, k,h,j \tag{14}$$

C. Constraints Related to Maintenance Scheduling

To simplify the formulation of the constraints related to scheduling the AMR maintenance, we define two input parameters. The first one is R^M, a binary array that informs whether each AMR has to be selected and complete the maintenance during the next working period, i.e., r_i^M equal to 1 to schedule maintenance, and 0 otherwise. The second one is P, a matrix of potential solutions for scheduling the maintenance. The size of P is $|(T-m)+1| \times |T|$, where T is the total number of working periods and m is the average maintenance duration given as input by the fleet manager. Specifically, each row of P denotes a potential set of m consecutive decision periods where maintenance would be performed. Thus, only m consecutive elements $p_{t,k}$ ($t \in \left[1,(T-m)+1\right]$) of each row have value 1 (i.e., maintenance). As discussed in the beginning of Section III, we consider similar maintenance duration for all AMRs. If heterogeneous duration are necessary, each AMR can define its matrix P independently. For example, if the total number of decision periods is 5 and the maintenance duration is 3 periods long, the matrix P is equal to:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Thus, if MTC decides that AMR i has to start its maintenance in period $k=2$, then the maintenance vector variable $u_{.,i}$ of AMR i would be equal to $\begin{bmatrix}0 & 1 & 0\end{bmatrix}$ and the result of the product $u_{.,i}P$ can be used to define the periods when the AMR is in maintenance and cannot recharge or execute tasks (i.e., $\begin{bmatrix}0 & 1 & 1 & 1 & 0\end{bmatrix}$ in this example).

Enabling Maintenance. All AMRs i with $r_i^M = 1$ will go to maintenance only once during the whole working period:

$$\sum_{k=1}^{T-m} u_{k,i} = r_i^M \qquad \forall\, i \tag{15}$$

State Consistency. The four possible states for each AMR in the fleet are recharging, executing, waiting, and maintenance. The AMR cannot perform any tasks while it is recharging at one of the charging stations. One of the navigation tasks and at least one of the objective tasks are being carried out by the AMR when it is in the executing state. The AMR can not execute tasks or charge while being in maintenance. Finally, an AMR is in the waiting state while it is neither charging, executing tasks, or in maintenance. We can define these rules as follows:

$$\sum_{c\in\mathcal{C}} z_{k,i,c} + \sum_{h\in\mathcal{H}} n_{k,i,h} \leq 1 - u_{k,i}\cdot p_{t,k} \qquad \forall\, k,i,t \tag{16}$$

where $n_{k,i,h}$ is equal to 1 when at least one objective task of navigation task h is assigned to AMR i in period k. Note that the matrix P indicating the potential solutions for scheduling the maintenance, may exhibit a high degree of sparsity in some cases. However, this sparsity is not going to increase the time for the LP solver to find a solution. This is because each entry equal to 0 in the matrix P would lead to the exact same constraint in Equation (16), which can help the LP solver reduce the search space.

Binary Decision Variables. The decision variables can only have binary values:

$$\begin{aligned} x_{k,i,h,j} &= \{0,1\} \qquad \forall k,i,h,j \\ z_{k,i,c} &= \{0,1\} \qquad \forall k,i,c \\ u_{k,i} &= \{0,1\} \qquad \forall k,i \end{aligned} \tag{17}$$

D. MINLP Implementation of MTC problem

Due to the integer restriction imposed by Equation (17) and the non-linearities found in Equation (4), the MTC problem defined in Section III is a Mixed Integer Nonlinear Program (MINLP). For these kind of problems, determining the best solution typically exhibit exponential time complexity [42]. To determine a near-optimal solution we use the Gurobi solver [43] to implement the MTC problem as a centralized algorithm running either in the cloud or a nearby edge server. However, even this solver can take long time to find a solution to MINLP problems. In our previous work [2] we have designed a polynomial time algorithm based on a greedy approach to coordinate task and charge scheduling for minimized task downtime and improved battery quality of life. However, that algorithm plans the task and charging schedule for several successive periods, which makes it complicated to be modified to include maintenance scheduling. To solve the MTC problem, we thus employ more advanced methods based on Linear Program (LP) relaxation and the Kuhn-Munkres algorithm to find a solution to the MTC problem in polynomial time.

Algorithm 1 MTC

Input: set of AMRs $\mathcal{R}$, number of decision periods T, set of navigation tasks $\mathcal{H}$, set of charging stations $\mathcal{C}$, set of objective tasks $\mathcal{J}$, matrix of navigation to objective task assignment Γ, set of AMRs needing maintenance $\mathcal{M}$, duration of the maintenance m, lowest energy of an AMR e^l, maximum energy capacity of an AMR e^{bat}.

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: $\mathcal{A}^{\mathcal{R}} \leftarrow \mathcal{R}$ ▷ Set of available AMRs
6: $\mathcal{A}^{\mathcal{C}} \leftarrow \mathcal{C}$ ▷ Set of available charging stations
7: $\mathcal{C}^{\mathcal{R}} \leftarrow \emptyset$ ▷ Set of charging AMRs
8: $\mathcal{W}^{\mathcal{R}} \leftarrow \emptyset$ ▷ Set of AMRs waiting to recharge
9: $\mathcal{T}^{\mathcal{M}} \leftarrow$ Maintenance_LP$(\mathcal{R}, T, \mathcal{H}, \mathcal{J}, \Gamma, \mathcal{M}, m, e^l, e^{bat})$
10: **for** $k \in \mathcal{T}$ **do**
11: **for** $i \in \mathcal{R}$ **do**
12: $e_{k-1,i} \leftarrow$ Get current energy level from AMR i
13: **if** $i \in \mathcal{A}^{\mathcal{R}}$ **and** $e_{k-1,i} \leq e^l$ **then**
14: $\mathcal{A}^{\mathcal{R}} \leftarrow \mathcal{A}^{\mathcal{R}} \setminus \{i\}$
15: $\mathcal{W}^{\mathcal{R}} \leftarrow \mathcal{W}^{\mathcal{R}} \cup \{i\}$
16: **if** $i \in \mathcal{M}$ **and** $k \in \mathcal{T}_i^{\mathcal{M}}$ **then**
17: $\mathcal{A}^{\mathcal{R}} \leftarrow \mathcal{A}^{\mathcal{R}} \setminus \{i\}$
18: $\mathcal{W}^{\mathcal{R}} \leftarrow \mathcal{W}^{\mathcal{R}} \setminus \{i\}$
19: $\mathcal{C}^{\mathcal{R}} \leftarrow \mathcal{C}^{\mathcal{R}} \setminus \{i\}$
20: $\mathcal{W}^{\mathcal{R}}, \mathcal{C}^{\mathcal{R}}, \mathcal{A}^{\mathcal{R}}, \mathcal{A}^{\mathcal{C}}, Z \leftarrow$ Charge$(k, \mathcal{W}^{\mathcal{R}}, \mathcal{A}^{\mathcal{C}}, Z, \mathcal{H}, \mathcal{J}, E, e^{bat}, \mathcal{C}^{\mathcal{R}}, \mathcal{A}^{\mathcal{R}}, \Gamma, 0)$
21: $X, \mathcal{A}^{\mathcal{R}}, \mathcal{W}^{\mathcal{R}} \leftarrow$ Task_Alloc$(k, \mathcal{A}^{\mathcal{R}}, \mathcal{W}^{\mathcal{R}}, \mathcal{H}, \mathcal{J}, E, \Gamma)$
22: $\mathcal{W}^{\mathcal{R}}, \mathcal{C}^{\mathcal{R}}, \mathcal{A}^{\mathcal{R}}, \mathcal{A}^{\mathcal{C}}, Z \leftarrow$ Charge$(k, \mathcal{W}^{\mathcal{R}}, \mathcal{A}^{\mathcal{C}}, Z, \mathcal{H}, \mathcal{J}, E, e^{bat}, \mathcal{C}^{\mathcal{R}}, \mathcal{A}^{\mathcal{R}}, \Gamma, 1)$
23: Transmit decision to AMRs and wait for next period

IV. Polynomial-Time MTC Algorithm

In this section, we describe the MTC algorithm, which provides a joint task allocation, charging, and maintenance schedule solution to the MTC problem.

