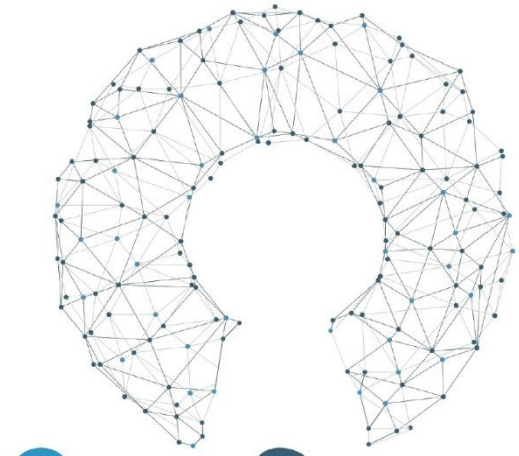

www.openswarm.eu

Call: HORIZON-CL4-2022-DATA-01

Type of action: RIA

Grant agreement: 101093046

Deliverable n°3.4 : Study on the Accuracy vs. Overhead Trade-offs of Energy-Aware Scheduling Methods for Swarm Objectives

Work Package n°3: Collaborative energy-aware AI

imec

WP Lead: imec

This project has received funding from the European Union's Horizon Europe Framework Programme under Grant Agreement No. 101093046.

Funded by the European Union

Document information	
Author(s)	Mohmmadsadegh Mokhtari, Ritesh Kumar Singh, Jeroen Famaey
Reviewers	Micael Couceiro
Submission date	31-Oct-2024
Due date	31-Oct-2024
Type	Report
Dissemination level	PU

Document history			
Date	Version	Author(s)	Comments
27-Sep-2024	01	Mohmmadsadegh Mokhtari	Preview
31-Oct-2024	02	Mohmmadsadegh Mokhtari	Deliverable

Table of contents

Executive Summary

This deliverable presents the outcomes of *Task T3.4 of WP3 (Energy Aware Scheduling methods for swarm)*. The research explores AI-based Energy Harvesting Aware Path Planning and Energy-Aware (EA) Multi-Robot Task Scheduling for ambiently powered robot swarms. The focus is on optimizing energy consumption while ensuring reliable task completion and maximizing operational efficiency.

The first part of the research investigates Energy-Aware Multi-Robot Scheduling using Heuristic Optimization methods for Ambiently-Powered Robot Swarms. This work integrates energy harvesting awareness into the task-scheduling process, aiming to minimize energy consumption and ensuring timely task completion. The proposed approach employs a centralized and autonomous architecture to handle Sequence-Dependent Setup Time (SDST) job shop scheduling demands. Key strategies include energy consumption estimation, charging contingency planning, and energy harvesting prediction, all of which minimize overall task execution time. To optimize the problem, Adaptive Particle Swarm Optimization (APSO), Particle Swarm Optimization (PSO), and Genetic Algorithm (GA) have been adopted. A case study was conducted using a Heterogeneous Pick Drop Delivery (HPDD) scenario inside a warehouse. The study was conducted utilizing the TurtelBot3 burger robot model within the Robotic Operating System (ROS) and Gazebo simulation environment. Simulation results demonstrated a 15% reduction in task completion time using the EA solution for multi-robot scheduling and task allocation problems, compared to Energy-Unaware (EU) methods, thus validating the approach's effectiveness.

The second part of the research focuses on Energy-Aware Path Planning for swarm robotics, where tasks are dynamically allocated based on energy availability. This work proposes the use of meta-heuristic methods, like GA and APSO for optimal EA task scheduling , and reinforcement learning (RL) for AI-based Energy Harvesting Aware Path Planning. The RL-based intelligent navigation algorithm aims to balance the objectives of minimizing travel distance and maximizing energy harvested. A comparative study was conducted using various RL techniques in simulated environments, evaluating performance based on several criteria. The results of this comparative study provide valuable insights into selecting optimal RL techniques for energy-efficient swarm robotics. Overall, the integration of energy harvesting into swarm navigation significantly enhances both the autonomy and mission success of robotic swarms, particularly in environments where recharging opportunities are limited.

1. Introduction

The advancements in robotics have led to the development of swarm robotics, where multiple robots work collaboratively to accomplish certain tasks efficiently. Swarm robotics shows significant potential in various applications, including pick-up and delivery services. For instance, in a factory setting, delivery robots can transport goods and materials from one point to another. However, efficient task scheduling in a swarm of delivery robots presents a challenging problem due to the dynamic environment and the need to handle heterogeneous tasks.

Multi-Robot Task Allocation (MRTA) algorithms are often customized to address specific challenges encountered by multi-robot teams within their operational environment. A major challenge for current multi-robot systems is the limited capacity of their batteries, which act as the primary power source. Researchers are exploring various solutions to address this limitation [28], especially for applications that require efficient utilization of robots over extended durations [26].

Energy-Aware (EA) scheduling in a swarm of robots involves scheduling multiple agents to accomplish tasks while optimizing energy consumption [21]. Robots equipped with energy harvesting capabilities are particularly suited for this approach, as they can collect energy from their environment [14], using mechanisms like solar panels, wind turbines, or kinetic energy converters. These energy-harvesting robots reduce their dependency on traditional energy sources. By integrating EA scheduling, the system can dynamically adapt task allocation to energy constraints and task requirements [4], thus minimizing charging times and avoiding battery depletion. This enhances operational reliability and reduces overall task completion time.

Providing energy for a swarm of robots is as crucial as the tasks themselves because it directly affects their endurance and ability to operate over extended periods. In this work, the proposed scheduling algorithm incorporates a predicted model of the energy consumption of individual robots, in conjunction with the total energy consumption of the swarm. Charging is treated as a priority task, which is planned for by the scheduling algorithm.

Given the complexity of scheduling problems - many of which are NP-hard [25] - a common approach to optimizing these tasks is the use of heuristic methods. There must be a balance between the computational burden and the schedule timing to ensure a compromise between precise solutions and fast response times. Selecting the appropriate optimization algorithm is critical [31]. Among these, Adaptive Particle Swarm Optimization (APSO) has demonstrated efficiency due to its multiple adaptive controls, resulting in faster convergence and more precise solutions, especially for complex optimization problems [33].

To address the challenges of resource management and task scheduling constraints, this work proposes a novel Energy-Aware Sequence-Dependent Setup Time (EASDST) open job shop scheduling method, which also considers

Travel Time (TT). The algorithm collects data from the robots, including current positions and battery levels, and makes predictions about energy harvesting and task completion times in collaboration with the robots themselves. The problem is optimized using APSO, and its performance is compared against the Particle Swarm Optimization (PSO) [8] and Genetic Algorithm (GA) [1]. The objective is to minimize the total task completion time. Although the proposed scheduling algorithm is generalized for multiple use cases, the current focus is on a Heterogeneous Pick, Drop, and Delivery (HPDD) setting in a warehouse. The system is designed and evaluated using ROS [24] and the Gazebo simulator [3].

The work also introduces a distributed scheduling framework in which each device autonomously decides when to execute its available tasks (e.g., sensing, AI-based data processing, or data dissemination). These decisions are based on energy constraints and swarm-wide objectives (e.g., exploring an area or detecting an object). For this, the complex collaborative behaviour of swarms is divided into smaller atomic subtasks, which can be distributed across devices and scheduled individually, as studied in T3.3. The energy consumption and interdependencies between subtasks are modelled and considered by the scheduler. This scheduling problem can be formulated as a distributed Mixed Integer Linear Program (MILP), which, given its NP-hardness, requires fast practical heuristics. As swarm-wide objectives must be achieved, each robot must observe the behaviour of neighbouring robots and adapt accordingly. This can be accomplished through direct observations using sensors or cameras, or through information exchange. This task also investigates the trade-offs between observation methods, balancing accuracy and communication overhead.

2. Key Performance Indicators (KPIs)

The KPI defined for T3.4 for the DoA is **"*target for successful collaborative task execution on field devices > 95%*"**. According to the results in Section 4.3.3, *Large-scale Simulation Results*, the target for successful collaborative task execution on field devices is theoretically 100% and practically (i.e., using the well-known ROS and Gazebo simulators) above 99%. In the evaluation, which included over 1,000 different trials, we did not encounter any incomplete tasks, as the task scheduler accounted for energy drainage and scheduled recharging accordingly. These findings indicate strong reliability and robustness, supporting the feasibility of real-world deployment with minimal risk of task failure.

However, due to an error in the final version of the DoA, another KPI was ultimately listed for T3.4, as ***"target average current draw of field device < 100μA"***. This KPI is instead addressed in Section 7 of D4.1. Here we developed a low-power controller for mobile robots that successfully reduced the power consumption of a sleeping robot to just 26.2 μW (i.e., 6.9 μA considering the 3.8V terminal voltage). This work introduces an Energy Management Module which can manage the robot idle period energy consumption using an ultra-low power Bluetooth connection. This achievement further enhances the energy efficiency of our system, making it more suitable for practical applications.

3. Technology Readiness Level (TRL)

The work on energy-aware task scheduling (Section 6), was at TRL 2 (Technology concept formulated) at project start, and was evolved to TRL 3 (Experimental proof of concept) during the execution of T3.4. The goal by the end of the project is to integrate the energy-aware scheduler into one of the project PoCs, aiming for TRL 4 or 5. This will involve demonstrating the system's functionality in a relevant environment, moving us closer to practical application and validation.

The work on energy harvesting aware path planning is at an earlier stage, and was at TRL 1 (Basic principles observed) at project start. It has been evolved to TRL 2 (Technology concept formulated) during the execution of T3.4.

4. List of publications

- M. Mokhtari, P. Haji Ali Mohamadi, M. Aernouts, R. Kumar Singh, B. Vanderborght, M. Weyn, J. Famaey, Energy-Aware Multi-Robot Task Scheduling using Meta-Heuristic Optimization methods for Ambiently-Powered Robot Swarms. Robotics and Autonomous Systems ,March 2024. (Under Review, Revision Submitted)

5. Background

To understand the methods and results presented in this work, this chapter provides an overview of important topics and several key concepts..

5.1.Ambiently-Powered Robot Swarms

Ambiently-powered robotic swarms refer to groups of robots that harvest energy from their surroundings, such as solar, thermal, or Radio Frequency (RF) energy. These robots are often deployed in scenarios where continuous recharging from traditional sources is infeasible, such as remote areas, hazardous environments, or space exploration. By autonomous harvesting energy, they operate for extended durations, completing tasks in areas with limited or no human intervention, or where conventional energy infrastructure is absent.

5.2. AI-based Energy Harvesting Aware Path Planning

Path planning for robotic swarms typically focuses on determining optimal routes to complete tasks while minimizing energy consumption and maximizing efficiency. In the context of ambiently-powered robots, path planning must account for fluctuating energy availability from the environment. AI-based path planning, particularly machine learning and deep learning, enhance this dynamically adjusting robot paths to optimize both energy consumption and task completion based on predicted energy availability. To achieve this primary goal, robots are guided through routes that allow them to collect energy while maintaining task completion efficiency. The challenge lies in balancing between task accomplishment reliability and the constraints of harvested energy. Environmental conditions (e.g., cloud cover affecting solar energy) can cause energy levels to fluctuate, requiring dynamic adjustments. Therefore, AI models that can predict energy availability by analyzing environmental factors and historical data need to be taken into account. Deep reinforcement learning and neural networks can help the swarm adapt in real-time, allowing robots to recharge opportunistically during missions and adjust paths based on energy collection opportunities.

5.3. Energy-Aware Multi-Robot Task Scheduling

Energy-aware multi-robot task scheduling allocates tasks to multiple robots while accounting for their current energy levels, task priorities, and energy harvesting potential. In ambiently-powered swarms, additional complexities arise, as tasks

may need to be reassigned dynamically based on robots' energy reserves and charging needs.

The goal here is to maximize the overall task accomplishment while minimizing the swarm task completion time. Tasks must be assigned in a way that considers both the current energy status of each robot and its potential to harvest energy during task execution. Meta-heuristic optimization methods, such as GA, PSO, and ant colony optimization (ACO), are commonly applied to this problem. These methods explore vast solution spaces to find optimal or near-optimal task allocation strategies that account for the dynamic and uncertain nature of ambient energy harvesting. Additionally, RL can be used to adapt the swarm to its environment in real-time.

5.4. Energy Consumption Reduction and Environmental Impact

Managing energy consumption is critical for ambiently-powered swarms, as their energy comes from unpredictable and sometimes scarce sources. AI-based path planning and task scheduling, combined with meta-heuristic optimization, help minimize the overall energy consumption of the swarm.

Ambiently-powered swarms reduce reliance on fossil fuels or battery replacement, lowering their environmental impact. By harvesting energy from renewable sources, these swarms can operate sustainably for long-term missions, significantly reducing operational costs by eliminating the need for manual recharging or battery replacements. Additionally, optimized task scheduling and path planning ensure that energy usage is kept to a minimum, preventing unnecessary expenditures on energy and maximizing the lifespan of robots.

5.5. Reinforcement learning

Machine learning (ML) is a field of computer science that enables computers to learn autonomously without explicit programming [83, 85]. At its core, there are three main groups of algorithms: supervised learning, unsupervised learning and RL [78].

In the first two categories of machine learning, data is gathered before the actual learning begins. In supervised learning, the algorithm learns from a labelled dataset. This means the model is trained with data consisting of input and the corresponding output. The model learns to make accurate predictions on new, unseen data by mapping the inputs to the outputs of the known labelled data. In unsupervised learning the algorithm learns from unlabelled data. Here, the model trains to find patterns, structures or groupings within the given data without any explicit guidance.

The last category of machine learning is RL, which also requires data to learn from. However, unlike the previous categories, this data not collected beforehand but instead is generated during the learning process [62].

RL is a type of ML technique that enables an agent to learn in an interactive environment using feedback from its own actions and experiences [70, 48]. In Figure 1 you will find a schema of a basic RL algorithm [43]. As you can see, there is an agent in a current state, that can perform an action. The environment will then respond to that action by providing a new state and interpreting that new state to generate a reward with the help of a reward function. This reward is what the agent tries to maximize which means it is what the agent will use to learn. This new state and reward are then passed along back to the agent, so it can decide what the next action will be and the learning process can continue.

The strategy the agent uses to decide which action it will take is called the policy. The purpose of an RL agent is to learn an optimal, or near-optimal, policy that maximizes the reward function. To build an optimal policy, the agent faces the Exploration vs Exploitation dilemma [62, 48]. On one hand, the agent wants to explore new states in the environment, allowing it to improve its current knowledge about each action (Exploration).

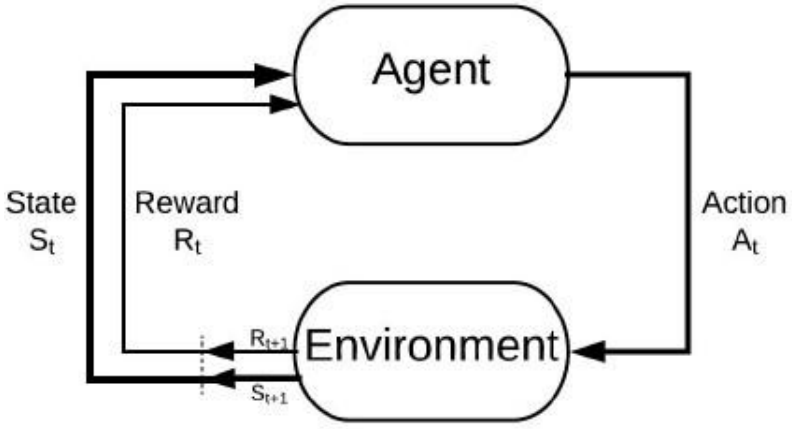

Figure 1. *Reinforcement Learning Diagram [43]*

And on the other hand, the agent also wants to maximize the overall reward (Exploitation). However, being greedy and only focusing on maximizing rewards may not actually get the most reward and lead to sub-optimal behavior. It is thus essential to balance between both exploration and exploitation.

There are three different types of RL implementations acknowledged: policy-based, value based, and model-based [43]. Policy-based RL focuses on developing a policy, or a deterministic/stochastic strategy, to maximize the cumulative reward. Next, value-based RL aims to maximize a value function that represents the expected reward. Lastly, model-based RL involves constructing a virtual model of a particular environment, allowing the agent to learn and operate within the environment's constraints. Another important way to categorize RL algorithms is by whether they are model-free or model-based [84]. Model-based RL builds a model of the environment to simulate outcomes before actions are taken.

Specific challenges in RL have led to the development of advanced techniques, like multi-objective RL [57], which handles conflicting objectives, and distributed RL [54], which involves multiple agents learning concurrently in a shared environment. Another technique, hierarchical RL [46], decomposes complex tasks into smaller, manageable subtasks.

Q-Learning [52] is a popular, value-based, model-free RL. At its core, Q-learning learns to estimate the value of taking a certain action in a particular state. The algorithm keeps those values in a Q-table for all possible state-action pairs [45].

Initially, the agent knows nothing and will have an empty Q-table, where each row is a state, and each column is an action. The agent will then explore the environment by taking actions. After taking an action, the agent observes the new state and reward and updates the q-value for the state-action pair using the Bellman equation.

We can also describe the Q-learning algorithm as a mathematical model [66]. We start by formally defining the attributes.

- **State (s):** An observation of the environment at a specific time.
- **Action (a):** A possible action that an agent can take in state s.
- **Reward (r):** The reward received after taking action a in state s, transitioning to state s'.
- **Q-value** $Q(s,a)$**:** The function that estimates the expected future reward of taking action a in state s.
- **Learning rate** α**:** The rate at which the agent updates the Q-values.
- **Discount factor** γ**:** A factor between 0 and 1 that discounts future rewards.

The Q-learning update rule is then given by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right) \quad (1)$$

Where:

- s_t is the current state at time t.
- a_t is the action taken in the current state at time t.
- r_t is the reward received after taking action a_t.
- s_{t+1} is the next state at time $t + 1$.
- $\max_{a'} Q(s_{t+1},a')$ is the maximum Q-value for the next state across all possible actions a'.
- α is the learning rate.
- γ is the discount factor.

5.6. Deep Reinforcement Learning

A problem arising in basic RL is that the state spaces in an environment can quickly become extremely large making it impractical for an agent to learn an effective policy by storing information about every possible state explicitly. As a solution, a Deep Neural Network can be used to approximate crucial components of the

learning process, such as policies or value functions [75, 42]. Instead of directly mapping states to actions using a tabular method, the neural network generalizes across similar states, enabling the agent to make decisions even in complex, high-dimensional environments.

Proximal Policy Optimization (PPO) [73] is a widely used, policy-based, model-free deep RL algorithm, and is the second algorithm utilized in this work. The three main advantages of PPO in comparison with other RL techniques are simplicity, stability, and sample efficiency, making it a preferred choice for many complex RL tasks.

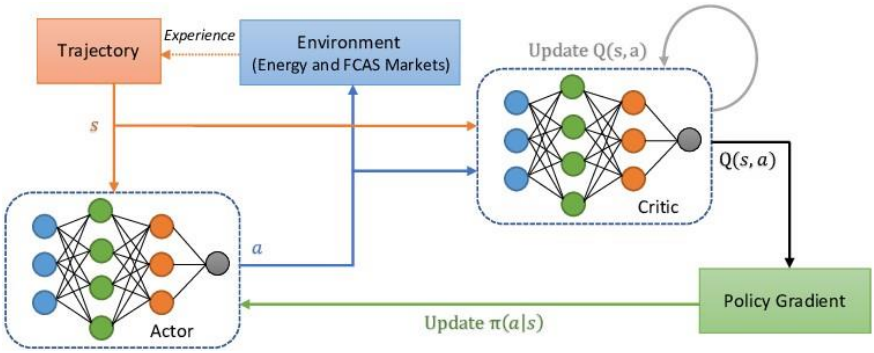

Figure 2. *PPO diagram [60]*

In Figure 2, the general workflow of the PPO algorithm is illustrated [68, 50]. As you can see, the algorithm identifies two main components: the Actor and the Critic. The Actor is a neural network responsible for determining the policy $\pi(a|s)$, which decides what actions to take based on the current state of the environment. The Critic is another neural network that evaluates how good these actions are by estimating the value function $Q(s,a)$ representing the future rewards of taking action a in state s.

The way PPO works starts with the Actor interacting with the Environment by choosing an action. This leads to a transition to a new state and the receipt of a reward which is given to both the Actor and Critic. The Critic network is then updated by minimizing the difference between the predicted value and the actual rewards received. Next, the Critic's output is used to compute the Policy Gradient which is in its turn used to update the Actor's policy.

This process loops continuously: the actor interacts with the environment, the critic evaluates the actions, and both networks are updated to improve the agent's performance over time.

Another critical component of PPO, that is not explicitly shown in the diagram, is called the clip function. It plays a vital role in the policy update process by ensuring that the update does not deviate excessively from the previous policy by limiting the change in the probability ratio. This constraint stabilizes training by preventing large, unstable updates that could degrade the policy's performance. The clip function effectively balances exploration and exploitation, ensuring that the agent learns efficiently without diverging from a reasonably good policy.

We summarize the working of PPO by giving the mathematical model [40].

- **Policy** $\pi_\theta(a|s)$**:** A probability distribution over actions given a state, parameterized by θ.
- **Value function** $V_\theta(s)$**:** An estimate of the expected return (cumulative future reward) from state s following the policy π_θ.
- **Advantage function** $A(s,a)$**:** The difference between the actual return and the estimated value, calculated as:

$$A(s,a) = Q(s,a) - V(s) \quad (2)$$

- **Clipped Objective:** PPO optimizes the following clipped surrogate objective function:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t,\ \text{clip} \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) A_t \right) \right] \quad (3)$$

Where:

- $\pi_\theta(a_t|s_t)$ is the policy's probability of taking action a_t in state s_t, under parameters θ.
- $\pi_{\theta_{\text{old}}}(a_t|s_t)$ is the policy's probability of taking action a_t in state s_t, under the old parameters θ_{old}.
- A_t is the advantage function at time step t.
- ϵ is a hyperparameter controlling the range for clipping.

Deep Q-Networks (DQN) [59] is one of the pioneering algorithms in deep RL that combines Q-Learning with deep neural networks to handle environments with high dimensional state spaces [79]. DQN leverages a neural network, often referred to as the Q-network, to approximate the Q-values $Q(s,a)$ for each state-action pair. Instead of maintaining a large Q-table as in traditional Q-Learning, DQN uses the Q-network to predict Q-values for all possible actions given the current state.

The DQN algorithm also introduces several key techniques to stabilize learning, such as experience replay and target networks [38]. Experience replay involves storing transitions (s,a,r,s') in a replay buffer and sampling mini-batches to update the Q-network, which helps break the correlation between consecutive transitions. The target network which is a copy of the Q-network, is updated less frequently to provide more stable target values during training.

Mathematically, the Q-network is trained by minimizing the loss:

$$L(\theta) = \mathbb{E}(s,a,r,s') \sim \text{Replay Buffer} \left[\left(r + \gamma \max a' Q_{\text{target}}(s',a';\theta^-) - Q(s,a;\theta) \right)^2 \right] \quad (4)$$

Where:

- θ are the parameters of the Q-network.

- θ^- are the parameters of the target network, which are periodically updated to the current parameters θ
- γ is the discount factor.
- $\mathbb{E}$ denotes the expectation over the mini-batch sampled from the replay buffer.

DQN, being a value-based method, differs significantly from Proximal Policy Optimization (PPO), which is a policy-based method. While DQN uses a Q-network to estimate the value of actions and selects actions greedily based on these estimates [79], PPO directly optimizes the policy by adjusting its parameters to maximize the expected reward. PPO generally handles continuous action spaces better and is often more stable due to its use of policy gradients and a clipped objective function to limit policy updates. In contrast, DQN is typically used in discrete action spaces and may struggle in environments with high-dimensional or continuous actions.

5.7. Gymnasium Library

The Gymnasium library is a maintained fork of OpenAI's Gym [74], providing an open-source toolkit designed to test and develop RL algorithms. It offers a standardized API for interacting with a variety of environments, allowing researchers to evaluate different algorithms consistently. Gymnasium allows the creation of custom environments, making it valuable for testing novel RL problems and algorithms. This flexibility supports a wide range of experimental setups in RL research.

