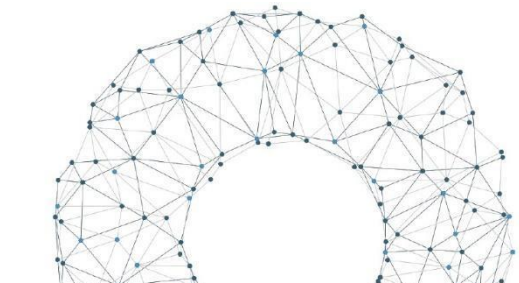


OpenSwarm

www.openswarm.eu

Call: HORIZON-CL4-2022-DATA-01

Type of action: RIA

Grant agreement: 101093046

Deliverable 4.4: Virtual Machines for Swarm Control and Learning

Work Package 4: Swarm Compiler

Task Lead: UOS

WP Lead: UOS

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)	Mohamed S. Talamali, Genki Miyauchi, Micael Couceiro, Roderich Groß
Reviewers	Micael Couceiro
Submission date	31-Oct-2024
Due date	31-Oct-2024
Type	R
Dissemination level	PU

Document history			
Date	Version	Author(s)	Comments
10-Jun-2024	01	Mohamed S. Talamali	Preview
31-Oct-2024	02	Roderich Groß	Deliverable

CONTENTS

Executive Summary

This document presents the work conducted under Task T4.4 which focused on creating virtual machines that enable compiled swarm controllers to operate seamlessly across multiple platforms, while supporting their run-time adaptation through the integration of learning frameworks. We developed virtual machine implementations for swarms of physical robots, including the Pi-puck, Kilobot and CapBot platforms. We also developed virtual machine implementations for swarms of simulated robots, interfacing with both a custom-built ultra-fast multi-agent drone simulator and ARGoS, a state-of-the-art physics-based robot swarm simulator. We showed that once a swarm controller has been compiled, the virtual machines make it possible to execute it on various swarm robotic platforms, demonstrating cross-platform compatibility. Moreover, we conducted two case studies focused on energy-efficient long-term operation of robot swarms. In the first study, a swarm of ground-based robots executed compiled controllers that ensured that either (i) each robot regularly returns to a base for recharging, or (ii) some robots distribute the energy to others. The swarm was shown to operate close to theoretically derived bounds for the work performed and its energy efficiency. In the second study, a fleet of delivery drones executed compiled swarm controllers enabling them to bid for, and fulfil, incoming delivery orders at a fulfilment centre. The fleet was heterogeneous, as the drones differed in battery health, the level of which was not known to them. We integrated a decentralised learning framework into the virtual machine, enabling the drones to learn during run-time their individual bidding policies, which were shown to correlate strongly with both battery health ground truth and task complexity. The findings demonstrate the advantages of combining correct-by-construction formal methods with decentralised online learning, and advance the state of the art in adaptive, energy-aware swarms.

Introduction

The objectives of WP4 are to design and implement tools that enable operators to intuitively program energy-aware, swarm-level behaviours that are fully adaptive, while adhering to formal specifications. WP4 focuses on developing energy management solutions for devices, allowing them to harvest and allocate energy adaptively, while monitoring and benchmarking consumption. Additionally, an intuitive swarm programming user interface enables operators to define device capabilities and specify learning objectives at both subsystem and swarm levels. The defined capabilities are processed by a swarm compiler, which automatically generates swarm controllers. Virtual machines will ensure that the resulting programs can run on multi-platform swarms.

The specific objectives for deliverable D4.4 are:

- To develop virtual machines capable of executing compiled swarm controllers on individual devices.
- To enable virtual machines to run the same compiled swarm controllers across different robotics platforms.
- To test virtual machines in simulation environments such ARGoS simulator.
- To ensure identical swarm controllers can be executed on both physical and simulated platforms.
- To employ learning frameworks, such as reinforcement or supervised learning, to optimise the stochastic selection of permissible controllable events based on operator-defined objectives (e.g., reducing energy consumption).

This document outlines the work completed in Task T4.4 and is structured as follows. Initially, a review of relevant research is presented. This is followed by an introduction of the virtual machines that have been developed, which facilitate the execution of the compiled swarm controllers across multiple platforms. The document then introduces two research studies conducted within the scope of this task. Both studies focus on energy-efficient long-term operation of robot swarms. In the first study, a swarm of

ground-based robots is tasked to perform work at a remote location. Two scenarios are considered: In one, each robot regularly returns to a base for recharging; in the other, some robots distribute the energy to others. The underlying virtual machines make it possible to execute the compiled controller on various platforms, including the Kilobot, CapBot, and simulated e-puck. In the second study, a smart cities scenario is considered where goods are delivered from a fulfilment centre using a fleet of UAVs under energy storage constraints. The fleet is heterogeneous, as the UAVs differ in battery health, and the level of which is not known to them. The UAVs execute a compiled swarm controller enabling them to bid for, and fulfil, incoming delivery orders. The UAVs learn to place bids only for tasks that align with their energy capabilities. In this context, a virtual machine, combined with online learning, is developed to allow individual robots to optimise their behaviour, thereby improving the overall performance of the swarm.

Relationship to Other Tasks and Deliverables

The work conducted in this task builds significantly upon the progress and developments achieved in other tasks and deliverables.

- The first study, involving a swarm of ground-based robots tasked with work in a remote region, relates to the power module capabilities of the CapBot robot platform which was developed as part of Task T4.1, specifically the capabilities of fast charging and energy sharing. A virtual machine was developed to execute the swarm controller on the physical CapBot robotics platform.For testing the virtual machine on the Kilobot platform, the new user interface from Task T4.2, designed for the Augmented Reality for Kilobots (ARK) system, was utilised to design, compile, and upload controllers onto swarms of robots.
- All swarm controllers used in this task were designed and compiled using the compiler developed in Task T4.3.

Technological Readiness Level (TRL)

Virtual Machines have been implemented and tested to execute controllers on different simulated and real robot platforms across a range of use cases. As a result, the technology has reached a Technology Readiness Level (TRL) of 4.

Key Performance Indicators (KPIs)

Table 3.2 in the DoA contains the following KPI for Task T4.4: “Memory footprint of the virtual machine” with a target below 100kB. We meet this KPI, as detailed below.

We have developed virtual machines to run compiled swarm controllers on different platforms, including both simulated and real robot swarms. Here, we report the size of the generated supervisory control theory (SCT) models and the memory size corresponding to each virtual machine and controller.

Use case	Role	Events	States	Transitions	YAML (bytes)	C++-based virtual machine size (bytes)	Python-based virtual machine size (bytes)
Swarms with global performance requirements (see also D4.3)	Leader	10	10	39	1143	15699	7343
	Worker	35	140	630	14335		
Swarms with human-in-the-loop (see also D4.3)	Leader	8	8	28	899		
	Worker	34	135	635	12038		
Swarms and energy management	Worker (without mobile charger)	10	32	124	2590		
	Worker (with mobile charger)	10	30	150	3029		
	Mobile charger	10	32	124	2646		

Optimising Swarm Behaviours Through Online Learning	UAV	9	6	39	1022		

List of Publications

- G. Miyauchi, M. S. Talamali and R. Groß, 'A Comparative Study of Energy Replenishment Strategies for Robot Swarms', in ANTS 2024, 14th International Conference on Swarm Intelligence, October 9-11, 2024. Konstanz, Germany, p 3-15, Lecture Notes in Computer Science, vol 14987, Springer
- M. S. Talamali, G. Miyauchi, Thomas Watteyne, Micael Couceiro and R. Groß, 'Ready, Bid, Go! On-Demand Delivery Using Fleets of Drones with Unknown, Heterogeneous Energy Storage Constraints', in the 24th International Conference on Autonomous Agents and Multiagent Systems, 2025, Detroit, US. (under review)

Related Work

In this section, we review the swarm robotics platforms used for testing virtual machines and the learning frameworks used in swarm robotics.

Execution and Evaluation of Swarm Controllers

Many robot platforms have been developed and used to study the behaviour of robot swarms. Some of these platforms are commercially available and have become popular choices for conducting swarm robotics experiments. A major shortcoming, however, is that each robot platform typically requires its own specific programming framework and unique API for accessing the sensors and actuators. This makes it difficult to transfer code written for one robot onto a different robot platform, even when the same programming language is used. This section highlights the challenges in transferring code between different robots by giving examples of platforms, simulators, and real-robot tracking systems used in the OpenSwarm project.

The work conducted in task T4.4 involves using the following swarm robotics platforms:

- **Kilobot:** The Kilobot [1] is a low-cost, small-scale robot platform designed specifically for large-scale swarm robotics research. Measuring just 33 mm in diameter and standing 34 mm tall, each Kilobot is powered by an ATmega328 microcontroller with 2 kB of RAM and 32 kB of flash memory. The platform is programmed using the C language, making it flexible for developing custom swarm algorithms. Kilobots communicate with each other using infrared signals, enabling coordination and proximity sensing within the swarm. Each robot moves via vibration motors, which allow for differential drive locomotion, allowing forward movement at a nominal forward speed of 1 cm/s and on-the-spot rotation at 45°/s. The Kilobot is powered by a coin-cell battery, providing 3-24 hours of operation depending on the robot's activity level. Its simplicity and scalability make it ideal for experiments with hundreds or thousands of robots,

facilitating research into collective behaviour, distributed algorithms, and emergent phenomena in swarm systems.

- **E-puck:** The e-puck [2] is a compact, autonomous mobile robot designed for educational and research purposes, particularly in swarm robotics. It measures 70 mm in diameter and weighs around 150 g, making it ideal for small-scale experiments. The robot is powered by a dsPIC microcontroller with 8 kB of RAM and 144 kB of flash memory, offering sufficient capacity for onboard programming. The e-puck supports programming primarily in C, allowing researchers and students to write and upload custom algorithms for various tasks. It is equipped with several sensors, including 8 infrared proximity sensors for obstacle detection, a 3D accelerometer, a gyroscope, a microphone array, and a VGA camera for vision-based tasks. Two stepper motors provide precise movement control, with a top speed of approximately 13 cm/s. Wireless communication is enabled via Bluetooth, facilitating coordination in multi-robot systems. Additionally, it is powered by 5Wh Li-ion rechargeable and removable battery that provides about 3 hours of autonomy.
- **E-puck2:** The e-puck2 [3] is an enhanced version of the original e-puck robot, designed for more advanced research and educational applications in swarm robotics. It features upgraded hardware, including a more powerful STM32F4 microcontroller with a Cortex-M4 core, providing 192 kB of RAM and 1 MB of flash memory, enabling the execution of more complex algorithms and data processing tasks. While the e-puck2 is primarily programmed in C, it also supports C++, offering the flexibility of object-oriented programming for more sophisticated and modular projects. The robot retains the versatile sensor suite of the original e-puck, including 8 infrared proximity sensors, a 3D accelerometer, gyroscope, magnetometer, microphone array, and a higher-resolution camera for vision-based tasks. Additionally, it introduces new sensors, such as a time-of-flight distance sensor for more precise ranging. The e-puck2 supports wireless communication via Bluetooth and now features Wi-Fi for enhanced networking capabilities. It uses the same stepper motors as the first version, maintaining a top speed of 13 cm/s, and shares the same Li-ion battery.

- **Pi-puck:** The Pi-puck [4] is an advanced extension of the e-puck and e-puck2 platforms, designed to provide increased computational power and flexibility for swarm robotics research. It integrates a Raspberry Pi single-board computer with the original e-puck base, significantly enhancing its processing capabilities with up to 1 GB of RAM and a quad-core ARM Cortex-A53 CPU. This upgrade allows for more complex algorithms and machine learning tasks to be run directly on the robot. The Pi-puck supports programming in a variety of languages, including Python, giving researchers more flexibility in developing swarm behaviours and real-time applications.

CapBot: The CapBot, developed as part of D4.1 [5], is a cutting-edge, battery-free swarm robotics platform designed to address power challenges in large-scale robotic deployments. Instead of traditional batteries, it utilises a supercapacitor array, allowing it to fully recharge in just 16 seconds and operate autonomously for up to 51 minutes at top speed. Equipped with a Cortex M4F processor, 256 kB of RAM, and 1 MB of flash memory, CapBot supports advanced computational tasks while enabling Bluetooth Low Energy (BLE) networking for communication. Its unique feature, trophallaxis, allows CapBots to transfer up to 50% of their charge to peers within 20 seconds, offering a dynamic energy-sharing solution for extended swarm operation. The robot features Mecanum wheels for omnidirectional movement and a lightweight design (285 g), making it versatile for various tasks. To expand the experimental capabilities of the above platform, we employ the following systems:

- **Augmented Reality for Kilobots (ARK) System:** The ARK system [6] enhances Kilobot robots by allowing them to interact with a virtual environment, overcoming the limitations of their minimal sensors. ARK uses an overhead camera tracking system, infrared emitters, and a base control station to provide real-time tracking and location-based information to each Kilobot. This enables dynamic interactions with virtual environments as if the robots had richer sensors. ARK also automates tasks like assigning IDs and positioning robots, supporting large swarms and making it a valuable open-source tool for scalable swarm robotics research.

- **SwarmHack:** SwarmHack [7] is an overhead tracking system that detects ArUco markers within its range. The system communicates with robots that carry ArUco markers via a local Wi-Fi network. The server-side code for SwarmHack is written in Python using WebSockets, and the robot-side code must also use Websockets for communication. In this task, we use the SwarmHack system to provide the CapBot robots with positioning and virtual sensing capabilities.

The aforementioned platforms require a mixture of C, C++ and Python, which can hinder code transferability between swarm robotic platforms. This incompatibility in programming languages and APIs means that re-programming and re-adjusting codes are often necessary when shifting from one platform to another. Addressing this issue is key to enabling more efficient and flexible development in swarm robotics.

In addition to experimenting with real robots, we extensively evaluate compiled swarm controllers and virtual machines in simulation. We chose to use the widely used swarm robotics simulator ARGOS. It is a physics-based multi-robot simulator designed for large-scale swarm robotics. ARGOS uses C++ for programming components such as controller and environment. While ARGOS supports native APIs (e.g., Kilobot), it does not yet provide smooth transferability to the e-puck and CapBot platforms used in this task.

Learning in Swarm Robotics

The integration of learning mechanisms in swarm robotics significantly enhances the adaptability and efficiency of these systems, enabling them to function effectively in dynamic and unpredictable environments. Various frameworks have been employed to implement learning capabilities in robot swarms, including social learning [8], evolutionary algorithms [9], supervised learning [10], reinforcement learning (RL) [11], multi-agent reinforcement learning (MARL) [12], and Turing learning [13].

In supervised learning, the system has access to a reference behaviour, which it uses to compute an error term that quantifies how much its actions deviate from the desired behaviour. This error term guides the system towards the correct behaviour. In reinforcement learning, the system lacks access to a reference behaviour and instead receives rewards based on the actions it takes, with the objective of maximising

cumulative rewards. In unsupervised learning, there is no supervision in the form of errors or rewards; instead, the system might be tasked with clustering data into groups with high similarity. In the following, we review selected works on adaptation and learning in multi-robot systems.

Social learning allows robots to acquire knowledge from their peers rather than relying solely on pre-programmed instructions. This approach is particularly useful in environments with unpredictable conditions, as it enables robots to adapt their behaviours based on real-time observations of other robots' actions [8]. For example, in agricultural applications, swarm robots can learn from one another to optimise tasks such as crop monitoring or pollination, thereby improving overall effectiveness. Furthermore, the concept of morphological computation, where robots use their physical interactions to learn and adapt, has been proposed to enhance swarm learning, particularly in sparse configurations where robots may experience temporary disconnection [14]. Tarapore *et al.* [15] present a decentralised algorithm for detecting faulty robots in a homogeneous swarm by learning the most common behaviour. Robots observe their neighbours' behaviours and exchange messages containing IDs and behavioural estimates. A voting process ensues, and each robot adjusts its behaviour to align with the majority. The least common behaviours, identified by multiple robots, are flagged as abnormal, and the robot responsible for such behaviour is considered faulty.

The use of evolutionary algorithms for designing controllers for a homogeneous swarm of robots is proposed in [9]. Two tasks are considered: one involves robots aggregating at a common location, and the other involves coordinating their movements when physically coupled. Each robot shares an identical controller. Populations of candidate controllers are optimised through task-specific fitness functions, and the controllers evolve over generations in computer simulations. This approach circumvents the need to manually break down the overall task into individual behaviours. However, identifying suitable fitness functions for complex tasks remains a significant challenge.

Reinforcement Learning (RL) is well-suited to swarm robotics, as it enables individual robots to learn optimal behaviours through trial and error, receiving feedback in the

form of rewards or penalties. This method has been used to develop end-to-end control policies for robotic swarms. For example, deep Q-learning, a popular RL algorithm, has enabled robots with limited sensory capabilities to collaborate on round-trip tasks, where they must travel repeatedly between two specific locations [16]. RL has outperformed many evolutionary algorithm-based approaches due to its ability to handle large parameter spaces and efficiently learn complex behaviours without requiring extensive prior knowledge of the environment [16] [17].

Multi-agent Reinforcement Learning (MARL) extends the principles of RL to environments involving multiple agents, where learning is influenced by the actions of other agents in the swarm. This is crucial for developing cooperative behaviours, as robots learn not only from their own experiences but also from interactions with their peers. Sadhu *et al.* [18] propose an extension to multi-agent Q-learning that accelerates the discovery of good solutions in cooperative task-planning scenarios. Two key enhancements are added: (i) an agent that reaches its goal waits for all other agents to reach their goals before starting a new learning cycle, and (ii) joint actions already taken at a given state are not repeated until all other actions are exhausted. These enhancements are theoretically proven to improve the algorithm's performance, and their effectiveness is demonstrated through experiments measuring speed, planning performance, run-time complexity, and real-time planning.

Supervised learning, though less common in swarm robotics than RL, still plays a key role in training robot controllers. In this paradigm, a supervisor demonstrates desired behaviours, which the robots learn to replicate. Thangavelautham *et al.* [19] propose a control strategy for lunar terrain excavation by a team of robots, termed Artificial Neural Tissue. This method combines a neural network with a coarse-coding mechanism. The neural network is trained using evolutionary algorithms, and the rules for crossover and mutation are outlined. The fitness function is discussed, along with the robot and task model. The method is validated through simulations and proof-of-concept physical experiments, with comparisons drawn against other neural network approaches.

Li *et al.* [13] introduce a system identification method known as *Turing Learning*, inspired by the Turing test. This method simultaneously constructs a population of behavioural

models, which imitate the behaviour of a real system, and classifiers, which distinguish between the behaviours of models and the real system. Both the models and classifiers are optimised using evolutionary algorithms. Classifiers are rewarded for correctly identifying whether behaviours are produced by models or the real system, while models are rewarded for deceiving the classifiers. Originally proposed by [20] and applied to swarms of simulated agents [21], the method was later rediscovered by [22] and is central to the state-of-the-art machine learning approach Generative Adversarial Networks. Li *et al.* [13] demonstrates that Turing Learning can infer the behavioural rules of a real swarm of robots from observation alone, outperforming metric-based approaches. Moreover, Turing Learning proves more efficient when interacting with the system rather than learning solely through observation [23].

