OpenSwarm

www.openswarm.eu

Call: HORIZON-CL4-2022-DATA-01

Type of action: RIA

Grant agreement: 101093046

Deliverable 4.3: Functional Specification, Architecture and Performance of the OpenSwarm Compiler

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

Document history

Date	Version	Author(s)	Comments
24-Jun-2024	01	Genki Miyauchi	preview
31-Oct-2024	02	Roderich Groß	Deliverable

Table of contents

Executive Summary

This document details the results achieved in the context of Task 4.3, specifically the development of a compiler that produces correct-by-construction adaptive swarm controllers and how they can be used in practice. The compiler takes as input a set of formal languages that describe the device's capabilities and specifications. Using supervisory control theory (SCT), it combines the capabilities and specifications into a minimally restrictive language that adheres to all specifications. The compiled swarm programs can subsequently be executed on different devices using the virtual machines developed as part of Task 4.4. To evaluate the capabilities, two use cases are presented. The first use case demonstrates that the swarm controllers can ensure that certain global performance requirements are met such as maintaining the connectivity of a network of mobile devices that are tasked to monitor simultaneously up to six remote locations. The second use case, which is based on a user study with 52 participants, demonstrates how multiple human operators can be formally integrated with a swarm of devices. enabling them to share the resources effectively. Both use cases illustrate how the dependencies among the swarms' components (i.e. individual devices or groups of devices within the swarm) can be formally captured by the swarm controllers which are produced by the compiler in less than a second.

Introduction

The objectives of WP4 are to design and implement the tools that enable operators to intuitively program energy-aware swarm level behaviours that are fully adaptive, while respecting formal specifications. WP4 develops energy management solutions for devices to harvest and allocate energy in adaptive ways, while monitoring and benchmarking consumption. It develops an intuitive swarm programming user interface for operators to describe what the devices can do and specify sub-system and swarm-level learning objectives. This feeds into a swarm compiler to automatically produce swarm controllers. Virtual machines will allow the resulting programs to run on multi-platform swarms.

The specific objectives for deliverable D4.3 are to develop a compiler that produces correct-by-construction adaptive swarm controllers. It takes as input a set of formal languages that describe the device capabilities and device/swarm specifications. Using supervisory control theory (SCT), these languages are combined into a minimally restrictive language that adheres to all specifications. To ease scalability, each device runs a "local modular" controller (also referred to as the supervisor). The dependencies among the swarm's components (individual devices or sub-swarms) will be formally captured using events shared across the swarm or sub-swarm. The supervisors stochastically choose among all permissible controllable events; the corresponding probability mass functions can be locally optimized at run time via reinforcement learning.

This document outlines the work completed in Task T4.3 and is structured as follows. First, a review of formal methods to design swarm controllers is presented. Second, the concept of SCT and the Nadzoru software, which is an open-source graphical user interface tool that contains the swarm compiler, is described. The extensions developed for Nadzoru as part of this deliverable are also described. After that, two use cases applying the swarm compiler in complex scenarios are presented. In the first use case, a swarm of robots splits into multiple teams and dynamically forms a mobile ad hoc network between the teams. The length of the network is shown to be near optimal. In the second use case, a swarm of robots are shown to dynamically engage with human operators who guide the robots to locations of interest and request robots to be exchanged between different teams.

Technical Readiness Level

The compiler technology, originally available at a TRL of 3, has been implemented and tested in a range of use cases, including a user study involving a total of 52 participants. Each trial involved two participants who collaboratively shared control over a swarm of simulated robots. We consider the technology to have reached a TRL of 4.

Key Performance Indicators

Table 3.2 in the DoA indicates KPI for T4.3 as “Compile time for realistic scenarios”, with a target below 60s. This KPI has been met, as described below.

We have tested controllers produced using the developed compiler in different use cases. Here we report the compilation time corresponding to every controller.

The models were compiled in the Nadzoru software using an Intel Core i7-8700 CPU @ 3.20GHz x 12 with 16GB of memory.

Use case	Role	Compile time (s)
Swarms with global performance requirements	Leader	0.0318
	Worker	0.2411
Swarms with human-in-the-loop	Leader	0.0264
	Worker	0.2630
Swarms and energy management	Worker (without mobile charger)	0.0609
	Worker (with mobile charger)	0.0590
	Mobile charger	0.0524

Optimising Swarm Behaviours Through Online Learning	UAV	0.0174

List of Publications

- G. Miyauchi, Y. K. Lopes, and R. Groß, 'Sharing the Control of Robot Swarms Among Multiple Human Operators: A User Study', in 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct. 2023, pp. 8847–8853. doi: 10.1109/IROS55552.2023.10342457.
- G. Miyauchi, M. S. Talamali, A. G. Millard and R. Groß, 'Towards Decentralised Formation of Minimal-Length Networks Using Swarms of Robots', in IEEE Robotics and Automation Letters (in preparation)
- J. Argote-Gerald, G. Miyauchi, P. Trodden, and R. Groß, 'On the Benefits of Robot Platooning for Navigating Crowded Environments' in 17th International Symposium on Distributed Autonomous Robotic Systems (DARS). Finalist *Best Paper Award*.

Related Work

Formal Design Methods in Swarm Robotics

Designing the behaviour of a robot swarm is challenging because changes made in the robot controller not only affect the actions of the particular robot but also the emergent behaviour of the entire swarm. The process of developing swarm behaviours usually requires trial and error, where the designer uses their intuition and experience to make modifications until the desired behaviours are achieved [1], [2]. However, as future swarms become more complex, they cannot be designed solely by such methods; there is a need for a systematic approach to support the swarm designer during the development of robot swarms [3].