Algorithm 1 shows the pseudo-code of the MTC algorithm. It takes the set of AMRs $\mathcal{R}$, the set of objective tasks $\mathcal{J}$, the set of navigation tasks $\mathcal{H}$, total number of decision periods T, the matrix Γ with elements $\gamma_{h,j}$ denoting which objective tasks j will be executed during each navigation task h, set of AMRs requiring maintenance $\mathcal{M}$, the minimum energy required by any AMRs to go back to any charging station e^l, and the maximum energy capacity of each AMR battery e^{bat}.

Initialization (Lines 1-8). The algorithm starts by initializing the variables in Lines 1-8. These include the set of decision periods $\mathcal{T}$, the task assignment array X with variables $x_{k,i,h,j}$, the array of charge scheduling Z with variables $z_{k,i,c}$ as elements, and the matrix E of AMR's energy level having elements $e_{k,i}$ are initialized to 0. We assume that at the beginning of the working period all the robots are available for task allocation (Line 5), and all charging stations are available to charge robots (Line 6). In addition, we assume no robot is charging (Line 7) or waiting to recharge (Line 8).

Maintenance Schedule (Line 9). The first decision made by MTC is when to schedule maintenance for each AMR in the input set $\mathcal{M}$. Given the complexity of the MTC problem described in Section III, it would be difficult to find a good heuristic. Thus, MTC in Line 9 calls the function Maintenance_LP, which is a linearized and relaxed version of the MTC problem that reads the current energy level of each AMR to obtain an infeasible but informative solution in polynomial time. In particular, we linearize the cost function in Equation (1) by adding auxiliary constraints and relax the integrality constraint in Equation (17) to transform the integer decision variables into non-integer variables with values between 0 and 1. The solution found by the relaxed LP includes fractional values for the decision variables. To obtain a feasible solution, for each AMR $i \in \mathcal{M}$ the Maintenance_LP function finds the highest $u_{k,i}$ fractional value across all time periods $k \in \mathcal{T}$ from the LP solution, rounds it to 1 (i.e., start period of maintenance for AMR i), and the rest to 0. This choice is based on the intuition that higher fractional values are generally correlated to a lower cost value.

Finally, the function creates a set of sets $\mathcal{T}^{\mathcal{M}}$ to be returned to the MTC algorithm, where $\mathcal{T}_i^{\mathcal{M}}$ is the set of time periods $k \in \mathcal{T}$ when AMR i is scheduled for maintenance (empty set if $i \notin \mathcal{M}$). Note that, different from the task allocation and charge scheduling decision (see below), the choice of the maintenance schedule is done only once at the beginning of the working period, which makes it easy for workers to determine the pick-up time for each selected AMR.

Task and Charge Scheduling (Lines 10-23). The algorithm then activates at the beginning of every decision period to determine the task and charge scheduling for the current period k. First, in Lines 11-19 it determines which AMRs are available for task allocation or need charging decision. Specifically, it reads the current energy level from each AMR (Line 12). If the energy level of any AMR in the set of available AMRs $\mathcal{A}^{\mathcal{R}}$ is less than the minimum energy necessary to go back to any charging station e^l, the algorithm makes it unavailable for further task allocation and adds it to the set of AMRs waiting for recharge $\mathcal{W}^{\mathcal{R}}$ (Lines 13-15). The algorithm also removes any AMR i scheduled for maintenance in the current period k from the set of AMRs available for task allocation, waiting for recharge, or charging (Lines 16-19).

Finally, MTC determines whether each AMR charging in the previous period $k-1$ should continue charging in the current period or stop charging to be available for task allocation (Line 20, Charge() function with last parameter 0). Then, it allocates the available tasks to the available AMRs (Line 21, Task_Alloc() function), and determines which AMR can start charging in the current period (Line 22, Charge() function with last parameter 1). The algorithm then transmits the allocation and charge scheduling decisions to the AMRs and waits for the beginning of the next decision period (Line 23). The details of the task allocation and charge scheduling functions in Lines 20-22 are given in the next sections.

A. Task Allocation

The pseudo-code of the Task_Alloc() function is provided in Algorithm 2. It first initializes the set $\mathcal{N}$ of navigation

Algorithm 2 Task_Alloc

Input: Current working period k, set of available AMRs $\mathcal{A}^{\mathcal{R}}$, set of AMRs waiting to recharge $\mathcal{W}^{\mathcal{R}}$, set of navigation tasks $\mathcal{H}$, set of objective tasks $\mathcal{J}$, AMR energy array E, navigation to objective task assignment Γ

1: $\mathcal{N} \leftarrow \mathcal{H}$ ▷ Set of available navigation tasks
2: **while** $\mathcal{A}^{\mathcal{R}} \neq \emptyset$ **and** $\mathcal{N} \neq \emptyset$ **do**
3: $W \leftarrow 0$ ▷ Weight matrix of size $(|\mathcal{N}| \times |\mathcal{A}^{\mathcal{R}}|)$
4: **for** $h \in \mathcal{N}$ **do**
5: **for** $i \in \mathcal{A}^{\mathcal{R}}$ **do**
6: $w_{h,i} \leftarrow$ Evaluate_Cost$(k, i, h, \mathcal{N}, \mathcal{J}, E, \Gamma)$
7: **if** $\sum_{l \in \mathcal{N}} \sum_{j \in \mathcal{J}} x_{k-1,i,l,j} \neq 0$ **then**
8: $w_{h,i} \leftarrow w_{h,i} \cdot ((100 - e_{k,i}) / \sum_{i=1}^{|\mathcal{A}^{\mathcal{R}}|} e_{k,i})$
9: $X, \mathcal{A}^{\mathcal{R}}, \mathcal{N}, \mathcal{W}^{\mathcal{R}} \leftarrow$ Munkres$(W, \mathcal{A}^{\mathcal{R}}, \mathcal{W}^{\mathcal{R}}, \mathcal{N}, \Gamma)$

Output: $X, \mathcal{A}^{\mathcal{R}}, \mathcal{W}^{\mathcal{R}}$

tasks available for allocation in Line 1. Then, in Lines 2-9 it assigns tasks to AMRs based on the Munkres algorithm [5], [44]. Specifically, the function creates a bipartite graph with nodes representing the AMRs available for allocation and the navigation tasks with at least one objective task left unallocated. Each AMR node is connected to a navigation node through a weighted edge. The Munkres algorithm finds the best allocation of AMRs to navigation tasks that minimizes the sum of the edge weights.

To do so, the Task_Alloc() function first initializes to zero a matrix of edge weights W of dimension $|\mathcal{N}| \times |\mathcal{A}^{\mathcal{R}}|$ (Line 3). Then, for each navigation task in $\mathcal{N}$ and every AMR in $\mathcal{A}^{\mathcal{R}}$, it calculates the weight of each edge of the bipartite graph (Lines 5-8). Each weight is calculated based on a cost function evaluation (i.e., Evaluate_Cost() in Line 6), which calculates the total cost of allocating the navigation task h to AMR i according to the Equation (1). It tries to first allocate all the objective tasks j that are associated with that particular navigation task h to the AMR i. Then, it calculates the total cost as the combined cost of what would be the task downtime and what would be the battery degradation cost if the AMR would recharge in the next period. The function then removes one objective task at a time and compares the current total cost with the previous total cost. If the current total cost is lower, the function keeps removing objective tasks one at a time until the current total cost can no longer be reduced. If no objective tasks of that navigation task h can be allocated to AMR i the function marks this edge as "DISALLOWED". Otherwise, it returns the least total cost it finds.