5.8. Energy Harvesting

Energy harvesting refers to capturing and storing energy from external sources to power devices [37, 77]. This technology is increasingly critical as it provides sustainable and efficient ways to meet growing energy demands. Despite its widespread relevance, there is no universally accepted definition of energy harvesting. The term broadly encompasses a variety of techniques and technologies that convert different forms of ambient energy into usable electrical energy.

The most relevant energy harvesting methods include:

- ***Solar Energy Harvesting [67]:*** Involves the use of photovoltaic cells to convert sunlight into electricity, being one of the most widely used and mature technologies for energy harvesting. Solar energy harvesting is particularly advantageous in environments where sunlight is consistently available, making it an excellent choice for outdoor applications such as in our problem.
- ***Kinetic Energy Harvesting [58]:*** Captures energy from movements or vibrations, being beneficial in natural settings where wind or water movement is present. It can be realized by converting mechanical stress or

vibrations into electrical energy using piezoelectric materials. This is beneficial when operating in environments with high levels of vibration, such as near machinery or in uneven terrains. But we can also use electromagnetic induction which allows generation of energy from our own movements or the movement of surrounding objects.

- ***Radio Frequency (RF) Energy Harvesting [41]:*** Captures energy from ambient radio waves emitted by communication devices, such as cell towers, Wi-Fi routers, or even other robots in a swarm. The approach utilizes antennas to capture electromagnetic energy from ambient RF signals. RF energy harvesting is useful in urban environments or any area with a dense network of communication signals. However, the power density is usually low, so it might be more suitable as a supplementary energy source rather than the primary one.
- ***Thermophotovoltaic (TVP) Energy Harvesting [65]:*** Involves converting radiant energy from a thermal source into electricity, similar to solar photovoltaic cells, but tuned for thermal radiation instead of sunlight. When operating in high-temperature environments (like near furnaces, industrial sites, or volcanic regions), TPV cells could harness the infrared radiation from these sources to generate power.

Among these methods, solar energy harvesting is the most suitable for mobile robots, given its high efficiency, lightweight nature, and compatibility with outdoor environments. Solar panels are easily integrated into robots, allowing them to harness an abundant energy source, reducing their dependency on traditional energy sources.

6. Energy-Aware Multi-Robot Task Scheduling using Meta Heuristic Optimization methods for Ambiently-Powered Robot Swarms

This section addresses the intricate challenge of energy aware multi-robot task scheduling, allocation, and planning in a robotic swarm. A central issue in such systems is ensuring efficient task scheduling while accounting for the energy constraints of individual robots. Scheduling tasks in such energy-constrained environments requires sophisticated methods that optimize both task completion and energy usage.

Previous studies have explored aspects of energy-aware task allocation. For instance, Gul [7] investigated energy harvesting in MRTA using a PSO algorithm, focusing on energy-related factors like energy consumption and battery utilization. Similarly, Latif et al. [11] proposed an energy-conscious distributed task allocation algorithm, focusing on minimizing charging transit times for robots.. Their approach singularly emphasizes efficiency derived from the energy expended during robot exploration and collection endeavors, mainly focusing on minimizing the charging transit time. However, their analysis overlooks pivotal aspects such as energy harvesting, robot energy models, and battery characteristics. Notably, their optimization objective centers on energy consumption reduction rather than the optimization of task completion time. Milan Tomy [26] presents a battery charge scheduling method for long-life autonomous robots that predicts efficient recharging time based on the workflow probability. Despite the valuable concept of this view, it is more related to workflow prediction and does not take into account the energy resource availability and variability.

The exploration of a PSO-based distributed Algorithm for dynamic task allocation in a robotic swarm, as conducted by Nedjah [20], yields insights into a task allocation algorithm inspired by the PSO methodology. Experimental validation underscores the efficacy of this algorithm in converging toward an optimal task allocation configuration. Notably, the communication aspect is paramount, employing a message structure transmitted by robot identifiers to a base station. Leu [12], introduced a robust task-planning framework for human robot collaboration within assembly lines. Hierarchical task modeling underpins their approach, which captures both sequential and parallel task relationships. While energy considerations are absent in their scheme, their hierarchical approach serves as a pertinent reference point for our task-planning strategy. Prospects of integrating their human-robot collaboration (HRC) framework into our future work emerge as a natural progression.

Further delving into the domain, Vazquez's formal task allocation and scheduling paradigm for multi-robot missions (KANOA) [29] is notable for accommodating task

constraints encompassing task order and joint tasks requiring collaborative robot engagement. However, the energy aspect remains unaddressed. In a similar vein, Xiong's work on swarm game and task allocation for autonomous underwater robots [32] sheds light on the dynamic evolution of robot swarm strategies during game scenarios. Although the primary objective centers on task completion time reduction, the energy facet is notably absent in their considerations. Moreover, Li et al.'s work [13] introduces a novel scheduling mechanism for multi-robot intelligent warehouse systems, leveraging PSO for task allocation. Despite the innovative aspects of their approach, this study notably overlooks crucial energy aspects, such as charging dynamics and real-world robotic characteristics. Wang [31] researched task scheduling for heterogeneous agents' pickup and delivery using recurrent open shop scheduling models. In this work, different kinds of open-shop scheduling algorithms for pick-up delivery tasks are investigated. This work is more concentrated on scheduling processes regardless of the agent's characteristics and energy considerations.

Certainly, the choice of the right optimization algorithm is pivotal for task scheduling. One approach proposed by the authors in [10] involves using the Hungarian method, a centralized solution that can allocate tasks optimally. However, this approach has some drawbacks, such as slow responses to dynamic changes. To address this issue, more distributed algorithms were proposed, including an auction-based method proposed in [19] that rebids uncompleted tasks every time a robot completes its previously assigned task. The authors in [31] adopted a specific Append-Beam-Christofides (ABC) heuristic, greedy local search (GLS), and simulated annealing (SA) to optimize the heterogeneous agent's pickup and delivery using recurrent open shop scheduling models. Additionally, [6] proposed a distributed solution to a multi-robot task allocation problem, and [27] introduced a distributed market based MRTA algorithm. Finally, the authors in [16] developed a pairwise distance-based matching algorithm to solve the dispatch problem in a multi-robot system. Several studies have been conducted to address the problem of assigning tasks to robots with different optimization criteria. For instance, Luo et al. [17] focused on the task-assignment problem with task-deadline constraints, while [18] formulated a single-robot task, single-task robot, time extended assignment, and MRTA problem with multiple nonlinear criteria. The proposed techniques in these studies used discrete variables to reduce computation burden and tackled low and large-scale MRTA problems using Branch and Bound (B&B) and GA, respectively. The MRTA techniques considered multiple optimization criteria such as traveled distance, task execution time, and energy consumption feasibility.

In summary, this research aims to bridge the gaps in previous studies by introducing a comprehensive framework for energy-aware multi-robot task scheduling. By integrating energy considerations into the scheduling process, we improve task efficiency and reduce the risk of battery depletion, resulting in faster task completion times and better overall resource management in robot swarms.

6.1. Problem statement

The problem scenario involves a set of tasks, a fleet of robots, and strategically positioned charging stations. The primary goal is to solve the Energy-Aware Sequence-Dependent Setup Time (EASDST) scheduling problem within the Heterogeneous Pick, Drop, and Delivery (HPDD) task structure. A detailed formulation and notation for this problem are provided in the next sections.

6.1.1. Set of Robots

In this problem, robots are programmable entities responsible for task execution. Each robot has a unique set of capabilities, including energy harvesting, navigation, and object manipulation. The set of N robots is represented as $\mathbb{R} = r_1, r_2, \dots, r_N$, where each robot $r \in \mathbb{R}$ has the following attributes:

- p_r^0: Start location.
- E_r^0: Initial energy level.
- E_{max}: Maximum battery capacity.

The location of robot r at any time t is defined as $p_r^t = (x_r^t, y_r^t, z_r^t)$. The energy harvested by robot r at a specific location p_r^t and time t is denoted as $H_{\text{r}}(t, p_r^t)$. The total energy harvested by robot r between time t_0 and t_1 is calculated as:

$$E_{harvest}(t_0, t_1) = \int_{t_0}^{t_1} H_{\text{r}}\,(t, p_r^t)\, dt \quad (1)$$

6.1.2. Energy and Time Variables

To ensure accurate predictive scheduling, we consider the energy and time consumption associated with each task. Specifically, these components consist of:

- E_{pick}: Energy required for the robot to pick up a package.
- T_{pick}: Time taken to pick up a package.
- E_{drop}: Energy required for the robot to drop off a package, including actions like unloading.
- T_{drop}: Time taken to drop off a package.
- $E_{\text{drive}}(p_1, p_2)$: Energy required for the robot to travel from location p_1 to location p_2.
- $T_{\text{drive}}(p_1, p_2)$: Time taken for the robot to travel from p_1 to p_2.

6.1.3. Set of Tasks

As mentioned before, the tasks are based on a warehouse setting. Each order $m \in \mathbb{M} = m_1, m_2, \dots, m_x$ contains a set of tasks $O_m = o_m^1, o_m^2, \dots, o_m^j$, with M tasks in total. Each task $o \in O_m$ is a drop with a specific weight w_o, picked from $p_o^\circ = (x_o^\circ, y_o^\circ, z_o^\circ)$ and dropped at $p_o^1 = (x_o^1, y_o^1, z_o^1)$. Orders also have deadlines, with td_m representing the deadline for order m.

The total energy required to complete task o is:

$$E_o = E_{\text{pick}} + E_{\text{drive}}(p_o^0, p_o^1) + E_{\text{drop}} \quad (2)$$

Similarly, the total time needed to complete task o is:

$$T_o = T_{\text{pick}} + T_{\text{drive}}(p_o^0, p_o^1) + T_{\text{drop}} \quad (3)$$

Different uncertainties, like other robots' movement and obstacles, need to be considered when calculating E_o and T_o. To predict these two variables, we used a predictive approach, leading to $\hat{E}_o$ and $\hat{T}_o$.

6.1.4. Set of Charging Stations

Although robots can harvest ambient energy, some tasks may require energy beyond what is harvested. Therefore, charging stations $\mathbb{C} = c_1, c_2, \dots, c_l$ are included in the system. Each charging station $c \in \mathbb{C}$ is located at $p_c = (x_c, y_c, z_c)$. The time required to charge a robot's battery from energy level E_0 to E_1 is given by $T_{\text{charge}}(E_0, E_1)$.

6.1.5. Decision Variables

To achieve task scheduling, specific decision variables are defined. Each robot r is assigned a sorted list of k tasks from each order m, represented as $O_{r,m} = o_1, o_2, \dots, o_k$, where $O_{r,m} \subseteq O_m$. No task can be assigned to more than one robot, ensuring:

$$O_{r,m} \cap O_{q,m} = \emptyset \quad \forall q \in \mathbb{R}\{r\} \quad (4)$$

6.1.6. State Variables

To account for the dynamic task allocation process, we define state variables that continuously update the robot's location, energy, and timing. The energy levels for robot r at the start and end of task o_i, with $0 \leq i \leq k$, are denoted as $E_{\text{start}}^r(o_i)$ and $E_{\text{end}}^r(o_i)$, respectively:

$$\begin{aligned} E_{\text{start}}^r(o_0) &= E_r^0 \\ E_{\text{start}}^r(o_i) &= E_{\text{end}}^r(o_{i-1}) \end{aligned} \quad (5)$$

If robot r lacks sufficient energy to complete a task o_i, i.e.,

$$E^r_{\text{start}}(o_i) < E_{\text{drive}}\left(p^1_{o_{i-1}}, p^0_{o_i}\right) + E_{o_i} \quad (6)$$

it will first recharge at the nearest charging station c_i:

$$\begin{aligned} c_i \quad = \underset{c \in C_r}{\text{argmin}} \big(& T_{\text{drive}}\left(p^1_{o_{i-1}}, p_c\right) + T_{\text{charge}}(E^r_c, E_{\text{max}}) \\ & + T_{\text{drive}}\left(p_c, p^0_{o_i}\right) \big) \end{aligned} \quad (7)$$

$\mathcal{C}_r$ is the set of charging stations that can be reached by r, given its current energy supply, i.e.,

$$\forall c \in \mathcal{C}_r : E^r_c > 0,$$

where E^r_c represents the available energy of robot r when it reaches recharging station c, i.e.,

$$\begin{aligned} E^r_c = \quad & E^r_{\text{start}}(o_i) - E_{\text{drive}}\left(p^1_{o_{i-1}}, p_c\right) \\ & + E^r_{\text{harvest}}\left(T^r_{\text{start}}(o_i), T^r_{\text{start}}(o_i) + T_{\text{drive}}\left(p^1_{o_{i-1}}, p_c\right)\right) \end{aligned} \quad (8)$$

We can then calculate $E^r_{\text{end}}(o_i)$ as follows:

$$\begin{aligned} E^r_{\text{end}}(o_i) \quad = E^r_{\text{max}} & + E^r_{\text{harvest}}\left(T^r_{\text{start}}(o_i), T^r_{\text{end}}(o_i)\right) \\ & - E_{\text{drive}}\left(p_{c_i}, p^0_{o_i}\right) - E_{o_i} \end{aligned} \quad (9)$$

Otherwise, if the robot has enough energy to complete the task without charging, the end energy $E^r_{\text{end}}(o_i)$ can be computed as follows:

$$\begin{aligned} E^r_{\text{end}}(o_i) \quad = E^r_{\text{start}} & + E^r_{\text{harvest}}\left(T^r_{\text{start}}(o_i), T^r_{\text{end}}(o_i)\right) \\ & - E_{\text{drive}}\left(p^1_{o_i}, p^0_{o_i}\right) - E_{o_i} \end{aligned} \quad (10)$$

To continually monitor and update the temporal time status of robots during scheduling to measure the problem objective function, for each robot r, the start and end time of a task o_i, with $0 \leq i \leq k$, as $T^r_{\text{start}}(o_i)$ and $T^r_{\text{end}}(o_i)$, are defined as:

$$\begin{aligned} T^r_{\text{start}}(o_0) \quad &= 0 \\ T^r_{\text{start}}(o_i) \quad &= T^r_{\text{end}}(o_{i-1}) \end{aligned} \quad (11)$$

$T^r_{\text{end}}(o_i)$ can be calculated as follows:

$$T^r_{\text{end}}(o_i) = T^r_{\text{start}}(o_i) + T^r_{setup} + T_{o_i} \quad (12)$$

where T^r_{setup} is the Travel Time (TT) between the task o_{i-1} and o_i for robot r.

$$T^r_{setup} = \begin{cases} T_{\text{drive}}(p^1_{o_{i-1}}, p^0_{o_i}) & \text{if } C_r \neq \emptyset \\ T_{\text{drive}}(p^1_{o_{i-1}}, p_c) + T_{\text{charge}}(E^r_c, E_{\max}) & \\ +T_{\text{drive}}(p_c, p^0_{o_i}) & \text{if } C_r = \emptyset \end{cases} \quad (13)$$

6.1.7. Objective Function

After task assignment, we aim to minimize the longest time taken by any robot to complete its tasks. Mathematically it can be formulated as below:

$$min\left(\max_{r\in\mathbb{R}, o\in O_{r,m}} \left(T^r_{end}(o)\right)\right) \quad (14)$$

The goal is to assign a subset of tasks $O_{r,m}$ to each robot r, which minimizes the objective function defined in the equation above, under the following constraints:

- **Task Completion:** Each assigned order must be completed before its deadline, underscoring the importance of timing in this process:

$$\forall m \in \mathbb{M}: \max_{o\in O_m}\left(T^r_{end}(o)\right) \leq td_m) \quad (15)$$

- **Energy Constraints:** Robots must have sufficient battery to reach the nearest charging station, being pivotal in this work due to the energy-aware features of task scheduling:

$$\forall r \in \mathbb{R}, \forall o \in O_m: E^r_{\text{end}} \geq E_{r_c} \quad (16)$$

  where E_{r_c} is the energy that the robot needs to reach the nearest charging station.

- **Sequential Task Completion:** Orders must be completed sequentially, though tasks can be executed concurrently in an open job shop fashion:

$$\forall m_i \in \mathbb{M}: T^{m_i}_{\text{end}} > T^{m_{i-1}}_{\text{end}} \quad (17)$$

  where $T^{m_i}_{\text{end}}$ is the

$$T^{m(r,i)}_{end} = \max\left(\sum_{j=1}^{m}\left(\sum_{i=1}^{k} T_{O_i} \mid o_i \in O_{r,m}\right)\right) \quad (18)$$

$$T^{mi}_{end} = \max\left(T^{m(r,0)}_{end}, T^{m(r,1)}_{end}, \ldots, T^{m(r,i)}_{end}\right) \quad (19)$$

6.2. Optimization algorithm and energy modelling

6.2.1. APSO optimization algorithm

Scheduling problems, particularly those involving multi-robot task allocation, are widely recognized as NP-hard. This complexity means that finding optimal solutions in polynomial time is often infeasible, especially as the problem size increases. Metaheuristic algorithms, such as APSO, offer a practical approach for finding high-quality solutions within reasonable computational limits, even though they do not guarantee global optimality.

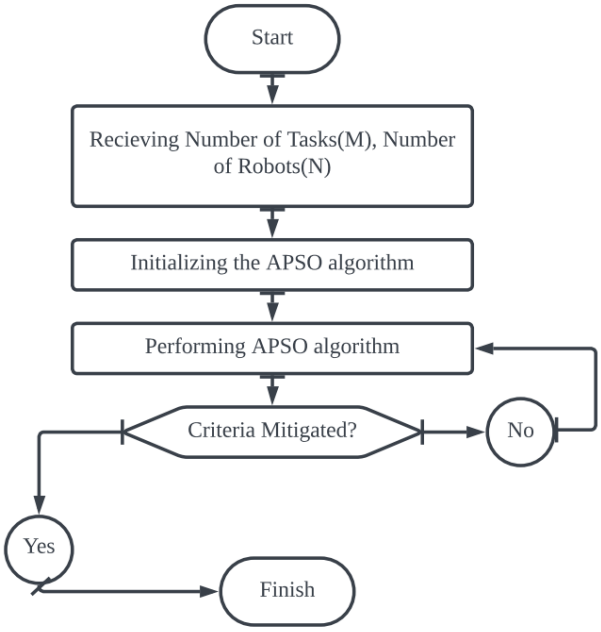

Figure 3. *APSO optimization flowchart. After receiving the number of robots N and tasks M the required parameters for starting the APSO process are set. APSO optimizes the EASDST algorithm. Once the stopping criteria are mitigated the process terminates.*

Figure 3 shows an overview of the APSO-based approach used to optimize the EASDST scheduling algorithm. going through the following stage: (i) initialization; (ii) fitness evaluation; (iii) velocity update; (iv) distance and evolutionary factor; (v) adaptive parameter control; and (vi) position update.

Initialization

During the initial stage, the scheduling algorithm is provided with data, including the set of robots, set of tasks, set of charging stations, and the initial value of the state variables as elaborated in section 3. Key parameters for the APSO algorithm, such as swarm size S, inertia weight W, acceleration coefficients $c1$ and $c2$, and the stopping criterion for iterations I_{iter}, are initialized. Additionally, the initial position

of each particle p_i^0 is set randomly. The swarm consists of S particles, each representing a potential solution to the task scheduling problem.

Each particle in the swarm is represented as an array of length equal to M (the total number of tasks). The first row of the particle array P_i^t contains the number of tasks from order m_i while the second row shows the robot ID r_i assigned to each corresponding task $o_m^i \in O_m$. Figure 4 demonstrates the particle configuration, illustrating how the APSO algorithm assigns robots to tasks in a sequential order.

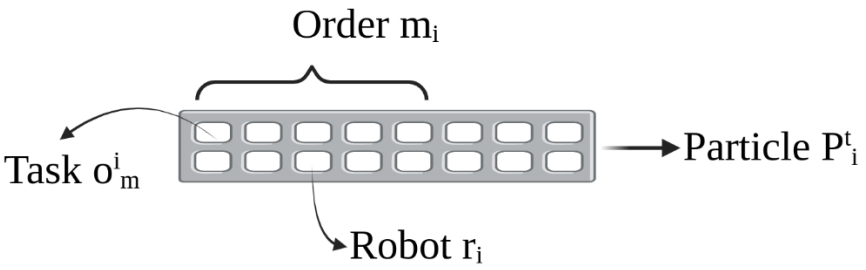

Figure 4. *Particle array structure. This Figure shows the particle configuration. It shows how the APSO algorithm assigns the robot to each task in each sequential order.*

Fitness Evaluation

After initialization, the fitness function Algorithm 2 is employed to evaluate each particle's performance. The objective function, described in Section 3.3, is used to calculate the overall value $F_{fitness}$ for each particle P_i^t. The best-performing particle, denoted as g_b^t, represents the global best solution found by the swarm. In each iteration, the algorithm compares the current solution of all S particles with the global best solution, upgrading g_b^t whenever a particle's performance exceeds that of the previous best.

Velocity Update

Before proceeding to the next iteration, the APSO algorithm updates particles' velocities based on the best particle g_b^t position. This is key to balancing exploration (searching for new areas of the solution space) and exploitation (refining known good solutions). The velocity of each particle is updated using the the velocity update equation of the traditional PSO algorithm:

$$V_i^{t+1} = W \cdot V_i^t + c_1 \cdot U_1^t \cdot (g_b^t - p_i^t) + c_2 \cdot U_2^t \cdot (g_b^t - p_i^t) \quad (20)$$

In this equation:

- V_i^{t+1}: Updated velocity of particle i at time $t + 1$.
- W: Inertia weight, controlling the trade-off between exploration and exploitation.

- c_1 and c_2: Acceleration coefficients influencing the particle's tendency to move towards the global best solution.
- U_1^t and U_2^t: Random numbers uniformly distributed between 0 and 1, introducing stochasticity to the search process.
- p_i^t: Current position i of particle P_i^t.

The implemented algorithm uses adaptive rules that can involve monitoring the convergence rate of the optimization process. During the early stages of optimization, W is typically set to a high value to encourage exploration. As the algorithm progresses, W decreases to encourage convergence on the best solution. In APSO, the inertia weight is controlled using the following adaptive equation:

$$W(f) = \frac{1}{1+1.5e^{-2.6f}} \in [0.4,0.9] \quad \forall f \in [0,1] \text{ (21)}$$

Distance and Evolutionary Factor:

To calculate the inertial weight W, the mean distance d_i of each particle P_i to all the other particles is computed using the Euclidian metric:

$$d_i = \frac{1}{(S-1)} \sum_{j=1, j\neq i}^{S} \sqrt{\sum_{k=1}^{D} \left(x_i^k - x_j^k\right)^2} \qquad (22)$$

where S and D are the swarm and dimension size, respectively ($D = 1$ in our specific problem).

The evolutionary factor f is calculated as:

$$f = \frac{dg - d_{\min}}{d_{\max} - d_{\min}} \quad \text{where} \quad f \in [0,1] \quad (23)$$

Here, d_g is the distance of the globally best particle, while d_{max} and d_{min} represent the maximum and minimum distances, respectively.

Adaptive Parameter Control:

To control the acceleration coefficients of $c1$ and $c2$, the evolutionary factor f is classified into four states: exploration ($S1$), exploitation ($S2$), convergence ($S3$), and jumping out ($S4$). Figure 5 illustrates the adaptive parameter control flow, which adjusts the values of $c1$ and $c2$ based on the current state of the swarm.