Formal methods are system design techniques that can help reduce the amount of ad-hoc development. It involves creating a model of the system to be controlled and a set of specifications that the system must meet. The resulting model can be analysed by a variety of existing model checking and verification tools [4]. The explicit representation of capabilities increases the designer's confidence in the correctness of the system [5]. In this section, we review previous work on formal methods applied to the development of robot swarms.

Model checking is a method that uses temporal logic to analyse whether a given model matches the specifications [6].

One of the first works that applied formal methods to swarms was [7], which showed how swarm connectivity could be maintained using temporal logic by specifying the desired properties of the swarm. The authors used integer programming alongside linear temporal logic to perform subswarm assignment on 20 ground robots, although this was based on a centralised approach.

Brambilla et al. [8] proposed *property-driven design*, a top-down approach to swarm development based on describing swarm properties and model checking. It consists of four steps, (i) formally define the desired properties of the swarm, (ii) design and verify a prescriptive model using Markov chains, (iii) implement the prescriptive model and (iv) implement and test the swarm. A swarm designer following these steps would be able to concentrate on the design of the global swarm behaviour first, before focusing on the implementation details. While this approach offers methods to check whether certain properties can be met even before running tests using simulated or real robots, the automatic generation of source code has yet to be shown; implementing the model still requires creative insight from the designer.

Finite-state machines are commonly used to define robot controllers as they are generally easily readable to humans. Franchesca et al. [9] proposed *AutoMoDe-Vanilla*, an automatic design method for robot swarms. First, a designer defines a set of atomic behaviours called modules, which may include parameters. This is followed by an optimisation process that creates a controller represented as a probabilistic finite state machine. Experiments showed that the automatically designed controller performed better in an aggregation task than the controllers manually designed by human experts. The AutoMoDe method has since been extended to a class of automatic modular design methods [10], [11], [12], [13], [14], [15].

Behaviour trees have also been used to model the controllers for robot swarms. The tree defines a sequence of actions and conditions that a robot performs. While being easily readable to humans, behaviour trees are hierarchical so it is possible to detach a subtree and attach it to a different part of the behaviour tree, providing modularity and reusability of a particular behaviour [16]. This can be useful when evolving a behaviour tree since the designer can read to understand the evolved behaviour and make modifications if necessary [17]. Recently, AutoMoDe has also been extended to use behaviour trees [18].

Supervisory control theory (SCT) is a formal method proposed by [19]. SCT is used to restrict the behaviour of a controller (also known as a supervisor) so that the given logical control specifications are satisfied. A unique aspect of SCT is its explicit representation of uncontrollable events; events that cannot be disabled by the controller. Lopes et al. [4] applied SCT for the control of robot swarms. Through four case studies, they demonstrated how state-of-the-art swarm algorithms can be represented using SCT. The approach involved the automatic generation of control code from the developed supervisor. Further extensions have been proposed for modelling probabilistic decision making [20] and communication between robots [21].

Formal methods can help reduce the complexity involved in the swarm development cycle. This is especially needed for swarms that are intended to interact with multiple humans as they further complicate the interaction. However, the application of formal methods focusing on HSI is currently very limited. Providing certain formal guarantees on a swarm's capabilities during proximal interaction could ensure the safety of humans working near them.

Swarm Compiler Development

Supervisory Control Theory (SCT)

The SCT framework is used to control systems that can be represented as a discrete event system (DES) [19]. In other words, there is a discrete set of states that can express the state of the system at any time. In DES, transitioning between states is triggered by *events*. Events in SCT are classified as either an *uncontrollable* or *controllable*. Uncontrollable events can occur at any time (e.g. sensor inputs), while controllable events only occur when triggered by the controller (e.g. actions initiated by a robot). When designing a controller in SCT, the designer must define formal languages that express (i) what the robot can do in principle (called *free behaviour models*) and (ii) how it should behave (called *control specifications*). The corresponding languages are then automatically synthesised into a coherent language to produce the controller (called the *supervisor*) that is executed on each robot. The supervisor guarantees that, at any time, only valid sequences of events can occur. This is realised by restricting the set of controllable events that a robot can choose from at each state.

This thesis uses SCT to model the behaviour of robots based on a class of approaches proposed by [4], [22]. The remainder of this section provides an overview of how models can be designed in SCT using an example system and how a swarm designer would apply the approach in practice.

Generators

In SCT, regular languages are often used to represent the capabilities and control specifications of a system. Languages are expressed as generators, which are similar to automata. The difference is that automata checks whether a given word is part of a language, while generators produce words that are part of the language.