With this weight calculation, there is a possibility for frequent navigation task switching across AMRs over consecutive decision periods, which would cause an unnecessary waste of energy. To limit such occurrence, if the selected AMR i was previously allocated the selected navigation task h (Line 7), the Task_Alloc() function updates the weight $w_{h,i}$ by multiplying it by the relative difference of the energy level of AMR i with respect to the total energy level of all the AMRs in the set of available AMRs $\mathcal{A}^{\mathcal{R}}$ (Line 8). After calculating all the edge weights of W, the Munkres() function is called to find the best task to AMR allocation (Line 9).

Algorithm 3 Charge

Input: current working period k, set of AMRs waiting to recharge $\mathcal{W}^{\mathcal{R}}$, set of available charging stations $\mathcal{A}^{\mathcal{C}}$, charging array Z, set of navigation tasks $\mathcal{H}$, set of objective tasks $\mathcal{J}$, AMR energy array E, max energy level e^{bat}, set of charging AMRs $\mathcal{C}^{\mathcal{R}}$, set of available AMRs $\mathcal{A}^{\mathcal{R}}$ matrix of navigation to objective task assignment Γ, start/stop charge d

if $d = 0$ **then** ▷ Stop Charge Decision
Sort charging AMRs in non-increasing order energy. Let $V_{\sigma(1)}, V_{\sigma(2)}, ..., V_{\sigma(|\mathcal{C}^{\mathcal{R}}|)}$ be the order.
for $i = 1 \ldots |\mathcal{C}^{\mathcal{R}}|$ **do**
$s \leftarrow \text{argmax}_{c \in \mathcal{C}} \{z_{k-1,\sigma(i),c}\}$
if $e_{k-1,\sigma(i)} = e^{bat}$ **then**
$\mathcal{C}^{\mathcal{R}} \leftarrow \mathcal{C}^{\mathcal{R}} \setminus \{\sigma(i)\}$ ▷ Stop Charging
$\mathcal{A}^{\mathcal{R}} \leftarrow \mathcal{A}^{\mathcal{R}} \cup \{\sigma(i)\}$
$\mathcal{A}^{\mathcal{C}} \leftarrow \mathcal{A}^{\mathcal{C}} \cup \{s\}$
else
if Compare_Cost$(k, \sigma(i), \mathcal{H}, \mathcal{J}, E, \Gamma)$ **then**
$\mathcal{C}^{\mathcal{R}} \leftarrow \mathcal{C}^{\mathcal{R}} \setminus \{\sigma(i)\}$ ▷ Stop Charging
$\mathcal{A}^{\mathcal{R}} \leftarrow \mathcal{A}^{\mathcal{R}} \cup \{\sigma(i)\}$
$\mathcal{A}^{\mathcal{C}} \leftarrow \mathcal{A}^{\mathcal{C}} \cup \{s\}$
else
$z_{k,\sigma(i),s} \leftarrow 1$ ▷ Continue Charging
else ▷ Start Charge Decision
while $\mathcal{W}^{\mathcal{R}} \neq \emptyset$ **and** $\mathcal{A}^{\mathcal{C}} \neq \emptyset$ **do**
for $i \in \mathcal{W}^{\mathcal{R}}$ **do**
for $c \in \mathcal{A}^{\mathcal{C}}$ **do**
$z_{k,i,c} \leftarrow 1$ ▷ Start Charging
$\mathcal{C}^{\mathcal{R}} \leftarrow \mathcal{C}^{\mathcal{R}} \cup \{i\}$
$\mathcal{W}^{\mathcal{R}} \leftarrow \mathcal{W}^{\mathcal{R}} \setminus \{i\}$
$\mathcal{A}^{\mathcal{C}} \leftarrow \mathcal{A}^{\mathcal{C}} \setminus \{c\}$
break

Output: $\mathcal{W}^{\mathcal{R}}, \mathcal{C}^{\mathcal{R}}, \mathcal{A}^{\mathcal{R}}, \mathcal{A}^{\mathcal{C}}, Z$

After the first round of assignment is done, the function updates the task assignment array X, the set of AMRs available for task allocation $\mathcal{A}^{\mathcal{R}}$ (by removing the AMRs that were allocated a task in this round), the set of navigation tasks with at least one objective task left to allocate $\mathcal{N}$, and the set of AMRs waiting for recharge $\mathcal{W}^{\mathcal{R}}$. In particular, the function adds AMRs to the set $\mathcal{W}^{\mathcal{R}}$ if all the edges to that AMR are "DISALLOWED", i.e., it cannot execute any task and needs recharge. In addition, for instances with more AMRs than navigation tasks it may be possible that, according to the energy levels and the cost evaluation in Line 6, some navigation tasks have some objective tasks left unallocated. Thus, the function loops over Lines 2-9 until either there are no more AMRs or tasks allocatable. The final allocation decision, updated set of available AMRs, and updated set of AMRs waiting for recharge are returned to the MTC algorithm.

B. Charge Scheduling

The pseudo-code of the Charge() function is given in Algorithm 3. It executes to either stop or start charging AMRs.

Stop Charge (Lines 1-15). To determine which AMRs charging in the previous period should stop charging and become available for task allocation, or continue charging in the current period, the function first sorts the AMRs in the set $\mathcal{C}^{\mathcal{R}}$ in non-increasing order of energy (Line 2). Then, it determines which charging station was used by the highest-priority AMR in the previous period (Line 4). If the AMR has reached the maximum energy level, the function removes it from the charging AMRs, makes it available for task allocation, and makes the charging station available for another AMR (Lines 5-8). Otherwise, it uses the cost function Equation (1) to compare, using the Compare_Cost() function, the partial cost of leaving this AMR charging for another period (and cause eventual downtime) against making it available for task allocation (and possibly cause degradation). The function returns true for the latter case, i.e., interrupting the charge for this AMR is convenient in terms of total cost (Lines 10-13), and returns false for the former case, i.e, it is more convenient to continue charging despite some potential downtime (Line 15).
Start Charge (Lines 16-24). The task allocation algorithm described in Section IV-A may discover some AMRs in need of recharging starting the current period. Thus, the Charge() function is called again to allocated AMRs waiting for recharge to any of the available charging stations. Since all AMRs waiting for recharge are at a similar energy level, the function simply allocates each AMR in $\mathcal{W}^{\mathcal{R}}$ in one of the available charging stations in $\mathcal{A}^{\mathcal{C}}$ until either there are no more AMRs waiting or no more charging stations available.

C. Complexity Analysis of MTC

The MTC Algorithm 1 iterates through every decision period and the total number of decision periods is $|\mathcal{T}|$. The function Maintenance_LP() in Line 9 solves an LP problem, which takes polynomial time. In every decision period, the functions Charge() with parameter 0 and 1 and Task_Alloc() are executed. The Charge() function with parameter 0 in Line 20 takes $O(|\mathcal{C}|(|\mathcal{C}| + |\mathcal{H}||\mathcal{J}|))$ to execute. The Task_Alloc() function in Line 21 takes $O(|\mathcal{R}|^4 + |\mathcal{H}|^2|\mathcal{R}|^2|\mathcal{J}| + |\mathcal{H}||\mathcal{R}|^3)$. The Charge() function with parameter 1 in Line 22 takes $O(\max(|\mathcal{R}|, |\mathcal{C}|)|\mathcal{R}||\mathcal{C}|)$. Thus, the MTC algorithm finds a solution in polynomial time.