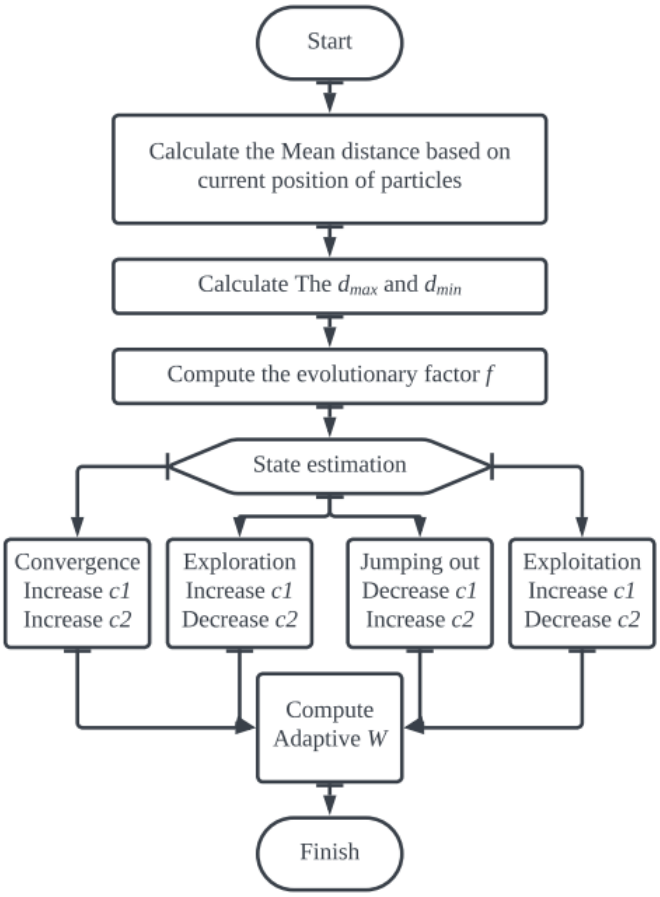

Figure 5. *The Figure illustrates the process of adaptive parameter control which considers the position of particles, estimates their current state, and adaptively controls the* $c1$, $c2$, *and W coefficients.*

Position Update:

The position update for each particle is subsequently executed using the updated velocity, and it is a pivotal step in the optimization process. The velocity update equation ensures that particles explore and exploit the search space effectively, contributing to the convergence of the algorithm toward an optimal or near-optimal solution. As the positions in the particle are discrete and identify the robot's number, the absolute value function is used to discretize the position value:

$$p_i^{t+1} = Abs(p_i^t + V_i^{t+1}) \in [0, N] \qquad (24)$$

where P_i^t is the current position of particle i at time t, and V_i^{t+1} is the updated velocity of particle i at time $t + 1$.

The velocity and position updates are performed iteratively until a stopping criterion of iterations I_{iter} is met.

Algorithm 1: APSO algorithm

Input : Parameters, population size, termination criteria
Output: Best solution found

Initialize population of particles with positions and velocities;
Randomly initialize global best solution set g_b^t;
while *termination criteria not met* **do**
  foreach P_i^t *in the Swarm* **do**
    Evaluate fitness of P_i^t based on Alg. 2;
    if $F_{fitness}$ *of* P_i^t *is better than fitness of* g_b^t **then**
      Update global best: $g_b^t \leftarrow P_i^t$;
    end
  end
  Update W, $c1$ and $c2$ using adaptive rules based on Eq. 21, Eq. 22, Eq. 23 and Fig. 3 flow chart.
  foreach p_i^t *in the particle* P_i^t **do**
    Update velocity and position of each position p_i^t in each P_i^t in the Swarm based on Eq. 20 and Eq. 24;
  end
end

Algorithm 2: Fitness function

Data: Input particle set P_i^t
while $i < m$ **do**
  Calculate task o_i completion duration T_{o_i} and energy use E_{o_i} corresponding to robot ID located in position P_i^t;
  Perform energy harvesting prediction and calculate E_{harvest};
  if $C_r = \emptyset$ **then**
    Add a recharging task to the schedule;
    Update the T_{end}^r and E_{end}^r state variables based on Eq. 9 and Eq. 12;
  end
  else
    Update the T_{end}^r and E_{end}^r state variables based on Eq. 10 and Eq. 12;
  end
  if *Constraint Eq. 15 or Eq. 16 or Eq. 17* = `false` **then**
    Result: $F_{fitness} = \infty$
    Break
  end
  Increment i;
  if $i = m - 1$ **then**
    Calculate $F_{fitness}$ from Eq. 14;
    Break
  end
end

6.2.2. Tasks energy consumption and execution time estimation model

One of the key constraints for mobile robots is their limited energy storage capacity, making effective management a crucial issue. Therefore, developing an energy estimation and prediction model that can be integrated into the scheduling phase is essential.

In general, the subsystems of a mobile robot can be categorized as power supply, computing (including high-level embedded computers and low-level microcontrollers), sensing, actuation, and communication subsystems. The energy consumption model of the robot is determined by the power demands of these subsystems. As such, the general power consumption model of a robot can be expressed as:

$$P_r = P_{sen} + P_{com} + P_{comm} + P_{act} \quad (25)$$

where P_r is the robot's total power consumption, and $P_{sen}, P_{com}, P_{comm}$ and P_{act} are sensing, computing, communication, and actuation subsystems, respectively.

In our test scenario, based on experimental data, we found that instantaneous power consumption of the computing, sensing, and communication subsystems remains relatively constant during different robot tasks. Thus, they can be grouped

as a static power (P_s), while the actuating power consumption (P_{act}) is dynamic and varies depending on the robot's movement and payload. As a result, the equation can be simplified to:

$$P_r = P_s + P_{act} \quad (26)$$

In this work both P_s and P_{act} are modeled using the experimental data from the TurtleBot3 (burger) robot.

For ground-wheeled robots, like TurtleBot3, actuating power is affected by payload, motor losses, road grade, rolling friction, and the robot's velocity and acceleration profiles. In our warehouse scenario, we assume that the rolling friction coefficient remains constant throughout the warehouse and the road grade is zero. Moreover, we neglect E_{drop} and E_{pick}, as these are relatively small in comparison to the overall task energy.

Based on the fundamental relationship between power and energy, the energy consumption for a given task can be calculated as:

$$E_o = \int (P_s + P_{act}) dt \quad (27)$$

If the robot is moving at a constant speed, the dynamic power p_{act} can be expressed as:

$$p_{\text{act}} = p_{\text{stdby}} + \sum_{i=1}^{N_w} \left(\sigma_0 + \sigma_1 \dot{\theta}_{w_i}^2 + \sigma_2 \text{sgn}\left(\dot{\theta}_{w_i}\right) \dot{\theta}_{w_i}\right) + \sigma_3 m \quad (28)$$

where:

N_w Number of active wheels (for TurtleBot3, $N_w = 2$)

- $\sigma_0, \sigma_1, \sigma_2$ and σ_3: Motor friction and loss coefficients.
- $\dot{\theta}_{w_i}$: wheel speed.
- m: total weight of the robot and the drop.
- P_{stdby}: motors' standby power.

To estimate energy consumption for a task, we need the following information: (i) the energy consumption model; (ii) task duration; (iii) task route distance; and (iv) robot constraints. In autonomous mobile robots, the global path planner generates the optimal route to the destination based on a known map. The local path planner navigates the robot along the waypoints while accounting for constraints and avoiding obstacles. Assuming that the local planner directs the robot at a constant speed between waypoints, the estimated task time and energy consumption are given by:

$$\begin{aligned}\hat{T}_o &= \sum_{j=1}^{w} \Delta t^j \\ &= \sum_{j=1}^{w} \left(\frac{d_l(wp^j, wp^{j-1})}{v} + \frac{d_a(wp^j, wp^{j-1})}{\omega} \right)\end{aligned} \quad (29)$$

where $\hat{T}_o$ stands for estimated task time, and v and ω are robot linear and angular velocities (which can be converted to wheels' velocity using the kinematic model of the robot). Furthermore $d_l(.)$ and $d_a(.)$. denote the functions for calculating the linear and angular distance of two waypoints.

The total energy consumption $\hat{E}_o$ for the task is then estimated as:

$$\hat{E}_o = \int_0^{T_o} (P_s + P_{act})\, dt \quad (30)$$

6.2.3. TurtleBot3 (burger) energy consumption model

For estimating the coefficients of power estimation and modeling the static power P_s under a specific payload, we equipped a TurtleBot3 with power meters. By conducting different experiments, the power consumption of each subsystem is measured, and the data set is used for training and testing the models.

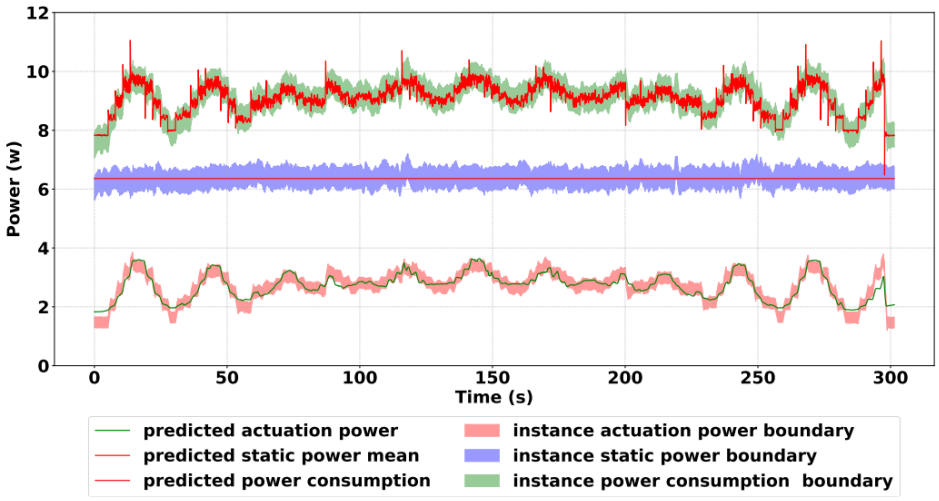

Figure 6. *Predicted power consumption of static power and actuation power of TurtleBot3 (burger) compared to the boundaries of measured instance powers with moving averages with window size=20.*

For modeling the static power P_s, a Gaussian distribution with mean = 6.357(w) and standard deviation = 0.343(w) is fitted to the data as shown in Figure 7.

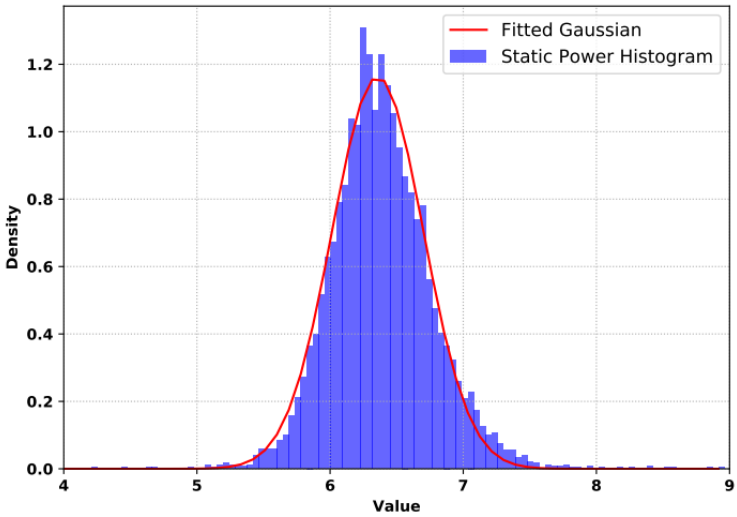

Figure 7. *The static power consumption of TurtleBot3 can be represented by a fitted Gaussian distribution (red line) on the histogram of static power measurements.*

Furthermore, using the well-known least squares regression method, actuating power coefficients are estimated and evaluated in different experiments. The model performance validation is shown in Figure 8. The results show the independent behavior of estimation error to the predicted values and a mean absolute percentage error of 5.7%.

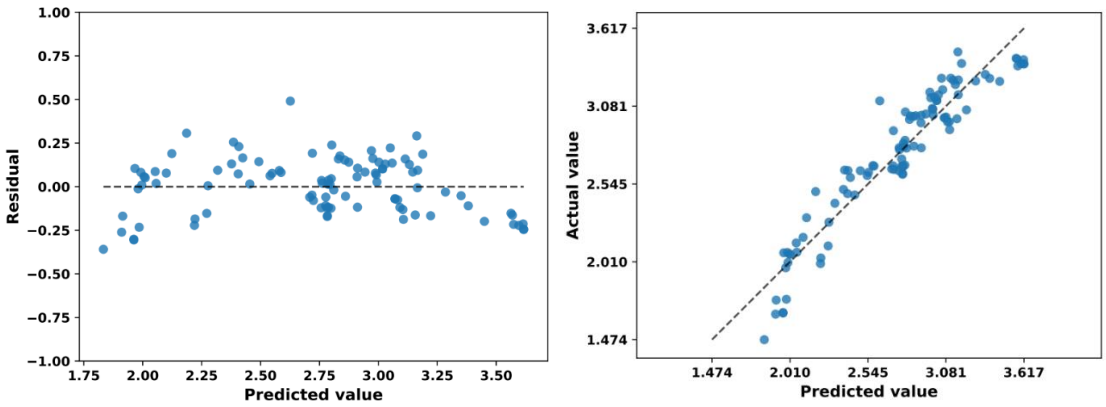

Figure 8. *Predictions validation with the test data set shows a bounded error for actuation power estimation with a mean absolute percentage error of 5.7% and* $r^2 score = 0.84$.

6.2.4. Energy harvesting prediction

Predicting energy harvested from a photovoltaic (PV) system is inherently complex, as it depends on a variety of factors, including weather conditions, historical data, the PV model, tilt angle, and geographic position. The overall equation representing the energy harvested by the robot from its solar panel can be expressed as:

$$\widehat{H}_{\mathrm{r}}(t, p_r^t) = F \cdot \left[f\big(W(t), PV(t), \mathrm{T}(t), P(t)\big) \right] \quad (31)$$

where:

- F: Maximum PowerPoint Tracking (MPPT) efficiency.
- $W(t)$: Accounts for weather-related historic data factors, including solar irradiance, which would typically affect outdoor energy harvesting.
- $PV(t)$: PV model, incorporating the electrical characteristics of the solar panel.
- T(t): Tilt angle of the PV system, which affects how much sunlight the panel can capture.
- $P(t)$: Robot's position, which can affect the amount of light captured, particularly in an indoor environment where lighting conditions may vary across different areas.

6.2.5. Energy harvesting prediction for the test-bed environment

In the specific context of this work, the energy harvesting scenario is based on a controlled indoor environment, where robots equipped with 20 cm^2 solar panels harvest energy from artificial light sources with an efficiency of 8 to 14 percent. The light intensity within the warehouse is modeled using a lighting histogram generated by six 10,000-lumen LED bulbs. These bulbs are installed on a ceiling that is 4 meters high in a 20 meters (length) by 14 meters (width) warehouse. Due to the fact of indoor energy harvesting, the tilt angle and weather data are neglected.

By considering these components, an analytic model can provide valuable insights into the energy harvesting potential of a solar panel system in a warehouse environment, enabling better decision-making for optimizing energy production and efficiency. The lighting simulation and harvested energy histogram of the warehouse are shown in Figure 9, which is an estimated result based on reference methodology.

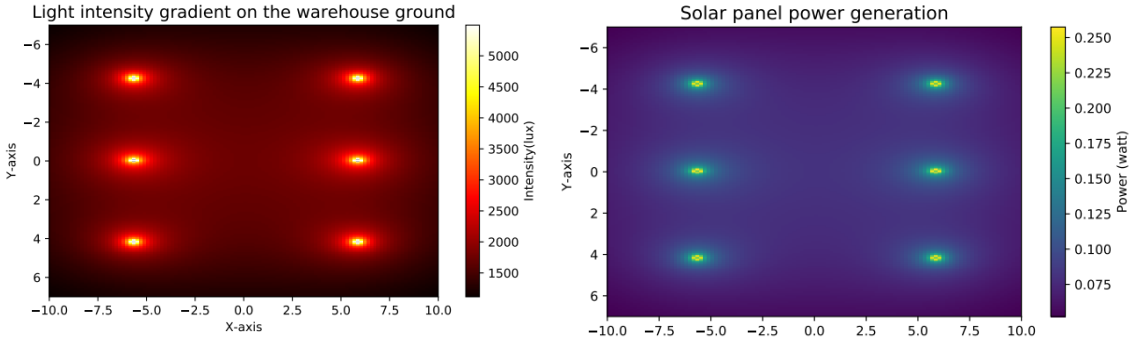

Figure 9. *Inside warehouse lighting and energy harvesting histogram. The histogram shows the energy a robot can harvest in every spot inside the warehouse.*

6.3. System architecture and results

We have developed a comprehensive autonomous architecture using ROS to implement task scheduling for practical applications. This architecture leverages

queue models and parallel processing and consists of 12 distinct nodes that collaborate seamlessly.

Figures 10 and 11 provide a visual representation of the system's flowchart, illustrating the interactions between these nodes and the overall system architecture. Figure 10 illustrates how the central nodes coordinate scheduling, monitoring, task queue management, and task duration/energy estimation, all within the ROS framework. Figure 11 provides a flowchart depicting the architecture of the task queuing system. It highlights how tasks are assigned to robots in parallel, based on the scheduling plan, ensuring efficient task distribution across the robotic swarm.

The first critical component is the **Task Acquisition** node, which continuously monitors the **Task Publisher** node - an external source responsible for providing task information. In some future scenarios, the **Task Acquisition** process may be autonomously handled by the robots themselves. This feature, which we plan to explore in future research, could further enhance the system's adaptability in dynamic environments.

Another pivotal node is the **Monitoring** node. This node oversees robot availability, energy levels, and maintenance planning. Additionally, it gathers essential data about both tasks and robots, playing a key role in workload-based scheduling. The **Monitoring** node communicates with the **Scheduling** node using a service and request protocol. This ensures that the **Scheduling** node - which is the most resource-intensive component of the architecture - activates only when necessary. To optimize energy consumption, the **Monitoring** node can place the system in an idle state when appropriate.

The **Scheduling** node initiates the task scheduling process by communicating directly with the robots, obtaining critical data such as their current positions, battery status, and energy harvesting rates. This collaborative connection forms the backbone of the task scheduling approach presented in this work. The **Scheduling** process relies on the notations and methodology described further in this section. A crucial aspect of the **Scheduling** is its integration with the **Duration and Energy Consumption Estimator (DEE)** node.

The **DEE** node is optimized to significantly reduce energy consumption. Rather than relying on continuous positioning data, we transitioned to discrete positioning, thereby creating a lookup table of feasible energy points. This shift not only reduces computational overhead, but also minimizes energy consumption during task execution. The **DEE** node utilizes information from ROS and Gazebo, such as the global planner and robot goal setter, to estimate task energy requirements, path planning, and task duration. Additionally, to ensure that robots avoid collisions during task execution, we implemented the **Collision Avoidance** node. This node relies on the robots' onboard sensors and employs a periodic sensing algorithm to prevent collisions.

Finally, the **Queue Management** node uses parallel processing to manage the execution of tasks by the robots, following the scheduled plan.

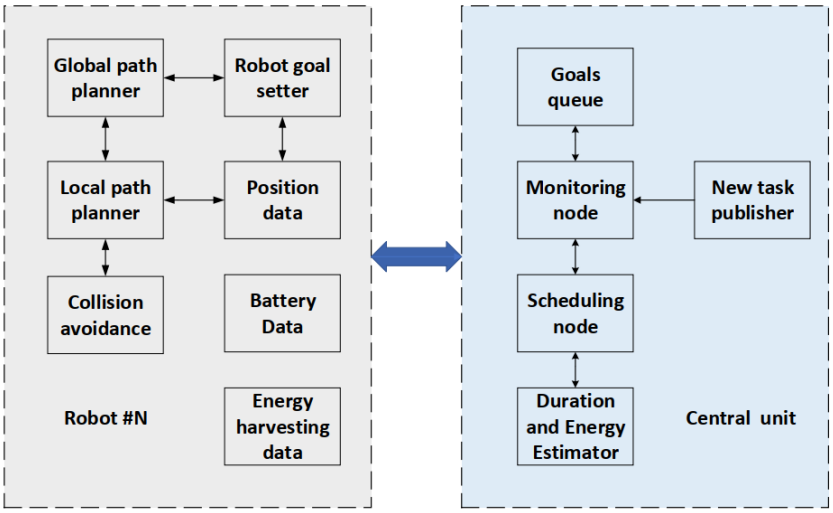

Figure 10. *The central nodes schedule, monitor, manage the tasks queue and estimate each task's duration and energy. This architecture is implemented using ROS.*

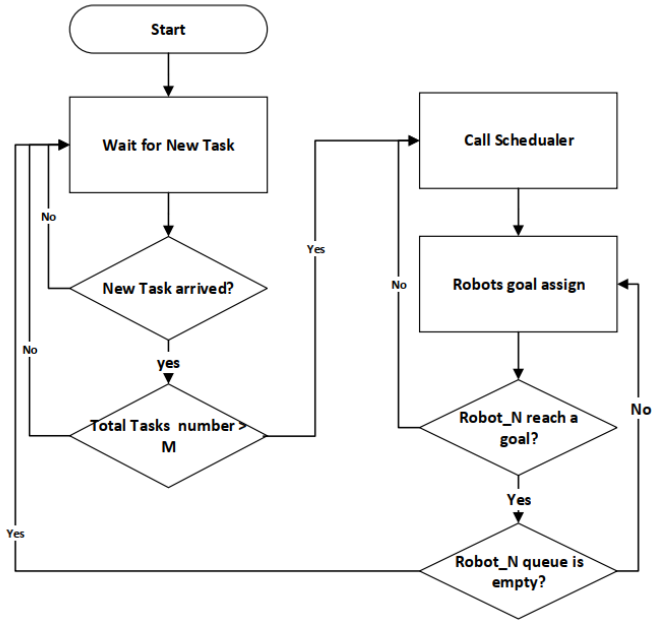

Figure 11. *Task queuing architecture flowchart. It illustrates the tasks queuing architecture that parallelly assigns the scheduled tasks to the robots.*

6.3.1. Simulation setup

As illustrated in Figure 12, the simulation environment represents a 20m × 14m warehouse space. The warehouse contains six storage areas, each consisting of six 5m×1m shelves arranged in a grid. Four sorting stations are positioned at the bottom of the environment. A charging station, marked by the green sign in Figure 12, is deployed within the system. This charging station is capable of charging up

to two robots simultaneously; a constraint that affects both the task scheduling and charging times of the robots. The objective is for the robots to deliver drops from the storage shelves to the sorting stations, while optimizing their task and energy consumptions.

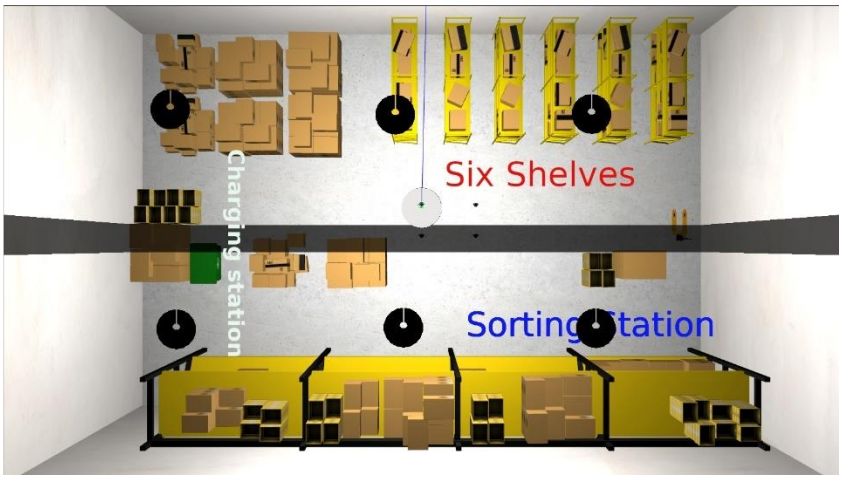

Figure 12. *Warehouse simulation environment, featuring six storage shelves and four sorting stations.*

In this setup, the APSO algorithm is compared against two baseline algorithms: traditional PSO and GA. The optimization performance depends on specific control parameters as previously introduced, like swarm size S, number of iterations I_{iter}, inertia weight W, and acceleration coefficients $c1$ and $c2$.

In the experimental design, the control parameters for APSO an PSO are defined as follows: $c1 = c2 = 2$, $W = 0.9$ (although in the case of APSO, these variables adapt dynamically), $S = 100$, and $I_{iter} = 100$. For the GA, the population size is set to 100 and the mutation rate is set to 0.1. The scheduling results were visualized using a Gantt chart, which displays the execution order of the different tasks. Different colors are used to represent different orders (i.e., groups of tasks with the same deadline), allowing for easy interpretation of the scheduling outcomes.