Generator G is defined as the following 5-tuple:

$$G = (Q, \Sigma, \delta, q_0, Q_m)$$

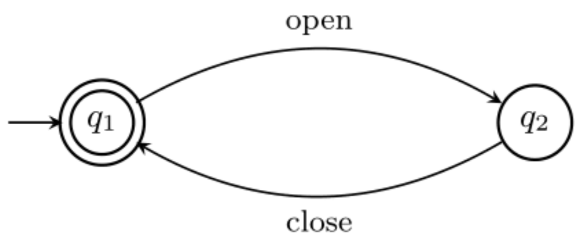
(1)

where $\boldsymbol{Q}$ is the finite set of states, $\boldsymbol{\Sigma}$ is the finite set of controllable and uncontrollable events, $\boldsymbol{\delta}: \boldsymbol{Q} \times \boldsymbol{\Sigma} \rightarrow \boldsymbol{Q}$ is the partial transition function, $\boldsymbol{q_0} \in \boldsymbol{Q}$ is the initial state, and $\boldsymbol{Q_m} \subseteq \boldsymbol{Q}$ is the set of marked states. Marked states are states that are considered to be safe and are usually used to represent a state after a process has finished or an idle state of the system. Generators are used to represent both free behaviour models and control specifications, which are discussed in the following sections.

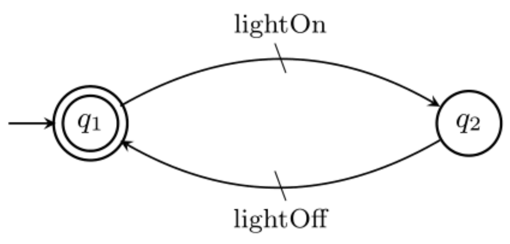
Free Behaviour Models.

Free behaviour models define the capabilities of a system. In other words, it describes all the possible actions that a system *could* perform. A system can be composed of multiple free behaviour models. Each model represents a certain functionality of the system. The models essentially act as the building blocks of the system and are assumed to be independent of one another. This means new functionalities can be added without affecting the existing models.

Figure 1 shows two free behaviour models for controlling the light inside a fridge. $\boldsymbol{G_1}$ represents the state of the fridge door. It indicates that events `open` and `close` occur alternatively. $\boldsymbol{G_2}$ represents the capability to control the light inside the fridge. The initial state is represented by an unlabelled arrow. An arrow with and without a stroke each represents a controllable and uncontrollable event, respectively. A double-circled or single-circled state each represents marked and non-marked states, respectively. In the given example, the marked state is defined to be when the fridge door is closed in $\boldsymbol{G_1}$ and when the light inside the fridge is turned off in $\boldsymbol{G_2}$.



(a) G_1

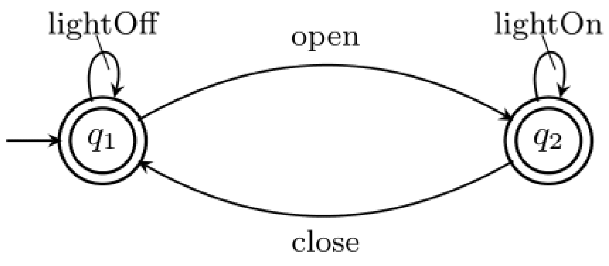


(b) G_2

Figure 1. Example free behaviour models for a fridge representing (a) a sensor to detect the state of the fridge door; (b) the capability to control the light inside the fridge.

Control Specifications.

Control specifications define the desired behaviour of a system. In other words, it describes the actions that the system *should* do. To do so, we disable some controllable events from occurring in certain states to achieve the desired behaviour of the system. Using the free behaviour models $\boldsymbol{G_1}$ and $\boldsymbol{G_2}$ from Figure 1, we can define a simple behaviour that turns on the light inside the fridge when the door opens and turns off



when it is closed.

Figure 2 presents an example control specification. In the initial state q_1, the only action allowed is `lightOff`. If the door opens, the state transitions to q_2. The only action allowed in q_2 is `lightOn`. It returns to the initial state when the door closes, where `lightOff` is again the only allowed action that can be triggered by the system. All states in control specifications are generally labelled as marked states unless it is used to represent a system with a buffer that needs to reach a state where the buffer is empty.

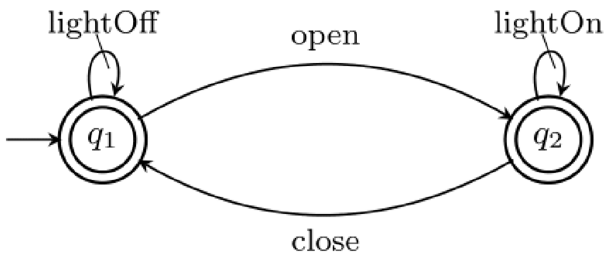


Figure 2. Example control specification to turn the light inside the fridge on when the door opens and off when it is closed.

Supervisor Synthesis

So far, we have seen that the free behaviour models define all possible events that can occur and the control specifications restrict controllable events from being triggered in certain states. The supervisor represents the control logic of the system, which can be obtained from the free behaviour and control specification models through an automatic process that synthesises them together [4].

The synthesised supervisor guarantees the following properties.

- **Controllable.** There are no outgoing uncontrollable events in any state that are disabled by the control specification.
- **Accessible.** Every state can be reached from the initial state.
- **Coaccessible.** From every state, at least one marked state can be reached.

These properties guarantee that the supervisor is non-blocking, in other words, there will be no deadlocks. During supervisor synthesis, any states that are non-accessible, non-coaccessible, or have an event that makes the supervisor uncontrollable are called *bad* states and are removed from the supervisor. The removal of these states is performed iteratively since removing one state could turn others into bad states, which are also removed. An advantage of the synthesis process is that when there is an error in the designed automata, it usually results in the final supervisor having a significantly small number of states or it may be missing certain events introduced in the free behaviour models. This is because models that conflict with each other result in the creation of more bad states, which are entirely removed during the synthesis. This is useful for the designer as any error in the logic can be detected at an early stage of the design process even before testing the system in simulation.