Virtual Machines for Cross-Platform Swarm Controllers

In this section, we describe the concept of virtual machines, which enables the seamless implementation of swarm robot controllers across different platforms. We refer to the control logic designed and synthesised using the swarm compiler described in deliverable D4.3.

The Concept of Virtual Machines

In the context of the OpenSwarm project, a *virtual machine* refers to a piece of template code written for each language or robot platform that can evolve the *supervisor* (i.e. the control logic) obtained from synthesised models based on the supervisory control theory framework. The supervisor is evolved whenever an uncontrollable event (i.e. events that cannot be controlled by the robot) or a controllable event (i.e. events that can be triggered by the robot) occurs.

One of the reasons for using a virtual machine is overcoming the difficulty in guaranteeing that behaviours modelled using formal approaches (e.g. state machines) are correctly implemented by human programmers. There is a risk that the actual implementation may not reflect the behaviour intended during the design process.

Instead, the virtual machine presented in this deliverable receives the set of supervisors as input (see Figure 1). These supervisors are stored in a text format independent of the robot platform. Once the robot is running, the virtual machine evolves the state of each supervisor according to the events that trigger in real time. Since the swarm compiler generates partially implemented source code for specific robot platforms (given that template code is available), the programmer only needs to implement the callback functions. These functions determine (i) whether uncontrollable events have occurred – these typically relate to the robot perceiving something in its environment, receiving

a message from other robots, or meeting an internal condition such as a timeout; (ii) what action the robot should perform when a controllable event is triggered.

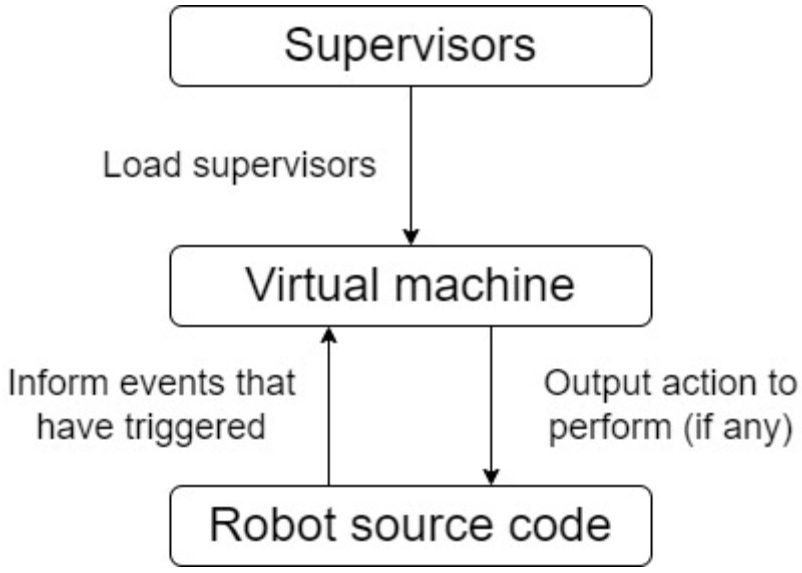


Figure 1. The virtual machine acts as a middleware to execute behaviour defined by the supervisors.

The virtual machine evolves the state of the supervisors according to Algorithm 1. It checks whether any uncontrollable events have occurred by executing the corresponding callback functions and if this is the case evolves the supervisors accordingly. Otherwise, the virtual machine selects one controllable event (if any) that can be triggered in the current states of the supervisors. It then also executes the corresponding callback function to trigger the corresponding action.

Algorithm 1: Virtual Machine

Let N be the number of supervisors
for all $j \in \{1,2,\dots,N\}$ **do**
    set current sate c_j to initial state q_{0_j}
end for
while *true* **do**
    if uncontrollable event $e_u \in \sum_u$ occurred **then**
        evolve the state with event e_u for all $j \in \{1,2,\dots,N\}$
    else
        find the set of enabled controllable events $\psi(c_1, c_2, \dots, c_n)$
        if $\psi(c_1, c_2, \dots, c_n) \neq \emptyset$ **then**
            select one controllable event $e_c \in \psi(c_1, c_2, \dots, c_n)$
            evolve the state with event e_c for all $j \in \{1,2,\dots,N\}$

13: execute the callback function of e_c
14: **end if**
15: **end if**
16: **end while**

The reduced need for ad-hoc implementation of control logic helps minimise the potential risk of introducing bugs during the implementation phase, especially in highly decentralised and heterogeneous systems like robot swarms. Moreover, it improves the transferability of the same controller when moving from simulation to real robot experiments, and across various robot platforms that often have different robot-specific codes. Overall, it helps swarm designers build confidence and trust in the designed swarm controller.

Implementation of Virtual Machines

The robot controllers reported in this document and Deliverable 4.3 all use supervisors to achieve their respective underlying robot behaviours. The behaviours are hence guaranteed to respect the formal specifications that the user (i.e. swarm designer) specified at design time. Since each robotic platform and simulation use different application programming interfaces (APIs) and programming languages, we implemented virtual machines for the corresponding languages used on the robots used in our works. Table 1 summarises these virtual machine implementations.

Table 1. Overview of the virtual machine implementations.

Platform	Real/Simulation	Programming language	Link
e-puck	Simulation (ARGoS)	C++	https://github.com/openswarm-eu/minimal-length-swarm-networks
Pi-puck	Real	Python	https://gitlab.com/genki_miyauchi/swarmhack/-/tree/iros-submission?ref_type=tags

CapBot	Real	C	https://github.com/jessicajayakumar/FreeBot/tree/work_and_charge
Kilobot	Real	C	https://github.com/mstalamali/Kilobots_SCT_Energy_Replenishment_Scenario
UAV	Simulation (ultra-fast multi-agent simulator)	Python	https://github.com/mstalamali/EA_Drone_Delivery_SCT_Implementation

To integrate the control logic into a new robot platform (either real or simulated), the developer follows the steps below:

1. Store the Virtual Machine and Compiled Supervisors.
2. Implement Callback Functions.
3. Register the Callback Functions.
4. Execute the Virtual Machine.

Store the Virtual Machine and Compiled Supervisors. After designing the robot's control logic, Nadzoru 2[1] generates two types of files: (i) the virtual machine source code in the chosen programming language, and (ii) the control logic stored in a text-based format. The developer must store these files under a suitable path so that they can be accessed either during compilation or at run-time. The virtual machine source code is a library that can be imported and used by the robot to parse the generated control logic represented in a text-based format and evolve the state of the control logic while the robot is operating. The virtual machine has been implemented in C/C++ and Python to work with most robot platforms. The control logic is stored in a YAML format. It is

[1] Nadzoru 2 is an openly available software for designing control logic based on SCT and has been substantially extended as part of the OpenSwarm project, as described in Deliverable 4.3.

intended to be provided to the virtual machine, which executes the behaviour according to it. Separating the virtual machine and the control logic allows the developer to update the control logic without having to re-generate the virtual machine source code.

Implement Callback Functions. The control logic consists of a set of events that have been defined during the design stage. To run the control logic on the robot, callback functions must be implemented for each event. Callback functions define what the robot should do when a certain event is triggered.

For example, assume an event called `turnLeft` is defined, which changes the movement of the robot. For the e-puck2, this may be written as a callback function written in C as shown below.

```
void callback_turnLeft(void* data){
     int left_speed = 0;
     int right_speed = 600;
     left_motor_set_speed(left_speed);
     right_motor_set_speed(right_speed);
}
```

For the Pi-puck, which is programmed using Python, the same callback function can be written as below.

```
def callback_turnLeft(data):
     left_speed = 0
     right_speed = 1
     pipuck.epuck.set_motor_speeds(left_speed, right_speed)
```

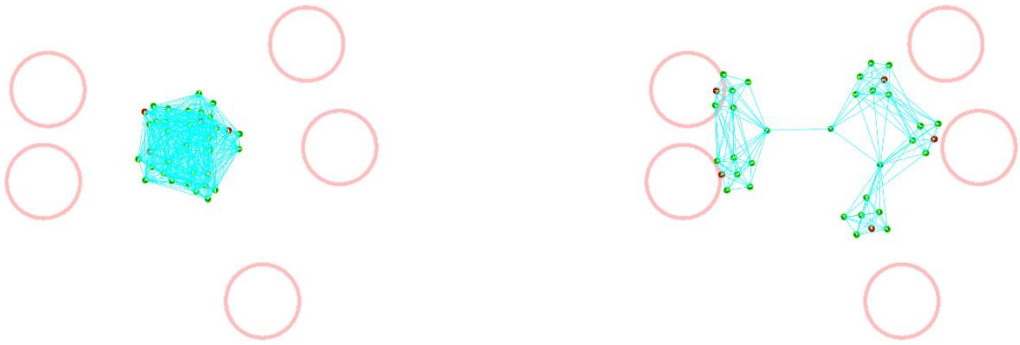
Register the Callback Functions. Once the callback functions have been implemented, they must be registered to the virtual machine. This can be done during the initialisation of the robot using the functions provided by the virtual machine source code. This

allows the virtual machine to execute the callback functions when corresponding events are triggered.

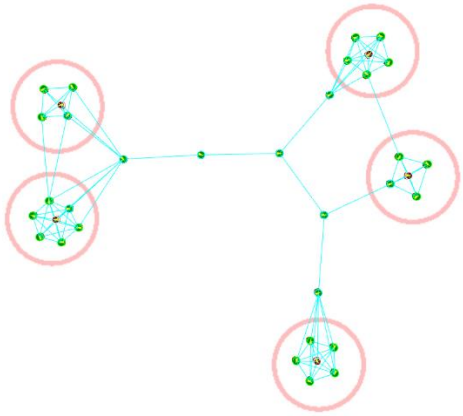
Execute the Virtual Machine. During the operation, the behaviour of the robot can be updated by calling the virtual machine from the robot's infinite loop. The robot updates its sensors and informs the virtual machine, which determines whether an event has been triggered. The virtual machine then chooses an action that can be triggered in the current state, if there are any.

Below, we illustrate the use cases of virtual machines in our works, demonstrating the transferability of compiled swarm controllers across different robot platforms and programming languages.

In deliverable D4.3's section "Swarms with global performance requirements", a subset of the robots in the swarm was required to dynamically maintain global connectivity among the lead agents moving towards their assigned target interests. The robot controller was written in C++, using a C++ implementation of the virtual machine (see Figure 2). Later, this multi-operator experiment was conducted using real Pi-puck robots, with the robot controller written in Python (see Figure 3). In both cases, the same supervisors were used, ensuring identical behaviour.

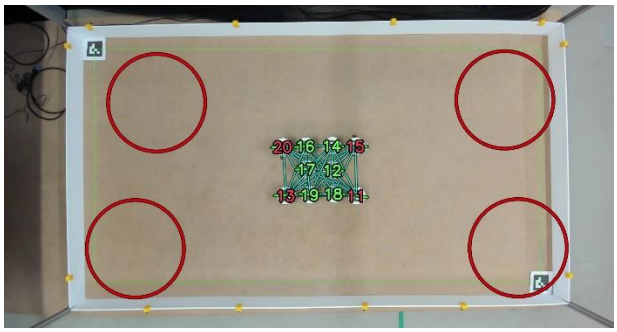


(a) (b)

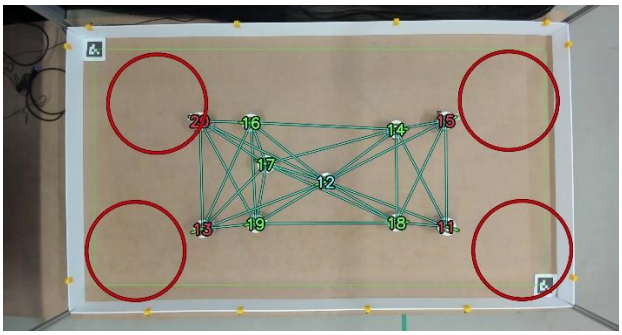


(c)

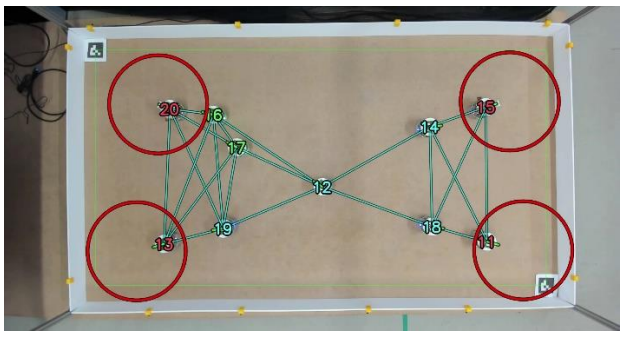
Figure 2. Screenshots of the ARGoS simulator, where a swarm of simulated e-pucks use the virtual machine to execute their behaviours to maintain connectivity between the leaders moving towards the locations of interest (red circles).



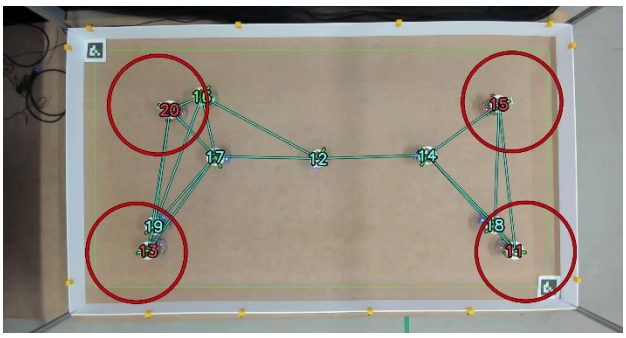
(a)



(b)



(c)

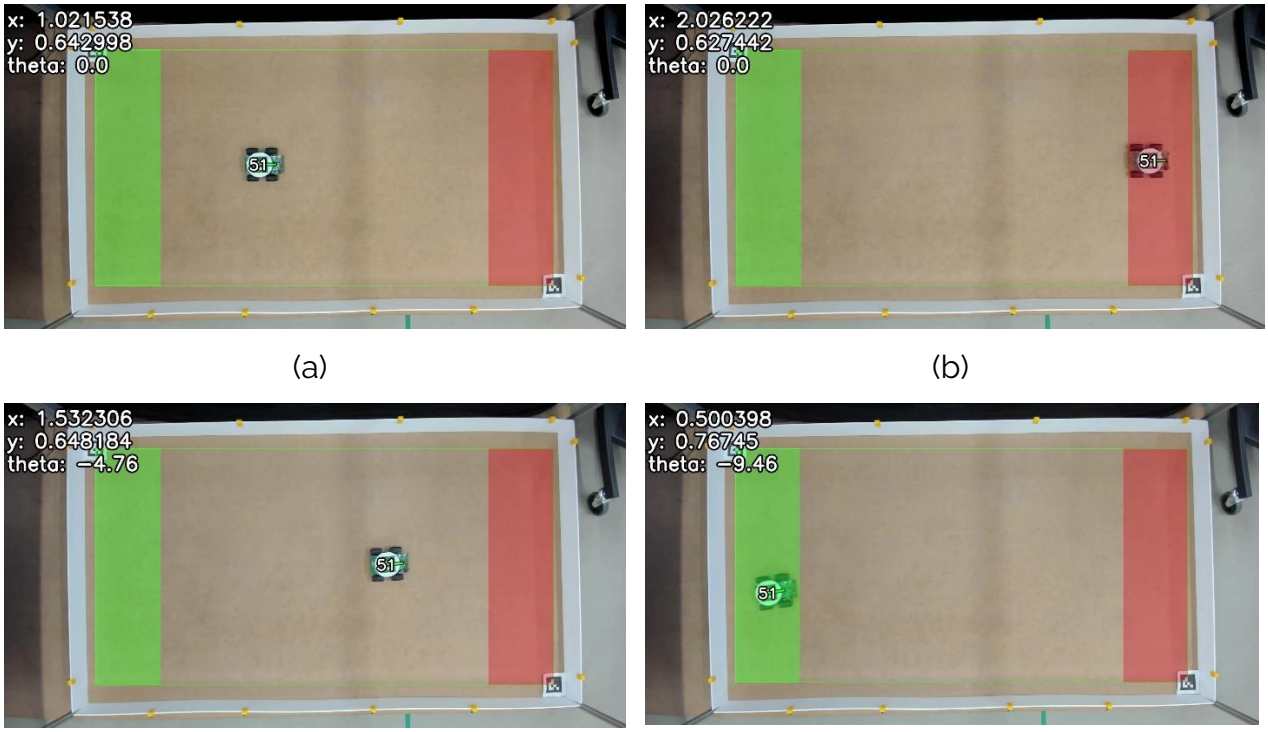


(d)

Figure 3. Screenshots of the real robot trials, where a swarm of Pi-pucks use the virtual machine to execute their behaviours to maintain connectivity between the leaders moving towards the locations of interest (red circles).

Similarly, in deliverable D4.3, Section "Swarms with human-in-the-loop", a user study investigated how two human operators could effectively share control of simulated robot swarms via a graphical user interface. The robot controller was initially written in C++, but later validated using real Pi-puck robots programmed in Python, again showing consistent performance across different platforms.

In Section Towards energy-efficient long-term operations of robot swarms, a swarm of simulated energy-constrained robots was tasked with performing a virtual task, consuming energy based on their actions (e.g. moving, performing the task, remaining idle). Two robot types – worker and mobile charger – were considered, both using SCT to define their behaviours. Demonstrations of transitions between moving to work and moving to recharge were shown using a CapBot (see Figure 4) and Kilobots (see Figure 5), which both use the C language but different APIs, further demonstrating the flexibility and cross-platform capabilities of virtual machines.



(c) (d)

Figure 4. A sequence of screenshots of the real robot trials, where a CapBot uses the virtual machine to execute its behaviour to move between charging and work regions (indicated in green and red, respectively).

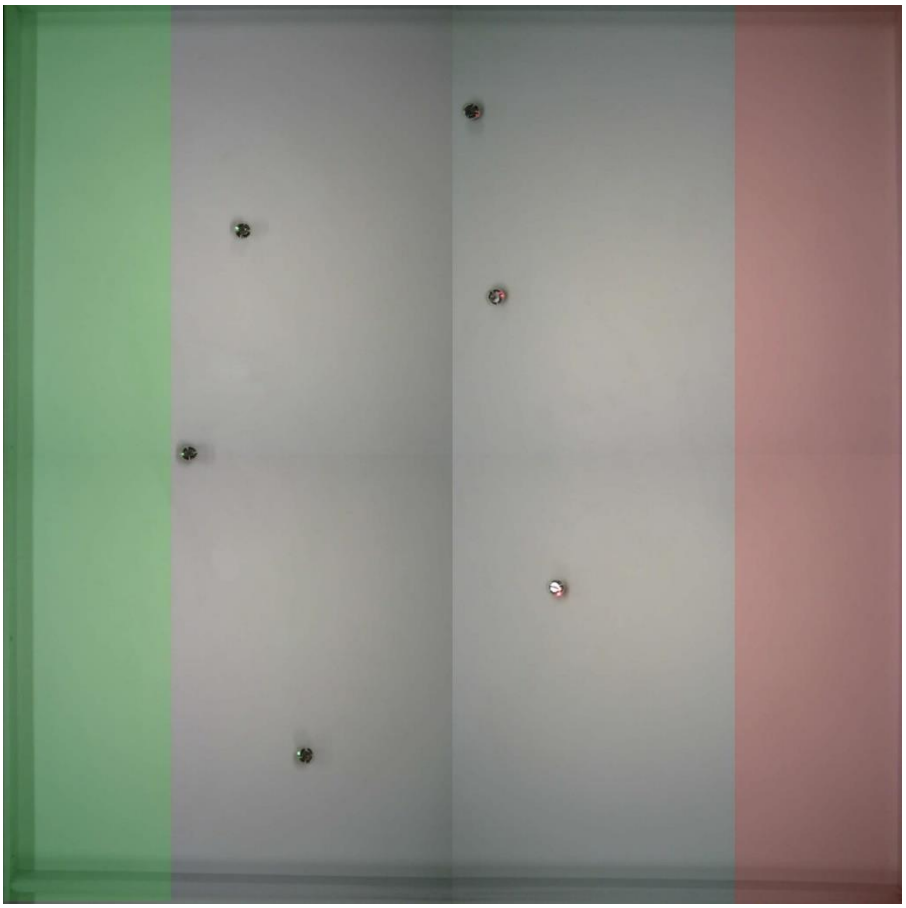


Figure 5. A screenshot of the real robot trials shows a swarm of Kilobots using the virtual machine to execute their behaviour and move between charging and work regions (indicated in green and red, respectively).