6.3.2. Multi-robot task scheduling validation

In this section, we present the results of our comprehensive robot task scheduling simulation, focusing on the effectiveness of the EA approach in dynamically optimized task scheduling for four ambiently powered robots, with a workload of $M = 50$ tasks. The results, as shown in Figure 13, provide the Gantt chart scheduling output of the EASDST algorithm.

To evaluate the algorithm, system architecture, and energy and time estimation model, a predicted schedule obtained from the EASDST optimized using the APSO algorithm is sent to the ROS-based architecture and simulated in the Gazebo environment. The different colors in the chart represent the set of orders $\mathbb{M} = m_0, m_1, m_2, m_3$, with each order containing a set of tasks. Specifically:

- $O_m^0 = o_m^0, \dots, o_m^{12}$

- $O_m^1 = o_m^{13}, \ldots, o_m^{20}$
- $O_m^2 = o_m^{21}, \ldots, o_m^{35}$
- $O_m^3 = o_m^{36}, \ldots, o_m^{49}$.

The initial robot battery charge are set as follows:

- $E_0^0 = 25\%$ (robot 0)
- $E_1^0 = 26\%$ (robot 1)
- $E_2^0 = 11\%$ (robot 2)
- $E_3^0 = 18\%$ (robot 3)

The charging times for a full battery charge is $T_{charge}(0, E_{max}) = 1350$ second. The red lines on the Gantt chart indicate the deadlines for each order, which must be met according to:

- $t_0^d = 550$ seconds
- $t_1^d = 850$ seconds
- $t_2^d = 1200$ seconds
- $t_3^d = 1700$ seconds

These deadlines are based on the predicted schedule, and in the case of the predicted schedule, they are fully met by the optimization algorithm. Table 1 shows the performance validation metrics for the model. In total, 50 experiments were conducted to compare the model in four different situations with random numbers of tasks (M) and robots ($N \in [0,3]$).

As illustrated in Table 1, the average error observed grows with the number of robots. This error between the predicted schedule and the simulation output can arise for several reasons. One major source of error is that the power consumption estimator relies on route information provided by the robot's global path planner. However, potential changes in the robot's path due to moving obstacles (e.g., other robots) can lead to deviations in power consumption. Additionally, improper tuning of robot parameters (e.g., PID controller) can cause excessive oscillations around set points, consuming extra energy. Communication and computation delays between ROS nodes may also introduce time gaps between tasks.

In general, the results from the Gantt chart demonstrate a high level of accuracy between the predicted model and the Gazebo simulations. The error metrics presented in Table 1 offer a comprehensive assessment. These metrics can be used to account for worst-case planning, and adding them to the scheduling can improve the performance of the system by coordinating the model with real-world conditions to mitigate deadline violations.

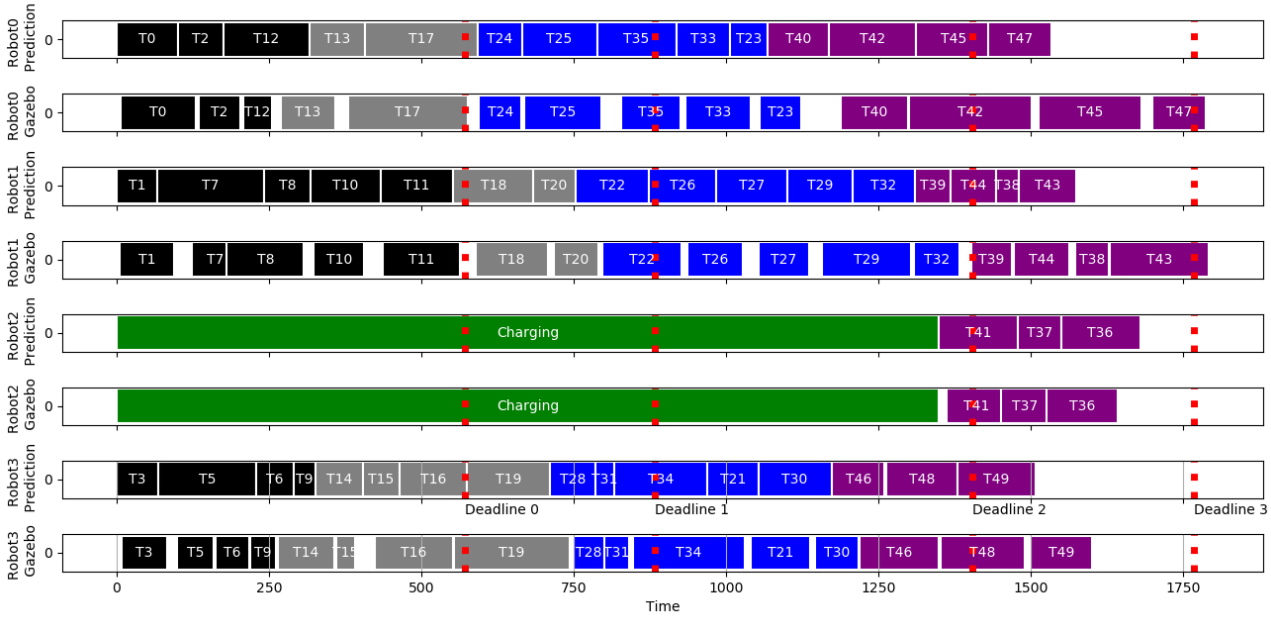

Figure 13. *Gantt chart illustrating the details of task scheduling and its corresponding Gazebo results for four robots, 50 tasks, and 4 orders using the EA algorithm.*

Table 1: *Error and standard deviation metrics for comparison of predicted scheduling model and Gazebo simulation.*

Metrics	1 Robot	2 Robots	3 Robots	4 Robots
Task Completion Time Average Error (%)	6.28	7.58	9.58	11.81
Task Completion Time Standard Deviation (%)	1.18	1.86	2.11	2.48
Task Energy Consumption Average Error (%)	7.43	7.47	10.38	11.63
Task Energy Consumption Standard Deviation (%)	1.61	1.78	2.22	2.32

6.3.3. Large-scale simulation results

In this section, we sought to demonstrate our proposed EA model compared to the EU approach in terms of completion time performance. Additionally, a comprehensive comparison between the different optimization methods - APSO,

PSO, and GA – is provided, highlighting their optimization performance and computation time.

Some assumptions were made for this section. First, the stopping criterion for all algorithms was set to terminate after 15 consecutive identical best answers, meaning that if the best solution is repeated over 15 continuous iterations, the algorithm stops. Secondly, the main distinction between the EA and EU approaches lies in energy awareness. The EA approach considers energy consumption, task timing estimation, and energy harvesting prediction, while the EU approach uses a traditional SDST model without considering energy harvesting. In both approaches, robots are ambiently powered, but the EU approach only accounts for battery recharging when the battery level drops below 10% of total capacity E_{max}.

For analyzing the optimality and simulation time, two scenarios are planned. Both scenarios are simulated using a computer with a 12th Gen Intel core i7-1270P, and 16 gigabytes of RAM. Every simulation for each point is done 25 times and the graphs show the average amounts.

The Relative Error metrics are provided in Table 2 and Table 3 for scenario A. The Optimal Relative Error (ORE) in these tables refers to the relative error between the optimal value obtained from 25 runs $\hat{x}_{\text{optimal}}$ and the reference optimal value $\hat{x}_{\text{reference}}$. The ORE can be calculated as:

$$\text{ORE} = \left|\frac{\hat{x}_{\text{optimal}} - \hat{x}_{\text{reference}}}{|\hat{x}_{\text{reference}}|}\right| \times 100\% \quad (32)$$

The average relative error (ARE) is the relative error between the average value $\tilde{x}$ and the reference optimal value $\hat{x}_{\text{reference}}$.

$$\text{ARE} = \left|\frac{\tilde{x} - \hat{x}_{\text{reference}}}{|\hat{x}_{\text{reference}}|}\right| \times 100\% \quad (33)$$

The worst relative error (WRE) is the relative error between the worst value $\hat{x}_{\text{worst}}$ obtained in each run and the reference optimal value.

$$\text{WRE} = \left|\frac{\hat{x}_{\text{worst}} - \hat{x}_{\text{reference}}}{|\hat{x}_{\text{reference}}|}\right| \times 100\% \quad (34)$$

Due to GA's best performance, its optimal solution is considered as the $\hat{x}_{\text{reference}}$.

Scenario A: Increasing the Number of Tasks

In **Scenario A**, we evaluated the effects of increasing the number of tasks on task completion time and simulation time. The number of robots was fixed to $N = 6$, while the number of tasks M varied between 10 and 80. The initial battery charges

of the robots are set to a given percent of nominal battery charge $E_{max} = 20Wh$ as follows:

- $E_0^0 = 42.9\%$ (robot 0)
- $E_1^0 = 27.6\%$ (robot 1)
- $E_2^0 = 16.7\%$ (robot 2)
- $E_3^0 = 48.1\%$ (robot 3)
- $E_4^0 = 57.3\%$ (robot 4)
- $E_5^0 = 16.9\%$ (robot 5)

To compare the different optimization approaches, we evaluated six configurations: APSOEA, PSOEA, and GAEA (for the energy-aware approach - solid lines), and APSOEU, PSOEU, and GAEU (for the energy-unaware approach – dashed lines).

Figure 14 illustrates that the EA approach outperforms the EU approach in terms of total task completion time for the same input. As the number of tasks exceeds 20, at least one robot in the EU approach requires recharging, while this threshold rises to 50 tasks in the EA approach. After this point, the gap between the two approaches begins to close, reflecting the benefits of energy-aware scheduling, especially with energy harvesting considered. Additionally, Figure 17 confirms that the EA approach better manages energy durability, particularly when robots require recharging. Experimental results demonstrate an average 15% reduction in task completion time for the EA approach over 50 experiments with random task and robot configurations.

To compare the optimization algorithms, two metrics are considered: the completion time objective function, which measures the algorithm's optimization performance, and the simulation time, which evaluates the algorithm's speed. As shown in Figures 15 and 16, GA consistently provides the best optimization results but is much slower than APSO, particularly in large-scale simulations. Table 2 and Table 3 provide relative error metrics for various task sizes, showing that as the number of tasks increases, the relative error decreases, especially when the population size is increased from 100 to 1000. This indicates that both APSO and PSO solutions converge toward GA's optimal results as the task size grows.

In terms of simulation time, however, GA is significantly more time-consuming compared to APSO, especially when the population size increases. In contrast, APSO demonstrates superior simulation efficiency, maintaining a reasonable simulation time even with larger populations.

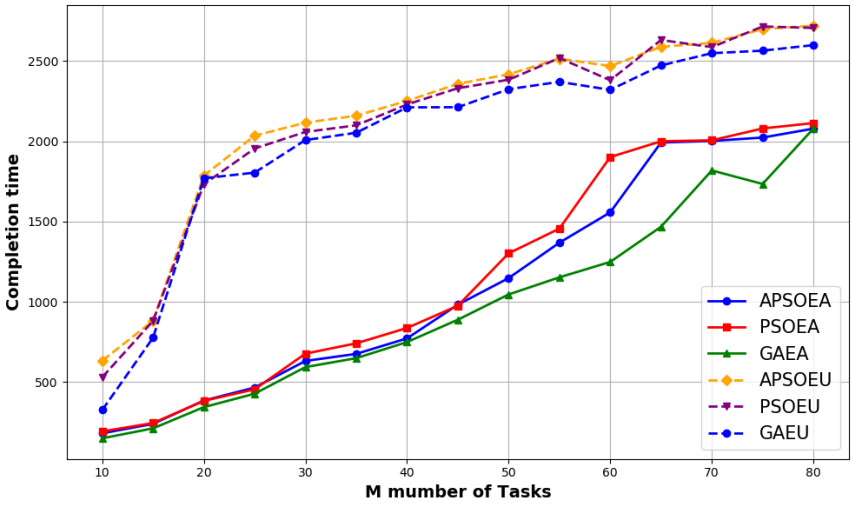

Figure 14. *Completion time (the unit is second) analysis of different algorithms with $S = 100$. The dashed lines are the EU approach and solid lines show the EA approach. In this scenario (A) the total number of tasks M varies and the number of robots N is fixed.*

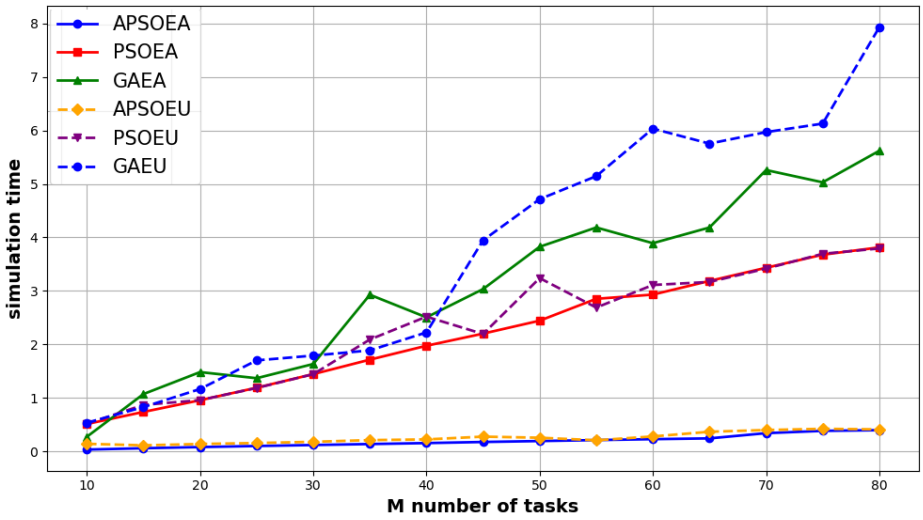

Figure 15. *Simulation time (the unit is second) analysis of different algorithms for a population size of $S = 100$. The dashed lines are the EU approach and solid lines show the EA approach. In this scenario (A) the total number of tasks M varies and the number of robots N is fixed.*

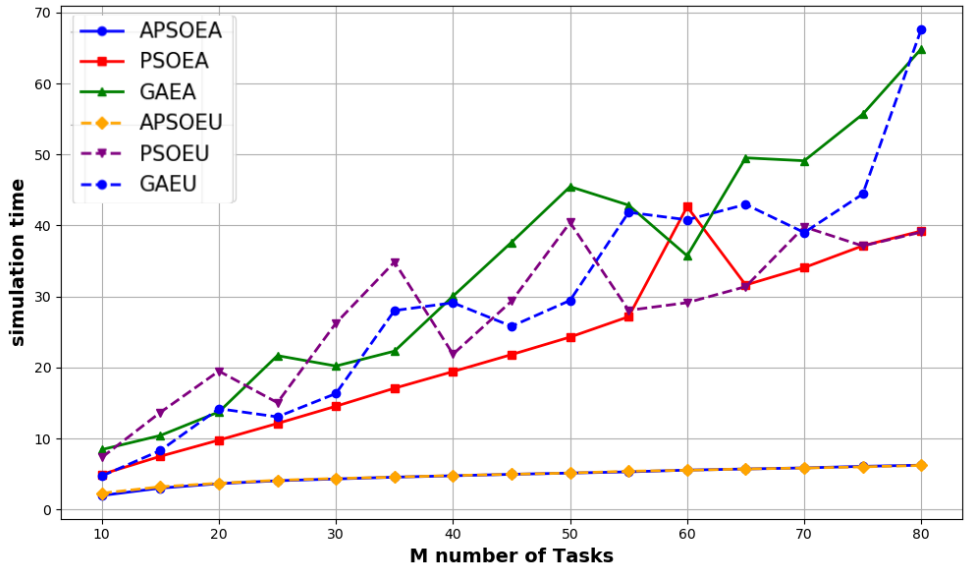

Figure 16. *Simulation time analysis of different algorithms for a population size $S = 1000$. The dashed lines are the EU approach and solid lines show the EA approach. In this scenario (A) the total number of tasks M varies and the number of robots N is fixed.*

Scenario B: Increasing the Number of Robots

In **Scenario B**, we analyzed the effect of increasing the number of robots on task scheduling. Here, the number of tasks was fixed to $M = 100$, and the number of robots N increased from 6 to 20. Figure 17 shows that the EA approach outperforms the EU in completion time, particularly when robots require recharging. As the number of robots increases, the total amount of energy harvested increases, reducing the overall charger usage. This confirms the advantage of EA scheduling in managing energy resources in ambiently powered robot swarms.

Figure 18 shows that, for APSO and PSO, the number of robots does not significantly affect simulation time. However, for GA, there is a slight increase in simulation time as the number of robots increases. This scenario reaffirms the superior optimization performance of GA and the better simulation time of APSO.

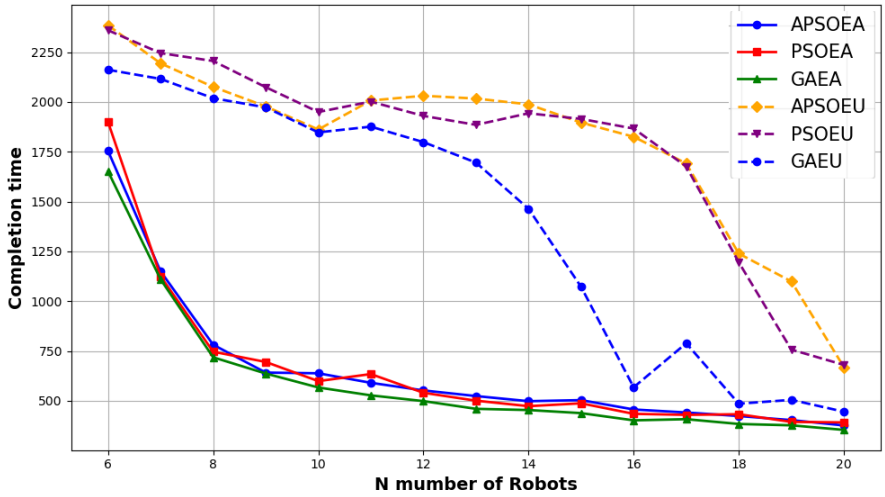

Figure 17. *Completion time (the unit is second) analysis of different algorithms with $S = 100$. The dashed lines are the EU approach and solid lines show the EA approach. In this scenario (B) the total number of tasks M is fixed and the number of robots N varies.*

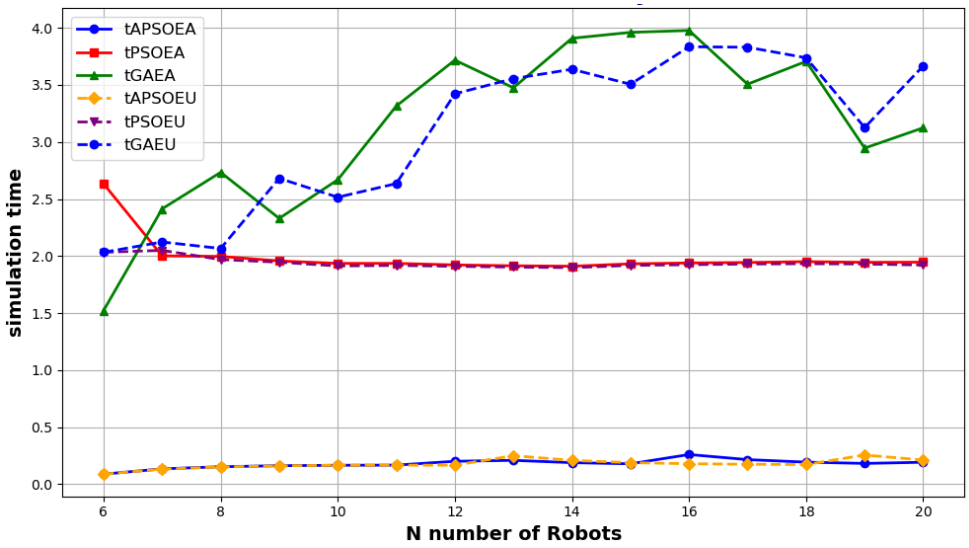

Figure 18. *Simulation time (the unit is second) analysis of different algorithms with $S = 100$. The dashed lines are the EU approach and solid lines show the EA approach. In this scenario (B) the total number of tasks M is fixed and the number of robots N varies.*

Table 2. *Relative error metrics which show the difference between the total task completion time of different EA optimization algorithms with a population size S = 100 in scenario A.*

Number of Tasks	GAEA ORE	GAEA ARE	GAEA WRE	PSOEA ORE	PSOEA ARE	PSOEA WRE	APSOEA ORE	APSOEA ARE	APSOEA WRE
10	0.0	1.1	4.0	2.8	17.8	31.6	1.9	29.6	33.0
15	0.0	11.5	18.7	22.4	24.8	28.4	13.6	17.9	31.8
20	0.0	0.3	1.9	8.9	13.8	21.3	4.5	13.1	18.4
25	0.0	0.6	2.1	10.0	17.7	24.7	8.4	14.6	21.5
30	0.0	4.9	8.1	11.1	13.2	19.5	10.6	13.2	21.3
35	0.0	3.7	6.1	8.2	11.1	13.0	5.6	10.7	16.5
40	0.0	3.7	6.9	6.1	10.9	13.1	6.8	9.6	14.2
45	0.0	2.9	6.7	4.0	8.1	15.9	7.1	10.1	14.1
50	0.0	2.3	3.4	7.2	12.5	15.1	6.7	9.3	12.7
55	0.0	2.3	3.6	6.0	9.5	13.6	10.2	10.8	13.6
60	0.0	3.4	4.4	8.9	12.2	16.5	5.2	8.7	16.6
65	0.0	4.8	6.9	7.3	11.9	14.5	8.8	13.8	15.5
70	0.0	2.3	3.6	5.4	8.3	11.5	3.9	9.5	11.8
75	0.0	1.0	1.9	3.5	6.4	12.1	1.1	3.3	5.8
80	0.0	0.7	2.1	4.4	5.7	8.5	1.5	4.7	8.8

Table 3. *Relative error metrics which show the difference between the total task completion time of different EA optimization algorithms with a population size S = 1000 in scenario A.*

Number of Tasks	GAEA ORE	GAEA ARE	GAEA WRE	PSOEA ORE	PSOEA ARE	PSOEA WRE	APSOEA ORE	APSOEA ARE	APSOEA WRE
10	0.0	1.1	2.9	1.8	9.2	12.9	6.2	8.1	12.9
15	0.0	2.3	5.8	10.7	15.8	20.5	8.7	12.2	19.3
20	0.0	3.1	8.4	14.5	16.1	19.6	11.4	12.7	14.4
25	0.0	0.9	3.3	8.1	10.3	10.8	4.1	6.7	14.9
30	0.0	0.8	1.8	0.7	4.0	12.2	3.6	8.9	11.7
35	0.0	2.1	5.3	8.5	9.8	11.8	7.1	9.5	12.2
40	0.0	1.3	3.1	3.9	6.5	9.5	5.8	8.1	11.2
45	0.0	0.4	1.2	5.7	8.0	10.1	4.6	5.6	6.9

50	0.0	5.5	6.9	5.5	10.1	12.3	7.8	9.8	13.4
55	0.0	2.9	4.4	5.0	6.5	8.3	0.9	4.1	8.1
60	0.0	0.7	1.1	4.7	7.5	8.8	4.6	6.7	8.2
65	0.0	0.9	1.5	4.1	5.7	6.9	3.5	4.4	5.0
70	0.0	1.3	1.9	4.9	6.6	7.6	0.7	2.4	3.0
75	0.0	1.1	2.1	0.7	4.7	7.9	3.2	6.7	8.0

7. AI-based Energy Harvesting Aware Path Planning

Swarm Robotics is a field within AI that focuses on designing, building, and deploying large groups of relatively simple robots. These robots coordinate in a distributed and decentralized manner to cooperatively complete complex tasks [35, 72, 26]. Inspired by the collective behavior of natural self-organizing systems, such as the movements of fish schools or bird flocks, swarm robotics leverages local interactions to achieve coordinated actions [69, 72]. Each robot within the swarm possesses local processing, communication, and sensing capabilities, enabling it to interact with its surroundings and adapt autonomously to environmental changes. These features grant swarms their core advantages: adaptability, robustness, and scalability [39]. Although bridging the gap between theoretical models and real-world implementations remains challenging, swarm robotics continues to attract significant research interest [69, 72, 82].