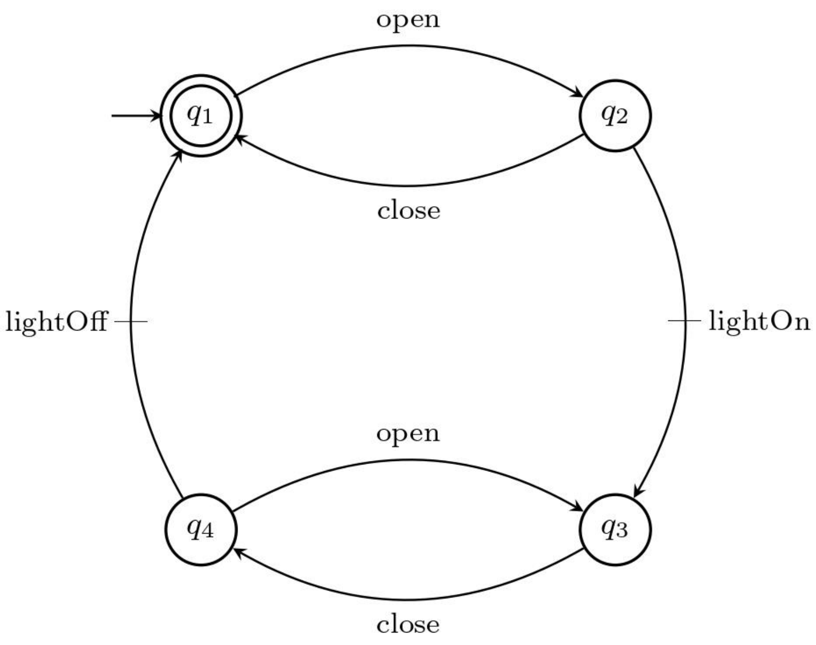


Figure 3. The synthesised supervisor for controlling the light inside a fridge.

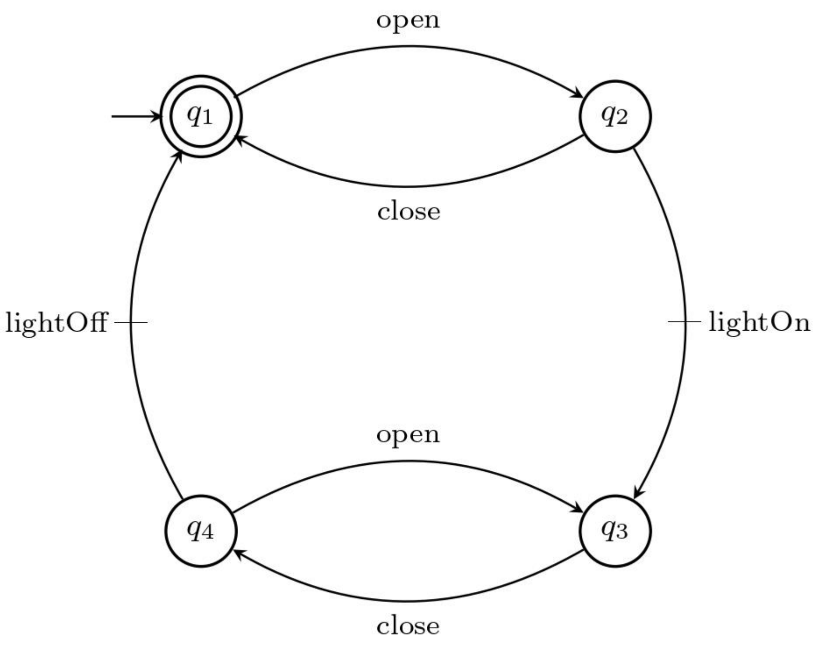


Figure 3 shows the supervisor that was synthesised using the models from Figure 1 and

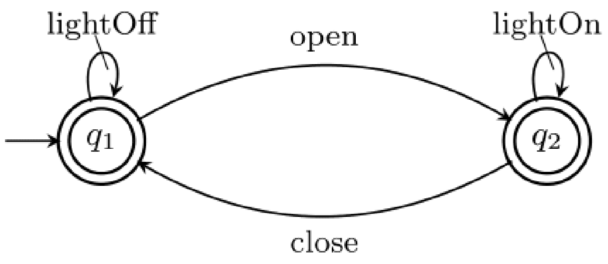


Figure **2**. The controllable events `lightOn` and `lightOff` are only triggered when uncontrollable events open and close occur, respectively. Note that the event `close` from state q_2 to q_1 is not disabled as it is an uncontrollable event. This covers the situation when the door opens but immediately closes before the system is able to trigger `lightOn`, in which case the light will not be turned on. An analogous situation is also covered by event `open` from state q_4 to q_3 when the door is opened immediately after it has been closed.

Designing free behaviour models and control specifications and synthesising the models to produce the supervisor can be done using the openly available software tool Nadzoru[1] [23].

When using the supervisors on a real system, it is necessary to link the events to the actual actions that are executed by the system. To do so, the SCT framework defines the *operational procedures* layer. Here, the designer programs a set of callback functions, which implements the behaviour of the robot when each event is triggered. While some programming is required, this is usually less than the amount of code that has to be written otherwise as the hardware-specific functionalities are the only parts that need to be coded; the control logic is already defined by the supervisors.