Finally, in the Section Optimising swarm behaviours through online learning a swarm of simulated energy-constrained UAVs was tasked with performing on-demand parcel delivery. The UAV's control logic was implemented using SCT, and the virtual machine was programmed using Python. Figure 6 shows a screenshot of the UAV carrying out parcel delivery in the simulator.

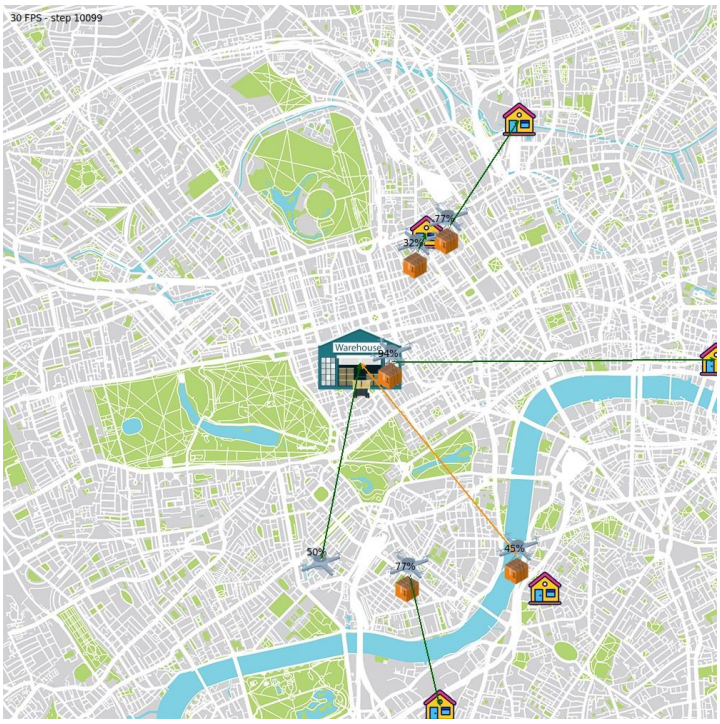


Figure 6. Screenshot of the ultra-fast Python-based multi-agent simulator, where a fleet of simulated UAVs use the virtual machine to execute their behaviours.

Towards Energy-efficient Long-term Operations of Robot Swarms

In real-world applications, robot swarms must operate autonomously over extended periods. Examples include environmental monitoring, surveillance, agriculture, construction, and mining [24], [25], [26]. A critical factor in such operations is energy management, where both energy in-flow (i.e. how robots replenish their energy) and out-flow (i.e. how robots utilise their energy to perform tasks) must be carefully optimised. At any given moment, robots must balance their energy consumption between performing useful work and replenishing energy reserves.

Several studies have explored swarms where individual robots alternate between performing tasks and visiting charging stations to recharge their finite energy storage [27] [28] [29] [30] [31]. To minimise charging times, systems with battery-swapping mechanisms at charging stations (hot-swap) have been considered [32]. Additionally, strategies to optimise the placement of (mobile) charging stations have been examined to reduce travel time for robots [33].

Another promising approach is energy sharing between robots, commonly referred to as energy trophallaxis [34] [35] [36] [37]. This approach can enhance division of labour, allowing some robots to focus on specific tasks [38], while reducing congestion at shared resources [39], such as charging stations. Battery-swapping between robots has been suggested to lower transfer times [40], and platforms such as CapBot have demonstrated efficient energy transfers with low transfer times and duty cycles of up to 98% [5]. However, challenges such as inefficient energy-sharing strategies or high energy transfer losses may hinder the uptake of this technology.

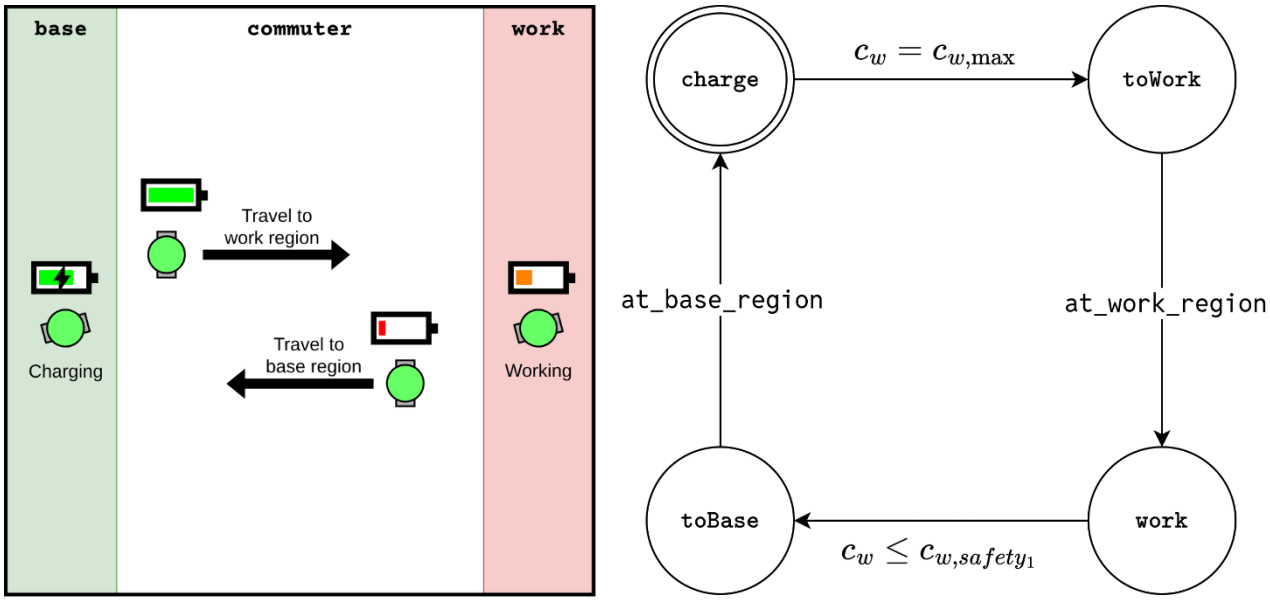
In this study, we compare two strategies for managing energy in-flow and out-flow for a swarm of robots tasked with long-term operation at a remote location. The first strategy relies solely on fixed charging stations, where each robot alternates between working at the remote site and returning to the base for recharging. The second strategy introduces mobile charging units with limited energy storage, which deliver energy to working robots and then return to the base for recharging. Robots in the swarm alternate between working and receiving energy from the mobile chargers. We derive formal upper bounds on the amount of work performed and the energy efficiency of both strategies. Through physics-based simulations, we explore the conditions under which either strategy becomes more favourable, considering variables such as the cost of idle time, movement, and task execution, as well as the ratio of energy storage capacity between mobile chargers and worker robots.

In the following sections, we first present the problem formulation and analysis for energy replenishment using fixed charging points. We then formulate the problem and analysis for energy replenishment using mobile chargers, followed by the implementation of both strategies and a discussion of the results.

Energy Replenishing Using Fixed Charging Stations

Problem Scenario

Consider the 2D environment illustrated in Figure 7a. The environment consists of three regions: (i) the *base* region (green); (ii) the *commuter* region (white); and (iii) the *work* region (red).



(a) (b)

Figure 7. Energy replenishment using fixed charging stations. (a) Workers (green disks) accumulate energy in the base region (green rectangle) before moving to the work region (red rectangle) to perform work. When a worker's energy storage runs low, it returns to the base region to recharge. (b) Finite-state machine executed by each worker robot.

The environment includes a population of n_w robot *workers* (green circles) . At each time step, a robot can either move or remain stationary. When located in the work region, the robot can also choose to perform work, with each working robot producing one unit of work per time step.

Each worker has a maximum energy storage of $c_{w,max}$ units. Its energy consumption is described as follows:

1. A stationary worker not performing work consumes $\nu_{w,min}$ units of energy per time step for core operations.
2. A moving worker not performing work consumes $\nu_{w,min} + \nu_{w,move}$ units of energy per time step.
3. A stationary worker performing work consumes $\nu_{w,min} + \nu_{w,work}$ units of energy per time step.
4. A moving worker performing work consumes $\nu_{w,min} + \nu_{w,move} + \nu_{w,work}$ units of energy per time step.

While within the base region, a worker accumulates energy at a gross rate of $\nu_{w,charge}$ units per time step. If the worker remains stationary, its net energy accumulation is $\nu_{w,charge} - \nu_{w,min}$, otherwise, it becomes $\nu_{w,charge} - \nu_{w,min} - \nu_{w,move}$. To transition from the base region to the work region, the worker must traverse the commuter region and vice versa.

The mission lasts for a finite duration τ. Initially, each worker's energy storage is assumed to be at full capacity. The workers aim to perform as many units of work as possible without depleting their energy while outside the base region. As $\tau \to \infty$, the workers, in theory, can continue to operate autonomously indefinitely.

Strategy

Figure 7b depicts a finite-state machine executed by each worker. Initially, all workers reside within the base region. At time zero, the worker is charging (state `charge`). Once the worker's energy reaches full capacity, it travels towards the work region (state `toWork`). Upon arrival, the worker performs work (state `work`). When the worker's energy level falls below the safety threshold, $c_{w,safety_1}$, it returns to the base region to recharge (state `toBase`).

Analysis

We formally derive an upper bound on the performance of the workers. In this theoretical model, we assume: (i) no interference among the workers (i.e., workers can move through each other); (ii) the base, commuter, and work regions are closed sets; (iii) all workers commute between the respective boundaries of the commuter region shared with the base and work regions; and (iv) all workers follow optimal paths.

At time 0, the worker leaves the base region at full capacity $c_{w,max}$. Let $\Delta_{w,commute}$ denote the time taken to reach the work region. Upon arrival at the work region at time $\Delta_{w,commute}$, the worker's remaining energy is $c_{w,max} - (\nu_{w,min} + \nu_{w,move})\Delta_{w,commute}$, after which it begins performing work. Once the worker's energy level decreases to $(\nu_{w,min} + \nu_{w,move})\Delta_{w,commute}$ units, it returns to the base region, arriving precisely when its energy reaches zero. It then recharges to full capacity in $\Delta_{w,charge} = \frac{c_{w,max}}{\nu_{w,charge} - \nu_{w,min}}$ time units. This cycle then repeats.

In each cycle, the time spent performing work is given by:

$$\Delta_{w,work} = \frac{c_{w,max} - 2(\nu_{w,min} + \nu_{w,move})\Delta_{w,commute}}{\nu_{w,min} + \nu_{w,work}}$$

The worker's duty cycle, D_w, is the proportion of time spent working, given by:

$$D_w = \frac{\Delta_{w,work}}{\Delta_{w,cycle}}$$

where $\Delta_{w,cycle} = \Delta_{w,work} + 2\Delta_{w,commute} + \Delta_{w,charge}$ is the duration of the worker's entire cycle.

The total amount of work performed by the system is:

$$W = n_w D_w \tau \tag{1}$$

Here we define the system's energy efficiency as the proportion of a worker's energy spent performing work[2], given by:

$$E = 1 - \frac{(v_{w,min} + v_{w,move})2\Delta_{w,commute} + v_{w,min}(\Delta_{w,work} + \Delta_{w,charge})}{c_{w,max}} \tag{2}$$

$$= 1 - \Delta_{w,cycle}\frac{v_{w,min}}{c_{w,max}} - 2\Delta_{w,commute}\frac{v_{w,move}}{c_{w,max}} \tag{3}$$

Energy Replenishing Using Mobile Charging Stations

Problem Scenario

Consider the 2D environment illustrated in Figure 8a. In comparison to the scenario with fixed charging stations, this scenario introduces a fourth region (blue), called *transfer* region, which sits between the commuter and work regions. The environment contains $\boldsymbol{n_w}$ workers and $\boldsymbol{n_m}$ mobile chargers. The workers retain the same capabilities as described in the previous scenario. Mobile chargers, however, are unable to perform work and can only transport energy.

[2] Note that devoting energy to core operations while performing work reduces a worker's efficiency.

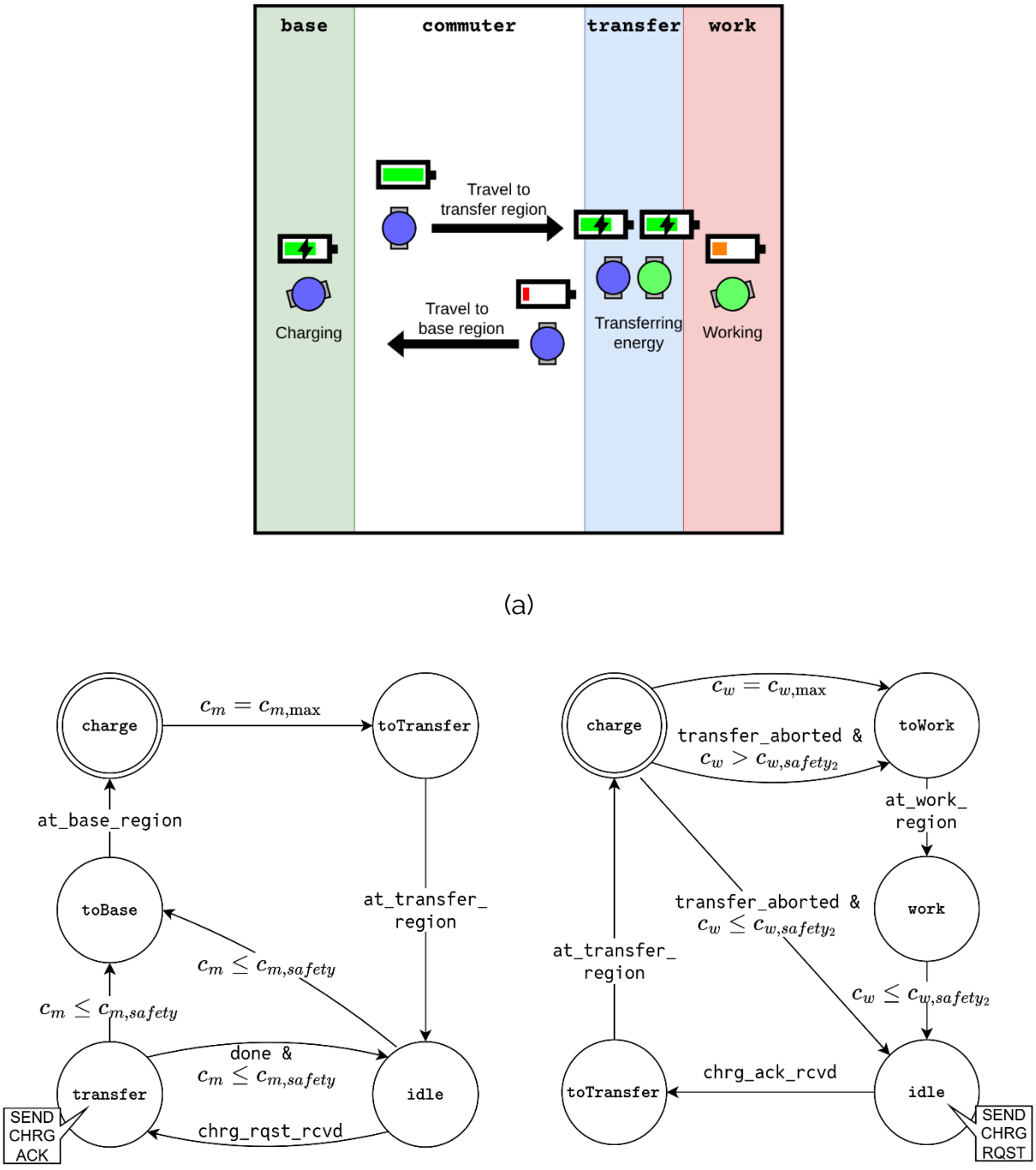


Figure 8. Energy replenishing using mobile chargers of limited storage capacity. (a) Mobile chargers (blue disks) transport energy from the base region to the transfer region (blue rectangle), where workers with low energy levels recharge. (b) Finite-

state machine executed by each mobile charging unit. (c) Finite-state machine executed by each worker robot.

Each mobile charger has a maximum energy storage capacity of $c_{m,max}$ units. Its energy consumption is as follows:

1. A stationary mobile charger consumes $\nu_{m,min}$ units of energy per time step to support core operations such as sensors reading, charge level monitoring, and communication.
2. A moving mobile charger consumes $\nu_{m,min} + \nu_{m,move}$ units of energy per time step.

While in the base region, a mobile charger accumulates energy at a gross rate of $\nu_{m,charge}$ units per time step. If stationary, the net energy gain is $\nu_{m,charge} - \nu_{m,min}$ and otherwise it is $\nu_{m,charge} - \nu_{m,min} - \nu_{m,move}$.

In the transfer region, a mobile charger can donate energy at a gross rate of $\nu_{m,transfer}$ units per time step to a worker. The worker receives energy at a gross rate of $\xi\nu_{m,transfer}$ units per time step, where $\xi \in (0,1]$ represents the transfer efficiency, accounting for energy loss during the process. During energy transfer both the mobile charger and the worker continue consuming energy for core operations and movement, if applicable.

Strategy

Figure 8b depicts the finite-state machine executed by each mobile charger. Initially, all mobile chargers reside in the base region charging at time zero (state `charge`). Once a mobile charger's energy reaches full capacity, it moves to the transfer region (state `toTransfer`). Upon arrival the mobile charger pauses (state `idle`), awaiting a transfer request from a worker. If its energy level falls below the safety threshold, $c_{m,safety}$, the mobile charger returns to the base region (state `toBase`). Otherwise, if it receives a transfer request, the mobile charger starts transferring energy to the worker robot (state `transfer`). The transfer ends when the worker's energy reaches full capacity, or the mobile charger's energy drops below $c_{m,safety}$, after which the mobile charger either returns to the `idle` state or transitions to the `toBase` state.

Figure 8c depicts the finite-state machine executed by each worker robot. All workers initially reside in the base region and are charging at time zero (state `charge`). Once fully charged, a worker moves to the work region (state `toWork`) and begins performing work (state `work`). If the worker's energy drops below the safety threshold, $c_{w,safety_2}$, it suspends work (state `idle`), repeatedly requesting energy transfers. Upon receiving acknowledgement from a mobile charger, the worker approaches it (state `toTransfer`), and the transfer process starts (state `charge`). Once the worker's energy is fully restored, it returns to the work region (state `toWork`). If the transfer ends prematurely, the worker evaluates whether its energy level exceeds the safety threshold, $c_{w,safety_2}$. If so, it moves towards the work region (state `toWork`); otherwise, it seeks energy from another mobile charger (state `idle`).