V. Experimental Results

A. Experimental Setup

We used HydraOne, our AMR prototype [45], to train the model parameters described in Section III using regression analysis and ensure realistic results. The prototype is equipped with an NVIDIA Jetson AGX, 156Wh Li-ion battery, Dual 250W motors, a Slamtec RPLidar S1 lidar, 2 Intel RealSense cameras, and an Arduino Mega 2560. We conducted an instruction count analysis of a representative navigation task based on HydraNet[1] and a representative object detection task based on MobileNet-SSD [40]. To address the variance in instruction counts across different tasks, we then generated problem instances using two uniform distributions. These distributions were centered around the mean instruction counts observed for HydraNet in navigation tasks and for MobileNet-SSD in objective tasks.

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

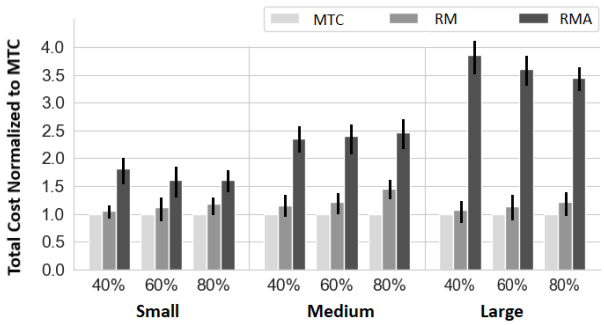


Fig. 1. Comparison of the total cost of the baselines normalized to MTC for small, medium, and large problem instances.

In our experiments, we have used the Gurobi solver to find the solution of the MINLP baseline and the LP of our MTC algorithm. We have chosen Gurobi because it has a free license for academic researchers [46]. Thus any researchers can use it to freely reproduce our results and build research on top of our code [47]. However, the problem can be easily rewritten for any other open source solvers like MIPCL [48], SCIP [49], GLPK [50], COIN-OR [51] and LP_Solve [52].
To run our experiments, we have generated several different problem instances categorizing them into small, medium and large instances. For small instances, we have considered 3-5 AMRs, 1-5 charging stations, 3-5 navigation tasks, 4-6 objective tasks, and working period of 3-8 hrs dividing it into decision periods of 10 mins. For medium instances we have considered 5-10 AMRs, 3-10 charging stations, 5-10 navigation tasks each having 4-6 objective tasks, and working period of 4-14 hrs. For large instances we have considered up to 15 AMRs and varied the number of other elements accordingly along with working period of 8-24 hrs. For each type of problem instance, we have considered 40%, 60% and 80% of the AMRs going to the maintenance and selected the maintenance duration of any AMR to be 1 hr.
Baselines. Given that there are no previous works solving the MTC problem and that our previous task and charge scheduling algorithm is too complex to be modified to host maintenance scheduling [2], we have designed three representative baselines to compare with our proposed MTC algorithm. The first one is RMA, which selects both the maintenance period and the task allocation randomly but uses the same charging policy. The second one is RM, which randomly selects the maintenance period but uses the same task allocation and charging policy of our MTC algorithm. The third one is MINLP, which is the MTC problem defined in Section III implemented in the Gurobi solver. Given that the solver can take long time to find a solution, we force it to provide the best solution found within a decision period.

B. Performance Metrics

In our previous work [2] we used, among other metrics, the energy-usage effectiveness (EEF) to measure the energy waste. However, in this paper we do not use this metric since the baselines use the same start-of-charge policy and achieve

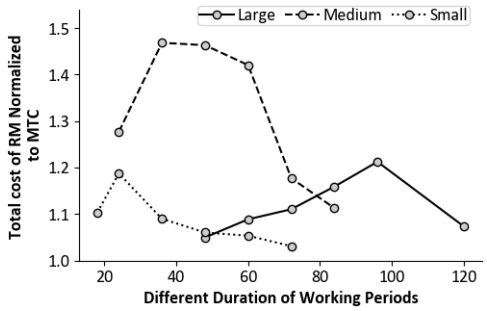


Fig. 2. Normalized total cost variation of RM for an increased duration of working periods for small, medium, and large problem instances.

similar EEF, which allows us to focus on the efficiency of the LP relaxation for maintenance decision and the Munkres algorithm for task allocation. Thus, the performance metrics used in our evaluation are the percentage of allocated tasks (TA) and the state of charge violation (SOC_V) for battery degradation. TA is the total percentage of objective tasks allocated to AMRs. SOC_V is defined as the total number of times an AMR violated the battery SOC thresholds throughout the battery charging cycle. The battery longevity is reduced by exceeding these thresholds:

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

where e^{bat} is the maximum energy capacity of the AMR battery, $e_{k,i}$ denotes the energy level of AMR i in period k, e^{max} and e^{DoD} are the thresholds of maximum and minimum energy given as input by the user. It is important to mention that MTC would accommodate any input threshold defined by the users. In accordance with previous research on Lithium-ion batteries [53], the lifespan of these batteries deteriorates when the energy level surpasses specific thresholds, falling short of the user's intended duration. Consequently, the SOC_V only rises when the energy level falls below the e^{DoD} or surpasses the e^{max}. The closer the SOC_V approaches zero, the more closely aligned the battery's lifespan becomes with the desired duration. More details on how to choose the energy thresholds can be found in Section VI.

C. Total Cost Comparison

Here we evaluate the performance of our proposed MTC algorithm using the small, medium, and large problem instances compared to the baselines RM and RMA. Due to its long running times, we do not include MINLP here but we will compare with it in Section V-E. We run each set of problem instance 50 times by varying the AMRs initial energy levels, randomizing the objective task priority to be selected from the range 0.1 to 1, and randomizing the number of instruction used for the navigation and objective tasks. Figure 1 shows the total cost given by the baselines normalized to the total cost of our MTC algorithm for small, medium, and large problem instances considering 40%, 60% and 80% of AMRs needing maintenance. Among the two baselines, RMA performs the worst with the highest total cost across all the cases. For example, considering the highest percentage of AMRs needing maintenance (e.g. 80%), the solution found by RMA shows 1.62, 2.47 and 3.45 times increased total cost compared to MTC for small, medium, and large problem instance, respectively.

RM, which performs the task allocation using the same approach as MTC but randomly selects maintenance periods for the AMRs, improves compared to RMA thanks to the allocation strategy described in Section IV-A. MTC further improves the solution of RM thanks to the LP relaxation strategy described in Section IV. Considering 80% of the AMRs needing maintenance, the solution found by RM leads to 1.18, 1.46 and 1.21 times increased total cost compared to MTC for small, medium, and large problem instance, respectively. Thus, MTC reduces the total cost by up to 147% and 46% compared to RMA and RM, respectively.