Swarm robots can exhibit various collective behaviors, which can serve as building blocks for different applications. Brambilla et al. [64] categorized these behaviors into four main groups: spatially organizing behaviors, navigation behaviors, collective decision-making and other collective behaviors. Figure 19 shows a classification of research in swarm robotics based on Brambilla et al. [64]. The herein proposed approach focuses on navigation behaviors, which involve coordinating the path planning of a swarm. Swarm navigation is a critical area of research due to its potential for real-world applications, such as environmental monitoring, search and rescue, precision farming, and transportation [47]. Efficient navigation is essential for optimizing the performance of swarm missions.

One of the persistent challenges in swarm robotics is energy management. Traditional swarm navigation algorithms tend to optimize for path efficiency and task completion without factoring in the energy levels of individual robots. This oversight is problematic, as many autonomous robots operate in environments where recharging infrastructure is either limited or unavailable. As a consequence, robots risk depleting their energy reserves, leading to mission failure or the need for frequent manual intervention to recharge or replace batteries.

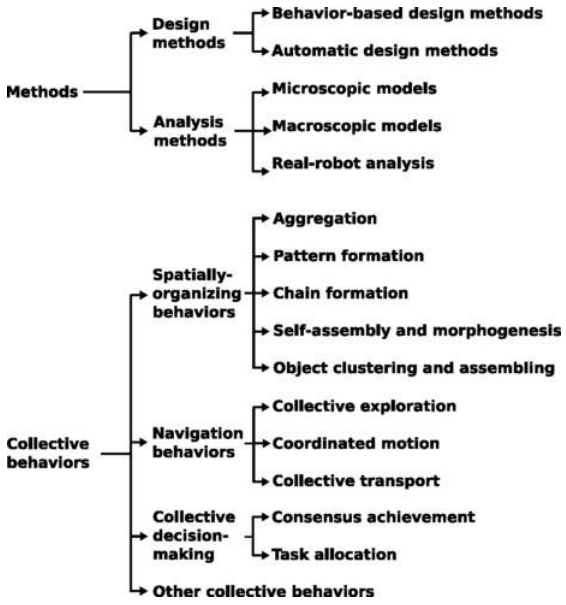

Figure 19. *Classification of research in swarm robotics into two taxonomies (proposed by Brambilla et al. [64])*

Incorporating energy harvesting capabilities into the navigation strategy of swarm robots offers a sustainable solution to this energy management challenge. Energy harvesting allows robots to collect energy from their environment through various means, such as solar panels, kinetic energy harvesters, or thermoelectric generators [81]. Since ambient energy availability often varies depending on location, integrating energy harvesting into the navigation process enables robots to adjust their paths to areas with higher energy potential. This not only extends the operational time of the robots, but also reduces dependency on human-operated charging systems, improving the efficiency of swarm missions.

Given the stochastic and dynamic nature of energy harvesting and navigation tasks, machine learning, and more specifically reinforcement learning (RL), is a promising approach to developing intelligent navigation algorithms. RL is a type of machine learning in which agents learn optimal behaviors through trial-and-error interactions with their environment [61, 48]. By receiving feedback in the form of rewards, RL agents can develop strategies that maximize cumulative rewards over time. This makes RL particularly effective for swarm navigation, where the robots must balance reaching their target destinations, avoiding collisions, and maximizing energy harvesting opportunities [80].

7.1. Problem statement

One of the primary challenges in the field of robotics is limited battery life, a significant issue for robotic swarms that need to operate autonomously over extended periods. Efficient energy management is essential to enhance the autonomy and operational longevity of these swarms, especially in environments where energy resources are limited or unevenly distributed.

The challenge is compounded by the fact that the availability of harvestable energy can vary significantly across different locations. Factors such as shadows cast by obstacles, terrain features, and time-dependent changes in sunlight exposure can affect energy harvesting potential within a given area. This spatial variability directly impacts the robots' ability to sustain themselves over time.

Incorporating these spatial dependencies into the swarms' navigation behavior offers a path to more efficient energy utilization. RL presents a promising approach for developing an intelligent navigation algorithm that not only considers the shortest path but also optimizes for areas with higher energy availability. This solution could significantly extend the swarm's operational time and improve overall mission effectiveness.

Our goal is to develop an intelligent navigation algorithm that integrates energy harvesting capabilities within a swarm of robots. To achieve this, we conduct a proof-of-concept in a simulation environment and evaluate the performance under various scenarios. This problem is broken down into several sub-problems, leading to the following research questions:

- **RQ1** How can a RL-based algorithm be designed to integrate energy harvesting and multi-objective navigation, balancing both the shortest path and optimal energy harvesting for robotic swarms?
    - **RQ1.1** What RL algorithms would be more effective for this task?
- **RQ2** How does the performance of robots that incorporate energy harvesting in their navigation compares to those that do not?
- **RQ3** How do environmental and system factors influence the algorithm's performance?
- **RQ4** How do different RL techniques compare in terms of performance and effectiveness for finding optimal paths while considering energy harvesting?

Energy management has drawn substantial research attention in robotics, particularly in developing energy-efficient autonomous systems. While recent work has started incorporating energy considerations into navigation strategies, energy harvesting-aware navigation in robotic swarms using RL remains relatively underexplored. There is little related research that directly addresses our specific problem. The relevant literature can be categorized into three areas:

- RL in robot navigation
- Energy harvesting in robotic swarms
- RL in energy-aware navigation

Each of these categories provides foundational insights for our approach, which combines RL-based navigation with energy harvesting.

For instance, Na et al. [34] propose a Federated Learning (FL) based Deep Reinforcement Learning (DRL) training strategy (FLDDPG) for swarm robotics,

focusing on environments with limited communication bandwidth (e.g. high-radiation, underwater, or subterranean environments). They demonstrate that the FLDDPG method exhibits greater robustness and generalization compared to baseline strategies which are limited by communication bandwidth.

Zhu et al. [59] introduce a rule-based RL (RuRL) algorithm for efficient robot navigation in complex spaces, combining rule-based and RL methods to reduce sample complexity and time cost. RuRL uses wall-following to create closed-loop trajectories, a reduction rule to minimize redundant exploration, and the Pledge rule to guide navigation within this optimized space, achieving scalable results in single and multi-room tests with SLAM-mapped robots.

Wei et al. [53] review energy harvesting techniques, particularly solar and mechanical methods, to power self-sustaining UAVs. They discuss combining these techniques with MEMS and flexible electronics to enhance UAV performance, highlighting dynamic soaring as a promising mechanical energy strategy and noting future challenges in UAV energy harvesting. Dolev et al. [71] explore caterpillar swarm-inspired nano-robots for energy-efficient movement in biological systems. Their design emphasizes coordinated movement, weight adaptation to blood flow, and effective in-vivo energy use, supporting advanced nano-robotic applications in medical contexts.

Visca et al. [51] propose an adaptive energy-aware planning framework for unmanned ground vehicles in unknown terrains, using a deep meta-learning model to develop an energy-efficient path planner that adapts to local terrain conditions, showing strong results in 3D simulations. Bouhamed et al. [63] develop a Deep Deterministic Policy Gradient (DDPG)-based UAV navigation system for obstacle-dense environments, optimizing for energy awareness with a dynamic threshold to return UAVs to charging stations, reducing collision risk and power loss. Niaraki et al. [36] design a Q-learning path planning framework for UAVs navigating in windy conditions, optimizing power use and goal detection in large search domains and demonstrating adaptability to various wind patterns.

These studies reflect advances in RL for navigation, energy harvesting, and energy-aware navigation, yet the combined focus on energy-aware navigation in robotic swarms remains underexplored. Although the literature provides valuable insights, it is important to distinguish between "energy-aware" navigation and "energy harvesting-aware" navigation. Energy-aware navigation typically involves optimizing paths or behaviors based on current energy levels, aiming to minimize consumption without considering dynamic replenishment. In contrast, energy harvesting-aware navigation actively plans routes to maximize opportunities for energy harvesting, extending mission duration and reducing reliance on external charging infrastructure. Literature, although not directly addressing our specific problem, still provides valuable insights and methods that can inspire our approach. This way we can obtain a general idea of how to divide the process for our research and verify the questions to be relevant.

RQ1 is a rather obvious question since this is the main goal of our work. In our research, we found a lot of overlapping approaches, thus we decided to only use two different RL techniques. Since energy harvesting is the focus in this research, we naturally started by looking at which RL techniques are used in our different research topics. In context of energy-aware path finding we found an interesting approach using Q-learning.

In the context of other research in RL, most studies have focused on deep reinforcement learning and policy gradient methods. Given this trend, we decided for our first RL algorithm to be Q-learning. We wanted to start with a simple RL basic RL and deep RL in our problem. For our second RL algorithm we chose PPO. In particular this selection for deep RL, is informed by its widespread use in complex environments and its robust performance in policy gradient methods, making it a better fit than alternatives like DDPG for the adaptive and dynamic requirements of our robotic swarm system.

The performance comparison posed in RQ2 is critical, as it examines the practical impact of energy harvesting on the efficiency of robotic swarms, a topic that has been insufficiently addressed in current research since we barely found anything matching this specific subject.

Additionally, RQ3 and RQ4 delve into the influence of environmental factors and the comparative effectiveness of different RL techniques, respectively. These questions are based on observed variations in performance across different settings and algorithms, highlighting the importance of adapting our methods to diverse operational conditions.

In summary, our research addresses key gaps in the literature. By focusing on RL-based energy harvesting navigation in robotic swarms, we aim to make meaningful contributions to this emerging field.

7.2. Methodology

In this section, we describe the design and development of the proposed solution for AI-based energy harvesting-aware path planning in swarm robotics. The goal is to develop an intelligent navigation algorithm that accounts for energy harvesting capabilities in a swarm of robots.

The full implementation of this solution is available in our GitHub repository[1]. The repository includes all the source code and supplementary materials for replicating and extending our work.

Our approach follows three main steps:

1. **Building the environment:** Designing a simulation environment to model swarm behavior and interactions between robotic agents.
2. **Training the environment using RL:** Implementing RL techniques to optimize energy-aware path planning.
3. **Design and execution of tests:** Testing the performance of the trained algorithms in various scenarios.

The overview of these steps is presented In Figure 20. The following sections provide detailed explanation of each step.

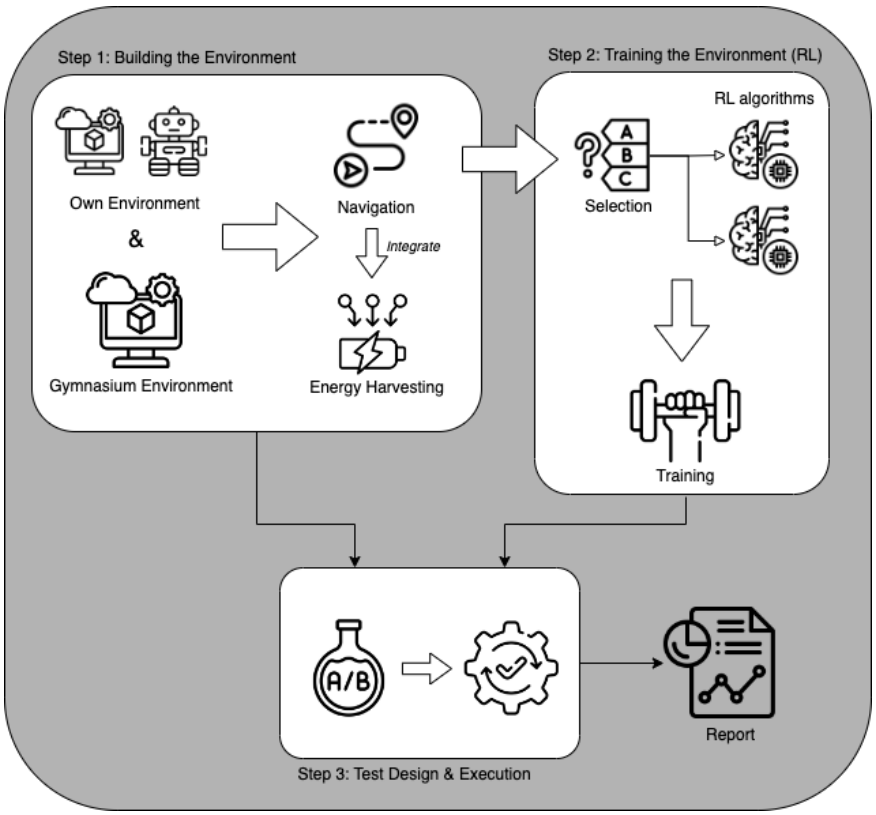

***Figure 20**. Design of the Solution*

1 https://github.com/openswarm-eu/AI-based-Energy-harvesting-Navigation

7.2.1. Step 1: Building the Environment

The first step involves creating a simulation environment to model the swarm behavior and environmental interactions of robotic agents. Initially, we implemented a Q-learning environment tailored to our specific context. Later, to validate our approach and compare it with other learning tools, we developed a second environment using the Gymnasium library for the PPO algorithm. An important note is that we must ensure both environments are model-free, meaning they do not require an explicit model of the environment's dynamics. This feature is crucial for RL-based training, which relies on learning optimal policies through trial and error.

The main goal remains to implement a navigation algorithm for a swarm of robots inside an environment that considers energy harvesting. Because the energy harvesting part of this is of much importance and different from other research done so far, we wanted to highlight how we implemented this part in each of the environments. This is why we will first explain the implementation of the navigation algorithm, and afterwards focus on how we integrated energy harvesting.

We implemented our own environment by simply using Python classes. Since the objective was to incorporate swarm behavior, we separated the logic between the environment and robots.

Environment Class:

The environment is structured as a 2D grid where robots can navigate and interact. Each robot operates autonomously and makes decisions based on local observations, such as the position of obstacles, light sources, and other robots. The class is initialized with several key attributes:

- ***Grid Size*:** The environment is modeled as a 2D grid where each robot r_i in the robot set R, can navigate within a bounded area. The environment's dimensions (width and height) are adjustable, allowing flexibility to simulate various scenarios.
- ***Obstacles*:** Static obstacles are introduced in the environment to simulate real-world physical barriers that the robots must avoid.
- ***Robots*:** A list of active robots (R) is maintained, each of which interacts with the environment and makes decisions autonomously.
- ***Light sources*:** Light sources represent energy sources for energy harvesting. Robots are rewarded for moving towards and harvesting energy from these sources (detailed in the Energy Harvesting section).
- ***Energy Harvesting*** : Covered in energy harvesting section.

We allow robots to be added and 'connected' to our environment. This connection enables robots to access necessary observation information for decision-making. Specifically, we assume that robots have access to certain environmental data such as the locations of obstacles, energy sources, and other robots within the environment. This setup allows the environment to serve as a shared context in

which robots can sense, move, and interact with both the environment and each other, mimicking the behavior of a swarm retrieving information from its surroundings.

The class includes a method to reset the environment, ensuring that each robot starts from a random valid position. This randomization is important for training RL algorithms, as it exposes the RL to a variety of scenarios and challenges, meaning it provides more learning data.

The *step* function is the core of the environment's navigational interaction with each robot individually. It processes a given action and evaluates the outcome of it. An individual robot will try to take the given action which represents a movement in a particular direction. The environment determines whether the move is valid, meaning whether it is in between the bounds of the grid, and whether it causes a collision with either an obstacle or another robot. These factors are then used to compute a reward, which guides the learning process for the RL algorithm later. We define our reward function as follows: we do multiple checks and depending on which check clears, it will return a certain reward based on the following criteria:

1. **Done:** A positive reward for reaching the destination.
2. **Out of Bounds:** A penalty if the robot moves outside the grid.
3. **Collision:** A penalty for colliding with obstacles or other robots.
4. **Otherwise:** If none of the checks clear, a small penalty to incentivize finding the shortest path.

The episode ends when all robots reach their respective goals, signaling that the navigation task is complete.

Robot Class:

The Robot class encapsulates the behavior of each robot within the swarm. The primary objective of each robot is to navigate in to reach a specific goal. Key attributes include:

- Id: Unique identified of each robot r_i, enabling the system to manage multiple robots simultaneously without confusion.
- $Position$: Robot r_i position pr_i in the grid.
- $Goal$: Navigation task t_i for robot r_i consisting of reaching a specific goal position gr_i within the environment in the grid.
- Env: The environment which robot operates in is considered as a grid map containing obstacles and other robot position. The connection between the robot and the environment is essential for the robot to interact with other robots and avoid obstacles.
- $Q\text{-}learn\ Agent$: Each robot has a Q-learning agent responsible for learning the optimal actions during navigation.
- $Harvested\ Energy$: Tracks the energy harvested by the robot during its operation (detailed later).

The robot class has a reset functionality that assigns a given starting position as the current position. Remember that these starting positions were initialized in the *reset* function of the environment class. Since the environment and the robot are connected, the method represents purely one of the communications taking place between those two classes. Other critical information that robot accesses from communicating with the environment are the layout of obstacles and the positions of other robots. This connection is thus vital as it enables the robot to make informed decisions about its movements.

The movement logic allows robots to move in four possible directions (up, down, left, or right), with each potential move evaluated for validity (bounds and collision checks). The robot's done method checks whether it has reached its goal, at which point the robot is considered to have completed its task.

Swarm Behavior:

As previously addressed, we separated the logic into two classes to incorporate swarm behavior. Here, we will explain in detail the specific mechanisms implemented to achieve swarm intelligence within our system.

Swarm behavior typically refers to the collective behavior of decentralized, self-organized systems, often involving multiple autonomous agents that interact with each other and their environment. We implemented the following swarm characteristics:

Decentralization, meaning swarm systems have multiple control centers rather than a single central control mechanism, with individual agents making decisions based on local information. In our implementation we ensured that each robot operates autonomously, making decisions based on its local environment, goals, and interactions with other robots. There is also no central controller directing the robots' actions. Each robot has its own Q-learning agent that is trained so that it can decide those actions.

Self-organization, this refers to the fact that agents adjust their behavior based on local interactions and environmental feedback, leading to a globally coordinated behavior without centralized control. We can find local interaction in multiple parts of our classes. Robots not only avoid collisions with each other and with obstacles based on local sensing. But also, each robot has its own goal and moves towards it independently. As for the self-organizing part, our agents adjust their behavior based on rewards and Q-values learned through interactions with the environment. However, there's no explicit mechanism for self-organization beyond the Q-learning algorithm.

Emergent behavior, this means that the group as a whole should exhibit complex collective behavior that emerges from the interactions of the individual agents. While the implementation provides the mechanisms for robots to avoid collisions and move towards goals, it doesn't explicitly program complex group behaviors. However, emergent behavior could potentially arise from the combination of

these individual behaviors, especially as they interact in the environment. But keep in mind that that behavior is driven by the Q-learning algorithm and predefined rules rather than emergent properties arising from local interactions.

Coordination and cooperation, is a feature relating to robots working together towards a common goal, sometimes without explicit communication. In our environment we can find this characteristic in two cases. First our robots have mechanisms to avoid collisions, which is a form of implicit coordination. The environment checks if all robots have reached their goals. This reflects the idea that the task is only complete when the entire swarm has successfully achieved its objectives, highlighting a collective goal rather than individual success. We also want to note that each robot is aware of the goals of other robots. And while this awareness isn't directly utilized in the provided code, it could be used to influence movement decisions, ensuring that the swarm collectively optimizes its path to achieve all goals without interference.

Robustness and scalability, Swarm systems are often robust to individual agent failures and scalable to large numbers of agents. Our implementation doesn't explicitly address robustness to individual failures but does allow scalability to not only more robots but also scalability regarding the dimension of the grid environment itself.

In conclusion, our classes have multiple characteristics of swarm behavior, particularly in terms of decentralization, local interaction, and the potential for emergent behavior. While the system demonstrates these aspects effectively, it lacks explicit mechanisms for robustness and advanced self-organization. However, the elements that do exhibit swarm behaviour work together to create a system where robots, though operating independently, contribute to the swarm's overall success.

***Energy harvesting*:**

Energy harvesting has been integrated into the navigation system, wherein light sources have been added to the environment class. This level of configurability allows comparing the results when we train the RL algorithms with and without considering energy harvesting.

- ***Light sources:*** These represent the energy sources within the environment that robots can utilize for energy harvesting. The positions of these light sources are fixed, and their locations are critical for the energy management of the robots.
- ***use EH:*** This Boolean flag determines whether energy harvesting is enabled in the environment. If set to True, the environment integrates energy harvesting into the reward structure, influencing the robots' navigation decisions.

Inside the *step* function of the environment class, we now not only let the robot make a move but also have the robot calculate how much energy can be harvested during that step.

Next, the amount of harvested energy is also integrated into the reward calculation. The total reward is now a weighted sum of the previous movement reward with the new energy harvesting reward (which is equal to the total amount of harvested energy to keep everything simple).

Moving to the robot class, we also add a critical new attribute:

- ***Harvested Energy:*** This attribute tracks the cumulative energy harvested by the robot which is a crucial aspect of the robot's operation.

Inside the *reset* function, we ensure that not only the position is reset to the new given start position. But the amount of harvested energy is reinitialized at zero, since there is no energy harvested at the beginning.

We also add the ability to harvest energy in a new method to our robot class. This method estimates the energy a robot can collect based on its proximity to the light sources. The closer a robot is to a light source, the more energy it can harvest, with maximum energy being collected when the robot is directly on a light source. The robot also needs to be in a certain proximity to be able to do energy harvesting. However, the amount of energy that can be harvested will become smaller at a linear rate when moving further away while still being inside the proximity range. The function thus works as follows:

1. *Calculate the distances to each light source:* We first calculate the distance of the position of the robot to each of the light sources. This calculation is simply the Euclidian distance:

$$\mid (x_{pr_i} - x_{l_i} \mid + \mid (y_{pr_i} - y_{l_i}) \mid \quad (5)$$

   We also have a range to which robots are able to harvest energy. If they are more steps away from the light source than the range, they will not be able to harvest any energy. This range is initially set to three.

2. *Calculate the amount of energy:* After we have calculated the relevant distances in which a robot is able to harvest energy, we can now calculate the amount of energy it will harvest based on this distance. In a real-life scenario a robot would have sensors for which the findings can be used to calculate the amount of energy is present in a location. In our simplistic grid environment, we took the following approach. When the robot has the same position as a light source, the amount of energy harvested is equal to the maximum energy emitted by the light source. The farther away we move, the lesser amount of energy should be present.

$$\sum_{i=1}^{L} \begin{cases} r_{EH} & \text{if } d_i = 0 \\ \frac{1}{d_i} & \text{otherwise} \end{cases} \quad (6)$$

where r_{EH} represents the maximum energy harvesting constant. This functionality supports the broader goal of optimizing energy usage within the swarm, as robots must balance their navigation tasks with the need to gather energy.

PPO Environment Model:

This second environment was developed using the Gymnasium library. Both the logic from the environment class and the robot class has been adopted to model the Gymnasium environment keeping the same methods. That way makes it easier to compare the outcome when using Q-learning in our own environment and PPO in the Gymnasium Environment.

A custom Gymnasium environment inherits from the abstract class gymnasium.Env, which serves as the foundation for creating reinforcement learning environments. This inheritance mandates the implementation of certain attributes and methods to ensure the environment can interact seamlessly with Gymnasium's reinforcement learning algorithms. Specifically, custom environments must define the $action\ space$ and $observation\ space$ attributes and implement the $reset$ and $step$ methods. Optionally, the environment can also include $render$ and $close$ methods for visualization and cleanup, respectively.

***Navigation*:**

We created a custom Gymnasium environment tailored for simulating multiple robotic agents navigating a grid-based world. At initialization, the environment is configured with several parameters. Most of them you will recognize having the same purpose as in our own environment implementation.