Public Events

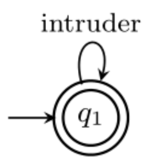
The original SCT framework assumes only the events that are observable by each robot. These events are referred to as *private* events. Lopes et al. [21] proposed incorporating *public* events; an extension to SCT that allows multiple robots to share the occurrence of events that were previously private. In this extension, both uncontrollable and controllable events can be either private or public. This allows communication among robots to be modelled within SCT. Generator G with public events is defined as the following 7-tuple:

$$G = \left(Q, \Sigma, \delta, q_0, Q_m, \Sigma_{u,pub}^{ext}, M\right)$$

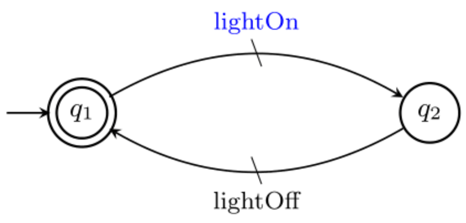
[1] https://gitlab.com/genki_miyauchi/nadzoru

where $\boldsymbol{\Sigma}$ comprises private uncontrollable and controllable events $\boldsymbol{\Sigma_{u,priv}}$, $\boldsymbol{\Sigma_{c,priv}}$ as well as public uncontrollable and controllable events $\boldsymbol{\Sigma_{u,pub}}$, $\boldsymbol{\Sigma_{c,pub}}$. $\boldsymbol{\Sigma_{u,pub}^{ext}}$ is the set of public uncontrollable events in $\boldsymbol{G}$ and other generators. Finally, $\boldsymbol{M}: \boldsymbol{\Sigma_{c,pub}} \rightarrow \boldsymbol{\Sigma_{u,pub}^{ext}}$ is a mapping from public controllable events to the related public uncontrollable events. $\boldsymbol{Q}$, $\boldsymbol{\delta}$, $\boldsymbol{q_0}$ and $\boldsymbol{Q_m}$ are identical to Equation (1).

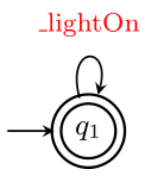
For example, consider a swarm of robots that are tasked to find an intruder. We would like only the last robot that has spotted the intruder to turn its LEDs on. Figure 4 shows the free behaviour models for these robots. $\boldsymbol{G_1}$ represents a sensor capable of detecting an intruder. $\boldsymbol{G_2}$ represents the robot's ability to turn its LEDs on and off. Event `lightOn` is specified as a public controllable event, so it will inform other robots that it has turned its lights on. $\boldsymbol{G_3}$ represents the corresponding public uncontrollable event `_lightOn` which is triggered when it detects that another robot has turned its lights on.



(a) G_1



(b) G_2



(c) G_3

Figure 4. Free behaviour models for last spotted location. (a) A sensor to detect an intruder. (b) The capability to control the robot's own LEDs. (c) A receiver to detect the occurrence of another robot turning their LEDs on. Public uncontrollable and controllable events are shown in red and blue, respectively.

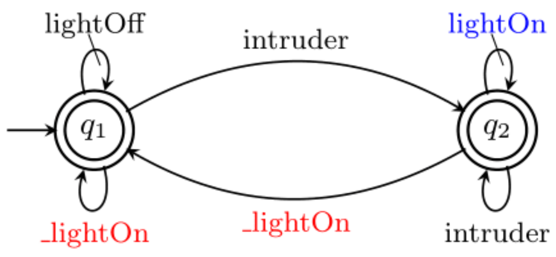


Figure 5 defines the control specification. It defines that when an intruder is detected (event `intruder`), the robot will turn its LEDs on (event `lightOn`) which is communicated to all other robots. If it detects that another robot has turned its lights on (event `_lightOn`), the robot will turn off its own lights (event `lightOff`). By using public events, it is possible to model the behaviour of robots when they receive signals from other robots. The synthesised supervisor is shown in

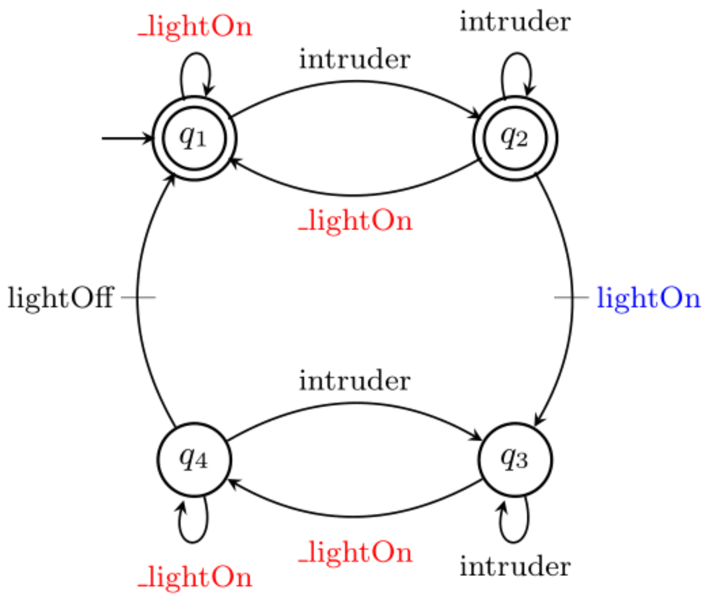


Figure **6**.