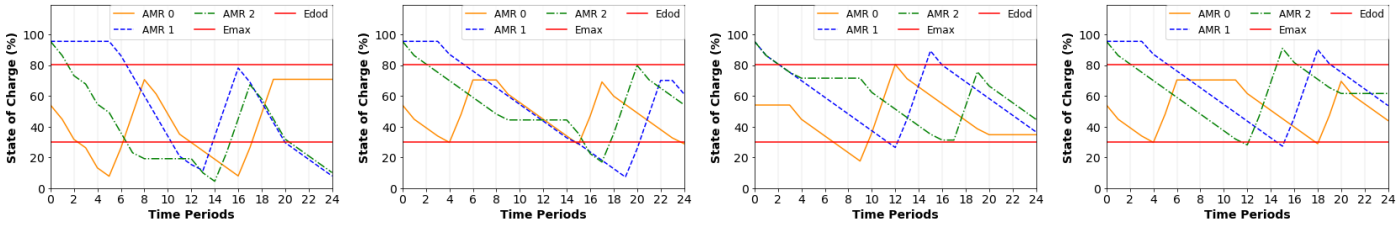
D. Working Period Effect on Total Cost

The total cost improvement achieved by MTC compared to the baselines also depends on the difference between the length of the working period and that of the maintenance period. In fact, if the maintenance period is very short the benefit of the relaxed LP can only minimally impact the total cost. On the other end, if the maintenance period is comparable to the working period the difference in choice between the LP relaxation and the random one would not be sufficient to cause much cost variation. Thus, it is expected that the ratio of total cost between the random baselines and MTC has the highest values somewhere in the middle. In addition, the bigger the problem instance is, the longer the duration of the working period should be for MTC to provide a solution with substantially reduced total cost.

Figure 2 shows how increasing the duration of total working period affects the solution found by RM compared to MTC using the same maintenance duration of previous section (six periods) for small, medium and large instances. We can see that for medium problem instances, if we consider the duration of working period to be 36 decision periods, the solution provided by RM increases the total cost by 1.46 times compared to MTC. Afterwards, the total cost decreases with the increase of the working period. This behavior also depends on how many AMRs are scheduled for maintenance. This is why for larger problem instances where the number of AMRs needing maintenance is higher, we need a larger working period to observe a larger improvement in the total cost provided by MTC compared to RM. As seen from the figure, when the working period duration is 24, the solution found by RM shows a larger increase in the total cost by 1.27 times for small problem instance. A similar increase can be seen for the larger problem instance considering a longer working period length of 96 periods. As expected, for all cases the solution found by RM becomes closer in cost to that of MTC when the length of the working period further increases.

E. Example Case Study

In this section, we describe in detail the operation of the baselines, RMA, RM, MINLP and our proposed MTC with



(a) (b) (c) (d)

Fig. 3. Comparison of the battery State-of-charge (SOC) among the baselines (a) RMA, (b) RM, (c) MINLP, and (d) proposed MTC algorithm considering a single case with SOC threshold of 30% and 80% of battery capacity.

an example scenario. The values of the performance metrics obtained from the four approaches is given in Table I. Figure 3 shows the SOC profiles of 3 AMRs in the case of baselines and MTC. Here, we have considered a single case study that consists of 24 decision periods of 10 minutes each, three AMRs, AMR 2 selected for maintenance, three charging stations, two navigation tasks with five objective tasks each.

Figure 3a shows the SOC profile of AMRs with the task, charging, and maintenance schedules found by RMA. Because the task allocation is done randomly, we can observe that the AMRs perform tasks for fewer periods compared to other approaches. For instance, AMR 1 could perform tasks only for 8 periods with RMA, whereas with RM AMR 1 performs tasks for 16 periods because with random task allocation, the AMR switches more frequently between the navigation tasks. Consequently, frequent transiting of AMRs between the navigation tasks causes additional energy consumption that is not utilized towards objective tasks. Furthermore, we notice that AMR 2 is selected to start its maintenance randomly in period 7, leaving only AMR 1 available to perform tasks as AMR 0 is charging in that period. This maintenance schedule resulted in insufficient availability of AMRs to perform navigation tasks and thus increased task downtime. For these reasons RMA shows a higher SOC violation of 18.59%, the lowest task allocation percentage at 91%, and the highest total cost compared to other baselines and MTC.

Figure 3b shows the SOC profile of AMRs with RM in which the task allocation schedule is performed similar to MTC using Kuhn-Munkres algorithm, but the maintenance schedule is selected randomly. Compared to task allocation schedule of RMA, we observe that with RM the AMRs remain on one particular navigation task until they start recharging leading to a lower task downtime. However, due to random maintenance schedule, AMR 2 is selected for maintenance in period 9 when it is at a lower SOC. As a result, when the AMR 2 is available to perform tasks after completing the maintenance, the task allocation schedule performs allocation even while violating the SOC threshold. Doing so RM obtains a similar task allocation percentage of 98% to MTC but has 36.45% higher total cost due to a higher battery degradation.

Figure 3c shows the SOC profile of AMRs with MINLP in which the maintenance and task allocation schedule is obtained through our formulation implemented in Gurobi. MINLP performs maintenance, task allocation, and charging schedule while considering the objective in Equation (1) to provide near optimal solution. It obtains the highest task allocation percentage of 100% and lowest SOC_V of 2.03%. Therefore, the total cost is lowest among all baselines and MINLP. However, due to its high time complexity, we had to put a time limit of 10 minutes to get the solution, which is a 20,000 times longer execution time than that of MTC. For larger problem instances, the solver is unable to find a better solution within that time limit.

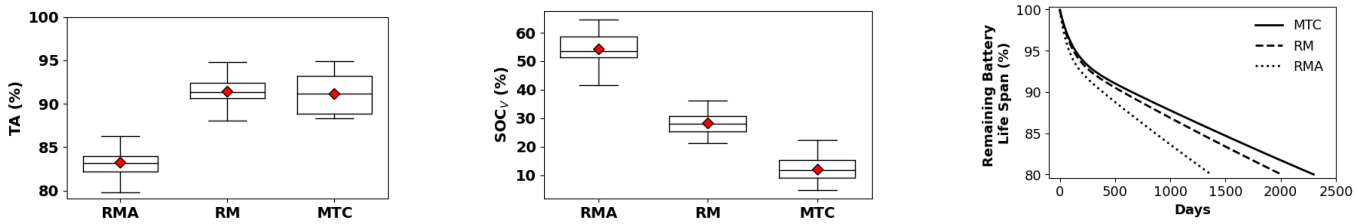
Figure 3d shows the SOC profile of AMRs with our proposed polynomial-time MTC algorithm. MTC performs maintenance using the LP relaxation as well as task allocation using the Munkers algorithm and charging schedule considering the total cost in Equation (1). Thus, it is able to obtain a task allocation percentage of 98%, which is similar to that of RM, 7.69% higher compared to RMA, and 2% lower than MINLP. Thanks to the LP relaxation, the maintenance period is set to start in period 20, where the remaining two AMRs are able to execute the rest of the tasks without violating SOC thresholds. Thus, MINLP and MTC perform a balanced trade-off between TA and SOC_V to provide a joint maintenance, task allocation, and charging schedule that ensures minimized task allocation and battery degradation. However, MTC is able to find a reasonable solution within a fraction of time.

TABLE I
PERFORMANCE COMPARISON OF MTC WITH BASELINES

Approach	Total Cost	Task Allocation	SOC_V	Exec. Time (sec)
RMA	116.93	91%	18.59%	0.001
RM	59.32	98%	6.06%	0.002
MINLP	24.62	100%	2.03%	600.03
MTC	41.87	98%	3.69%	0.03

F. Performance Metrics Comparison

In Figure 4, we show the performance comparison of MTC with the other two baselines, RMA and RM, considering all the problem instances. Figure 4a shows the percentage of task allocation provided by the three approaches. We can see that RMA has the lowest percentage of task allocation with an average of 83%. This is because RMA chooses randomly to allocate the tasks to the AMRs without considering the total cost, which includes also how much downtime would be caused because of the allocation of tasks. RM shows a better performance than RMA in allocating 88% to 95% of the tasks. This is because RM uses the same approach as MTC for task assignment to the AMRs. On the other hand, MTC performs the best among two of them by being able to allocate 90% to 96% of the tasks. The result shows the advantage of using the



(a) (b) (c)

Fig. 4. Comparison of (a) task allocation, (b) per-run SoC threshold violation, and (c) remaining battery life span for RMA, RM and MTC.