- *Grid size:* Instead of having a width and height, we have a combined parameter which defines the dimensions of the grid. It remains flexible only now we limited ourselves to square grids.
- *Obstacles:* As before in the environment class we represent possible barriers as static obstacles within the grid.
- *Robots:* Now instead of keeping tabs on a list of robot class objects, we simply just save the position of the robots in an array. The indexes of the array will serve as the id of each robot.
- *Goals:* Again, we don't have a separate robot class anymore, so we keep the target positions in another array. In this case the index of the robot's position array will match with the index of the robot's goal array.
- *Light sources:* Covered in energy harvesting section.
- *Harvested Energy:* Covered in energy harvesting.

The following new attributes have been introduced for the Gymnasium environment:

- *Action Space:* This attribute specifies the set of all possible actions that agents can take in the environment. In our implementation we define this as a Multi Discrete space, where each robot can take four possible actions:

move up, down, left, or right. This allows for simultaneous action selection by multiple agents.

- *Observation Space:* The observation space represents the state of the environment as seen by the (RL) agents. In our environment it is implemented as a dictionary, containing the positions of robots, goals, and obstacles. This comprehensive representation provides all necessary information for the agents to make informed decisions.
- *Rendering Attributes:* Optionally, a custom environment could include rendering to have a better visualization. We chose to implement a render method (and close method for cleanup afterwards) using the Pygame library. This required some additional attributes used for rendering purposes. However, we won't be explaining this in detail since this is not of relevance for this research.

The class includes a reset method which initializes the environment to a starting state at the beginning of each episode. This method randomly places the robots on the grid, avoiding overlaps with obstacles, other robots, and goals.

The core functionality of the custom environment's dynamics remains to be described in the *step* function. The main difference is that we now consider all robots inside a single step, instead of considering them individually. This means that instead of one action, we receive an array of actions with the indices corresponding to the right robots. The method processes each robot's movement, checks for collisions and validity, and calculates rewards, while also checking if all robots have reached their goals. Each of these operations is implemented in smaller functions which match the functions in our previous own environment and robot classes like e.g. the move method inside the robot class. The reward calculation also takes place as before, using the same reward values. This way we can easily compare the outcomes of both environments.

Lastly, we added the (optional) methods *render* and *close*. The *render* method was especially useful for visualizing the environment during development and debugging. While the *close* method ensured that resources used for rendering are properly released when the environment is no longer in use, preventing memory leaks and other issues. However, as said before, we won't go into further detail since this is irrelevant to the subject of this work. Figure 21 is an example of what the rendering looks like.

After the implementation of the navigation algorithm, we can proceed with the integration of energy harvesting into the custom Gymnasium environment. Again, you will see a lot of resemblance with our previous own environment.

We start by adding the essential new attributes. This will be a combination of the additional attributes to both the environment and robot classes from before.

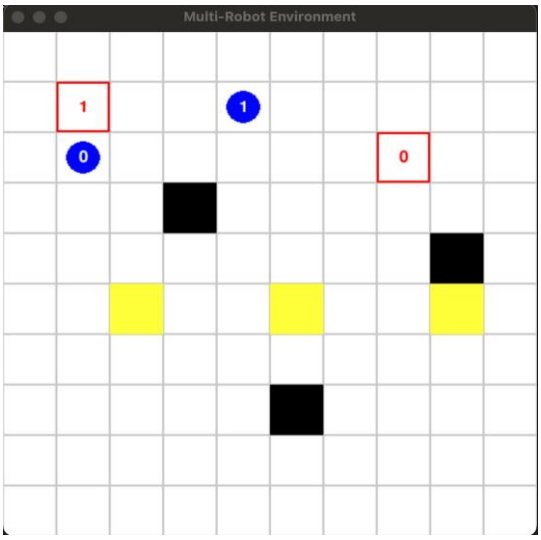

Figure 21. *An Example of how we render the Grid Environment*

- *Light sources:* As before, we need energy sources within the environment that robots can utilize for energy harvesting. Important to note is that this is also added to the observation space attribute, just like goals and obstacles.
- *use EH:* We still want to be able to compare the results with and without the use of energy harvesting. So, we again introduce a Boolean flag which determines whether energy harvesting is enabled.
- *Harvested Energy:* Because we don't have a separate robot class anymore, we keep track of the harvested energy in another array just like the positions and goals.

The reset method is adapted so that after resetting the robot's positions, we also rest the harvested energy to zeroes for all robots.

We added the same functionality to calculate the amount of energy that can be harvested as in the robot class. We estimate the energy a robot can collect based on its proximity to the light sources. This process is then added to the *step* method and utilized when calculating the weighted sum as the total reward instead of just calculating the moving reward.

7.2.2. Step 2: Training the Environment Using RL

RL Training: After completion of the first step, we can proceed to the second step in which we will implement and train our reinforcement learning algorithms. However, first a decision has to be made on which RL algorithms are chosen for training our complex environment. Based on prior analysis of widely used RL algorithms, we decided to use Q-learning and PPO as our chosen RL algorithms. We also need to define the state and action space, which is the same for both RL techniques. The action space consists of four actions since we can only move in four different directions inside a grid: up, down, left and right. For the state space we only consider the position of the robot. When training these RL techniques, we try to retrieve the optimal action for each state-action pair. If we were to also consider the amount of energy harvested in our state space, this would lead to an

exponential rise in number of state-action pair. This would make the execution time for training much larger, thus we decided it is better to only have the position of the robot as the state.

***Q-Learning Algorithm*:**

In this section we will discuss the Q-learning algorithm we implemented for our own environment. The logic of the Q-learning agent was designed as a class. In general, the algorithm learns the value of actions taken in various states, guiding the agent to make better decisions over time. In Figure 22 a flow chart is provided, which offers a schematic overview of the iterative learning process of the Q-learning algorithm.

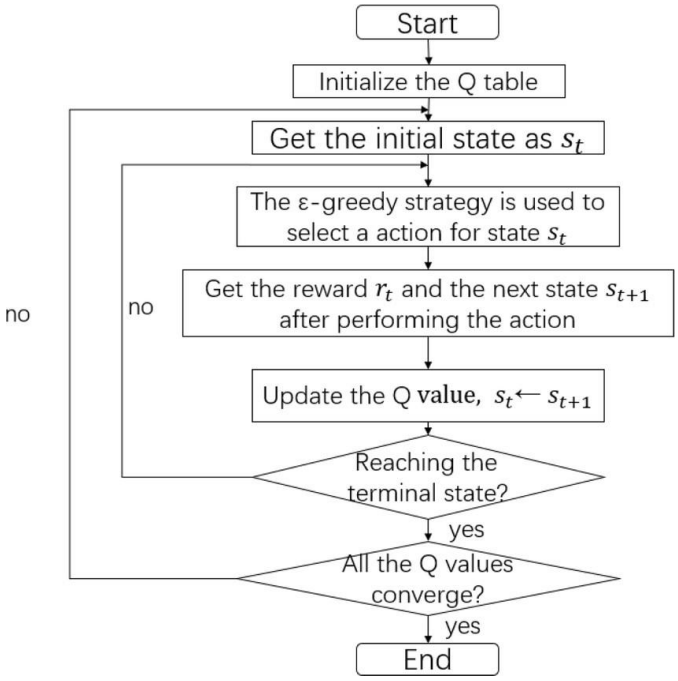

Figure 22. *Flowchart of the Learning Process of Q-learning [76]*

The implementation involves several key attributes that are essential for the agent's functioning:

- *Action Size:* This specifies the number of possible actions the agent can take. In our grid environment this means a robot can take actions up, down, left and right.
- *Epsilon ϵ:* This constant controls the exploration rate for the epsilon-greedy policy.
- *Gamma γ:* This constant is the discount factor. It represents the importance of future rewards compared to immediate rewards. It is used to calculate the value of future states.
- *Learning Rate α:* A constant which determines the rate at which Q-values are updated based on new information.
- *Q-Table:* A dictionary storing Q-values for every possible state-action pair.

Epsilon, gamma and learning rate are constants for which we predefined the value. An overview of these constant values can be found in Table 4.

Table 4. *Values of Q-Learning Constants*

Parameter	Value
Q Epsilon ϵ	0.4
Q _Gama γ	0.9
Q Learning Rate α	0.1

The RL algorithm consists of two core methods: act and $learn$.

The act method chooses an action based on the epsilon-greedy policy. With probability epsilon ϵ, the agent selects a random action to explore new possibilities. Otherwise, meaning probability $1 - \epsilon$, it selects the action with the highest Q-value for the given state. If the state does not yet exist in the Q-table, it means that that state has never been visited before and it initializes that state with $-\infty$ values for each possible action to represent unknown Q-values.

The $learn$ function updates the Q-values based on the observed transition from the current state to the next state. First, it ensures that both the current state and the next state are present in the Q-table, initializing them if necessary. Next, the Q-value for the state-action pair is then updated using the Q-learning update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \quad (7)$$

where s is the current state, a is the action taken, r is the reward received, s' is the next state, and a' represents possible actions in the next state.

Now we use this Q-learning agent class to train our own environment. Training involves repeatedly interacting with an environment to learn the best actions to take in various states to maximize cumulative rewards.

Thus, we will loop over a numerous number of episodes. In each episode we start by resetting the environment, making the robots start from new random positions. This ensures varied experiences across episodes.

Next, we loop over a maximum number of steps. This represents the maximum number of steps robots can take before reaching their destination. If they exceed that number, it means the algorithm didn't find the right q-values yet for all state-action pairs. Otherwise, at the end of this loop we will check whether all robots have reached their destinations, skipping to the next episode if that is the case.

Now, since we have a swarm of robots, a robot works in a decentralized way. This means every robot has such a Q-learning agent class attribute of its own. This means every robot trains their own agent separately. Thus, the next step will be to loop over all robots, ensuring each robot will get the chance to let their Q-learning agent learn.

For each robot we first check if the robot has already reached its destination. If so, we can move on to the next robot since the current one is done. Then we chose an action with the use of the act function from the Q-learning agent of the robot. After that we take that action by taking a step in our own environment. And lastly, we update the Q-values of the agent by learning from the next state and reward we obtained after taking that step.

***PPO Algorithm*:**

We implemented a custom Gymnasium environment for modeling PPO. In the Gymnasium library, different RL algorithms are already implemented and updated to the newest advances. This means we no longer need our own implementation of PPO. We could simply use the PPO implementation of *stable baselines3*. This is another library in Python that provides a set of reliable and well-tested implementations of popular RL algorithms. It is built on top of PyTorch and is a successor to Stable Baselines, which was built on TensorFlow. As a reminder of how the internal flow of the training process of PPO words, we provide a PPO flowchart in Figure 23.

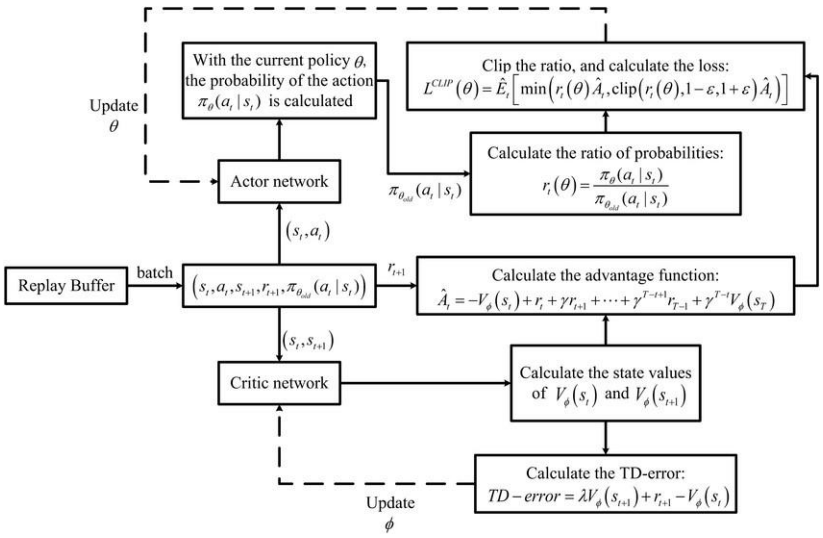

Figure 23. *Flowchart of the Learning Process of PPO [44]*

Now we use *stable baselines3*'s PPO to train our custom Gymnasium environment. We start by creating the PPO model. This model had most parameters already correctly initialized, except for the policy which in our case is called *'Multi Input Policy*' and the environment itself of course. The model can then use it learn method to properly train the PPO with the environment. The total timesteps we need for learning depends on the complexity of the environment, meaning it depends on the number of robots are present in our environment. Note that this

parameter becomes exponentially larger the more robots we use which unfortunately also means training takes more time.

7.2.3. Step 3: Design and Execution of Tests

Lastly, we can save the trained model so when testing we do not need to waste time on retraining it every time. This third and final step involves the design and execution of test cases, which we refer to as "environment instances" trained using specific RL techniques. After executing the tests, a test report will be generated containing performance information about each environment instance, as well as specific path information based on initial robot positions.

Test Framework:

Testing an environment instance involves three main steps:

- **Building the environment:** This step requires defining the size of the grid, the positions of robot goals, obstacles, light sources, and whether energy harvesting is enabled.
- **Training the environment:** This is done using the selected RL algorithm (Q-learning for our custom environment or PPO for the Gymnasium environment). Each algorithm is trained over a specified number of episodes or timesteps.
- **Navigating the environment:** After training, the environment is navigated based on the RL model predictions to evaluate performance.

For both our custom environment and the Gymnasium-based PPO environment, the process starts by building the environment instance. In our custom environment, we also separately construct the robots based on their given goals, before connecting them to the environment.

For Q-learning, training is conducted over several episodes, with a maximum number of timesteps per episode. Setting a maximum timestep helps prevent the agent from getting stuck in infinite loops and ensures efficient exploration. Conversely, in PPO, the total number of timesteps is set across all episodes, and episodes end dynamically based on the environment's characteristics.

The final step is navigation, where we start from fixed initial positions for each robot and use the RL model's predictions to navigate through the environment. During this process, we track the path length, total reward, and total energy harvested. If a robot fails to reach its goal within the maximum number of steps, we consider the attempt a failure.

Notably, in our custom environment, each robot trains its own Q-learning agent. In contrast, in the PPO model, all robots are trained collectively, which improves collision avoidance. During navigation, we introduce an additional check in the custom environment to prevent collisions: if the best predicted action leads to a collision, the robot reverses its step and selects the next best action.

Finally, the result are saved in the following types of files:

- **Q-tables:** For each robot in our custom environment, we save Q-tables that visualize the states as a grid, showing the Q-values and the optimal action for each state.
- **Path comparison files:** These files showcase four paths for environments trained with and without energy harvesting, illustrating the difference in behavior.
- **Results tables:** These tables summarize the average results, including path lengths, total rewards, energy harvested, and success/failure rates for each RL technique.

Parameter Test

When training an RL model, choosing the right parameter values is critical. Since these values cannot be known beforehand, we conducted an extensive parameter test using a trial-and-error approach. This test investigates which parameter values yield the best performance for each environment instance.

We identified several key parameters that influence performance:

- **Number of episodes/timesteps:** Training the RL model for too few episodes results in underfitting, while training for too many episodes leads to overfitting.
- **Reward function values:** We varied several aspects of the reward function, including:
    - **Reward Done:** When a robot has reached it's individual goal in the environment, we assign a positive reward to teach the robot to navigate and end up here.
    - **Penalty Invalid Move:** When a robot tries to go out of the bounds of the grid, we assign a negative reward to learn the robot to stay inbound.
    - **Penalty Collision:** When a robot collides with either an obstacle or another robot in the environment, we assign a negative reward to ensure the robot learns to not cause collisions.
    - **Penalty Move:** When a robot takes a step, without making an invalid move or causing a collision, we assign a small negative reward to ensure the robot tries to find the goal in a minimum number of steps.
    - **Reward EH & EH Power:** A positive reward is given for energy harvested based on proximity to light sources, with two possible formulas:

        1. A linear function that reduces the reward as the robot moves further from the light source $\frac{1}{distance}$.

        2. An exponential function that penalizes distance more sharply, potentially offering better learning outcomes for energy harvesting behavior $\frac{1}{distance^2}$.

We assigned several possible values to each parameter and tested the resulting environment instances. Since the number of parameter combinations grows exponentially, we limited our tests to two basic cases:

1. **Basic Case 1:** One robot, three obstacles, and three light sources in a 10x10 grid.
2. **Basic Case 2:** Two robots, the same obstacles, and light sources, also in a 10x10 grid.

Testing with two basic cases allowed us to explore how the presence of multiple robots affects training, reward values, and overall performance. For each case, we trained both the custom Q-learning environment and the Gymnasium PPO environment.

Finally, we used the results of the PPO parameter test (which is more sensitive to parameter variations) to inform the optimal reward values for the Q-learning environment. This ensured that the reward function was consistent between the two environments, allowing us to make meaningful comparisons between their performances.

7.3. Results

After running the extensive parameter test, we analyzed the results to determine the optimal parameter values that maximize performance. Since we varied several different values for each parameter, we generated 864 different environment instances from a single basic case. An overview of all tested parameters is provided in Table 5. We analyzed the results separately for the two basic cases (one robot and two robots).

The process of selecting the best parameter-values is the same for both base cases so we will focus only on the first base case with one robot. We run the parameter test several times, each time considering a different number of starting points when executing the navigation of the test. It is important to note is we chose to run the navigation with deterministic predictions from PPO. This resulted in three CSV files: one containing the results of 864 different environment instances when executing the navigation with 10 starting points, one with all results when executing the navigation with 20 starting points and the last one with 50 starting points.

Table 5. *Overview of Tested Values for Each Parameter*

Parameter	Base Scenario 1	Base Scenario 2	Both Scenarios
Training Timesteps	20000, 50000	50000, 100000	-
Reward Done	100, 500	500, 1000	-
Penalty Invalid Move	-	-	-1, -2, -5
Penalty Collision	-	-	-2, -5, -10

Penalty Move	-	-	-0.1, -0.2, -0.5
Reward EH	-	-	2, 5
Reward EH Power	-	-	1, 2

Each of these resulting csv-files were analyzed so that we could generate the top 10 best choices for each set. For the analysis, we added an $Anomaly$ column to the data. This column checks whether the average energy harvesting enabled is greater than the average harvested energy when training without energy harvesting. This is crucial, as we expect the algorithm to prioritize energy harvesting even if it takes a longer path.

Once anomalies were filtered out, we computed the ratios of average harvested energy and average path length for both cases (with and without energy harvesting). These ratios reveal how much energy harvesting is considered. To evaluate the efficiency of the trade-off between energy harvesting and path length, we created a trade-off column that calculates the ratio of the energy harvesting gain to the path length increase. This trade-off measures how well the algorithm balances energy harvesting with the cost of potentially longer paths.

After filtering out failure cases, we sorted the results by this trade-off value in descending order, generating a list of the top 10 best parameter sets for each results set.

Table 6. *Overview of Optimal Parameter-Values for Base Case 1 values that will optimize the results.*

Parameter	Value
reward done	100
penalty invalid move	-2
penalty collision	-5
penalty move	-0.2
reward EH	5
EH power	2
total timesteps	50000

Table 7. *Overview of Optimal Parameter-Values for Base Case 2*

Parameter	Value

reward done	1000
penalty invalid move	-5
penalty collision	-2
penalty move	-0.2
reward EH	5
EH power	2
total timesteps	100000

This process is then repeated for our second basic instance that contains two robots. Remember, we varied over different values since the testing and training with two robots becomes more complex. Table 7 shows the resulting optimal parameter as mentioned in Section Parameter Test, we used the custom gymnasium environment trained with PPO for our original parameter test. Now we have a small parameter test testing our own environment trained with Q-learning, to discover the optimal episode and maximum timesteps. Again, both base cases are considered, and we will use the optimal reward parameter-values from each case in our environment instances as well. The process of selection is again repeated with filtering out anomalies and failures, calculating the energy harvesting ratio and path length ratio, using this to calculate the trade-off and ordering that trade-off in a descending way. This results in the optimal values for our first base case shown in Table 8. And these optimal values for the second base case are found in Table 9.

Table 8. *Additional Optimal Parameter-Values for Base Case 1*

Parameter	Value
Episodes	1000
Maximum timesteps	100

Table 9. *Additional Optimal Parameter-Values for Base Case 2*

Parameter	Value
Episodes	10000
Maximum timesteps	200

7.4. Evaluation

In this section, we define the evaluation criteria used to validate our algorithm and then describe the different scenarios tested to assess the implementation. Finally, we examine the results for each test scenario based on these criteria.

Evaluation Criteria:

To validate our algorithm against the test scenarios, we use the following three evaluation criteria:

- **Navigation performance:** We evaluate the average path length and the amount of energy harvested, both with and without energy harvesting.
- **Comparison of RL techniques:** We compare the two RL techniques (Q-learning and PPO) based on execution time and the quality of results.
- **Impact of environmental and system factors:** We analyze how varying environmental affect algorithm's performance and efficiency.
- **systematic factors:** The system parameters impact on algorithm's performance is investigated through different examination.

These criteria were chosen to directly address the research questions outlined in our work. The first criterion, focused on path length and energy harvesting, is aligned with RQ1 and RQ2, which investigate the balance between efficient navigation and energy harvesting. The second criterion evaluates the efficiency and effectiveness of each RL technique (RQ1.1 and RQ4). The third criterion assesses the robustness and adaptability of the algorithm to different conditions (RQ3).

Test Scenarios:

We designed two types of test scenarios: environmental changes and system changes. Table 10 provides an overview of the parameters and values used to generate each scenario.

Table 10. *Overview of All Different Parameter-Values of All Scenarios*

	Parameter Values			
Environmental Scenarios				
Multi Robot	1	2	-	-
Grid size	5x5	10x10	20x20	-
Light source	1	3	7	7 (clustered)
System Scenarios				
Reward Weight	40/60	50/50	60/40	80/20
EH Range	2	3	5	-

These scenarios modify the physical environment, such as the number of robots or obstacles, or make systematic changes to the implementation, such as the range for energy harvesting or the reward weight.

To ensure consistency, we use basic cases introduced earlier as starting points when generating different scenarios, modifying one variable per test. This approach allows us to make meaningful comparisons across different scenarios while keeping most factors constant. The basic cases are derived from the parameter tests described earlier, which identified the most optimal parameter values for performance.

Environmental Test Scenarios:

We designed the following three types of environmental test scenarios:

- **Multi Robot Scenarios:** These scenarios vary the number of robots in the environment. Training an environment with multiple robots is significantly more complex than with a single robot, so we examine both the single-robot base case and the two-robot base case to observe differences in training difficulty and results. Although it would be ideal to test with more than two robots, the exponential increase in computation time makes this impractical within the scope of this work.
- **Grid Size Scenarios:** These scenarios vary the size of the grid, examining environments with a 5x5 grid and a 20x20 grid. Larger grids present more possible states and increase the complexity of training. We created two smaller base scenarios for the 5x5 grid, ensuring that all positions (goals, obstacles, light sources) fit within the smaller grid.
- **Light Source Scenarios:** We also vary the number of light sources, a critical aspect of energy harvesting. In addition to the three light sources in the base cases, we examine environments with one light source, seven scattered light sources, and seven clustered light sources. These scenarios allow us to observe how the number and distribution of light sources affect energy harvesting efficiency.

System Test Scenarios:

Systematic test scenarios modify internal system parameters, such as the reward weights and energy harvesting range:

- **Reward Weight Scenarios:** These scenarios adjust the weights in the reward function, which balances movement and energy harvesting rewards. While the base cases use a 50/50 weighting, we also test with 40/60, 60/40, and 80/20 weightings to observe the impact of emphasizing one reward over the other.
- **Energy Harvesting Range Scenarios:** Here, we vary the range within which robots can detect and harvest energy. The base cases use a range of three

steps, but we also test with a smaller range (two steps) and a larger range (five steps). This allows us to investigate how the distance from light sources affects energy harvesting behavior.

Results Overview:

After running all the test scenarios, the results are analyzed according to the defined evaluation criteria. These results are stored in a test report, which includes general performance data as well as detailed information like Q-tables and specific navigation paths. The evaluation of these results will help us determine how well the algorithm performs under different conditions and whether it meets the expected outcomes.

7.4.1. EC1: Navigation Performance

For the first criterion, we review navigation performance in terms of the average path length and the amount of energy harvested, both with and without considering energy harvesting. An overview of the results for the basic scenarios can be found in Table 11.