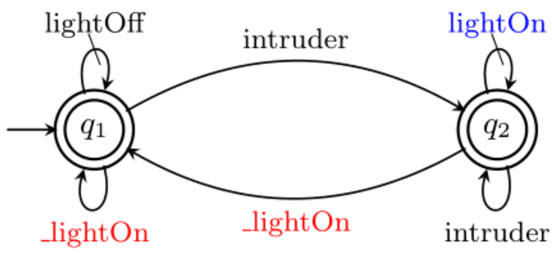


Figure 5. Control specifications for last spotted location to turn its own LEDs on when it detects an intruder and turn it off when another robot turns its LEDs on.

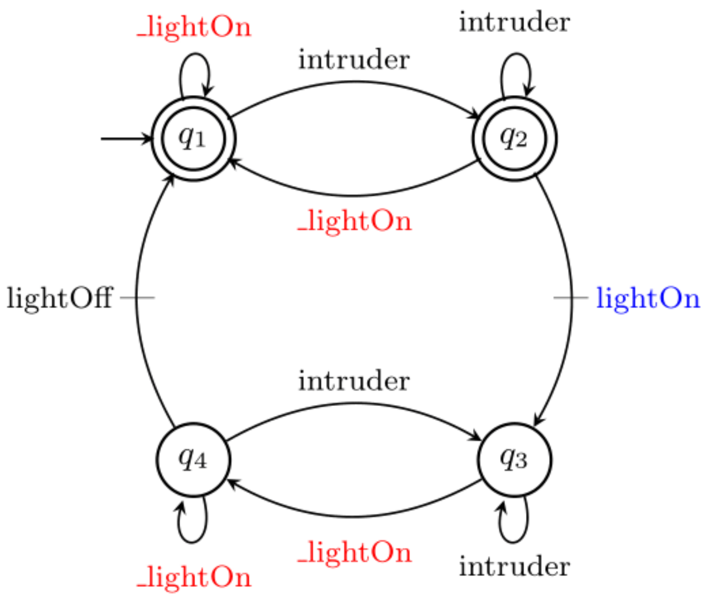


Figure 6. The synthesised supervisor for detecting an intruder using public events.

Probabilistic Events

There can be situations where the standard SCT framework allows more than one controllable event to be triggered at a certain state. In this situation, each permissible event will be given equal probability and one event will then be picked at random. While this ensures that the system continues to operate in a safe state, it always treats the events equally and does not allow one event to be more likely to be picked when multiple events can be triggered. This can limit the robot's ability to choose an action that is more likely to produce a better outcome.

The probabilistic SCT (pSCT) framework [20] is an extension of the standard framework that enables a designer to specify probabilities for each event transition to adjust the likelihood of the events from being chosen.

Consider an example where a swarm of simple robots must explore a bounded environment containing obstacles. The designer might want a robot to turn left more often than to turn right so that the robots are more likely to explore the environment anti-clockwise. In the pSCT framework, the probabilities can be specified for each event transition as shown in Figure 7. It defines that when the events `turnLeft` and `turnRight` are both permissible events, `turnLeft` will be chosen with a probability of $\boldsymbol{p = 0.6}$, while turnRight will have a probability of $\boldsymbol{p = 0.4}$. Note that if there is only one controllable event that can be triggered in the current state, that event will be chosen with a probability of 1.

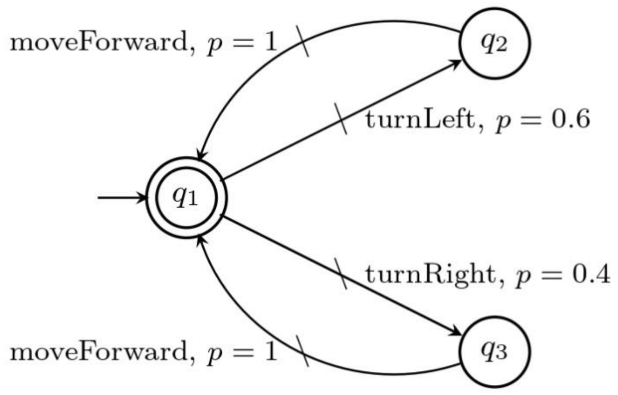


Figure 7. Example specification using probabilistic SCT. Each controllable event has an associated probability that can be specified at design time. The event turnLeft is more likely to be selected than turnRight when both are permissible events.

Nadzoru: A Graphical User Interface to design SCT Models

Nadzoru is an open-source software tool that can be used to design and synthesise SCT models, and generate template source code for a variety of platforms and programming languages. In this section, we introduce Nadzoru 2[2], which is a modern succession of the original Nadzoru 1, followed by the extended capabilities developed as part of this deliverable for Nadzoru 2 to enable public and probabilistic events, and the preparation of template source code for C++ and Python.