Analysis

We formally derive an upper bound for the performance of workers. We assume that (i) there is no interference among the agents' bodies (i.e., the agents may move through each other); (ii) the base, commuter, and work regions are closed sets; (iii) the transfer region is merely the boundary between the commuter and work region, (iv) all mobile units commute between the subsets of the boundary of the commuter region that are shared with the base region and work region, respectively, and follow optimal trajectories, and (v) workers reside on the boundary that the commuter and work regions share, and hence do not require any movement to switch between states (resting, working, or energy transfer).

We first consider the case of a system comprising a single mobile charging unit and a single worker robot, which have an identical cycle length, Δ_{cycle}. Hence, we have $\Delta_{cycle} = \Delta_{w,work} + \Delta_{w,rest} + \Delta_{transfer}$.

Given some value of $\Delta_{transfer}$, and a readily available mobile charging unit with sufficient energy, the robot can perform work for a maximum duration of

$$\Delta_{w,work} = \frac{\min\left(\Delta_{transfer}\left(\xi\nu_{transfer} - \nu_{w,min}\right), c_{w,max}\right)}{\nu_{w,min} + \nu_{w,work}}$$

where ξ is the transfer efficiency. In other words, $1 - \xi$ is the proportion of energy that is lost during transfer. In this case, we have $\Delta_{w,rest} = 0$, and $\Delta_{w,cycle_{min}} = \Delta_{w,work} + \Delta_{transfer}$.

The worker's duty cycle is the proportion of time it is working. It is given by

$$D_w = \frac{\Delta_{w,work}}{\Delta_{cycle}}$$

Given some value of $\Delta_{transfer}$, and a readily available mobile charging unit with sufficient energy, the robot can rest for a maximum duration of

$$\Delta_{w,rest} = \frac{\min\left(\Delta_{transfer}\left(\xi\nu_{transfer} - \nu_{w,min}\right), c_{w,max}\right)}{\nu_{w,min}}$$

In this case, we have $\Delta_{w,work} = 0$, and $\Delta_{w,cycle_{max}} = \Delta_{w,rest} + \Delta_{transfer}$. The worker's duty cycle is given by $D_w = 0$.

Let us consider the mobile charging unit. During the transfer period, the mobile unit consumes $\Delta_{transfer}\left(\nu_{m,min} + \nu_{transfer}\right)$. During commuting, it consumes $2\Delta_{commute}\left(\nu_{m,min} + \nu_{move}\right)$. In other words, the unit needs $c_{m,desired} = \Delta_{transfer}\left(\nu_{m,min} + \nu_{m,transfer}\right) + 2\Delta_{commute}\left(\nu_{m,min} + \nu_{m,move}\right)$ of energy. This is possible only if $c_{m,desired} \leq c_{m,max}$.

The mobile charger unit has to charge for a duration of

$$\Delta_{m,charge_{min}} = \frac{c_{m,desired}}{\nu_{m,min} + \nu_{m,charge}}$$

Hence, the minimum feasible cycle length is

$$\Delta_{m,cycle_{min}} = \Delta_{m,charge_{min}} + \Delta_{transfer} + 2\Delta_{commute}$$

If $\Delta_{m,cycle_{min}} < \Delta_{w,cycle_{min}}$, the mobile charging unit remains at the charger for a duration of $\Delta_{m,charge_{min}} + \Delta_{w,cycle_{min}} - \Delta_{m,cycle_{min}} > \Delta_{m,charge_{min}}$. The worker never rests, maximising its duty cycle for the given transfer period. However, a longer transfer period could be considered.

If $\Delta_{m,cycle_{min}} \geq \Delta_{w,cycle_{max}}$, no solution exists, and a shorter transfer period needs to be considered. Otherwise, the transfer period is feasible, but the worker is required to rest for some time, reducing its duty cycle.

All transfer periods for which the worker never rests are equally good in terms of duty cycle (assuming they exist). It can hence be desirable to determine the transfer period that, in addition, minimises the energy expenditure of the whole system per unit of work performed. We expect this to be the largest possible transfer period for which the worker never rests as such a configuration would minimise energy expended while commuted per unit of work performed.

The system's total amount of work performed is

$$W = n_w D_w \tau$$

where n_w is the number of workers, and τ is the overall duration, which would have to be multiple of the cycle length.

Here we define the energy efficiency of the whole system as the proportion of energy consumed by the entire system (i.e. the worker and mobile unit) that is spent by the worker to perform work. Formally,

$$E = \frac{\Delta_{work} \nu_{w,work}}{\Delta_{cycle}(\nu_{w,min} + \nu_{m,min}) + 2\Delta_{m,commute}\nu_{m,move} + (1-\xi)\Delta_{transfer}\nu_{transfer} + \Delta_{work}\nu_{w,work}}$$

The equations could be extended to address scenarios with multiple workers and mobile units.

Implementation

To evaluate the energy replenishment strategies, we implement the state machines shown in Figure 7b, Figure 8b and Figure 8c on the e-puck robot platform [2]. The e-puck is a mobile differential-wheeled robot with a 7 cm diameter and a maximum speed of $v_{max} = 12$ m/s. Each robot is equipped with a range-and-bearing system that enables relative localisation and communication within a local neighbourhood of radius 0.533 m.

All robots update their states using the state machines and employ virtual forces to determine their movement direction. Let p_{ij} denote the position of robot j in the local coordinate system of robot i. The virtual force acting on robot i is defined as:

$$u_i = \alpha u_i^a + \beta u_i^{rn} + \gamma u_i^{ro}$$

where α, β and γ are positive scalars that weigh the influence of the different components. The components are as follows:

- Attraction component (u_i^a): Represents the attraction towards a goal g_i, Which can be a point in the base region, work region, or transfer region, defined as:

$$u_i^a = \frac{g_i}{||g_i||} \min(||g_i||, v_{max})$$

  where g_i is a position vector of the target goal, relative to the position of robot i.

- Repulsion from Neighbours (u_i^{rn}): Ensures that robots avoid collisions with nearby robots by calculating repulsion forces based on their distance as:

$$u_i^{rn} = -\frac{1}{|N_i|} \sum_{j \in N_i} \frac{\sigma^\lambda}{||p_{ij}||^\lambda} \frac{p_{ij}}{||p_{ij}||}$$

where N_i denotes the set of robots in the neighbouring of robot i, σ is the desired separation between robots, and λ is the exponent that controls the strength of the repulsive force.

- Repulsion from Obstacles (u_i^{ro}): Prevents collisions with obstacles (e.g. walls) using the e-puck's eight proximity sensors, which are placed around the robot's circumference [2], defined as: $u_i^{ro} = -\frac{1}{8d_{max}}\sum_{j=1}^{8}(d_{max} - d_j)\hat{v}_j$

  where $d_{max} = 10\,cm$ is the sensors' range, d_j is the distance reported by the j^{th} sensor, and $\hat{v}_j$ is the unit vector pointing from the robot's centre to the j^{th} sensor. If sensor j detects no obstacle, we set $d_j = d_{max}$.

Results

We use physics-based simulations that account for robot collisions to quantify the performance of energy replenishment strategies in terms of both work performed and energy efficiency. Simulations are run in ARGoS [41], with the physics and state machines updated every 0.1 seconds.

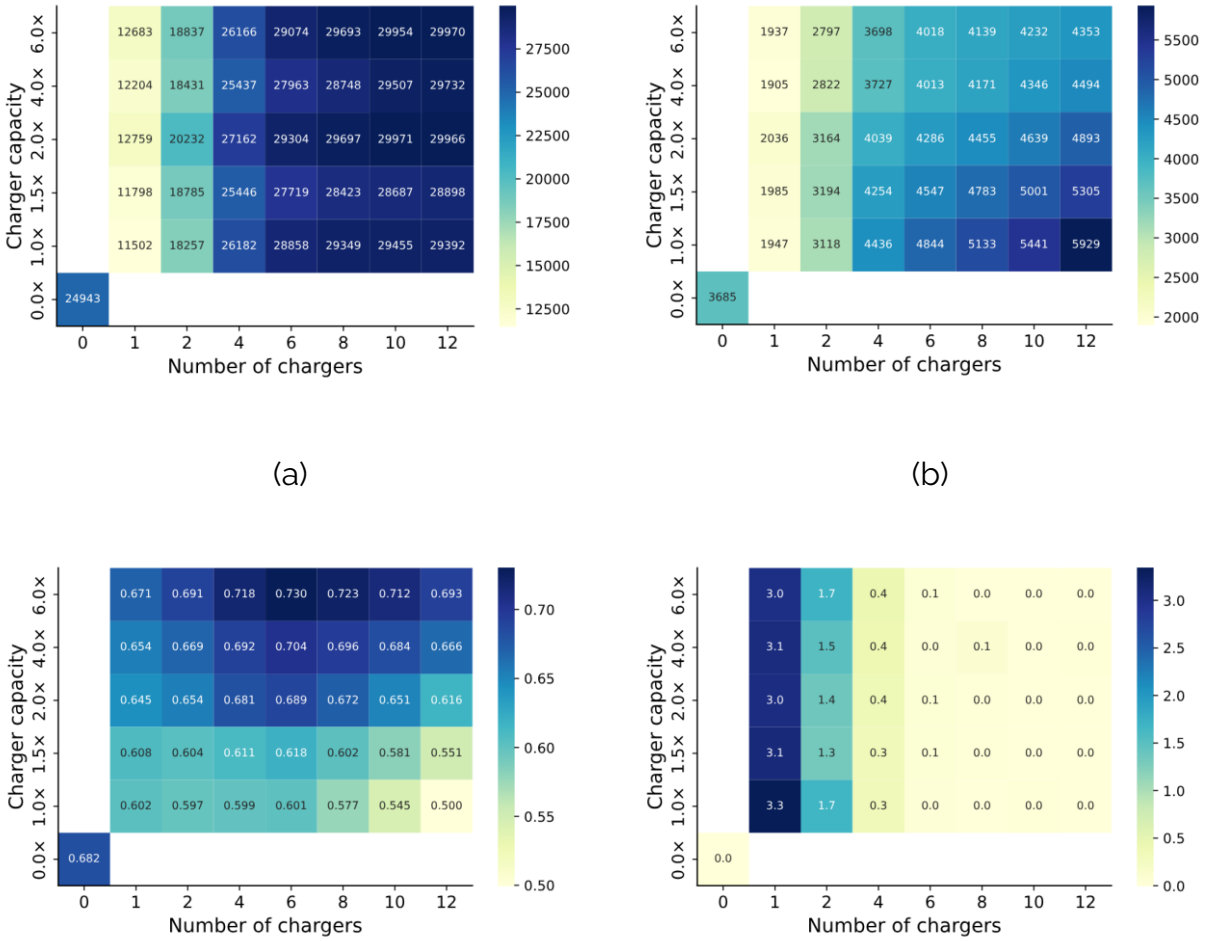
The arena is $1.6m \times 1.6m$ (H×W) and is bounded within $x, y \in [-0.8, 0.8]$. The base, work, and transfer regions each have dimensions of $0.3m \times 1.6m$. We use $\alpha = 1$, $\beta = 100$, $\gamma = 1$, $\sigma = 0.1m$ and $\lambda = 24$ for the robot's motion, and, by default, there is no transfer loss ($\xi = 1$). Each trial lasts for 10 minutes. Video recordings of the simulation are available at [42], and the source code can be found at [43].

Fixed vs Mobile Charging Stations

For both strategies, the same number of workers is used, while one strategy introduces mobile chargers. This approach is based on the observation that replacing a worker with a mobile charger does not increase the amount of work performed and even leads to a reduction[3]. However, mobile chargers can be useful as they do not need to perform work and thus do not require costly work tools. Additionally, they can be designed for more efficient movement and offer increased energy storage.

We conducted trials with $n_w = 6$ workers And, for the mobile charger strategy, we varied the number of chargers $n_m = \{1,2,4,6,8,10,12\}$ and their storage capacity $c_{m,max} = \phi c_{w,max}$ where $\phi = \{1,1.5,2,4,6\}$ to examine their effects.

[3] This is because the transfer of energy from mobile chargers to workers requires time, and during this time all parties consume a base level of energy.



(a) (b)

(c) (d)

Figure 9. The effect of the number of mobile chargers and their storage capacity on (a) the amount of work performed, (b) the total energy consumed, (c) the proportion of energy used to perform work, and (d) the number of depleted workers. The results of the static charger strategy are shown in 0 chargers and 0× charger capacity cells. Each configuration was tested for 50 trials.

Figure 9a shows that 24,943 units of work are performed when workers charge at the base. This represents 92.8% of the theoretical upper bound (derived from equation 2), suggesting near-optimal performance in the simulation.

When workers obtain energy from mobile chargers, the amount of work performed increases with the number of mobile chargers, $\boldsymbol{n_m}$, and plateaus after around $\boldsymbol{n_m = n_w}$. With 6 and 12 mobile chargers, the work performed is 89.3% and 92.0%, respectively, of

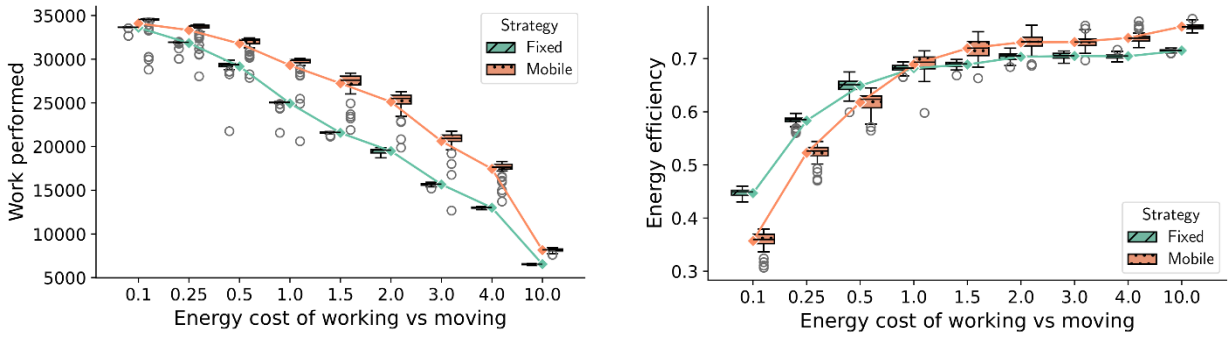
what could be achieved if the transfer region offered a constant energy supply (equivalent to setting $\Delta_{w,commute} = \mathbf{0}$ in Equation (1). Performing substantially more work would require additional workers.

Figure 9b shows the total energy consumed by all robots, whereas Figure 9c presents the energy efficiency - i.e., the proportion of total energy used for work. Workers charging at the base achieve an energy efficiency of 68.2%, only 12.6% below the upper bound obtained from Equation (3. For workers receiving energy from mobile chargers, the highest energy efficiency occurs consistently when $\boldsymbol{n_m} = \boldsymbol{n_w}$, indicating that an equal number of mobile chargers and workers leads to optimal collaboration. Moreover, deploying many mobile chargers with small capacity proves inefficient. Nevertheless, mobile chargers were able to help perform more work and even improve energy efficiency when their storage capacity was larger than the workers.

Finally, Figure 9d reveals that a small number of mobile chargers, even those with high capacity, cannot supply sufficient energy to all workers due to only being able to serve one robot at a time. As a result, some robot workers may deplete their energy while waiting.

Impact of Workload

In the previous section, we assumed equal energy consumption rates for work and movement (i.e., $\nu_{*,move} = \boldsymbol{\nu}_{*,work}$). Here, we investigate the impact of changing the energy consumption rate for work. We use $\boldsymbol{n_m} = \mathbf{6}$ and $\boldsymbol{c_{m,max}} = \mathbf{2}\boldsymbol{c_{w,max}}$ for this analysis. This setting performed more work while maintaining a similar energy efficiency to the strategy without mobile chargers (see Figure 9c).



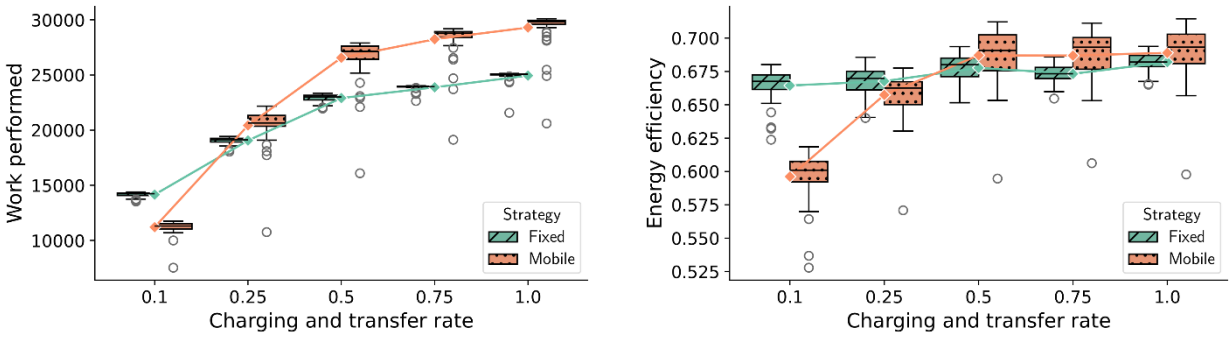
(a) (b)

Figure 10. Comparison of the two strategies across different energy consumption ratios $\frac{v_{*,work}}{v_{*,move}}$. The solid lines connect the average values indicated by the diamond marker for each strategy. For mobile chargers, results are shown for 6 chargers and 2× charger capacity. (a) The amount of work performed and (b) the proportion of energy used to perform work. Each configuration was tested for 50 trials.

Figure 10a shows that as the energy required for performing work increases, less work is completed. The strategy using mobile chargers continues to outperform the strategy without them. Figure 10b shows that mobile chargers are more energy-efficient when work consumes as much or more energy than movement. Otherwise, sending workers back to the base for recharging proves more energy-efficient.

Impact of Charging Rate

We also explore the effect of charging and transfer rates. For the mobile charger strategy, we use $n_m = 6$ and $c_{m,max} = 2c_{w,max}$.