Table 11. *Overview of the Results of the Base Scenarios*

Multi-Robot Tests

1 Robot

		Number of Failures	Number of Successes	Average Path Length	Average EH	Anomaly	EH Ratio	Path Ratio	Trade-off (EH / Path Length)
Q-learning	No EH	0	10	6.8	3.61	False	280.33 %	126.47 %	221.66 %
	EH	0	10	8.6	10.12				
PPO	No EH	0	10	6.8	3.09	False	425.57 %	155.88 %	273.0 %
	EH	0	10	10.6	13.15				

2 Robots

		Number of Failures	Number of Successes	Average Path Length	Average EH	Anomaly	EH Ratio	Path Ratio	Trade-off (EH / Path Length)
Q-learning	No EH	0	10	7.9	3.66	False	100.27 %	100.0 %	100.27 %
	EH	0	10	7.9	3.67				
PPO	No EH	0	10	13.15	3.62	False	228.45 %	95.44 %	239.38 %
	EH	0	10	12.55	8.27				

In **Basic Case 1**, we observe that the average path length increases slightly when the algorithm incorporates energy harvesting, while the average energy harvested rises more significantly. In the Q-learning environment, the path length increases by 126%, while the energy harvested increases by 280%, resulting in a trade-off percentage of 222%. This trade-off percentage measures the algorithm's ability to increase energy harvesting with minimal impact on path length. The PPO environment model shows even better performance, with the path length increasing by 155% and energy harvesting rising by 425%, leading to a higher overall trade-off of 273%.

In **Base Case 2**, the results are similar, although there are a few anomalies. In Q-learning, the average path length remains unchanged, which is unexpected but not necessarily problematic, as energy harvesting still improves. In PPO, the path length is slightly longer when energy harvesting is not considered, which

contradicts our expectations that path lengths would increase with energy harvesting. This discrepancy is likely due to non-deterministic action predictions during training, which occasionally lead to suboptimal paths.

Figure 24 shows an example of robot navigation after training with Q-learning while considering energy harvesting. The timestamps on the paths illustrate that although the robots cross paths, they do so at different times, demonstrating that the algorithm has effectively learned to avoid collisions.

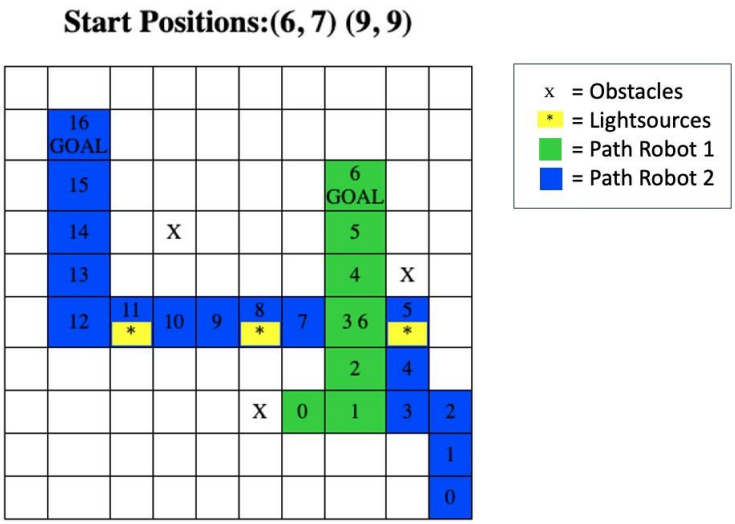

Figure 24. *A single path example after training with Q-learning considering Energy Harvesting*

In summary, these results align with our expectations. Training with energy harvesting capabilities leads to a significant rise in the amount of energy that is harvested, while the difference in average path length stays at a minimum. This means that our algorithm can successfully balance the trade-off between these ratios. When comparing the results of our two algorithms we can state that the overall performance of the PPO algorithm exceeds the performance of Q-learning.

7.4.2. EC2: Effectiveness of RL techniques

The second criterion compares the effectiveness of the two RL techniques (Q-learning and PPO) in terms of execution time and results. Since we already discussed the results in the previous section, we now focus on the execution times.

Our process includes three main steps, with Tables 12 and 13 presenting the execution times for each step:

1. **Build Time:** In Q-learning, the build time is negligible, even in more complex environments. In PPO, the build time is slightly longer, especially in the first test (**Basic Case 1** without energy harvesting), but this can be attributed to loading the necessary libraries. This delay only affects the initial build and is not a concern for subsequent tests.
2. **Training Time:** Training times increase significantly in the second base case (multi-robot environment), as expected. PPO, being a deep reinforcement learning technique, has longer training times compared to Q-learning,

which uses a simpler learning formula. Training with energy harvesting also takes longer due to the more complex reward function that considers both movement and energy harvesting.

3. **Test Time:** Test times for both Q-learning and PPO are negligible, similar to build times.

Table 12. *Execution times for Q-learning*

	Base case 1		
	Build Time	Train Time	Test Time
No EH	5.48e-05	0.6817	0.0008869
EH	0.0006337	2.1972	0.0028450
	Base case 2		
	Build Time	Train Time	Test Time
No EH	0.0002930	17.8919	0.0049081
EH	0.0002639	19.6207	0.0047820

Table 13. *Execution times for PPO*

	Base case 1		
	Build Time	Train Time	Test Time
No EH	1.3543	188.6143	0.6185
EH	0.2887	195.3210	0.2676
	Base case 2		
	Build Time	Train Time	Test Time
No EH	0.2553	444.4896	0.5017
EH	0.2047	473.9146	0.5313

We start by focusing on the execution times to build the environment. As you will notice for our Q-learning technique this costs us less than a millisecond of time, even when the environment becomes more complex. We can safely say that these timings are negligible.

Figures 25 and 26 show examples of navigation paths after training with Q-learning and PPO, respectively. We used the same starting points for both algorithms to allow for direct visual comparison.

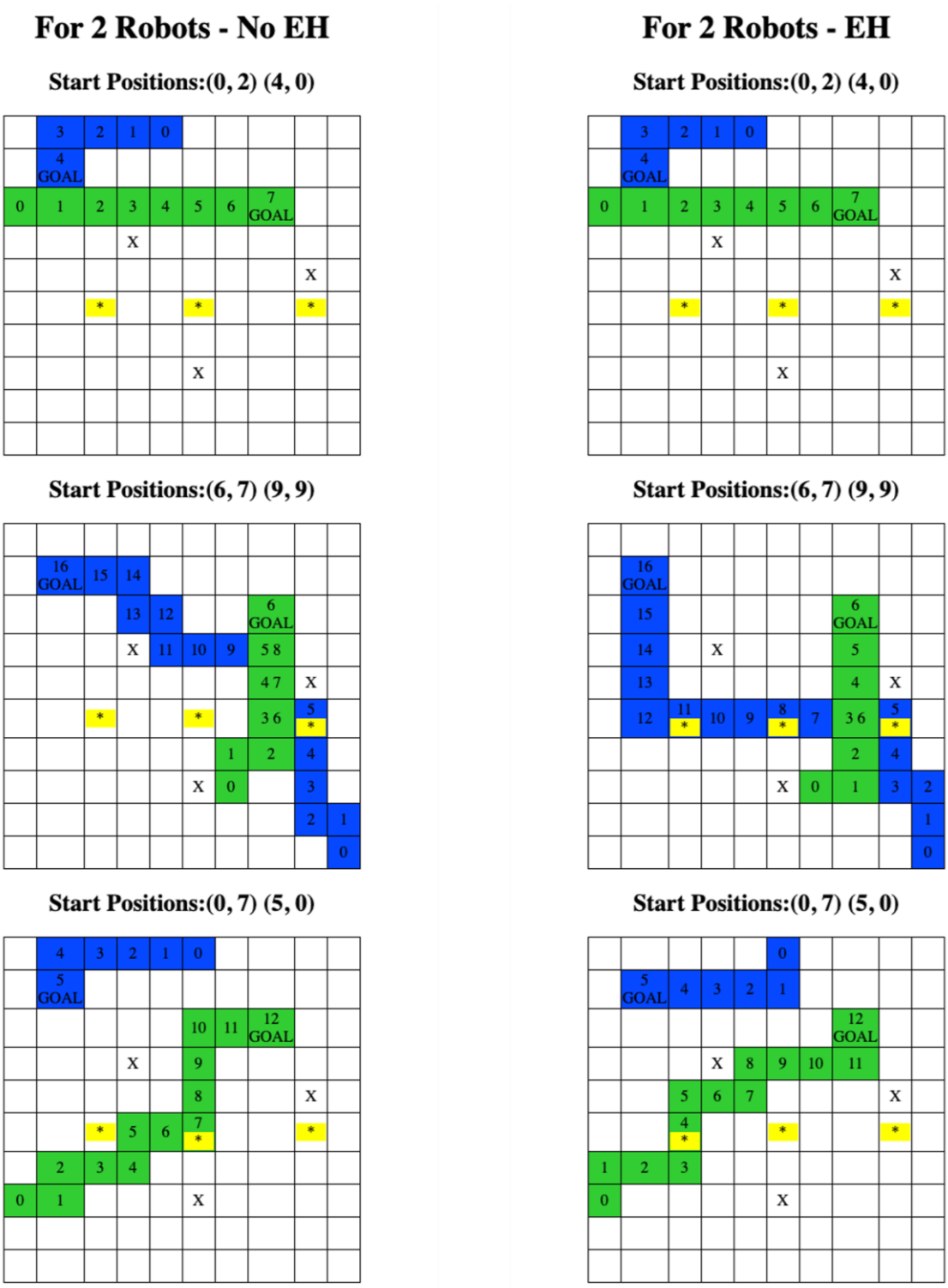

Figure 25. *Path examples after training with Q-learning*

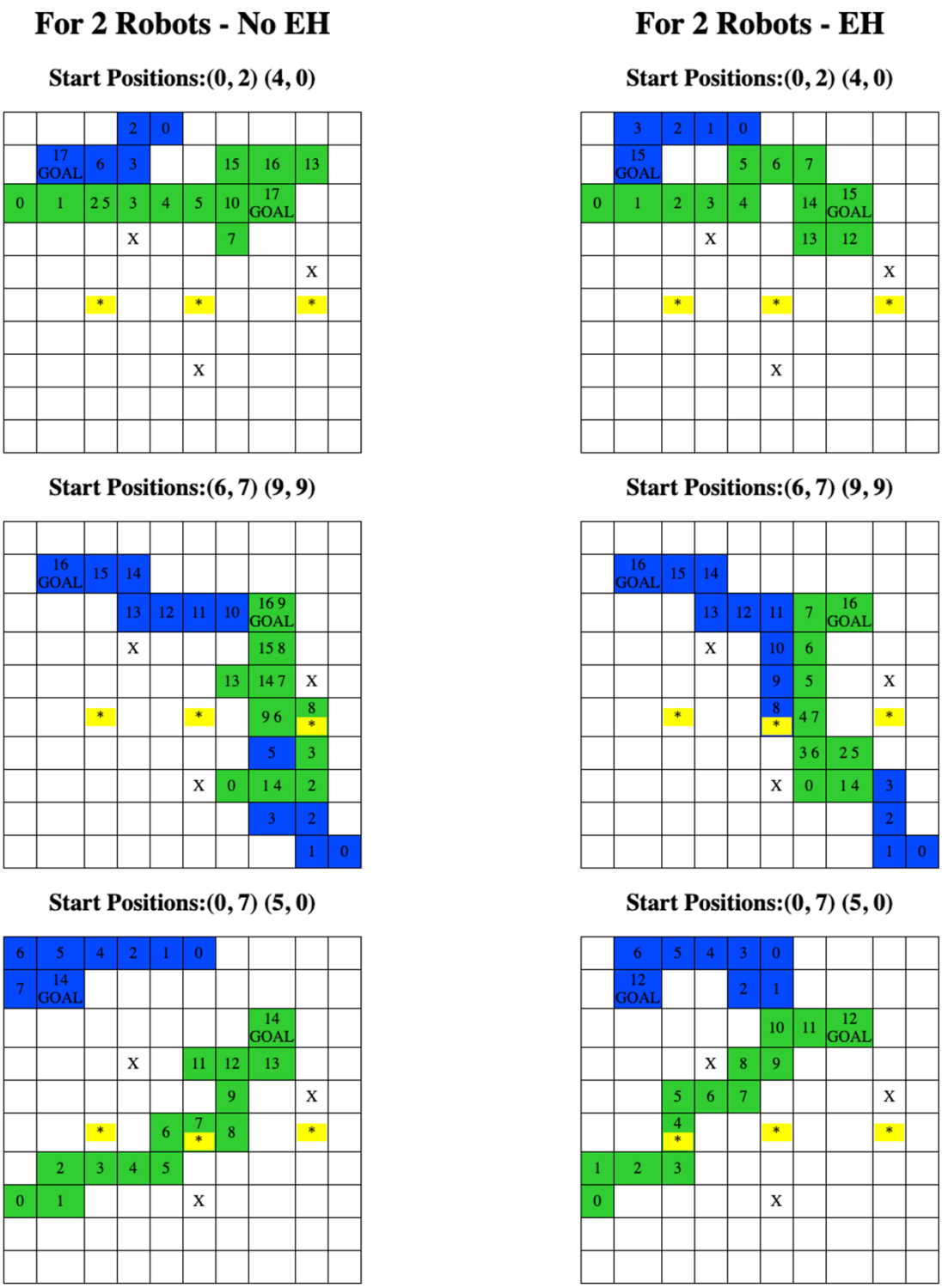

Figure 26. *Path examples after training with PPO*

7.4.3. EC3: Environmental Test Results

This scenario considers the multi-robot case, which essentially retests the basic cases. Since these results were already discussed, we move on to the grid size and light source scenarios.

- **Grid Size Scenarios:** In the grid size tests, we initially used the optimal parameters from the base cases. However, these resulted in suboptimal outcomes, particularly in the smallest grid (5x5), where over-training likely occurred. We ran a smaller parameter test (Table 15) to adjust the parameters for the grid size. The results (Table 16) indicate that smaller grids lead to shorter paths and less energy harvesting, while larger grids increase both path length and energy harvested. The only anomaly occurred with PPO in the 20x20 grid, where non-deterministic predictions likely caused suboptimal performance.
- **Light Source Scenarios:** As expected, reducing the number of light sources decreases the amount of energy harvested, while increasing the number of light sources boosts energy collection, particularly when the sources are clustered. These results highlight the importance of light source distribution in energy harvesting.

Table 14. *Overview of the results of the grid size scenarios after execution with base case optimal parameter-values*

Gridsize Tests

Gridsize = 5x5

			Number of Failures	Number of Successes	Average Path Length	Average EH	Anomaly	EH Ratio	Path Ratio	Trade-off (EH / Path Length)
1 Robot	Q-learning	No EH	0	10	2.9	0.97	False	149.48 %	100.0 %	149.48 %
		EH	0	10	2.9	1.45				
	PPO	No EH	0	10	5.3	1.47	False	108.16 %	107.55 %	100.57 %
		EH	0	10	5.7	1.59				
2 Robots	Q-learning	No EH	1	9	2.45	0.73	False	154.79 %	118.37 %	130.77 %
		EH	0	10	2.9	1.13				
	PPO	No EH	0	10	8.5	3.87	True	72.35 %	57.65 %	125.51 %
		EH	0	10	4.9	2.8				

Gridsize = 20x20

			Number of Failures	Number of Successes	Average Path Length	Average EH	Anomaly	EH Ratio	Path Ratio	Trade-off (EH / Path Length)
1 Robot	Q-learning	No EH	0	10	6.8	1.55	False	387.74 %	114.71 %	338.03 %
		EH	0	10	7.8	6.01				
	PPO	No EH	0	10	6.8	2.66	True	16.54 %	45.59 %	36.28 %
		EH	8	2	3.1	0.44				
2 Robots	Q-learning	No EH	0	10	8.7	4.19	False	172.32 %	95.4 %	180.62 %
		EH	0	10	8.3	7.22				
	PPO	No EH	1	9	10.55	3.0	True	57.67 %	114.69 %	50.28 %
		EH	1	9	12.1	1.73				

Table 15. *Overview of values in the parameter test for the grid size scenarios*

Grid Size	Base Case	Parameter Values		
		Total Timesteps	Episodes	Max Timesteps

5	Base Case 1	10000, 20000, 50000	500, 1000, 2000	50, 100
	Base Case 2	20000, 50000	1000, 2000, 3000	100, 200
20	Base Case 1	20000, 50000, 100000	1000, 5000, 10000	100, 200
	Base Case 2	50000, 100000	5000, 10000, 20000	200, 400

As in our parameter test, in the process of selecting the optimal parameter values, the results need to hold the following properties:

1. We do not want our results to have failure cases when testing the navigation after training.
2. We do not want the results to contain anomalies, meaning the average amount of harvested energy when training without energy harvesting properties should not be greater than the average amount when training with energy harvesting considered.
3. We do not want the path length when training only without energy harvesting, to be larger than the path length of training while considering energy harvesting.

Afterwards, we sort the trade-off between the average energy harvesting ratio and the average path length ratio in a descending way. Remember that this trade-off represents how well the results balance energy harvesting against the cost of a potentially longer path. This will reveal the best options for our parameter values in descending order. Table 16 provides an overview of the new optimal parameter-values for the grid size scenarios.

Table 16. *Overview of new optimal parameter-values for the grid size scenarios*

Grid Size	Base Case	Parameters		
		Total Timesteps	Episodes	Max Timesteps
5	Base Case 1	20000	2000	100
	Base Case 2	50000	3000	100
20	Base Case 1	20000	1000	100
	Base Case 2	100000	5000	200

Now that we have the optimal parameter values, we can execute the tests to obtain the results. In Table 17, we constructed an overview of all the results.

Table 17. *Overview of the Results of the Grid size Scenarios*

Gridsize Tests

Gridsize = 5x5

			Number of Failures	Number of Successes	Average Path Length	Average EH	Anomaly	EH Ratio	Path Ratio	Trade-off (EH / Path Length)
1 Robot	Q-learning	No EH	0	10	2.9	2.6	False	103.85 %	100.0 %	103.85 %
		EH	0	10	2.9	2.7				
	PPO	No EH	0	10	4.9	3.7	True	86.49 %	69.39 %	124.64 %
		EH	0	10	3.4	3.2				
2 Robots	Q-learning	No EH	0	10	2.9	0.82	False	119.51 %	96.55 %	123.78 %
		EH	0	10	2.8	0.98				
	PPO	No EH	0	10	7.45	3.24	False	214.2 %	104.7 %	204.59 %
		EH	0	10	7.8	6.94				

Gridsize = 20x20

			Number of Failures	Number of Successes	Average Path Length	Average EH	Anomaly	EH Ratio	Path Ratio	Trade-off (EH / Path Length)
1 Robot	Q-learning	No EH	0	10	7.2	5.3	False	190.57 %	108.33 %	175.91 %
		EH	0	10	7.8	10.1				
	PPO	No EH	6	4	10.0	9.9	False	142.42 %	137.0 %	103.96 %
		EH	0	10	13.7	14.1				
2 Robots	Q-learning	No EH	0	10	8.4	2.44	False	318.03 %	105.95 %	300.17 %
		EH	0	10	8.9	7.76				
	PPO	No EH	0	10	14.1	5.78	False	162.46 %	84.4 %	192.49 %
		EH	1	9	11.9	9.39				

When operating in smaller grids, we expect to find the path faster but also harvest only little amounts of energy. This is natural since we reach the destination faster and thus have less opportunity to harvest. These characteristics are clearly found in the results of the 5x5 grid when training with both RL techniques. Note that although PPO has an anomaly, it still performs better than our Q-learning algorithm. When operating in larger grids, we expect both the average path length and average energy harvesting to be larger. This is also found in the 20x20 grid scenario results. The average energy harvesting increases for both techniques as well as the average path length. Note that the only situation where this is not the case is when training our PPO with 2 robots in a 20x20 grid. The overall performance here is also worse than that of Q-learning this time. We can explain this because our navigate execution in PPO predicts the actions non-deterministically. So there is a (small) chance we won't receive the best action when we predict, but instead get a random action.

When we compare the results of the different grid sizes to assess the impact of this type of environmental changes, we observe that the increase in grid size leads to longer path lengths and lower rewards across all conditions. The effect is more noticeable when we train with energy harvesting, where the average harvesting and path ratios are significantly impacted in the larger grid. The results matched our overall expectations but were not as originally thought.

The last scenario regarding environmental changes revolves around the light source scenario. Here we not only differ in the number of light sources but also consider the way they are distributed over the grid. Again, we mentioned we might have to research some alternative optimal parameter values. However, after executing the scenarios with the optimal parameter values of the basic cases, we found the results to be good, so we deemed running yet another computation-intensive parameter test to be unnecessary.

The overview of these results is found in Table 18. Originally, we expected that the average amount of energy harvesting will dramatically decrease when we limit the number of light sources in our environment. Especially for the scenarios that do not consider energy harvesting.

The results of only one light source in the environment match our expectations fully. In comparison to our base cases that have 3 light sources, the average amount energy harvesting is significantly lower. Unlike, the average path length stays the same overall.

When a larger number of light sources are present, our expectations are that there will be an increase in the amount of energy harvesting for energy harvesting based navigation strategies as well as navigation strategies without energy harvesting. However, depending on their distribution this increase will be dramatic. Imagine an environment in which the light sources are clustered together. Your path can only adjust a little so that we can pass through the cluster enabling us to harvest the energy of all those light sources at once. When the same number of light sources are distributed sparsely all over the grid, we would have to increase our path length drastically to be able to go past all of the light sources before reaching the goal. These expectations are only partially evident in our results. Unfortunately, not all results show these characteristics due to insufficient parameter-tuning. This proves that our algorithm works but is very sensitive to these parameter-values. However, the process of searching for these optimal parameters is very time-consuming.

7.4.4. EC4: Systematic Changes Test Results

The first systematic change we tested was the reward weight. As expected, increasing the weight of energy harvesting led to higher energy collection but also caused failures in PPO, as the algorithm prioritized energy harvesting over navigation. Conversely, increasing the importance of navigation reduced energy harvesting without significantly affecting path length. Next, we tested the energy harvesting range. Reducing the range had little impact on energy harvesting, likely because the change (from three to two steps) was too small. Increasing the range to five steps, however, resulted in a noticeable improvement in energy harvesting, with a slight increase in path length, especially in PPO.

Table 19 holds the overview of the results of the reward weight scenarios. For this type of scenarios, we expect to see longer path lengths when we make the energy harvesting reward more significant. While on the other hand we expect to see

declining values for energy harvesting the more importance we give to finding the optimal path.

Table 18.: Overview of the Results of the Light source Scenarios

Lightsource Tests

1 lightsource

			Number of Failures	Number of Successes	Average Path Length	Average Reward	Average EH	Anomaly	EH Ratio	Path Ratio	Trade-off (EH / Path Length)
1 Robot	Q-learning	No EH	0	10	7.0	99.4	4.96	False	298.99 %	120.0 %	249.16 %
		EH	0	10	8.4	50.89	14.83				
	PPO	No EH	0	10	6.8	99.42	5.94	False	177.78 %	111.76 %	159.06 %
		EH	0	10	7.6	50.41	10.56				
2 Robots	Q-learning	No EH	0	10	6.6	998.62	11.48	False	295.56 %	103.03 %	286.86 %
		EH	0	10	6.8	501.51	33.93				
	PPO	No EH	0	10	13.4	985.7	35.55	False	115.02 %	115.67 %	99.44 %
		EH	0	10	15.5	492.12	40.89				

7 lightsources (sparse)

			Number of Failures	Number of Successes	Average Path Length	Average Reward	Average EH	Anomaly	EH Ratio	Path Ratio	Trade-off (EH / Path Length)
1 Robot	Q-learning	No EH	0	10	7.4	99.36	22.43	False	132.23 %	100.0 %	132.23 %
		EH	0	10	7.4	52.23	29.66				
	PPO	No EH	0	10	7.1	99.2	20.67	False	217.42 %	95.77 %	227.01 %
		EH	3	7	6.8	37.34	44.94				
2 Robots	Q-learning	No EH	0	10	6.6	998.62	57.1	False	199.18 %	100.0 %	199.18 %
		EH	0	10	6.6	508.09	113.73				
	PPO	No EH	0	10	17.2	970.7	618.28	True	34.76 %	87.21 %	39.86 %
		EH	0	10	15.0	504.85	214.92				

7 lightsources (clustered)

			Number of Failures	Number of Successes	Average Path Length	Average Reward	Average EH	Anomaly	EH Ratio	Path Ratio	Trade-off (EH / Path Length)
1 Robot	Q-learning	No EH	0	10	7.6	99.34	52.47	False	126.93 %	105.26 %	120.58 %
		EH	0	10	8.0	55.92	66.6				
	PPO	No EH	0	10	6.9	99.22	40.54	True	0.0 %	0.0 %	nan %
		EH	10	0	0.0	0.0	0.0				
2 Robots	Q-learning	No EH	0	10	6.6	998.62	58.24	False	359.25 %	100.0 %	359.25 %
		EH	0	10	6.6	513.24	209.23				
	PPO	No EH	0	10	17.0	985.62	361.51	False	100.07 %	104.12 %	96.11 %
		EH	0	10	17.7	500.71	361.77				

In the results you can see that in case we make energy harvesting slightly more important, the average amount increases by a lot. However, we don't really notice a lot of difference in the average path length. However, after training PPO for one robot, all navigation executions fail. This means that the impact of the weights is far greater for this RL technique. The failures indicate that we failed to reach our terminal state within 50 steps. Since we reduced the importance of movement and increased the significance of energy harvesting, the model is stuck to keep harvesting energy instead of trying to navigate to the goal.