Linear Programming approach on deciding the maintenance schedule rather than selecting randomly.

Figure 4b shows the comparison of violation of SoC threshold per run (as shown in Eq. 18) for the three approaches. We can see RMA shows the worst SOC_V. On average, RMA results in violating the SoC thresholds more than 55% of the times resulting in a higher cost of battery life span degradation. In the case of RM, the SoC thresholds are violated on average 30% of the time, which is less than RMA because of the joint consideration of task allocation and battery degradation in the Munkres algorithm. However, in the case of RM, the maintenance periods are selected randomly for the AMRs without considering the effect of it on the total cost. As a result, the available AMRs have to keep executing tasks and violate the SOC thresholds more often. Compared to RMA and RM, MTC performs much better and violates the SOC thresholds less than 15% of the times on average. As MTC uses the linear programming approach to select the maintenance schedule, it gives a much better solution compared to RM, which leads to a lower battery lifespan degradation even though, at times, some AMRs may still need to violate the SOC thresholds to keep executing tasks when AMRs go to maintenance.

Figure 4c shows how the battery life span would be affected by the baselines and MTC in the long run. We have used the state of charge trace of AMR 1 from the scenario considered in Figure 3 and used the rainflow counting algorithm [54] to estimate the remaining battery life span of the AMR battery over the years. Because using all the AMRs' SOC traces in the rainfall algorithm is extremely time consuming, we only used one of the AMR's SoC traces to show an example results. We find that, in case of RMA, the SOC thresholds are violated more often, which results in the AMR battery capacity to degrade much faster than RM and MTC. We can see that the battery life span degradation reaches 20% in approximately 1400 days (3.8 years) when using RMA while in case of RM, it is approximately 2000 days (5.4 years). Compared to the baselines, MTC reduces the battery life span degradation and takes approximately more than 2400 days (6.6 years) for the battery capacity to degrade to 20%. Thus, MTC improves the battery capacity degradation by approximately 25% and 68% compared with the baselines RM and RMA, respectively.

VI. Limitations and Future Work

We have assessed the performance of the MTC algorithm by conducting simulations. However, the simulations are performed using well-established locomotion and energy models, whose parameters are trained based on data obtained from our real AMR prototype [45]. Therefore, we can assert that the results obtained from these simulations are reliable. Furthermore, running real-life experiments to directly observe the variations in battery lifespan is an intricate and time-consuming endeavor. As a result, we have utilized the rainflow counting algorithm [54] as it is a robust and user-friendly model for estimating battery lifespan, particularly in large-scale AMR setups. On the other hand, the proposed MTC may be improved in terms of scalability and handling variability of environmental conditions, but could also be used for choosing the best battery energy thresholds.

Scalability. Currently, the proposed MTC algorithm adheres to a centralized paradigm. Consequently, as the number of AMRs increases, it becomes imperative to address the issue of scalability. In our experimental evaluation, we rigorously assessed the algorithm's performance with fleet sizes of up to 15 AMRs where the algorithm exhibited an execution time of approximately 30 seconds, which is only 5% of the generally larger decision period (10 minutes in our experiments). Normally a fleet of AMRs operates in a limited area (e.g., a factory), which naturally limits the fleet size. Thus, the MTC algorithm would perform well in such scenarios. However, for very large fleets, for example having 100-1000 AMRs, we acknowledge that the algorithm may have scalability issues because the LP portion may take longer to provide a solution. A very large AMR fleet would normally serve an operational area that is also considerably large. In such a scenario, a possible solution is to divide the large area into smaller cells where each cell has its own set of AMRs to execute tasks and can be managed by the proposed MTC control algorithm. On the other hand, if the large area cannot be divided into smaller cells, a future direction is to develop a distributed version of the MTC algorithm that executes directly on each AMR.

Several studies proposed solutions on distributed task allocation in multi-robot systems operating in dynamic environment [55]–[58]. Similar to those works, a distributed framework can be employed for maintenance, task, and charge scheduling where each AMR can be allowed to autonomously execute various sub-steps of the Kuhn–Munkres algorithm based on its own state information and the information received from the other AMRs in the fleet. The communication among the AMRs can be helpful in achieving the allocation decision without a central manager. Also, a game theoretical approach [59]–[64] can be considered, where each AMR can be treated as a player in a strategic game. In this approach, AMRs make decisions based on their utility functions, which quantify the benefit or cost of performing specific tasks. Within

this structure, a distributed algorithm could be developed to find equilibrium solutions such as Nash or Bayes-Nash equilibrium ensuring an optimized and balanced task distribution among the AMRs. Another effective strategy based on mechanism design (often referred to as reverse game theory) can be considered, which crafts a distributed allocation system tailored specifically for resolving task allocation issues in a dynamic environment [65]. These decentralized decision-making paradigms are well-suited for situations where centralized control is unfeasible or undesirable because it not only improves system efficiency but also adjusts to dynamic surroundings.

Dynamic Environmental Conditions. The proposed MTC algorithm operates with a coarse time grain to coordinate AMR operations as an offline approach, i.e., it assumes prior knowledge of task characteristics, which is common in several scenarios where AMRs have to execute a predetermined set of tasks (e.g., factories). However, it is in our future work designing an online version of the TMC algorithm for online task scenarios, e.g., robot delivery. In addition, due to the following reasons, it could be improved in terms of (1) fault tolerance, and (2) consideration of fine time grain dynamic variables (e.g., energy variations, non-linear battery discharge profiles, communication delays, or sensor inaccuracies). First, while considering preventive maintenance scheduling severely reduces the chance of various failures, the MTC algorithm could handle sudden AMR failures either by simply re-executing when failure is detected, or by accounting for some degree of redundancy for risky (e.g., older) AMRs. Second, it is challenging to include fine grain energy consumption, non-linear discharge, and communication delays at coarse-grain task/charge/maintenance decision time, because it requires considering with a much finer time granularity each AMR's current location, network status, and detailed terrain conditions over a much longer period. Including such fine grain characteristics into coarse-grain decisions may be impractical. Even considering fine-grain dynamics into the coarse-grain fleet coordination may still be affected by such estimation errors. If there is a consistent and significant variation in the energy consumed by an AMR for a specific task, the energy model parameters can be updated at runtime according to the readings from the AMR sensors. In addition, a potential solution might be allowing the AMRs to autonomously adjust their energy consumption (e.g., varying CPU frequency, velocity) to match the energy budgeted by the MTC algorithm, which would improve coordination effectiveness. This will be addressed in our future work.

We also believe that relatively short (e.g., tens of seconds) communication delays would not impact too much the performance of the proposed MTC algorithm, since communication is relatively infrequent for decision update (e.g., ten minutes in our experiments). However, such delays might be critical in the distributed version discussed above. We will study the potential effects of communication delays on the distributed algorithm performance to make it more robust to environmental dynamics. Finally, sensor inaccuracies would affect more the local operations of each AMR, which should be handled locally at a fine grain control (as discussed above) rather than through coarse grained allocation decisions.