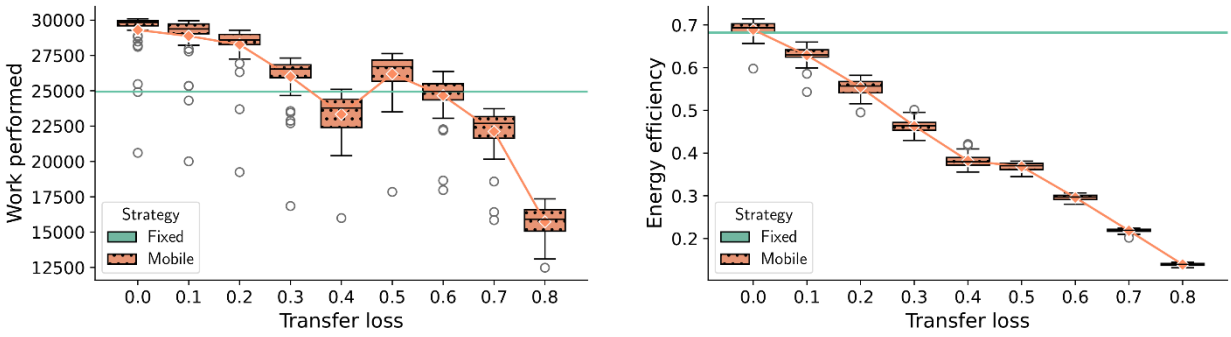
(a) (b)

Figure 11. Comparison of the two strategies across different charging and transfer rates, $v_{*,charge} = v_{m,transfer}$. The solid lines connect the average values indicated by the diamond marker for each strategy. For mobile chargers, results are shown for 6 chargers and 2× charger capacity. (a) The amount of work performed and (b) the proportion of energy used to perform work. Each configuration was tested for 50 trials.

Figure 11a shows that as charging and transfer rates, $v_{*,charge}$ and $v_{m,transfer}$, decreases, less work is completed, as robots spend more time charging. Figure 11b illustrates that energy efficiency for fixed chargers remains relatively constant, as movement tends to dominate charging time (see equation 4). For mobile chargers, however, energy efficiency decreases significantly when $v_{*,charge} \leq 0.25$ due to the longer charging times impacting performance. As the charging times start to dominate, this disproportionately impacts the strategy with mobile chargers, as it requires twice the time for workers to charge.

Impact of Energy Transfer Loss

Finally, we consider energy transfer loss, where some energy dissipates into the environment during transfers between mobile chargers and robot workers. For this test, we use $n_m = 6$ and $c_{m,max} = 2c_{w,max}$.



(a) (b)

Figure 12. Comparison of the two strategies for different levels of transfer losses ξ. The solid line connects the average values indicated by the diamond marker for each strategy. For mobile chargers, results are shown for 6 chargers and 2× charger capacity. (a) The amount of work performed and (b) the proportion of energy used to perform work. Each configuration was tested for 50 trials.

Figure 12a reveals a non-monotonic dependency. The amount of work performed decreases as transfer loss increases up to 40%, then rises at 50%, before decreasing again beyond 60%. This behaviour is due to the timings of the mobile chargers and workers' activities. At a 50% transfer loss, mobile chargers could serve a single worker before needing to return to the base. At 40%, they attempted to serve two workers, but often had to abort to return to the base prematurely.

As expected, Figure 12b shows that transfer loss negatively impacts energy efficiency, as some energy is lost during transfer.

Conclusion

This main section, addressing energy-efficient long-term operations of robot swarms, compared two energy replenishment strategies for a swarm of robots. In the first strategy, all robots were responsible for their own energy and commuted between charging stations and remote work locations. In the second strategy, robots capable of

performing work at remote locations depended on mobile chargers to deliver and transfer energy.

We compared the two strategies based on the amount of work performed and energy efficiency using physics-based simulations. The results reveal conditions under which both strategies approach their theoretical upper bounds. Mobile chargers prove beneficial when there is an adequate number of them, neither too few nor too many, and when they possess a large energy capacity. Moreover, they are particularly advantageous when more energy is required for performing work than for navigating. However, slow recharging rates and energy transfer losses can negatively impact both the amount of work performed and energy efficiency.

These insights provide a useful guide for designing energy-efficient swarm systems, particularly in scenarios where energy delivery systems must be optimised for long-term autonomous operations in dynamic environments.

Optimising Swarm Behaviours Through Online Learning

Unmanned Aerial Vehicle (UAV)-based on-demand delivery is expected to bring significant benefits to society, including reductions in delivery times, operational costs, and carbon emissions. By deploying a fleet of UAVs, multiple orders can be fulfilled simultaneously, enabling large-scale applications in parcel, meal, and medication deliveries.

Various methodologies have been explored to address delivery problems with fleets of energy-constrained UAVs, including those that optimise order allocation and route planning. These include mixed-integer linear programming (MILP) [44] [45] [46] [47], queuing theory [48], auctions [49], and various heuristic methods [6, 20, 21]. An interesting problem variant is on-demand deliveries, where orders arrive stochastically over time [50] [51] [52].

However, several key assumptions in these studies may not hold in real-world scenarios:

- **Pre-defined UAV energy models:**, Most studies assume that the UAVs know their energy consumption models and can be used them when planning their deliveries. This poses the risk of over-reliance on the given models: While various energy models have been proposed, it has been shown that different models can lead to divergent outcomes even under the same conditions [53].
- **UAV homogeneity:** UAVs are often assumed to be homogeneous, which is generally not the case, even for UAVs of the same brand or model. Variability can arise over time due to factors such as battery health and hardware wear (e.g., propellers and motors), resulting in unique energy models for each UAV.

- **Full battery assumption:** It is often assumed that UAVs charge instantly when visiting their base. This may hinder their application in practical scenarios, as it does not reflect the low duty cycles of most commercial delivery UAVs (e.g., 6.97% for DJI FlyCart 30 with one battery). Factoring in the significant proportion of time devoted to charging ultimately leads to a new class of problems. For example, UAVs opting to dispatch with only a partial charge could result in faster deliveries than those waiting to charge fully.

The contributions of this paper are threefold:

1. **Decentralised learning-based deployment strategy for UAVs with unknown battery health:** We propose a decentralised learning-based deployment strategy for on-demand delivery using UAV fleets. Each UAV decides whether to bid on incoming orders based on its battery's state of charge, parcel mass, and delivery distance. Unlike prior work, our approach requires no knowledge of the UAV's battery state of health (i.e., its true capacity) or its energy consumption model. Over time, each UAV learns to bid only on orders that it is likely to fulfil, enhancing the overall efficiency of the fleet.
2. **Long-term evaluation of heterogeneous fleets:** We evaluate the proposed strategy using simulated UAV fleets that vary in battery health due to factors such as ageing, usage, and storage patterns. Simulations using realistic UAV energy models are conducted over a period of eight weeks, with orders arriving stochastically at different rates. The results show that deploying the least confident bidding UAVs leads to the highest number of fulfilled orders and the lowest delivery times. Furthermore, the strategy outperforms traditional approaches that dispatch UAVs only when their battery charge exceeds predefined thresholds.
3. **Forecasting-enabled order commitment:** We propose a second variant of the strategy where learned bidding policies are used for forecasting. This enables UAVs with insufficient charge levels to commit to fulfilling current orders at a specific future time, improving prioritisation of earlier orders and aligning better with first-come, first-served scheduling.

It is noteworthy that although we focus on an on-demand UAV delivery scenario, our proposed framework can be applied to other robotic systems where agents are unaware of the full extent of their capabilities at the start of deployment.

In what follows, we present the problem formulation, and then we provide details about the proposed decentralised learning-based deployment strategy. After that, we present the simulator and results, respectively.

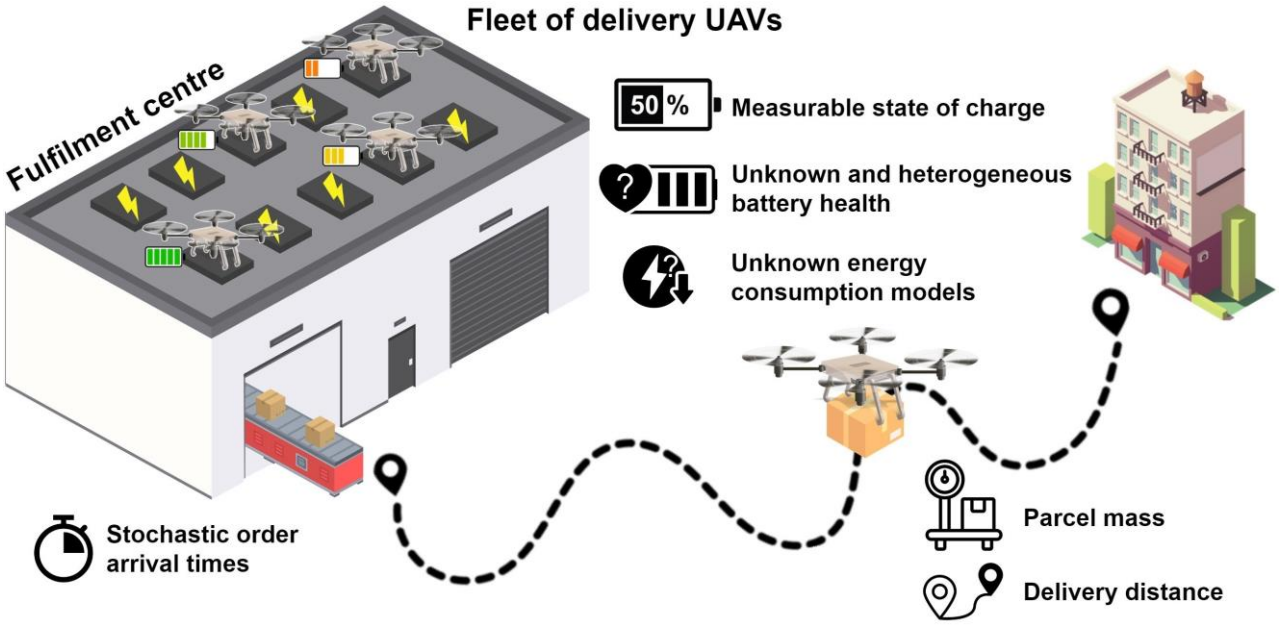


Figure 13 On-demand delivery scenario with a fleet of UAVs committing to deliver orders arriving at a fulfilment centre. The fleet is heterogeneous, as the UAVs differ in battery health, and hence in their true energy storage capacities. Each UAV learns a policy for placing bids on incoming orders based on its current charge level, parcel mass and delivery distance. Optionally, it can use this policy for forecasting, enabling it to plan the fulfilment of orders in the future.

Problem Statement

As illustrated in Figure 13, a fulfilment centre (FC) operates a fleet of delivery UAVs, denoted as $U = \{1, 2, \dots, S\}$. During an operational period of length T, the FC receives orders $0 = \{1, 2, \dots\}$, which arrive sequentially at times $t_1, t_2, \dots$, where $0 \leq t_1 < t_2 < \cdots < T$. Each order $j \in 0$ corresponds to a delivery task associated with a parcel of mass $m_j \in [m_{\min}, m_{\max}]$, to be delivered to a location at a distance $d_j \in [d_{\min}, d_{\max}]$ from the FC.

The FC maintains a communication infrastructure (e.g., a wireless network) to communicate with available UAVs (e.g., to announce delivery tasks) and allows UAVs to communicate with each other, facilitating collaborative decision-making.

All UAVs are powered by an onboard battery with a *theoretical capacity* $C_{\text{theoretical}}$. The battery of UAV $i \in U$ has a state of health (SoH), denoted by $SoH_i \in [0,1]$, which determines its true (maximum) capacity. Specifically, the true battery capacity is given by $C_{\text{true},i} = SOH_i \cdot C_{\text{theoretical}}$. In this study, SoH_i is assumed to remain constant throughout the operational period.

At any given time t, UAV $i \in U$ has a battery state of charge (SoC), denoted by $SoC_i(t) \in [0,100]$. which represents the percentage of $C_{\text{true},i}$that is currently available in its battery. At $t = 0$, all UAVs are assumed to be fully charged, such that $\forall i \in U: SoC_i(0) = 100$. The state of charge of UAV i changes based on its activity:

- *Charging*: While within the FC, its SoC increases at a rate of λ_{charge} per unit of time until it reaches the maximum value.
- *Delivering*: When outside the FC, its SoC decreases at a rate of $\lambda_{\text{delivery}}(m_p)$ per unit time, where m_p represents its payload. If the UAV is carrying a parcel with mass m, we have $m_p = m$; if not, $m_p = 0$.

Each UAV can measure its state of charge $SoC_i(t)$ at any time, but is unaware of its state of health SoH_i, true capacity $C_{\text{true},i}$ and energy consumption model $\lambda_{\text{delivery}}(m_p)$.

When outside the FC, a UAV moves at a constant speed of v_a. It first moves towards the parcel's delivery destination. Upon successful delivery, it immediately returns to the FC.[4] While transporting a parcel to its delivery destination, a UAV may choose to abort the delivery and return to the FC. We refer to this as an *aborted delivery attempt*. If a UAV's SoC reaches zero during delivery, it becomes unable to return to the FC and is considered lost.

The objective is to define a decentralised deployment strategy that assigns advertised delivery tasks to specific UAVs. Various performance criteria are explored,

[4] For simplicity, we assume UAVs can move directly towards their destinations. Our methods could be extended to accommodate UAVs navigating around static obstacles, provided a map of the environment is available.

including: (i) the number of successfully delivered parcels (the more, the better); (ii) the time elapsed from order arrival to successful delivery, referred to as *delivery time* (the lower, the better); (iii) the number of unsuccessful delivery attempts (the fewer, the better); and (iv) the cumulative backlog age which gives the total waiting time for all unfulfilled orders (the lower, the better).

Decentralised Learning-based Deployment Strategy

We propose a decentralised deployment strategy that augments a finite-state machine controller with auction-based and online learning approaches for assigning UAVs to advertised delivery tasks. The strategy runs onboard each UAV. It comprises four components (see Figure 14): (i) a *UAV controller* to define the overall logic and help a UAV transition among its behavioural states; (ii) a *bidding* policy to decide for any advertised delivery task whether to place a bid and a bid value, reflecting its confidence in bidding; (iii) a *bids evaluation* policy to determine the winner amongst the UAVs that bid for the same task; (iv) and an *online learning algorithm* to refine the individual bidding policy such that the UAV bids only for tasks it is likely capable of completing successfully.

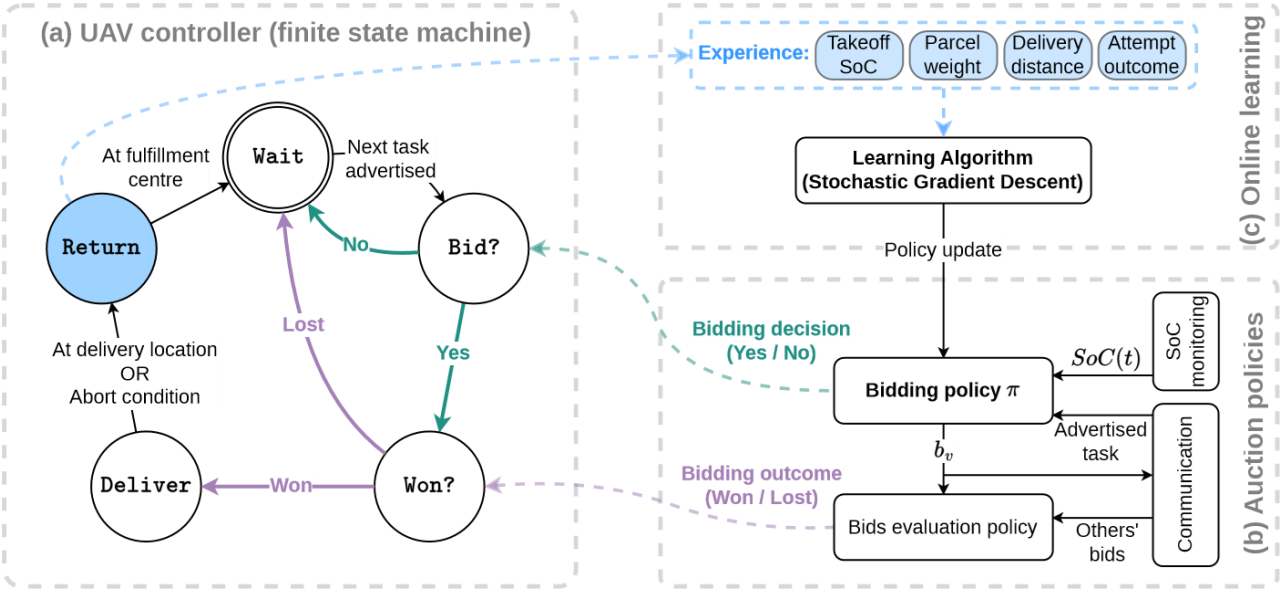


Figure 14. Overview of the decentralised learning-based deployment strategy: (a) A UAV's core logic is governed by a finite-state machine; (b) the UAV uses two auction policies: one for bidding for announced delivery tasks, and another for bids

evaluation to locally determine whether its bid was successful; (c) upon returning from a delivery attempt, the UAV updates its bidding policy.

UAV Controller

We assume that all UAVs initially reside within the FC. The UAV controller is depicted in Figure 14a. A UAV begins in the **`Wait`** state, where it awaits an announcement of the next delivery task along with the task ID. A task announcement consists of a unique task id, parcel mass and delivery distance. When a task announcement is received, the UAV transitions to the **`Bid?`** state. In this state, it uses its bidding policy to determine whether to bid, and a bid value, reflecting its level of confidence in the bid. If the UAV opts to bid, it proceeds to the **`Won?`** state; otherwise, it returns to the **`Wait`** state.

In the **`Won?`** state, the UAV broadcasts a tuple comprising (i) the task ID, (ii) its unique ID, and (iii) the bid value for placing the bid. In parallel, it records its bid and those of any other UAVs bidding for the same task. Subsequently, it provides all bids to its local *bids evaluation* policy, which enables it to determine the outcome of the auction. If it won, the UAV records its current state of charge, hereafter denoted as $SoC_{takeoff}$, and transitions to the **`Deliver`** state. Otherwise, it returns to the **`Wait`** state.

When in the **`Deliver`** state, the UAV retrieves the parcel (assumed to happen instantaneously) and flies to the delivery destination. It either (i) reaches the destination and the delivery is successful, or (ii) meets an abort condition, in which case the delivery is unsuccessful. In either case, the UAV transitions to the **`Return`** state. The abort condition is set such that the UAV returns to the FC if $SoC(t) \leq \xi\, SoC_{takeoff}$, that is, it returns once only a fraction ξ of its state of charge at the time of takeoff remains. The controller is modelled using supervisory control theory [54].

In the **`Won?`** state, the UAV broadcasts a tuple comprising: (i) its unique ID; (ii) the task ID; and (iii) the level of confidence for placing the bid. It simultaneously records the IDs and bid values of the other UAVs bidding for the same task. Subsequently, it provides all bids to its local *bids evaluation* policy, which enables it to determine whether it won or lost the auction. If it wins the auction, the UAV records its current state of charge,

hereafter denoted as SoC_{takeoff}, and transitions to the **`Deliver`** state. Otherwise, it returns to the **`Wait?`** state.

When in the **`Deliver`** state, the UAV retrieves the parcel (assumed to happen instantaneously) and flies to the delivery destination. It either (i) reaches the destination and the delivery is successful; or (ii) meets an abort condition, in which case the delivery is unsuccessful. In either case, the UAV then transitions to the **`Return`** state. The abort condition is set such that the UAV returns to the FC if $SoC_i(t) \leq \xi SoC_{\text{takeoff}}$, i.e., it returns once only a fraction ξ of the initial state of charge remains.

Auction Policies

As depicted in Figure 14b, the UAV's role in the auction-based task allocation process is implemented through two policies: one for bidding (to decide whether to bid and the bid value) and another for bids evaluation (to determine whether the bid was successful).