When considering navigation to be more important, the results show the more importance is given, the further the amount of energy harvesting decreases. The average path length stays overall the same for this which is also expected. One exception to this observation in the average energy harvesting, using two robots with PPO when the weights are divided in a 80% movement and 20% energy harvest significance. There is also a small increase in average path length for that same PPO trained environment. This can only be declared due to the combination of predicting our actions deterministically and the vulnerability of our algorithm to fine-tuning the parameter values.

Table 19. *Overview of the Results of the Reward Weight Scenarios*

Reward Weight Tests

40% Move - 60% Energy Harvest

		Number of Failures	Number of Successes	Average Path Length	Average EH
1 Robot	Q-learning	0	10	9.0	12.3
	PPO	10	0	0.0	0.0
2 Robots	Q-learning	0	10	7.9	5.13
	PPO	0	10	12.2	5.51

60% Move - 40% Energy Harvest

		Number of Failures	Number of Successes	Average Path Length	Average EH
1 Robot	Q-learning	0	10	8.8	12.1
	PPO	0	10	8.4	10.9
2 Robots	Q-learning	0	10	7.9	4.4
	PPO	0	10	12.1	4.23

80% Move - 20% Energy Harvest

		Number of Failures	Number of Successes	Average Path Length	Average EH
1 Robot	Q-learning	0	10	7.4	8.7
	PPO	0	10	6.7	6.9
2 Robots	Q-learning	0	10	7.9	3.66
	PPO	0	10	13.4	10.28

The other type of test scenarios we consider in these sorts of changes, is the energy harvesting range scenarios. Here, we vary the range in which robots are able to detect and harvest energy. These test cases will have an effect on training our RL techniques with and without energy harvesting. We will thus again execute the tests on both kinds for each base case.

Our expectation for these test cases is when the range is smaller, the overall average amount of energy harvesting decreases. Although, another logical expectation would be that the average path length increases in this case as well, we believe the algorithm will not do this since up until now the average path length stayed rather consistent. The other scenario is that the range will increase. Here we expect to see an increase in energy harvesting and maybe slightly in average path length as well.

These expectations are only somewhat evident in our results in table 20. When our range to detect and harvest energy becomes smaller, it barely affects the average amount of energy harvesting. We had expected a larger difference, but this was

probably a bit unrealistic since we only change the range from three to two when comparing it to the base cases. It does however have no effect on the average path length as we did predict. When we increased the range to five, the difference in results is more visible. We notice a slight increasing effect on the average path length when taking energy harvesting in consideration, specifically in the PPO algorithm. And also, a much more present difference in the amount of energy harvested.

Table 20. *Overview of the Results of the Energy Harvesting Range Scenarios*

Energy Harvesting Range Tests

Range EH = 2

			Number of Failures	Number of Successes	Average Path Length	Average EH	Anomaly	EH Ratio	Path Ratio	Trade-off (EH / Path Length)
1 Robot	Q-learning	No EH	0	10	7.2	1.32	False	721.97 %	119.44 %	604.44 %
		EH	0	10	8.6	9.53				
	PPO	No EH	0	10	6.6	3.12	False	318.27 %	139.39 %	228.32 %
		EH	0	10	9.2	9.93				
2 Robots	Q-learning	No EH	0	10	7.9	3.55	False	149.58 %	100.0 %	149.58 %
		EH	0	10	7.9	5.31				
	PPO	No EH	0	10	13.7	3.05	False	192.13 %	85.4 %	224.97 %
		EH	0	10	11.7	5.86				

Range EH = 5

			Number of Failures	Number of Successes	Average Path Length	Average EH	Anomaly	EH Ratio	Path Ratio	Trade-off (EH / Path Length)
1 Robot	Q-learning	No EH	0	10	7.6	3.4	False	254.41 %	102.63 %	247.89 %
		EH	0	10	7.8	8.65				
	PPO	No EH	0	10	7.6	4.36	False	406.42 %	197.37 %	205.92 %
		EH	0	10	15.0	17.72				
2 Robots	Q-learning	No EH	0	10	7.9	3.79	False	152.77 %	100.0 %	152.77 %
		EH	0	10	7.9	5.79				
	PPO	No EH	0	10	11.65	6.95	True	187.77 %	148.5 %	126.44 %
		EH	0	10	17.3	13.05				

In conclusion, the evaluation of the algorithm under varying environmental and systematic factors revealed several key insights into its performance and adaptability. Environmental tests showed that the algorithm adapts to different grid sizes and number of light sources. The results frequently matched our expectations. Like how in smaller grids, this leads to shorter paths and less energy harvesting. However, in larger grids, the results were rather substandard, probably because we wanted to try and keep the same overall structure without changing too much in one scenario. The light source scenarios confirmed that more sources, especially when clustered, significantly boost energy harvesting, highlighting the algorithm's ability to capitalize on favorable conditions.

Systematic changes, like adjusting reward weights and energy harvesting range, demonstrated that while the algorithm can balance navigation and energy collection, it is sensitive to parameter tuning—unexpected failures in PPO underscore this point.

Overall, the algorithm shows promising adaptability to a range of environmental and systematic variations, though its performance is not without limitations. Future work could focus on further optimizing parameter selection and enhancing the algorithm's robustness to cope with even more diverse and unpredictable scenarios.

8. Conclusion

In conclusion, our novel energy-aware scheduling algorithm, implemented and evaluated in the ROS/Gazebo environment, has demonstrated superior performance compared to traditional energy-unaware approach. The algorithm effectively predicts the energy data of robots and optimizes intelligent swarm task scheduling in a heterogeneous pick, drop, and delivery scenario using real data. The proposed ROS-based autonomous architecture further validates the practical applicability of our approach.

The comparative analysis of the APSO, GA, and PSO algorithms provides valuable insights. While the GA algorithm yields more accurate and optimal solutions, the APSO algorithm excels in faster simulation times. These findings contribute to the growing body of knowledge on energy-aware robotic swarm scheduling, offering researchers practical insights into how different AI-based algorithms can be utilized depending on the specific optimization goals.

Furthermore, in our exploration of reinforcement learning techniques (RQ4), we found notable performance differences. While PPO required significantly longer execution times compared to Q-learning, this was only critical during the training phase. PPO's training time ranged from 200 seconds for a single robot to 500 seconds for two robots, whereas Q-learning trained in just 2 seconds for a single robot and 20 seconds for two robots. This difference becomes important when dealing with larger state spaces or real-time applications.

In terms of overall effectiveness across different test scenarios, Q-learning often outperformed PPO in terms of execution speed and efficiency. This can be attributed to Q-learning's simpler approach, which was well-suited to the relatively straightforward state spaces and reward structures used in our scenarios. While PPO holds potential for handling larger action spaces and more complex environments, Q-learning proved to be the more efficient solution for the contexts examined here. These comparative findings provide valuable insights for future work, particularly in selecting appropriate RL techniques based on energy constraints and real-time application needs.

Looking ahead, there are several avenues for future research. One promising direction involves exploring the integration of human-robot collaboration (HRC) frameworks with AI-based algorithms to further enhance the capabilities of robotic swarms in dynamic and uncertain environments. Another area of interest could involve examining other AI-based algorithms, such as those incorporating deep learning or hybrid approaches, to improve decision-making in more complex and large-scale scenarios.

9. References

[1] Mahmood Akbari and Mojtaba Henteh. Comparison of genetic algorithm (ga) and particle swarm optimization algorithm (pso) for discrete and continuous size optimization of 2d truss structures. Journal of Soft Computing in Civil Engineering, 3(2):76–97, 2019.

[2] Robin Amsters and Peter Slaets. Turtlebot 3 as a robotics education platform. In Robotics in Education: Current Research and Innovations 10, pages 170–181. Springer, 2020.

[3] Nicol`o Ciuccoli, Laura Screpanti, and David Scaradozzi. Underwater simulators analysis for digital twinning. IEEE Access, 12:34306–34324, 2024.

[4] M. Egerstedt, J. N. Pauli, G. Notomista, and S. Hutchinson. Robot ecology: Constraint-based control design for long duration autonomy. Annual Review of Control, 46:1–7, 2018.

[5] Stanisław Gawiejnowicz. Time-Dependent Scheduling. Springer, 2008.

[6] S. Giordani, M. Lujak, and F. Martinelli. A distributed algorithm for the multi-robot task allocation problem. In International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, pages 721–730, 2010.

[7] O.M. Gul. Energy harvesting and task-aware multi-robot task allocation in robotic wireless sensor networks. Sensors, 23:3284, 2023.

[8] J. Kennedy and R. Eberhart. Particle swarm optimization. 4:1942–1948 vol.4, 1995.

[9] S. Kernbach. Collective foraging: Cleaning, energy harvesting, and trophallaxis. In Handbook of Collective Robotics, pages 377–436. Jenny Stanford Publishing, 2013.

[10] H. W. Kuhn. The hungarian method for the assignment problem. Naval Research Logistics Quarterly, 2:83–97, 1955.

[11] E. Latif, Y. Gui, A. Munir, and R. Parasuraman. Energy-aware multi-robot task allocation in persistent tasks. 2021.

[12] J. Leu, Y. Cheng, C. Liu, and M. Tomizuka. Robust task planning for assembly lines with human-robot collaboration. 2022.

[13] Z. Li, A.V. Barenji, J. Jiang, R.Y. Zhong, and G. Xu. A mechanism for scheduling multi-robot intelligent warehouse system face with dynamic demand. J Intell Manuf, 31:469–480, 2020.

[14] Zexi Liang, Jiarui He, Chuangang Hu, Xiong Pu, Hadi Khani, Liming Dai, Donglei (Emma) Fan, Arumugam Manthiram, and Zhong-Lin Wang. Next-generation energy harvesting and storage technologies for robots across all scales. Advanced Intelligent Systems, 5, 2022.

[15] Liangkai Liu, Ren Zhong, Aaron Willcock, Nathan Fisher, and Weisong Shi. An open approach to energy-efficient autonomous mobile robots. pages 11569–11575, 2023.

[16] M. Lukic and I. Stojmenovic. Energy-balanced matching and sequence dispatch of robots to events: Pairwise exchanges and sensor assisted robot coordination. In Proceedings of the 10th IEEE 10th International Conference on Mobile Ad-Hoc and Sensor Systems, pages 249–253. IEEE, 2013.

[17] L. Luo, N. Chakraborty, and K. Sycara. Distributed algorithm design for multi-robot task assignment with deadlines for tasks. In Proceedings of the 2013 IEEE International Conference on Robotics and Automation, pages 3007–3013. IEEE, 2013.

[18] J. G. Martin, J. R. D. Frejo, R. A. Garc´ıa, and E. F. Camacho. Multi-robot task allocation problem with multiple nonlinear criteria using branch and bound and genetic algorithms. Intelligent Service Robotics, 14:707–727, 2021.

[19] M. Nanjanath and M. Gini. Dynamic task allocation for robots via auctions. In Proceedings of the 2006 IEEE International Conference on Robotics and Automation, pages 2781–2788.

IEEE, 2006.

[20] N. Nedjah, R.M. de Mendonc¸a, and L. de M. Mourelle. Pso based distributed algorithm for dynamic task allocation in a robotic swarm. Procedia Computer Science, International Conference On Computational Science, ICCS 2015, 51:326–335, 2015.

[21] G. Notomista and et al. A resilient and energy-aware task allocation framework for heterogeneous multirobot systems. IEEE Transactions on Robotics, 38(1):159–179, Feb. 2022.

[22] Ramviyas Parasuraman, Keith Kershaw, Prithvi Pagala, and Manuel Ferre. Model based on-line energy prediction system for semi-autonomous mobile robots. pages 411–416, 2014.

[23] Bastien Politi, St´ephanie Parola, Antoine Gademer, Diane Pegart, Marie Piquemil, Alain Foucaran, and Nicolas Camara. Practical pv energy harvesting under real indoor lighting conditions. Solar Energy, 224:3–9, 2021.

[24] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. 3(3.2):5, 2009.

[25] Yu.N. Sotskov and N.V. Shakhlevich. Np-hardness of shop-scheduling problems with three jobs. Discrete Applied Mathematics, 59(3):237–266, 1995.

[26] Milan Tomy, Bruno Lacerda, Nick Hawes, and Jeremy L. Wyatt. Battery charge scheduling in long-life autonomous mobilerobots via multi-objective decision making under uncertainty. Robotics and Autonomous Systems, 133:103629, 2020.

[27] S. Trigui, A. Koubaa, O. Cheikhrouhou, H. Yousseff, G. Bennaceur, M. F. Sriti, and Y. Javed. A distributed market-based algorithm for the multi-robot assignment problem. Procedia Computer Science, 32:1108–1114, 2014.

[28] T Verstraten, MS Hosen, M Berecibar, and B Vanderborght. Selecting suitable battery technologies for untethered robots. Energies, 16(13):4904, 2023.

[29] G. V´azquez, R. Calinescu, and J. C´amara. Scheduling of missions with constrained tasks for heterogeneous robot systems. Electron. Proc. Theor. Comput. Sci., 371:156–174, 2022.

[30] Nathan D Wallace, He Kong, Andrew J Hill, and Salah Sukkarieh. Motion cost characterisation of an omnidirectional wmr on uneven terrains. IFAC-PapersOnLine, 52(22):31–36, 2019.

[31] Hanfu Wang and Weidong Chen. Task scheduling for heterogeneous agents pickup and delivery using recurrent open shop scheduling models. Robotics and Autonomous Systems, 172:104604, 2024.

[32] M. Xiong and G. Xie. Swarm game and task allocation for autonomous underwater robots. Journal of Marine Science and Engineering, 11:148, 2023.

[33] Zhi-Hui Zhan, Jun Zhang, Yun Li, and Henry Shu Hung Chung. Adaptive particle swarm optimization. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 39(6):1362–1381, January 2010.

[34] Seongin Na et al. “Federated Reinforcement learning for collective navigation of robotic swarms”. In: IEEE Transactions on Cognitive and Developmental Systems 15.4 (Dec. 2023), pp. 2122–2131. doi: 10.1109/tcds.2023.3239815.

[35] In˜aki Navarro and Fernando Mat´ia. “An introduction to swarm robotics”. In: ISRN Robotics 2013 (Sept. 2013), pp. 1–10. doi: 10.5402/2013/608164.

[36] Amir Niaraki, Jeremy Roghair, and Ali Jannesari. Visual Exploration and Energyaware Path Planning via Reinforcement Learning. 2021. arXiv: 1909.12217[eess.SP].

[37] H B Radousky and H Liang. “Energy harvesting: an integrated view of materials, devices and applications”. In: Nanotechnology 23.50 (Nov. 2012), p. 502001. doi:10.1088/0957-4484/23/50/502001.

[38] Melrose Roderick, James MacGlashan, and Stefanie Tellex. “Implementing the deep Q-Network”. In: arXiv (Cornell University) (Jan. 2017). doi: 10.48550/arxiv.1711.07478.

[39] Melanie Marianne Schranz et al. “Swarm robotic behaviors and current applications”. In: Frontiers in robotics and AI 7 (Apr. 2020). doi: 10.3389/frobt.2020.00036.

[40] John Schulman et al. “Proximal Policy optimization Algorithms”. In: arXiv (Cornell University) (Jan. 2017). doi: 10.48550/arxiv.1707.06347.

[41] W.A. Serdijn, A.L.R. Mansano, and M. Stoopman. Introduction to RF energy harvesting. Jan. 2014, pp. 299–322. doi: 10.1016/b978-0-12-418662-0.000192.

[42] Mohit Sewak. Deep reinforcement learning. Jan. 2019. doi: 10.1007/978-981-138285-7.

[43] Siddharth Sharma. “The Ultimate Beginner’s Guide to Reinforcement Learning”. In: (Dec. 2021).

[44] Wei Shi et al. “Efficient hierarchical policy network with fuzzy rules”. In: International Journal of Machine Learning and Cybernetics 13 (Feb. 2022). doi: 10.1007/ s13042-021-01417-2 (cit. on p. 34).

[45] Chathurangi Shyalika. “A Beginners Guide to Q-Learning - towards Data Science”.

[46] Ankita Sinha. “Hierarchical Reinforcement Learning - towards data science”. In: (Jan. 2024).

[47] Zhaochen Sun. “Robot Swarm Navigation: Methods, Analysis, and Applications”. In: International Conference on Big Data & Artificial Intelligence & Software Engineering (Sept. 2021). doi: 10.1109/icbase53849.2021.00131.

[50] Wouter Van Heeswijk PhD. “Proximal Policy Optimization (PPO) explained - towards data science”. In: (Aug. 2023).

[51] Marco Visca et al. “Deep Meta-Learning Energy-Aware Path Planner for unmanned ground vehicles in unknown terrains”. In: IEEE Access 10 (Jan. 2022), pp. 30055–30068. doi: 10.1109/access.2022.3155161.

[52] Christopher J. C. H. Watkins and Peter Dayan. “Q-learning”. In: Machine Learning 8.3-4 (May 1992), pp. 279–292. doi: 10.1007/bf00992698.

[53] Xingyu Wei et al. “Energy harvesting fueling the revival of self-powered unmanned aerial vehicles”. In: Energy Conversion and Management 283 (May 2023), p. 116863. doi: 10.1016/j.enconman.2023.116863.

[54] Gerhard Weiß. “Distributed reinforcement learning”. In: Robotics and Autonomous Systems 15.1-2 (July 1995), pp. 135–142. doi: 10.1016/0921-8890(95)00018.

[57] Dibin Zhu and Steve Beeby. Kinetic energy harvesting. Oct. 2010, pp. 1–77. doi:10.1007/978-1-4419-7566-9\{_}1.

[58] Yuanyang Zhu et al. “Rule-Based reinforcement learning for efficient robot navigation with space reduction”. In: IEEE/ASME Transactions on Mechatronics 27.2 (Apr. 2022), pp. 846–857. doi: 10.1109/tmech.2021.3072675.

[59] Amrani Amine. “Deep Q-Networks: from theory to implementation - Towards Data Science”. In: (Dec. 2021).

[60] Muhammad Anwar et al. Proximal Policy Optimization Based Reinforcement Learning for Joint Bidding in Energy and Frequency Regulation Markets. Dec. 2022. doi: 10.48550/arXiv.2212.06551 (cit. on p. 11).

[61] Andrew G. Barto. Reinforcement learning. Jan. 1997, pp. 7–30. doi: 10.1016/b978-012526430-3/50003-9.

[62] Shweta Bhatt. “Reinforcement Learning 101 - towards Data Science”. In: (Apr. 2019). url: https://towardsdatascience.com/reinforcement-learning-101e24b50e1d292 (cit. on p. 5).

[63] Omar Bouhamed et al. “A DDPG-based Approach for Energy-aware UAV Navigation in Obstacle-constrained Environment”. In: 2020 IEEE 6th World Forum on Internet of Things (WF-IoT). 2020, pp. 1–6. DOI: 10.1109/WF-IoT48130.2020.9221115 (cit. on p. 19).

[64] Manuele Brambilla et al. “Swarm robotics: a review from the swarm engineering perspective”. In: Swarm intelligence 7 (Jan. 2013), pp. 1–41. DOI: 10.1007/s11721-012-0075-2. URL: https://doi.org/10.1007/s11721-012-0075-2 (cit. on pp. 1,2).

[65] Loy Chuan Chia and Bo Feng. “The development of a micropower (micro-thermophotovoltaic) device”. In: Journal of Power Sources 165.1 (Feb. 2007), pp. 455–480. DOI: 10.1016/j.jpowsour.2006.12.006.

[[66] Jesse Clifton and Eric Laber. “Q-Learning: Theory and applications”. In: Annual Review of Statistics and Its Application 7.1 (Mar. 2020), pp. 279–301. DOI: 10.

1146/annurev-statistics-031219-041220. URL: https://doi.org/10.1146/ annurev-statistics-031219-041220 (cit. on p. 7).

[67] Maziar Dehghan et al. Introduction to solar energy harvesting and storage. Jan. 2023, pp. 1–23. DOI: 10.1016/b978-0-323-90601-2.00012-x.

[71] Shlomi Dolev et al. “In-vivo energy harvesting nano robots”. In: Nov. 2016. DOI:10.1109/icsee.2016.7806107.

[72] Marco Dorigo, Guy Theraulaz, and Vito Trianni. “Swarm Robotics: Past, Present, and Future [Point of View]”. In: Proceedings of the IEEE 109.7 (2021), pp. 1152– 1165. DOI: 10.1109/JPROC.2021.3072740 (cit. on p. 1).

[73] Adrien Lucas Ecoffet. “An intuitive explanation of policy gradient - towards data science”. In: (Jan. 2022)

[74] Farama Foundation. Gymnasium documentation. 2023. URL: https://gymnasium.farama.org/index.html (cit. on p. 15).

[75] Vincent Francois-Lavet et al. “An introduction to deep reinforcement learning”. In: Foundations and Trends® in Machine Learning 11.3-4 (Jan. 2018), pp. 219–354. DOI: 10.1561/2200000071.

[76] Tingting Fu et al. “Distributed reinforcement learning-based memory allocation for edge-PLCs in industrial IoT”. In: Journal of Cloud Computing 11 (Oct. 2022). DOI: 10.1186/s13677-022-00348-9 (cit. on p. 32).

[77] Adnan Harb. “Energy harvesting: State-of-the-art”. In: Renewable Energy 36.10 (Oct. 2011), pp. 2641–2654. DOI: 10.1016/j.renene.2010.06.014.

[78] Yash Kavaiya. “Types of machine learning: supervised, unsupervised, and reinforcement learning”. In: (Nov. 2023).

[79] Juha Kiili. Reinforcement Learning Tutorial Part 3: Basic Deep Q-Learning.

[80] Jens Kober, J. Andrew Bagnell, and Jan Peters. “Reinforcement learning in robotics: A survey”. In: The International Journal of Robotics Research 32.11 (Aug. 2013), pp. 1238–1274. DOI: 10.1177/0278364913495721.

[81] Zexi Liang et al. “Next-Generation energy harvesting and storage technologies for robots across all scales”. In: Advanced intelligent systems 5.4 (June 2022). doi:10.1002/aisy.202200045.

[82] Aleksis Liekna, Janis Grundspenkis, and Riga Technical University. “TOWARDS PRACTICAL APPLICATION OF SWARM ROBOTICS: OVERVIEW OF SWARM TASKS”. In: Engineering For Rural Development (2014), pp. 271–272.

[83] Qingyang Ling. “Machine learning algorithms review”. In: Applied and Computational Engineering 4.1 (June 2023), pp. 91–98. doi: 10.54254/2755-2721/4/20230355.

[84] Srikanth Machiraju. “Accelerate training in RL using distributed reinforcement learning architectures”. In: (Mar. 2022).