Deciding Battery Energy Thresholds. In this paper, the MTC algorithm decides high-level task allocation, charge, and maintenance scheduling for the AMR fleets based on user-defined battery energy thresholds. However, it might be challenging for the users to choose such thresholds ahead of time. To alleviate this challenge, the fleet manager can leverage the proposed MTC algorithm to run simulations with various threshold values and observe their consequences in terms of battery degradation and task downtime. In addition, the user can also choose different values for the weight q (used in Equation (1)) to give more or less importance to the battery degradation over task downtime for each threshold combination. The simulations with the MTC algorithm can thus serve as an aid to find a good set of input parameters before the actual deployment. Note that the battery lifespan model we employed [54] takes into account the operating temperature. Assuming that AMRs operate in a similar environment having similar temperatures, our framework can be used to accurately estimate the obtained lifespan. On the other hand, in our future work we will consider ways to modify the framework in cases where each AMR may experience extremely different temperatures during operations.

VII. Conclusion

The uninterrupted operation of AMRs allows for task optimization, increased throughput, and meeting demanding operational requirements. In this paper, we explored the significance of effective coordination between task allocation, charging schedules, and maintenance for autonomous mobile robots (AMRs) operating 24/7 in highly automated environments. Existing works have overlooked the importance of appropriately scheduling maintenance for AMRs, considering both task downtime and battery lifespan. To address this gap, we proposed MTC, a maintenance-aware task and charging scheduler for fleets of AMRs based on Linear Programming (LP) to help decide the best time to schedule maintenance for a given set of AMRs and Kuhn-Munkres algorithm to finalize the task assignment and carry out the charge scheduling that minimizes the combined cost of task downtime and battery degradation. The experimental results showed that the solution found by MTC, reduces the combined total cost by up to 3.45 times and provides up to 68% improvement in battery capacity degradation compared to the baselines. These findings highlight the potential benefits of our proposed approach in optimizing the operation of AMRs in real-world scenarios, contributing to the advancement of autonomous mobile robots and sustainable battery utilization.

References

[1] Global Market Insights, "Autonomous mobile robot market - type, by end-use & forecast, 2023-2032," https://one.nhtsa.gov/nhtsa/Safety1nNum3ers/august2015/S1N_Aug15_Speeding_1.html, 2023.

[2] A. S. Chavan and M. Brocanelli, "Towards high-quality battery life for autonomous mobile robot fleets," in *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, 2022, pp. 61–70.

[3] J. Carlson and R. Murphy, "How ugvs physically fail in the field," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 423–437, 2005.

[4] SMP Robotics, "Mobile robot technical maintenance service," https://smprobotics.com/security_robot/services/, 2023.

[5] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the society for industrial and applied mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[6] S. Park and S. Hashimoto, "Autonomous mobile robot navigation using passive rfid in indoor environment," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 7, pp. 2366–2373, 2009.

[7] J. Janet, R. Luo, and M. Kay, "Autonomous mobile robot global motion planning and geometric beacon collection using traversability vectors," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 1, pp. 132–140, 1997.

[8] I. Shnaps and E. Rimon, "Online coverage by a tethered autonomous mobile robot in planar unknown environments," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 966–974, 2014.

[9] T. Schmitt, R. Hanek, M. Beetz, S. Buck, and B. Radig, "Cooperative probabilistic state estimation for vision-based autonomous mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 670–684, 2002.

[10] E. A. Sisbot, L. F. Marin-Urias, R. Alami, and T. Simeon, "A human aware mobile robot motion planner," *IEEE Transactions on Robotics*, vol. 23, no. 5, pp. 874–883, 2007.

[11] I. Ohya, A. Kosaka, and A. Kak, "Vision-based navigation by a mobile robot with obstacle avoidance using single-camera vision and ultrasonic sensing," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 969–978, 1998.

[12] N. Tsourveloudis, K. Valavanis, and T. Hebert, "Autonomous vehicle navigation utilizing electrostatic potential fields and fuzzy logic," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 4, pp. 490–497, 2001.

[13] C. Lamini, S. Benhlima, and A. Elbekri, "Genetic algorithm based approach for autonomous mobile robot path planning," *Procedia Computer Science*, vol. 127, pp. 180–189, 2018.

[14] J. C. Mohanta, D. R. Parhi, and S. K. Patel, "Path planning strategy for autonomous mobile robot navigation using petri-ga optimisation," *Computers & Electrical Engineering*, vol. 37, no. 6, pp. 1058–1070, 2011.

[15] J. Luo and N. Jha, "Battery-aware static scheduling for distributed real-time embedded systems," in *Proc. of the 38th Design Automation Conference*, 2001, pp. 444–449.

[16] P. Chowdhury and C. Chakrabarti, "Static task-scheduling algorithms for battery-powered dvs systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 2, pp. 226–237, 2005.

[17] M. Jafari-Nodoushan, B. Safaei, A. Ejlali, and J.-J. Chen, "Leakage-aware battery lifetime analysis using the calculus of variations," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4829–4841, 2020.

[18] J. Kwak, K. Lee, T. Kim, J. Lee, and I. Shin, "Battery aging deceleration for power-consuming real-time systems," in *2019 IEEE Real-Time Systems Symposium (RTSS)*, 2019, pp. 353–365.

[19] L. He, Y.-C. Tung, and K. G. Shin, "Icharge: User-interactive charging of mobile devices," in *Proc. 15th Annual Int. Conf. on Mobile Systems, Applications, and Services*, ser. MobiSys '17. New York, NY, USA: ACM, 2017, p. 413–426.

[20] F. Van der Duyn Schouten and S. Vanneste, "Maintenance optimization of a production system with buffer capacity," *European journal of operational research*, vol. 82, no. 2, pp. 323–338, 1995.

[21] R. D. Meller and D. S. Kim, "The impact of preventive maintenance on system cost and buffer size," *European Journal of Operational Research*, vol. 95, no. 3, pp. 577–591, 1996.

[22] B. Chen, Y. Zhang, X. Xia, M. Martinez-Garcia, and G. Jombo, "Knowledge sharing enabled multirobot collaboration for preventive maintenance in mixed model assembly," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 11, pp. 8098–8107, 2022.

[23] D.-H. Lee, S. A. Zaheer, and J.-H. 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, 2014.

[24] D.-H. Lee, "Resource-based task allocation for multi-robot systems," *Robotics and Autonomous Systems*, vol. 103, pp. 151–161, 2018.

[25] 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.* IEEE, 2006, pp. 2757–2762.

[26] 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)*. IEEE, 2018, pp. 868–874.

[27] 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. IEEE, 2002, pp. 2746–2751.

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

[29] F. Sempé, A. Munoz, and A. Drogoul, "Autonomous robots sharing a charging station with no communication: a case study," in *Distributed Autonomous Robotic Systems 5*. Springer, 2002, pp. 91–100.

[30] J. Chen and J. Xie, "Joint task scheduling, routing, and charging for multi-uav based mobile edge computing," in *ICC 2022 - IEEE International Conference on Communications*, 2022, pp. 1–6.

[31] Steven Douglas Corp., "Preventative maintenance for industrial robots," https://sdcautomation.com/preventative-maintenance-for-industrial-robots,, 2023.

[32] B2E Automation, "Industrial robot maintenance - the complete checklist," https://www.b2eautomation.com/insights/industrial-robot-maintenance-the-complete-checklist,, 2022.

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

[34] 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*, vol. 22, no. 2, pp. 1231–1245, 2023.

[35] M. Zagha, B. Larson, S. Turner, and M. Itzkowitz, "Performance analysis using the mips r10000 performance counters," in *Proc. of the 1996 ACM/IEEE Conference on Supercomputing*, 1996, pp. 16–16.

[36] S. Tuli, S. Ilager, K. Ramamohanarao, and R. Buyya, "Dynamic scheduling for stochastic edge-cloud computing environments using a3c learning and residual recurrent neural networks," *IEEE Transactions on Mobile Computing*, vol. 21, no. 3, pp. 940–954, 2022.