Bidding Policy:

Let UAV i consider a delivery task $j \in O$ at time t. Its bidding policy, $\pi(\cdot)$, takes as input the task's distance d_j, the parcel's mass m_j, and the UAV's current state of charge $SoC_i(t)$. The features' vector is defined as $\boldsymbol{x}^T = [d_j, m_j, SoC_i(t)]$. The bidding policy outputs a binary bidding decision b_d (whether to bid or not) and a bid value b_v:

$$\pi(\boldsymbol{x}) = [b_d(\boldsymbol{x}), b_v(\boldsymbol{x})] \quad \textbf{(4)}$$

- Bidding Decision:

The function $b_d(\boldsymbol{x})$ classifies input features $\boldsymbol{x}$ into two categories: 1 if the UAV is capable of performing the task and 0 if not:

$$b_d(\boldsymbol{x}) = \begin{cases} 1\,, & f(\boldsymbol{x}) \geq 0 \\ 0\,, & f(\boldsymbol{x}) < 0 \end{cases} \quad \textbf{(5)}$$

where $f(\boldsymbol{x})$ is the decision function.

In this study, a Support Vector Machine (SVM) is utilised as the decision function:

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b \tag{6}$$

where $\boldsymbol{w}$ and b are the weight vector and bias, respectively. The values of these parameters are specific to each UAV and are updated online by the UAV after each delivery attempt. Here, the decision function $f(\boldsymbol{x})$ represents a hyperplane that separates the two categories of input features. Despite their simplicity, linear classifiers like this have demonstrated accuracy comparable to more complex nonlinear classifiers, while significantly reducing training times [55] [56].

- Bid Value:

When a UAV i decides to place a bid (i.e. $b_d(\boldsymbol{x}) = 1$), a bid value b_v has to be determined. It can be calculated as the distance of the input features $\boldsymbol{x}$ from the decision hyperplane $f(\boldsymbol{x})$:

$$b_v(\boldsymbol{x}) = \frac{f(\boldsymbol{x})}{\|\boldsymbol{w}\|_2} \tag{7}$$

where $\|\cdot\|_2$ is the Euclidean norm. This value of $b_v(\boldsymbol{x})$ reflects the UAV's perceived ability to complete the task, with a greater distance from the hyperplane indicating higher confidence.

Bids Evaluation Policy:

The bids evaluation policy processes the bids of all competing UAVs (including the UAV itself), returning True if the UAV is deemed the winner and False otherwise. We consider the following options for selecting the winner:

- *least confident*: the UAV with the lowest bid value, that is, of lowest confidence, is the winner;
- *most confident*: the UAV with the highest bid value, that is, of highest confidence, is the winner;

In both cases, if a tie occurs, the UAV with the highest ID is selected as the winner.

Online Learning

After each delivery attempt, each UAV i refines its bidding policy, as expressed by decision $f(\boldsymbol{x})$ through continuous online learning. When a UAV attempts a delivery task $j \in O$, it collects a labelled data point $(\boldsymbol{x_a}, y_a)$. The vector $\boldsymbol{x_a}$ represents the features used by the UAV earlier in the bidding process, given by $\boldsymbol{x_a} = [d_j, m_j, SoC_{\text{takeoff}}]$, where d_j is the delivery distance, m_j is the parcel mass, and SoC_{takeoff} is the UAV's state of charge at takeoff. The label y_a indicates the delivery outcome: 1 for a successful completion and 0 for an aborted delivery.

Upon collecting a labelled data point, the UAV updates its decision function $f(\boldsymbol{x})$, as defined in Equation 3, using the Stochastic Gradient Descent (SGD) algorithm [57]. To account for the sensitivity of SGD to the scale of input data, the features are standardised before updating:

$$\tilde{\boldsymbol{x}}_{\boldsymbol{a}} = \frac{\boldsymbol{x_a} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \tag{8}$$

where $\tilde{\boldsymbol{x}}_{\boldsymbol{a}}$ represents the standardised input features, $\boldsymbol{x_a}$ the original input features, and $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ denote the mean and standard deviation of the input features vector, respectively.

The decision function updates are carried out with the objective of minimising the regularised training error:

$$E(\boldsymbol{w}, b) = L\big(y_a, f(\tilde{\boldsymbol{x}}_{\boldsymbol{a}})\big) + \alpha R(\boldsymbol{w}) \tag{9}$$

where L is the loss function measuring the (mis)fit of the decision function, and R is a regularisation term. The hyperparameter $\alpha > 0$ controls the strength of the regularisation.

In our implementation, we employ a modified Huber loss function for its proven ability to enable fast and robust learning [58] [59]:

$$L\left(y_a, f(\tilde{\boldsymbol{x}}_{\boldsymbol{a}})\right) = \begin{cases} max\left(0, \left(1 - (2y_a - 1)f(\tilde{\boldsymbol{x}}_{\boldsymbol{a}})\right)^2\right), & \text{if } (2y_a - 1)f(\tilde{\boldsymbol{x}}_{\boldsymbol{a}}) \geq -1 \\ -4\,(2y_a - 1)f(\tilde{\boldsymbol{x}}_{\boldsymbol{a}}), & \text{otherwise} \end{cases}$$

(10)

and an ℓ_2 norm regularisation term:

$$R(\boldsymbol{w}) = \frac{\|\boldsymbol{w}\|_2}{2}$$

(11)

Using SGD, the parameters of the decision function are updated iteratively as follows:

$$\begin{cases} \boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\left[\dfrac{\partial L\left(y_a, f(\tilde{\boldsymbol{x}}_{\boldsymbol{a}})\right)}{\partial \mathbf{w}} + \alpha \dfrac{\partial R(\boldsymbol{w})}{\partial \boldsymbol{w}}\right] \\ b \leftarrow b - \eta \dfrac{\partial L\left(y_a, f(\tilde{\boldsymbol{x}}_{\boldsymbol{a}})\right)}{\partial b} \end{cases}$$

(12)

where η is the learning rate. To improve convergence and stability, η is progressively decayed over optimisation steps, following the heuristic approach proposed by [60].

The decision function is initially trained based on two assumed labelled data points: (i) $([d_{\min}, m_{\min}, 100.0], 1)$, suggesting that a fully charged UAV, that is, $SoC_{takeoff}$ = 100, would successfully deliver a parcel of minimum mass $m_{\min}$ for the minimum delivery distance $d_{\min}$; (ii) $([d_{\max}, m_{\max}, 0.0], 0)$, suggesting that a completely fully discharged UAV, that is, $SoC_{takeoff}$ = 0, cannot deliver a parcel of maximum mass $m_{\max}$ for the maximum delivery distance $d_{\max}$.

Multi-Agent Simulations

To enable a comprehensive performance evaluation of strategies in long-term deployment scenarios—with up to ca. 2.4 million auctions over 8 weeks of simulated time per trial, we developed an ultra-fast Python-based multi-agent simulator, the source code of which is available online at [61]. A screenshot of the simulator is shown in Figure 15. To implement the learning algorithm, the widely-used scikit-learn library [62] is used.

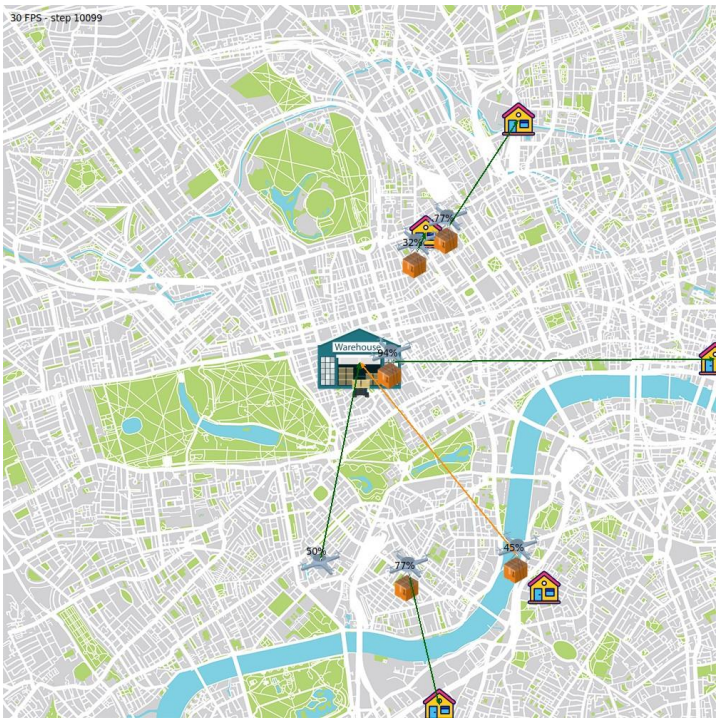


Figure 15 Screenshot from the purpose-built simulator showing the fulfilment centre and six drones, five of which are attempting a delivery, whereas the sixth is returning following delivery of a parcel.

UAV Energy Models

The developed multi-agent simulator incorporates realistic UAV energy models for both battery charging and consumption.

Charging Model:

The charging model is based on the following assumptions: (i) The charging rate decreases as the battery approaches maximum capacity due to increasing internal resistance. (ii) The charging rate is identical regardless of the battery's SoH. Although batteries with a lower SoH have reduced capacity, they are known to charge less efficiently [63] [64] [65].

When charging, the UAV's state of charge, $SoC(t) \in [0, 100]$, increases at the following rate:

$$\lambda_{\text{charge}}(t) = \frac{100 - SoC(t)}{3600} \frac{\gamma_c P_c}{C_{\text{theoritical}}}$$

(13)

where P_c represents the charger's power (in W) and γ_c denotes the efficiency. $C_{\text{theoretical}}$ refers to the UAV's theoretical battery capacity (in Wh).

Consumption Model:

We use the model proposed in [66] to compute the energy consumption (in $\frac{W}{s}$) per second for a UAV flying with a payload mass m_p at a constant speed v_a:

$$C_{ps}(m_p) = \frac{[g\,(m_f + m_b + m_p)]^{\frac{3}{2}}}{3600\sqrt{2\,n_r \rho \zeta}}$$

(14)

where m_f and m_b are the UAV frame and battery masses, respectively. n_r is the number of rotors on the UAV, ζ is the area of the spinning blade disc of one rotor, g is the gravitational constant (in m/s^2), and ρ is the air density at $15°C$.

When delivering, the UAV's state of charge, $SoC(t) \in [0, 100]$, decreases at the following rate:

$$\lambda_{\text{delivery}}(m_p) = -100\frac{C_{ps}(m_p)}{C_{\text{theoretical}} \cdot SoH}$$

(15)

Orders Arrival and Processing

In our simulations, the FC is responsible for receiving orders, preparing parcels, and announcing the corresponding delivery tasks to available UAVs.

Order Arrival:

Orders arrive stochastically, with an expected inter-arrival time $\bar{\tau}$. The arrival time of the next order is randomly drawn from an exponential distribution:

$$t_{j+1} = t_j + Z_j \qquad \text{where } Z_j \sim Exp\left(\frac{1}{\bar{\tau}}\right) \text{ with } t_1 = 0$$

(16)

where t_{j+1} denotes the arrival time of the next order, and t_j is the arrival time of the previous order.

Order Processing:

Orders arriving at the FC are stored in a queue which sorts them by their arrival times from the earliest to the latest placed orders. The advertising of tasks, extracted from these orders, starts with the first order in the queue. It adheres to the following rules:

- **(i)** for as long as no UAV resides at the FC, task advertisement is suspended;
- **(ii)** if a task is advertised but no UAV places a bid, the FC proceeds with the next unallocated order in the queue, if any;
- **(iii)** if a task is advertised, and at least one UAV places a bid, the order is considered allocated; the FC continues with the earliest unallocated order in the queue (i.e., considering again the front of the queue);
- **(iv)** after an aborted delivery attempt, the corresponding order is considered unallocated again; it remains in the queue according to the original arrival time of the order.

Results

This section presents a comprehensive performance evaluation of the proposed learning-based deployment strategy. **Table 2** provides an overview of the simulation parameters.

Table 2 Simulation variables

Variables	Denotation	Description	Value(s)	Unit
UAV	m_f	Frame mass	10	Kg
	n_r	Number of rotors	8	—
	ζ	Area of the spinning blade disc of one rotor	0.27	m^2

	v_a	Nominal speed	10	m/s
Battery	m_b	Mass	10	kg
	$C_{\text{theoritical}}$	Theoretical capacity	800	Wh
	SoH	State of Health	$\mathcal{U}(0.5, 1.0)$	—
Charger	P_c	Power	100	W
	γ_c	Efficiency	95%	—
Swarm	S	Swarm size	25	—
Orders	d_j	Delivery distance	$\mathcal{U}(1000,6000)$	km
	m_j	Parcel mass	$\mathcal{U}(0.5,5.0)$	kg
	$\bar{\tau}$	Inter-arrival time	$\{15,20,25,30,35,40\}$	min
	T	Operation period	8	weeks
Others	g	Gravitational acceleration constant	9.81	m/s^2
	ρ	Air density at $15°C$	1.225	kg/m^3

Unless otherwise stated, we simulate fleets of $\boldsymbol{S = 25}$ UAVs over an operational period of $\boldsymbol{T = 8}$ weeks.

We consider order inter-arrival times of $\bar{\boldsymbol{\tau}} \in \{\mathbf{15, 20, 25, 30, 35, 40}\}$ minutes. The order parameters are chosen at random using uniform distributions, $\boldsymbol{d_j \sim \mathcal{U}(1000, 6000)}$ metres and $\boldsymbol{m_j \sim \mathcal{U}(0.5, 5.0)}$ kilograms. If meeting the aforementioned conditions, orders are advertised one at a time, every $\boldsymbol{2s}$. **Table 2** provides as well the parameters used to simulate the UAV, including its battery, and the charger. For each UAV, the battery health is chosen randomly using a uniform distribution, $\mathrm{SoH} \sim \boldsymbol{\mathcal{U}(0.5, 1.0)}$. Assuming $\mathrm{SoH} = \mathbf{1}$ and using the energy models described earlier, the UAV can fly for 28.75 min while transporting a parcel of maximum weight. It requires 505 min to fully charge its battery once entirely depleted. Hence, its maximum duty is 5.4%, which is similar to the duty cycle of the commercial delivery drone DJI FlyCart 30 (6.97% with the single-battery configuration).

UAVs are set to abort delivery attempts when having consumed $\boldsymbol{\xi = 0.5}$ of their takeoff state of charge.

For the learning algorithm, we set both the regularisation strength parameter $\boldsymbol{\alpha}$ and the initial learning rate $\boldsymbol{\eta}$ to 0.01. Given that delivery distance $\boldsymbol{d_j}$ is sampled from $\boldsymbol{\mathcal{U}(1000, 6000)}$, parcel mass $\boldsymbol{m_j}$ is drawn from $\boldsymbol{\mathcal{U}(0.5, 5.0)}$, and robots can have a SoC ranging from 0 to 100, we set the mean vector and standard deviation vector in Equation to:

$$\boldsymbol{\mu} = (3500, 2.75, 50) \qquad \boldsymbol{\sigma} = (1443.38, 1.298, 28.87)$$

Each experimental condition is repeated over 20 random runs, leveraging an Intel Xeon Platinum 8358 CPU core in a High-Performance Computing (HPC) cluster running Python 3.10.12.

Effect of Winner Selection Rule

In Figure 16, we evaluate our learning-based deployment strategy under three winner selection rules: *Least confident* (green), *Most confident* (pink), and *Random* (orange). *Random* effectively chooses each bidder with equal probability[5].

[5] To achieve this distribution in a decentralised way, each bidder randomly samples their bid value, and once all bids have been broadcast, the highest bidder wins.

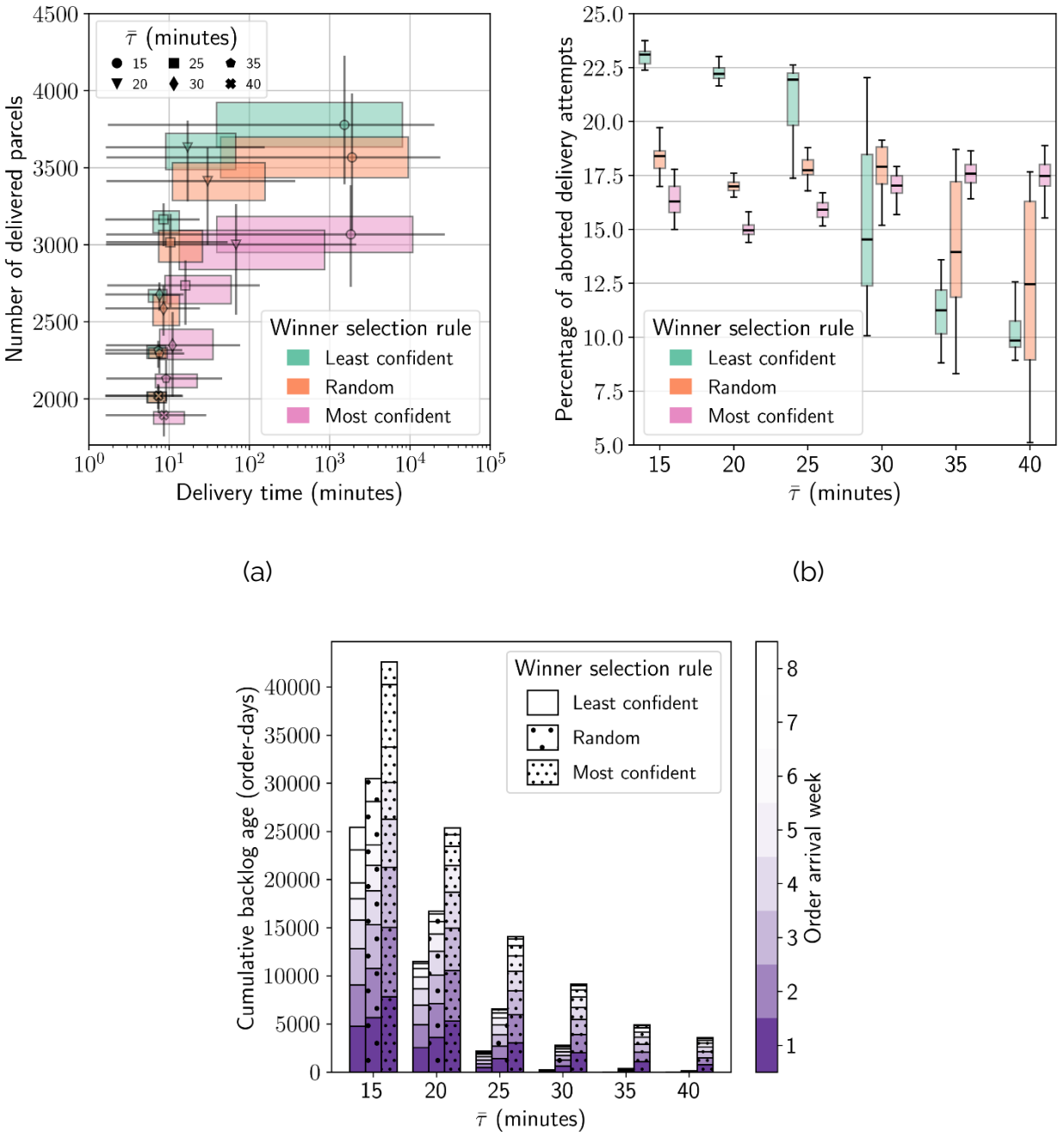


Figure 16 Performance of the learning-based deployment strategy for different winner selection rules: *Least confident* (green), *Most confident* (pink), and *Random* (orange). Metrics used are (a) number of delivered parcels and delivery time, (b) percentage of aborted delivery attempts, and (c) cumulative backlog age (segmented by order arrival weeks.