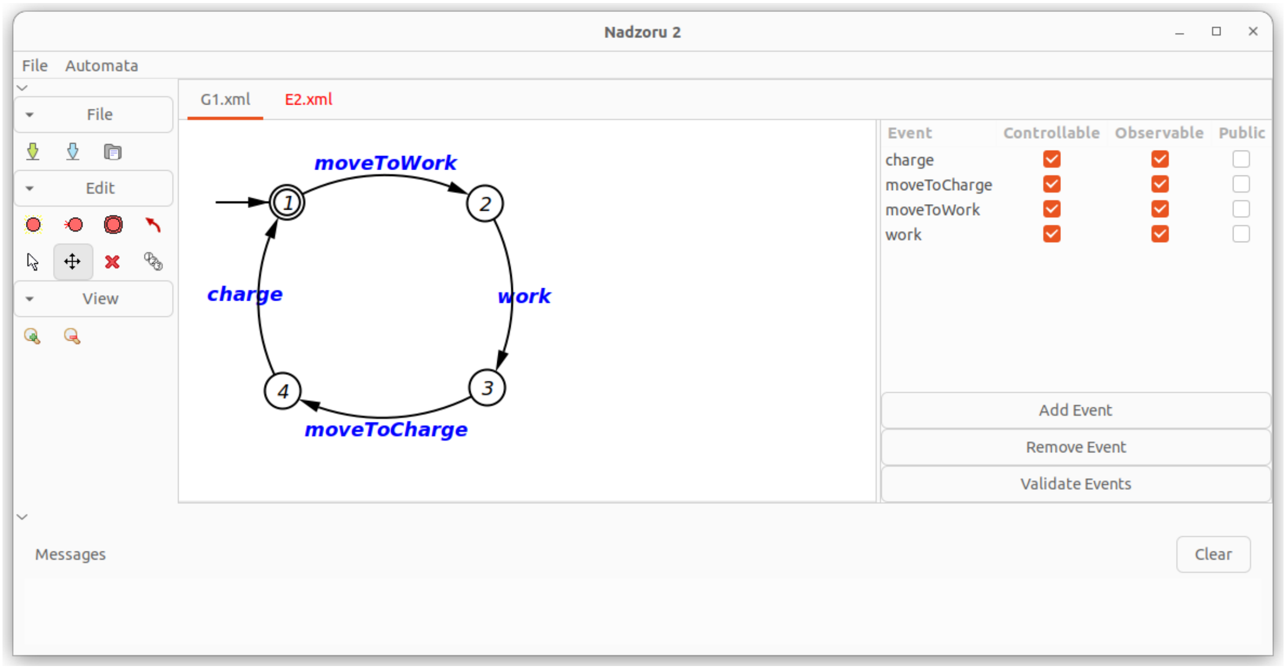
To design the behaviour of robots using Nadzoru 2, the designer follows the steps below:

1. Define free behaviour models.
2. Define specifications.
3. Synthesise the free behaviour models and specifications.
4. Export the synthesised models to generate source code.
5. Implement callback functions.

[2] https://github.com/openswarm-eu/Nadzoru2

We give an overview of how a designer interacts with Nadzoru 2 from starting to design a model to generating the source code run that can be run on a robot. We consider a robot that can move between two regions to either (i) perform work that is present in that region, or (ii) charge its own energy, depending on the specific region.

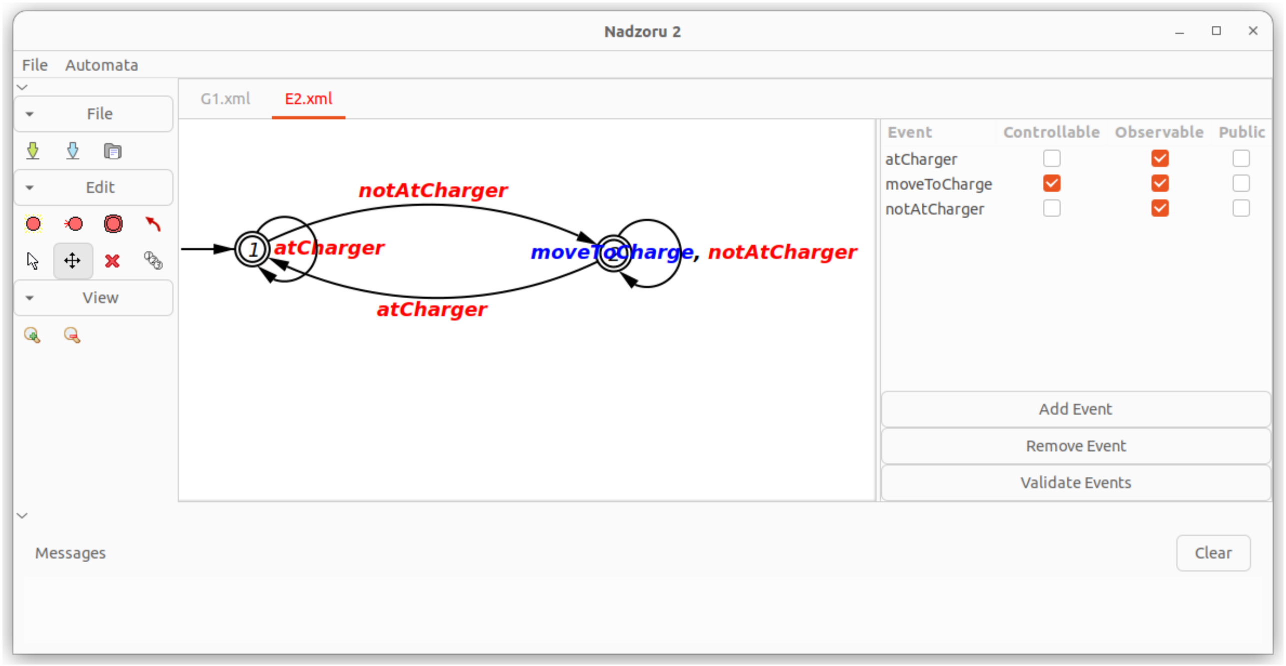
First, a set of free behaviour models (i.e. the capabilities of the system) need to be defined. Figure 8. The Nadzoru 2 interface showing a user defining the free behaviour model. The user lists the relevant events (right) and uses them to create a generator (left) shown using states and transitions. Figure 8 shows a user that has defined one of the capabilities. On the right panel, it lists the events (`work`, `charge`, `moveToWork`, and `moveToCharge`) relevant to this capability. The centre panel shows a user defining the free behaviour model that represents the robot's capability to indefinitely repeat the cycle of moving and either performing some work or charging its energy.