[37] V. Petrucci, O. Loques, D. Mossé, R. Melhem, N. A. Gazala, and S. Gobriel, "Energy-efficient thread assignment optimization for heterogeneous multicore systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 1, pp. 1–26, 2015.

[38] A. P. Kuruvila, X. Meng, S. Kundu, G. Pandey, and K. Basu, "Explainable machine learning for intrusion detection via hardware performance counters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4952–4964, 2022.

[39] M. Brocanelli and X. Wang, "Hang Doctor: Runtime detection and diagnosis of soft hangs for smartphone apps," in *Proc. of the 13th EuroSys Conference*, ser. EuroSys '18. New York, NY, USA: ACM, 2018, pp. 6:1–6:15.

[40] 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 (SEC 2019)*, 2019.

[41] 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.

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

[43] GUROBI OPTIMIZATION, LLC, "Gurobi optimizer," https://www.gurobi.com/solutions/gurobi-optimizer/?campaignid=193283256&adgroupid=138872523520&creative=596136082932&keyword=gurobi%20solver&matchtype=e&_bn=g&gclid=Cj0KCQjwhfipBhCqARIsAH9msbkX_kY3H3xXXbvxfLdngFzmrp1F_f5mTARVRpUg99yc1TmJOe3-ywkaAs5lEALw_wcB,, 2023.

[44] F. Bourgeois and J.-C. Lassalle, "An extension of the munkres algorithm for the assignment problem to rectangular matrices," *Communications of the ACM*, vol. 14, no. 12, pp. 802–804, 1971.

[45] 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.

[46] Gurobi, "Free licenses for academics & recent graduates," https://www.gurobi.com/academia/academic-program-and-licenses/, 2023.

[47] S. T. Atik, A. S. Chavan, D. Grosu, and M. Brocanelli, "Mtc repository - available after publication," 2023.

[48] NAGADOMI, "Mipcl solver," https://www.kaggle.com/code/nagadomi/mipcl-example-only-preference,, 2019.

[49] SCIP Optimization Suit, "Scip," https://www.scipopt.org/,, 2023.

[50] GNU, "Glpk (gnu linear programming kit)," https://www.gnu.org/software/glpk/,, 2012.

[51] COIN-OR Foundation, "Coin-or optimization suite binaries," https://www.coin-or.org/downloading/,, 2016.

[52] keikland, peno64, "lpsolve," https://sourceforge.net/projects/lpsolve/,, 2021.

[53] Battery University, "How to prolong lithium-based batteries," https://batteryuniversity.com/learn/article/how_to_prolong_lithium_based_batteries, 2019.
[54] B. Xu, A. Oudalov, A. Ulbig, G. Andersson, and D. S. Kirschen, "Modeling of lithium-ion battery degradation for cell life assessment," *IEEE Transactions on Smart Grid*, vol. 9, no. 2, pp. 1131–1140, 2018.
[55] G. P. Das, T. M. McGinnity, S. A. Coleman, and L. Behera, "A distributed task allocation algorithm for a multi-robot system in healthcare facilities," *Journal of Intelligent & Robotic Systems*, vol. 80, pp. 33–58, 2015.
[56] J. Blankenburg, S. B. Banisetty, S. P. H. Alinodehi, L. Fraser, D. Feil-Seifer, M. Nicolescu, and M. Nicolescu, "A distributed control architecture for collaborative multi-robot task allocation," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, 2017, pp. 585–592.
[57] S. Giordani, M. Lujak, and F. Martinelli, "A distributed algorithm for the multi-robot task allocation problem," in *Trends in Applied Intelligent Systems: 23rd International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2010, Cordoba, Spain, June 1-4, 2010, Proceedings, Part I 23*. Springer, 2010, pp. 721–730.
[58] L. Luo, N. Chakraborty, and K. Sycara, "Distributed algorithms for multirobot task assignment with task deadline constraints," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 3, pp. 876–888, 2015.
[59] K. Lu, G. Li, and L. Wang, "Online distributed algorithms for seeking generalized nash equilibria in dynamic environments," *IEEE Transactions on Automatic Control*, vol. 66, no. 5, pp. 2289–2296, 2021.
[60] G. Mitsis, E. E. Tsiropoulou, and S. Papavassiliou, "Price and risk awareness for data offloading decision-making in edge computing systems," *IEEE Systems Journal*, vol. 16, no. 4, pp. 6546–6557, 2022.
[61] L. Luo, N. Chakraborty, and K. Sycara, "Distributed algorithms for multirobot task assignment with task deadline constraints," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 3, pp. 876–888, 2015.
[62] Z. Sun, G. Sun, Y. Liu, J. Wang, and D. Cao, "Bargain-match: A game theoretical approach for resource allocation and task offloading in vehicular edge computing networks," *IEEE Transactions on Mobile Computing*, pp. 1–18, 2023.
[63] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, and Y. Yang, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 3, pp. 515–529, 2020.
[64] I. Jang, H.-S. Shin, and A. Tsourdos, "Anonymous hedonic game for task allocation in a large-scale multiple agent system," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1534–1548, 2018.
[65] T. E. Carroll and D. Grosu, "Distributed algorithmic mechanism design for scheduling on unrelated machines," *Journal of Parallel and Distributed Computing*, vol. 71, no. 3, pp. 397–406, 2011.

BIOGRAPHIES

Syeda Tanjila Atik is a Ph.D. candidate in the Department of Computer Science at Wayne State University and a student member of the Energy-aware Autonomous Systems Lab (EAS-Lab). She earned her B.Sc. and M.Sc. degrees in Information Technology from Jahangirnagar University, Bangladesh in 2015 and 2017. Her research interests are in the area of edge computing, Internet of Things, autonomous mobile robots, and machine learning.

Akshar Chavan is currently a Ph.D. candidate in the Computer Science Department at Wayne State University. He earned his Bachelor of Engineering degree in Mechanical Engineering from the Mumbai University, India, in 2014 and Masters of Science degree in Industrial Engineering from Wayne State University, USA in 2020. His research interest areas are in the area of Energy Aware Computing Systems, Autonomous Mobile Robots, Mobile Computing, and Cloud Computing, parallel and distributed computing.

Daniel Grosu (Senior Member, IEEE) received the Diploma in engineering (automatic control and industrial informatics) from the Technical University of Iaşi, Romania, in 1994 and the MSc and PhD degrees in computer science from the University of Texas at San Antonio in 2002 and 2003, respectively. Currently, he is an associate professor in the Department of Computer Science, Wayne State University, Detroit. His research interests include parallel and distributed computing, approximation algorithms, and topics at the border of computer science, game theory and economics. He has published more than one hundred peer-reviewed papers in the above areas and is an IEEE Computer Society Distinguished Contributor. He serves as associate editor and member of the editorial boards of *ACM Computing Surveys*, *IEEE Transactions on Parallel and Distributed Systems*, and *IEEE Transactions on Cloud Computing*. He is a senior member of the ACM, the IEEE, and the IEEE Computer Society.

Marco Brocanelli (Member, IEEE) is an Assistant Professor in the Department of Electrical and Computer Engineering at The Ohio State University and the director of the Energy-aware Autonomous Systems Lab (EAS-Lab). He received the B.E. and M.E. degrees in Control Systems from the University of Rome Tor Vergata (Italy). In August 2018, he received the Ph.D. degree in the Electrical and Computer Engineering program of The Ohio State University. His research interests are in the area of cyber-physical systems, energy-aware systems, Internet of Things (IoT), edge computing, embedded and real-time systems. In 2022 he received the CAREER award from NSF.