Figure 16a compares the winner selection rules in terms of delivery time (x-axis) and number of delivered parcels (y-axis) for different inter-arrival intervals. The distributions (over 20 runs) are visualised using a 2D box plot for each combination of winner selection rule and inter-arrival interval. Counter-intuitively, the *Least confident* rule outperforms the other two, consistently delivering more parcels, and in shorter times across all inter-arrival intervals. This can possibly be attributed to a more demand-oriented workload allocation where less capable UAVs end up fulfilling simpler tasks, preserving the more capable UAVs for more demanding tasks that may arise in the future. Moreover, when the least confident bidder is employed, it is possible that the UAVs learn to set their confidence levels conservatively. The *Most confident* rule leads to fewer deliveries and longer delivery times, whereas the *Random* strategy falls between the two.

Figure 16b illustrates the percentage of aborted delivery attempts (y-axis) for various expected inter-arrival times $\bar{\tau}$ (x-axis) across the three winner selection rules. For shorter inter-arrival times ($\bar{\tau} \leq 25$ minutes), the *Least confident* rule results in a higher percentage of aborted deliveries. This could be because less confident UAVs are more likely to fail near their operational limits. For longer inter-arrival times ($\bar{\tau} > 25$), the *Least confident* rule consistently exhibits the fewest aborted attempts of the three rules. The performance of the *Random* rule notably improves as well. By contrast, the percentage of aborted delivery attempts does not decrease for the *Most confident* rule as the inter-arrival time increases.

Figure 16c shows a bar plot of the cumulative backlog age at the end of the 8-week operation period for different inter-arrival times ($\bar{\tau}$) and the different winner selection rules. Each bar further breaks down the orders by their arrival week. As $\bar{\tau}$ increases from 15 to 40 minutes, all selection rules show a reduction in the cumulative backlog age, suggesting that less frequent order arrivals lead to lower backlogs. The *Least confident* rule consistently yields the lowest cumulative backlog age (for all inter-arrival times). This indicates better queue management and backlog reduction, especially as the inter-arrival time increases, being able to process almost all orders for $\bar{\tau} \geq 30$ By contrast, the *Most confident* rule results in the highest cumulative backlog age, implying a less

efficient resource use. Note that depending on the inter-arrival time, 252-672 orders are expected to arrive per week.

Analysis of Decision Accuracies

We evaluate the decision-making accuracy of the learned bidding policies at different stages (i.e., at the end of each week) of the eight-week operation period. All three selection rules are considered. For each rule, 500 UAVs are evaluated (i.e., 25 UAVs per run, and 20 runs in total). The inter-arrival time is $\bar{\tau} = 15$ minutes.

We computed the decisions made by the bidding policies (i.e., whether to bid or not) using 1000 randomly generated input feature vectors $\boldsymbol{x}^T = [d, m, SoC]$, where $d \sim \mathcal{U}(1000, 6000)$, $m \sim \mathcal{U}(0.5, 5.0)$, and $SoC \sim \mathcal{U}(0, 100)$. We then assessed the correctness of these decisions by comparing them to the ground truth. The ground truth uses the UAVs' true battery capacity and the simulated energy consumption model (both of which are unknown to the UAV) to determine whether the UAV can complete the task. A decision is considered correct only if the UAV chooses to bid for a task it is capable of completing, or if it chooses not to bid for a task it is incapable of completing.

Figure 17 displays the mean decision accuracy (y-axis) over time (x-axis, in weeks) for three winner selection strategies: *Least confident* (green), *Most confident* (pink), and *Random* (orange). The shaded areas around the line plots represent the 95% confidence intervals for the mean decision accuracy. The *Least confident* rule exhibited significantly better decision accuracy, reaching about 97% after eight weeks. This could be attributed to these UAVs tending to push their capabilities more frequently, leading to more refined decision-making over time. In contrast, the *Most confident* rule showed slower progress, remaining below 85% after eight weeks. Many of these UAVs have fewer opportunities to learn about their limits.

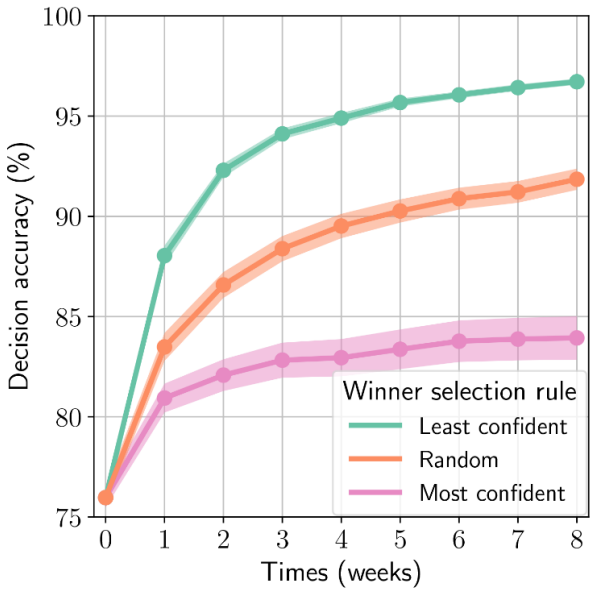


Figure 17 Decision accuracy over time for the three winner selection strategies: *Least confident* (green), *Most confident* (pink), and *Random* (orange).

We examine the extent to which a UAV's state of health (indirectly) impacts its bidding policy. Recall that the UAV is unaware of its state of health. We test the bidding policies that are obtained at the end of the 8-week operation period for UAVs that used the *Least confident* winner selection rule during learning. We then record the bid value for every UAV, assuming a $SoC \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$, a delivery distance $d_j \in \{1000, 1050, 1100, \dots, 4900, 4950, 5000\}$ metres and mass $m_j \in \{0.5, 0.55, 0.6, \dots, 4.9, 4.95, 5\}$ kilograms. For each combination of SoC, distance and mass, we record the SoH of the winning UAV (if any bids are placed). The reported values are the average SoH across 20 runs.

The results are depicted in Figure 18. Each subplot corresponds to a specific SoC. The colour gradient, from blue to red, represents the average SoH of the winning UAVs (the colour being white if no bids were placed). When SoC is 100%, the SoH of the deployed UAVs ranges from 0.5 to around 0.7, with more difficult tasks (longer distances and heavier parcels) corresponding to higher SoH values. This suggests that the *Least confident* strategy prioritises deploying the least capable UAVs that can still complete the task. As SoC decreases, the range of SoH values expands. For instance, when SoC is 60%, the SoH of the deployed UAVs ranges from 0.5 to 1.0, with more difficult tasks

leading to the deployment of UAVs with higher SoH. However, as SoC continues to drop, we observe that UAVs are no longer deployed for harder tasks (upper right region of the heatmaps). This occurs because, at low SoC levels, even the most capable UAVs (those with high SoH) are unable to handle these challenging deliveries. When SoC reaches very low levels (e.g., 10%), UAVs are only deployed for the simplest tasks, and the SoH values are restricted to higher ranges (above 0.8). At such low energy levels, only the most capable UAVs are selected for the remaining simpler tasks, ensuring that the system operates within the UAVs' remaining capacity.

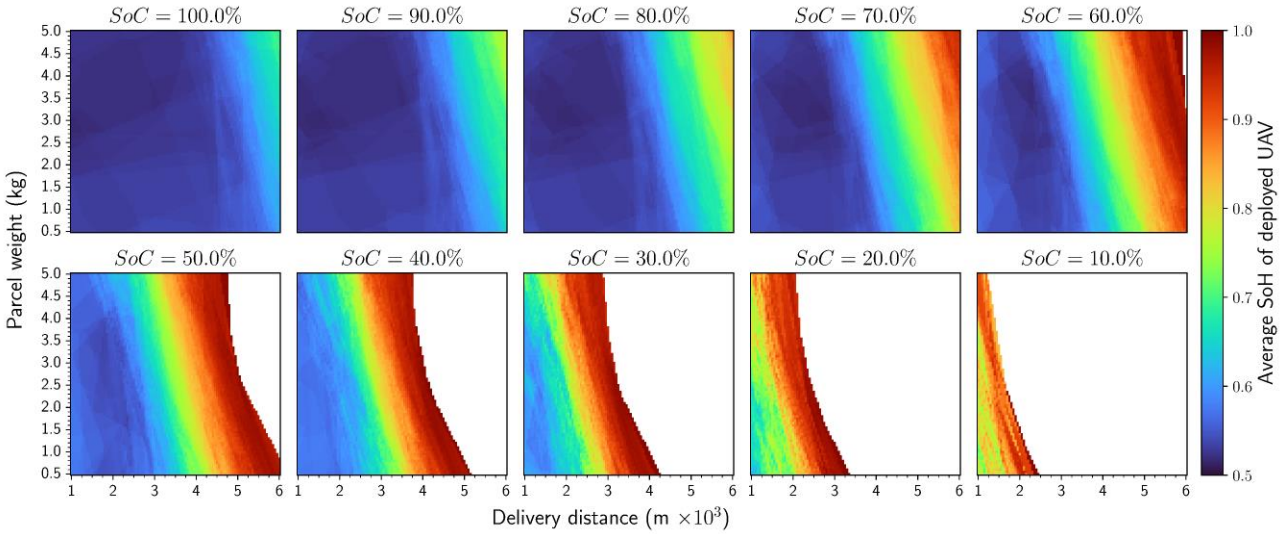


Figure 18 SoH of UAVs deployed by the *Least confident* winner selection rule for various tasks and SoC values.

Comparison Against Threshold-based Deployment Strategy

In the following, the learning-based deployment strategy is only considered in conjunction with the *Least confident* winner selection rule. We benchmark this strategy against a traditional *threshold-based* deployment strategy. In this baseline approach, UAVs are only deployed if their SoC exceeds a predefined threshold. This strategy operates using the same auction-based mechanism for task allocation as the learning-based approach (see methodology), but UAVs bid only when their SoC is above the threshold, and their bid value corresponds to the current SoC. We tested this threshold-based strategy with SoC thresholds of $\{50, 60, 70, 80, 90, 100\}\%$ and found that the threshold-based strategy achieved the highest number of deliveries and, on average,

the shortest delivery times when the UAVs deployed a SoC threshold of 80%. Note that the learning-based approach starts without such prior tuning, as the UAVs need to progressively refine their bidding strategies.

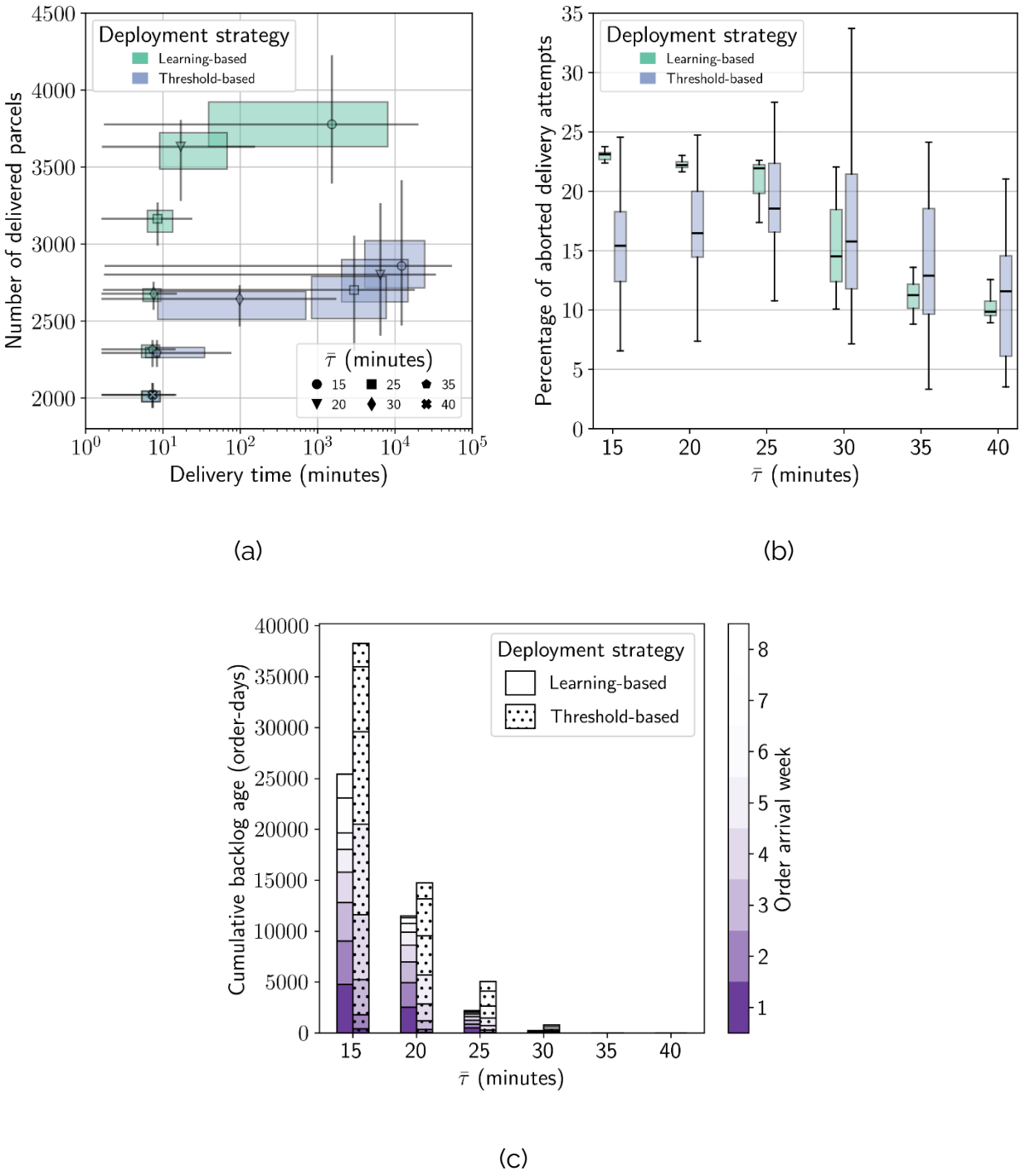


Figure 19 Comparing the deployment learning-based strategy against a threshold-based strategy. (a) The learning-based deployment strategy outperforms the

baseline approach in terms of the number of delivered parcels and delivery time. (b) It results in a higher number of failed delivery attempts, (c) but produces less backlog compared to the baseline approach.

Figure 19a presents the results in terms of the number of delivered parcels (y-axis) and delivery time (x-axis), across various inter-arrival times ($\bar{\tau}$) from 15 to 40 min. At the lowest expected inter-arrival time of 40 min, both strategies perform equally well regarding both criteria. As the expected inter-arrival time increases, both strategies manage to deliver more parcels, but the performance gap widens. The *Learning-based* strategy consistently outperforms the *Threshold-based* strategy in both delivery volume and delivery time. For example, at $\bar{\tau} = 20$ minutes, the *Learning-based* strategy delivers a median of approximately 3600 parcels, with a median delivery time of 17 minutes, whereas the *Threshold-based* strategy delivers approximately 2800 parcels, with a median delivery time exceeding 6500 minutes. This stark difference is may be attributed to the adaptive nature of the *Learning-based* approach, which allows UAVs to bid and be deployed at any SoC level, as long as considered capable of completing the given task, while the *Threshold-based* strategy waits for UAVs to reach the required SoC level, regardless of the task difficulty.

Figure 19b compares the two strategies in terms of aborted delivery attempts. The *Learning-based* strategy exhibits a higher number of aborted delivery attempts than the *Threshold-based* strategy. This is because the *Learning-based* strategy allows UAVs to attempt deliveries at lower SoC levels. However, despite the higher number of aborted attempts, the strategy achieves lower delivery times than the *Threshold-based* strategy.

In Figure 19c, we observe that the *Learning-based* strategy consistently results in a lower cumulative backlog age compared to the *Threshold-based* strategy but it is less good at prioritising orders from earlier weeks. This could be because the *Learning-based* strategy's responsiveness encourages UAVs to be deployed as soon they consider themselves capable of fulfilling the order, leading to faster task completion and less accumulation of pending orders. On the other hand, the *Threshold-based* strategy leads to delayed deployments, which exacerbates task backlog, especially when tasks arrive in rapid succession.

In summary, the *Learning-based* strategy not only improves delivery throughput, but also reduces delivery time and backlog, despite having a higher rate of aborted attempts compared to the *Threshold-based* strategy.

Forecasting Using Learned Bidding Policies

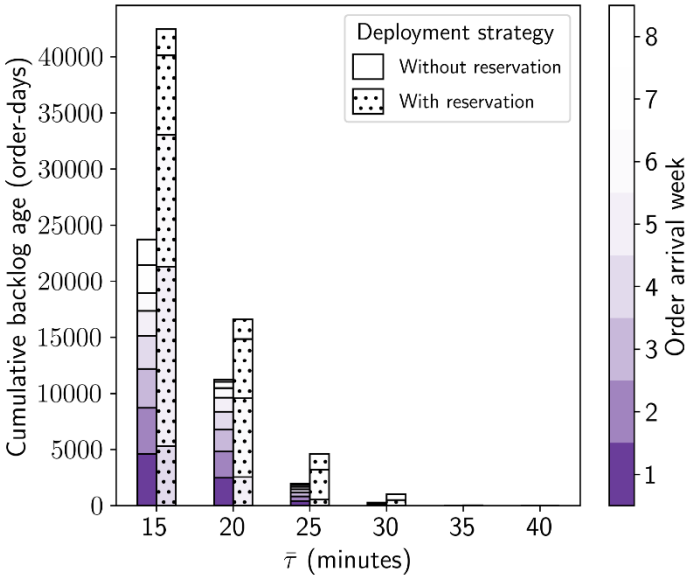


Figure 20 Pending orders after eight weeks with and without reservations.

A potential enhancement we explored is enabling UAVs with insufficient SoC to commit to fulfilling current delivery tasks at a later time, once they have accumulated enough charge. UAVs that choose not to bid for immediate delivery can submit a *reservation bid*. Each UAV uses its learned decision function $f(\boldsymbol{x})$ and the expected charging rate to determine whether to place a reservation bid and to calculate the appropriate bid value. A reservation bid is made if the UAV forecasts that it will be able to perform the task in the future. The bid value represents the forecasted time required to reach the sufficient charge and begin addressing the task. A UAV wins the reservation auction if no other UAV bids to address the task immediately, and its bid value is the lowest.

We evaluated this forecasting variant in simulation over an 8-week operational period. To assess the effect of forecasting, we used UAVs that had previously learned and tuned their decision functions over an earlier 8-week period. Figure 20 illustrates the impact of forecasting on task backlog. The results show a significant reduction in the

number of pending orders from earlier weeks when forecasting is enabled compared to when it is not. However, more backlogs will be created for later weeks. By allowing UAVs to commit to future tasks, the forecasting mechanism helps prioritise earlier orders, but increases backlog.

Summary

In this study, we proposed a decentralised learning-based deployment strategy for UAV-based on-demand delivery systems. Our strategy enables UAVs to make real-time bidding decisions for delivery tasks based on their current state of charge, delivery distance, and parcel mass, without requiring knowledge of their energy consumption models or battery health. The UAVs continuously refine their bidding policies through online learning, improving decision accuracy and task performance over time.

Through an extensive set of simulations, we demonstrated that the proposed strategy outperforms a traditional threshold-based approach, which only deploys UAVs when their state of charge exceeds a predefined amount. The learning-based approach consistently delivered more parcels and in shorter times, proving particularly effective in scenarios with frequent task arrivals. Our analysis revealed that deploying the least confident bidders led to more efficient workload distribution and better overall delivery performance. Our learning strategy made it possible for UAVs to adapt their decision-making to their individual energy constraints, including the state of charge and state of health, in addition to being responsive to the task parameters. The strategy was subsequently extended to enable forecasting via the use of the learned capabilities models. This enabled UAVs with sufficient charge levels to commit to fulfilling orders at specific future times. This also highlights the flexibility of our approach in handling real-world delivery challenges, such as fluctuating demands and task prioritisation.

Future work will focus on incorporating additional environmental factors, such as wind direction, temperature, and terrain characteristics, into the decision-making process. These factors significantly impact UAV energy consumption and operational feasibility. To accommodate these complexities, more sophisticated policy architectures, such as deep neural networks, may be required.

Conclusion

This deliverable focused on the development and testing of virtual machines (VMs) for swarm controllers, enabling these controllers to execute across multiple platforms while incorporating both formal coordination mechanisms and online learning strategies. Through this work, we successfully developed VMs capable of running compiled swarm controllers on a variety of platforms, ensuring seamless cross-platform compatibility.

The virtual machines provide a robust framework that minimises the manual effort needed for platform-specific implementations, reducing the likelihood of bugs during the deployment phase. This framework has proven its flexibility by enabling the deployment of identical controllers across physical and simulated platforms while maintaining consistent behaviour across different platforms, such as Kilobot, e-puck, and CapBot.

Moreover, the integration of online learning strategies – in the form of decentralised reinforcement learning, enhances the adaptability of the swarm controllers. In particular, it enabled robots to optimise their behaviour in real-time according to their own, a priori unknown hardware capabilities, thus improving both the swarm's energy efficiency and task performance in dynamic environments.

Two main areas were explored in the deliverable: energy management in long-term swarm operations and task allocation for UAVs in on-demand delivery scenarios. In the first case, we compared strategies where robots used either fixed or mobile charging stations, revealing conditions in which mobile chargers enhance energy efficiency and work output. In the second case, a heterogeneous fleet of UAVs autonomously learned to bid for delivery tasks based on their unknown, individual energy constraints, showcasing the potential of the learning-based approaches to leverage unknown individual differences in the heterogeneous fleets.

Our results demonstrate that the developed VMs, in conjunction with learning-based swarm controllers, provide a powerful solution for executing swarm robotics tasks across diverse platforms, making it possible to ensure formal specifications provided at design time are fulfilled while optimising swarm behaviour for long-term autonomous operations. Future directions of this work include incorporating more complex policies and optimising learning speeds to further enhance the capabilities of swarms in more challenging and diverse environments.

Bibliography

[1] M. Rubenstein, C. Ahler, N. Hoff, A. Cabrera and R. Nagpal, "Kilobot: A low cost robot with scalable operations designed for collective behaviors," *Robotics and Autonomous Systems,* vol. 62, no. 7, pp. 966--975, 2014.

[2] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano and A. Martinoli, "The e-puck, a robot designed for education in engineering," in *Proceedings of the 9th conference on autonomous robot systems and competitions*, 2009, pp. 59--65.

[3] "EPFL educational and research mini mobile robot VERSION 2," gctronic, [Online]. Available: https://www.gctronic.com/e-puck2.php. [Accessed 23 10 2024].

[4] A. G. Millard, R. Joyce, J. A. Hilder, C. Fleşeriu, L. Newbrook, W. Li, L. J. McDaid and D. M. Halliday, "The Pi-puck extension board: A raspberry Pi interface for the e-puck robot platform," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 741-748.

[5] M. Liu, F. Yang, S. Michiels, T. Van Eyck, D. Hughes, S. Alvarado-Marin, F. Maksimovic and T. Watteyne, "Demo Abstract: FreeBot, a Battery-Free Swarm Robotics Platform," in *The 21st ACM Conference on Embedded Networked Sensor Systems (SenSys '23), November 12–17, 2023, Istanbul, Turkiye*, 2023.

[6] A. Reina, A. J. Cope, E. Nikolaidis, J. A. R. Marshall and C. Sabo, "ARK: Augmented Reality for Kilobots," *IEEE Robotics and Automation Letters,* vol. 2, no. 3, pp. 1755-1761, 2017.

[7] A. G. Millard, R. Redpath, A. M. Jewers, C. Arndt, R. Joyce, J. A. Hilder, L. J. McDaid and D. M. Halliday, "ARDebug: an augmented reality tool for analysing and debugging swarm robotic systems," *Frontiers in Robotics and AI,* vol. 5, p. 87, 2018.

[8] N. Bredeche and N. Fontbonne, "Social learning in swarm robotics," *Philosophical Transactions of the Royal Society B: Biological Sciences,* vol. 377, p. 20200309, 2022.

[9] M. Dorigo, V. Trianni, E. Şahin, R. Groß, T. H. Labella, G. Baldassarre, S. Nolfi, J.-L. Deneubourg, F. Mondada, D. Floreano and L. M. Gambardella, "Evolving Self-Organizing Behaviors for a Swarm-bot," *Autonomous Robots,* vol. 17, p. 223–245, 2004.

[10] M. Otte, "An emergent group mind across a swarm of robots: Collective cognition and distributed sensing via a shared wireless neural network," *The International Journal of Robotics Research,* vol. 37, p. 1017–1061, 2018.

[11] M.-A. Blais and M. A. Akhloufi, "Scalable and cohesive swarm control based on reinforcement learning," *Cognitive Robotics,* vol. 4, pp. 88-103, 2024.

[12] M. Bettini, A. Prorok and V. Moens, "BenchMARL: Benchmarking Multi-Agent Reinforcement Learning," *Journal of Machine Learning Research,* vol. 25, p. 1–10, 2024.

[13] W. Li, M. Gauci and R. Groß, "Turing Learning: A metric-free approach to inferring behavior and its application to swarms," *Swarm Intelligence,* vol. 10, p. 211–243, 2016.

[14] M. Y. Ben Zion, J. Fersula, N. Bredeche and O. Dauchot, "Morphological computation and decentralized learning in a swarm of sterically interacting robots," *Science Robotics,* vol. 8, p. eabo6140, 2023.

[15] D. Tarapore, A. L. Christensen and J. Timmis, "Generic, scalable and decentralized fault detection for robot swarms," *PloS one,* vol. 12, p. e0182058, 2017.

[16] Y. Wei, X. Nie, M. Hiraga, K. Ohkura and Z. Car, "Developing End-to-End Control Policies for Robotic Swarms Using Deep Q-learning," *Journal of Advanced Computational Intelligence and Intelligent Informatics,* vol. 23, pp. 920-927, 2019.

[17] T. Yasuda and K. Ohkura, "Sharing experience for behavior generation of real swarm robot systems using deep reinforcement learning," *Journal of Robotics and Mechatronics,* vol. 31, p. 520–525, 2019.

[18] A. K. Sadhu and A. Konar, "Improving the speed of convergence of multi-agent q-learning for cooperative task-planning by a robot-team," *Robotics and Autonomous Systems,* vol. 92, p. 66–80, 2017.

[19] J. Thangavelautham, K. Law, T. Fu, N. A. El Samid, A. D. S. Smith and G. M. T. D'Eleuterio, "Autonomous multirobot excavation for lunar applications," *Robotica,* vol. 35, p. 2330–2362, 2017.

[20] W. Li, M. Gauci and R. Groß, "A Coevolutionary Approach to Learn Animal Behavior Through Controlled Interaction," in *Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2013)*, 2013.

[21] W. Li, M. Gauci and R. Groß, "Coevolutionary learning of swarm behaviors without metrics," in *Proc. of the Genetic and Evolutionary Computation Conference (GECCO 2014)*, 2014.

[22] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence and K. Q. Weinberger, Eds., Curran Associates, Inc., 2014, p. 2672–2680.

[23] R. Groß, Y. Gu, W. Li and M. Gauci, "Generalizing GANs: A Turing Perspective," in *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., 2017.

[24] J. V. A. Marques, M.-T. Lorente and R. Groß, "Multi-Robot Systems Research: A Data-Driven Trend Analysis," in *Distributed Autonomous Robotic Systems*, Cham, 2022.

[25] S. Pearson, T. C. Camacho-Villa, R. Valluru, O. Gaju, M. C. Rai, I. Gould, S. Brewer and E. Sklar, "Robotics and autonomous systems for net zero agriculture," *Current Robotics Reports,* vol. 3, p. 57–64, 2022.

[26] L. Xie, Y. Shi, Y. T. Hou and A. Lou, "Wireless power transfer and applications to sensor networks," *IEEE Wireless Communications,* vol. 20, p. 140–145, 2013.

[27] K. Chour, J.-P. Reddinger, J. Dotterweich, M. Childers, J. Humann, S. Rathinam and S. Darbha, "An agent-based modeling framework for the multi-UAV rendezvous recharging problem," *Robotics and Autonomous Systems,* vol. 166, p. 104442, 2023.

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

[29] M. Rappaport and C. Bettstetter, "Coordinated recharging of mobile robots during exploration," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 2017.

[30] Y. Warsame, S. Edelkamp and E. Plaku, "Energy-aware multi-goal motion planning guided by monte carlo search," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, 2020.

[31] K. Yu, A. K. Budhiraja, S. Buebel and P. Tokekar, "Algorithms and experiments on routing of unmanned aerial vehicles with mobile recharging stations," *Journal of Field Robotics,* vol. 36, p. 602–616, 2019.

[32] F. Vaussard, P. Rétornaz, S. Roelofsen, M. Bonani, F. Rey and F. Mondada, "Towards Long-Term Collective Experiments," in *Intelligent Autonomous Systems 12*, Berlin, 2013.

[33] G. Li, I. Svogor and G. Beltrame, "Long-Term Pattern Formation and Maintenance for Battery-Powered Robots," *Swarm Intelligence,* vol. 13, p. 21–57, 2019.

[34] M. Kubo and C. Melhuish, "Robot trophallaxis: Managing energy autonomy in multiple robots," *Proceedings of Towards Autonomous Robotic Systems,* p. 77–84, 2004.

[35] C. Melhuish and M. Kubo, "Collective energy distribution: Maintaining the energy balance in distributed autonomous robots using trophallaxis," in *Distributed Autonomous Robotic Systems 6*, Springer, 2007, p. 275–284.

[36] C. Moonjaita, H. Philamore and F. Matsuno, "Trophallaxis with Predetermined Energy Threshold for Enhanced Performance in Swarms of Scavenger Robots," *Artificial Life and Robotics,* vol. 23, p. 609–617, 2018.

[37] A. F. T. Winfield, S. Kernbach and T. Schmickl, "Collective Foraging: Cleaning, Energy Harvesting, and Trophallaxis," in *Handbook of Collective Robotics*, Jenny Stanford Publishing, 2013.

[38] J. J. Kreider, T. Janzen, A. Bernadou, D. Elsner, B. H. Kramer and F. J. Weissing, "Resource sharing is sufficient for the emergence of division of labour," *Nature Communications,* vol. 13, p. 7232, 2022.

[39] G. Pini, A. Brutschy, M. Birattari and M. Dorigo, "Interference Reduction through Task Partitioning in a Robotic Swarm," in *Sixth International Conference on Informatics in Control, Automation and Robotics–ICINCO*, 2009.

[40] H. Schioler and T. D. Ngo, "Trophallaxis in robotic swarms-beyond energy autonomy," in *2008 10th International Conference on Control, Automation, Robotics and Vision*, 2008.

[41] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella and M. Dorigo, "ARGoS: A Modular, Parallel, Multi-Engine Simulator for Multi-Robot Systems," *Swarm Intelligence,* vol. 6, p. 271–295, 2012.

[42] G. Miyauchi, M. S. Talamali and R. Groß, "Online supplementary material," 2024. [Online]. Available: https://doi.org/10.15131/shef.data.25561923.

[43] G. Miyauchi, M. S. Talamali and R. Groß, *Robot controller source code.,* 2024.

[44] N. Agatz, P. Bouman and M. Schmidt, "Optimization approaches for the traveling salesman problem with drone," *Transportation Science,* vol. 52, p. 965–981, 2018.

[45] P. Kitjacharoenchai, M. Ventresca, M. Moshref-Javadi, S. Lee, J. M. A. Tanchoco and P. A. Brunese, "Multiple traveling salesman problem with drones: Mathematical model and heuristic approach," *Computers & Industrial Engineering,* vol. 129, p. 14–30, 2019.

[46] D. Schermer, M. Moeini and O. Wendt, "A hybrid VNS/Tabu search algorithm for solving the vehicle routing problem with drones and en route operations," *Computers & Operations Research,* vol. 109, p. 134–158, 2019.

[47] B. D. Song, K. Park and J. Kim, "Persistent UAV delivery logistics: MILP formulation and efficient heuristic," *Computers & Industrial Engineering,* vol. 120, p. 418–428, 2018.

[48] P. Grippa, D. A. Behrens, F. Wall and C. Bettstetter, "Drone delivery systems: Job assignment and dimensioning," *Autonomous Robots,* vol. 43, p. 261–274, 2019.

[49] M. Rinaldi, S. Primatesta, G. Guglieri and A. Rizzo, "Auction-based Task Allocation for Safe and Energy Efficient UAS Parcel Transportation," *Transportation Research Procedia,* vol. 65, pp. 60-69, 2022.

[50] H. Chen, Z. Hu and S. Solak, "Improved delivery policies for future drone-based delivery systems," *European Journal of Operational Research,* vol. 294, p. 1181–1201, 2021.

[51] H. Huang, C. Hu, J. Zhu, M. Wu and R. Malekian, "Stochastic task scheduling in UAV-based intelligent on-demand meal delivery system," *IEEE Transactions on Intelligent Transportation Systems,* vol. 23, p. 13040–13054, 2021.

[52] Y. Liu, "An optimization-driven dynamic vehicle routing algorithm for on-demand meal delivery using drones," *Computers & Operations Research,* vol. 111, p. 1–20, 2019.

[53] J. Zhang, J. F. Campbell, D. C. Sweeney II and A. C. Hupman, "Energy consumption models for delivery drones: A comparison and assessment," *Transportation Research Part D: Transport and Environment,* vol. 90, p. 102668, 2021.

[54] Y. K. Lopes, S. M. Trenkwalder, A. B. Leal, T. J. Dodd and R. Groß, "Supervisory Control Theory Applied to Swarm Robotics," *Swarm Intelligence,* vol. 10, p. 65–97, 2016.

[55] G.-X. Yuan, C.-H. Ho and C.-J. Lin, "Recent Advances of Large-Scale Linear Classification," *Proceedings of the IEEE,* vol. 100, pp. 2584-2603, 2012.

[56] J. Pang, L. Qin, C. Zhang, W. Zhang, Q. Huang and B. Yin, "Local Laplacian Coding From Theoretical Analysis of Local Coding Schemes for Locally Linear Classification," *IEEE Transactions on Cybernetics,* vol. 45, pp. 2937-2947, 2015.

[57] N. Ketkar, "Stochastic Gradient Descent," in *Deep Learning with Python: A Hands-on Introduction*, Berkeley, CA: Apress, 2017, p. 113–132.

[58] S. Kim and W. J. Wilbur, "Classifying protein-protein interaction articles using word and syntactic features," *BMC Bioinformatics,* vol. 12, p. S9, 2011.

[59] Z. Guo, A. Min, B. Yang, J. Chen and H. Li, "A modified huber nonnegative matrix factorization algorithm for hyperspectral unmixing," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing,* vol. 14, p. 5559–5571, 2021.

[60] L. Bottou, "Stochastic Gradient Descent Tricks," in *Neural Networks: Tricks of the Trade: Second Edition*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, p. 421–436.

[61] M. S. Talamali, "Energy-aware drone delivery multi-agent simulation," 21 10 2024. [Online]. Available: https://github.com/mstalamali/EA_Drone_Delivery. [Accessed 21 10 2024].

[62] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, p. 2825–2830, 2011.

[63] X. Gong, R. Xiong and C. Mi, "Study of the characteristics of battery packs in electric vehicles with parallel-connected lithium-ion battery cells," *IEEE Transactions on Industry Applications,* vol. 51, no. 2, pp. 1872-1879, 2015.

[64] J. Lee, J. Kim, J. Yi and C. Won, "Battery management system algorithm for energy storage systems considering battery efficiency," *Electronics,* vol. 10, no. 15, p. 1859, 2021.

[65] S. Arof, P. Mawby, H. Arof and E. Noorsal, "Low harmonics plug-in home charging electric vehicle battery charger bms state of charge balancing using priority and sequencing technique," *IOP Conference Series: Earth and Environmental Science,* vol. 1261, no. 1, p. 012032, 2023.

[66] K. Dorling, J. Heinrichs, G. G. Messier and S. Magierowski, "Vehicle routing problems for drone delivery," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 47, p. 70–85, 2016.

[67] P. J. Ramadge and W. M. Wonham, "Supervisory Control of a Class of Discrete Event Processes," *SIAM Journal on Control and Optimization,* vol. 25, p. 206–230, 1987.

[68] S. Poikonen and B. Golden, "Multi-visit drone routing problem," *Computers & Operations Research,* vol. 113, p. 104802, 2020.

[69] K. Peng, J. Du, F. Lu, Q. Sun, Y. Dong, P. Zhou and M. Hu, "A hybrid genetic algorithm on routing and scheduling for vehicle-assisted multi-drone parcel delivery," *IEEE Access,* vol. 7, p. 49191–49200, 2019.

[70] A. Narayanan, E. Pournaras and P. H. J. Nardelli, "Large-scale Package Deliveries with Unmanned Aerial Vehicles using Collective Learning," *IEEE Intelligent Systems,* pp. 1-10, 2024.

[71] K. Dorling, J. Heinrichs, G. G. Messier and S. Magierowski, "Vehicle Routing Problems for Drone Delivery," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 47, pp. 70-85, 2017.

[72] D. Andrea, Battery management systems for large lithium-ion battery packs, Artech house, 2010.

[73] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano and A. Martinoli, "The E-Puck, a Robot Designed for Education in Engineering," *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions,* vol. 1, p. 59–65, 2009.

[74] L. E. Parker and C. Touzet, "Multi-robot learning in a cooperative observation task," in *Distributed autonomous robotic systems 4*, Springer, 2000, p. 391–401.

[75] K. S. Narendra and S. T. Venkataraman, "Adaptation and learning in robotics and automation," in *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, 1995.

[76] G. H. W. Gebhardt, K. Daun, M. Schnaubelt and G. Neumann, "Learning Robust Policies for Object Manipulation with Robot Swarms," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

Glossary

FC	Fulfilment Centre
SCT	Supervisory Control Theory
SGD	Stochastic Gradient Descent
SoC	State of Charge
SoH	State of Health
SVM	Support Vector Machine
UAV	Unmanned Aerial Vehicle
VM	Virtual Machines