Event	Controllable	Observable	Public
charge	✓	✓	☐
moveToCharge	✓	✓	☐
moveToWork	✓	✓	☐
work	✓	✓	☐

Figure 8. The Nadzoru 2 interface showing a user defining the free behaviour model. The user lists the relevant events (right) and uses them to create a generator (left) shown using states and transitions.

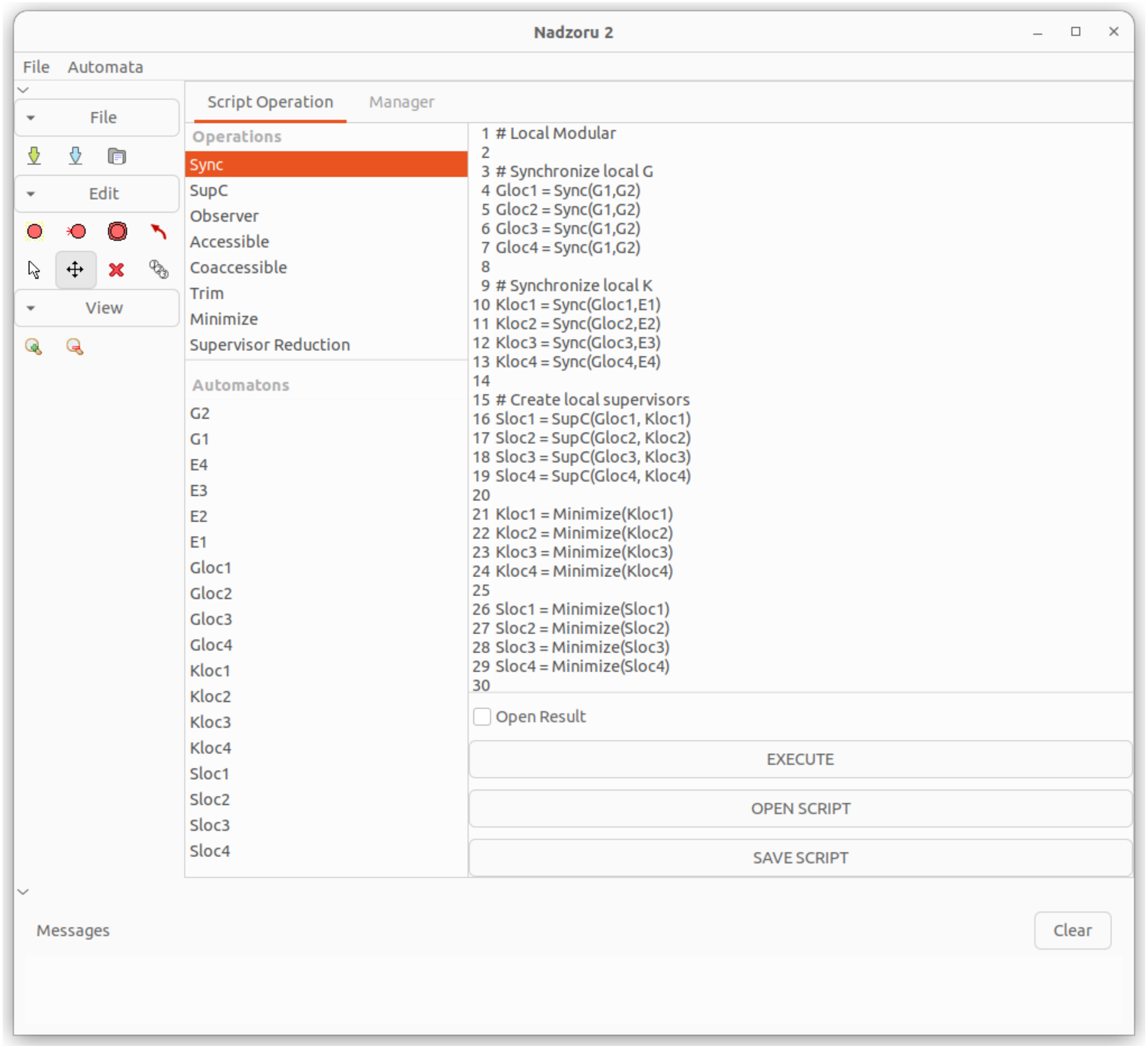
Next, a set of specifications (if any) must be defined. Figure 9 shows a user that has defined one of the specifications. Similarly, as before, the panels on the right and centre display the relevant events and the generator that the user has designed. The specific example shown restricts the event `moveToCharge` so that it can only be triggered when event `notAtCharger` has been triggered.



Event	Controllable	Observable	Public
atCharger	☐	✓	☐
moveToCharge	✓	✓	☐
notAtCharger	☐	✓	☐

Figure 9. The Nadzoru 2 interface showing a user defining a specification for the robot. The user lists the relevant events (right) and uses them to create a generator (left) shown using states and transitions.

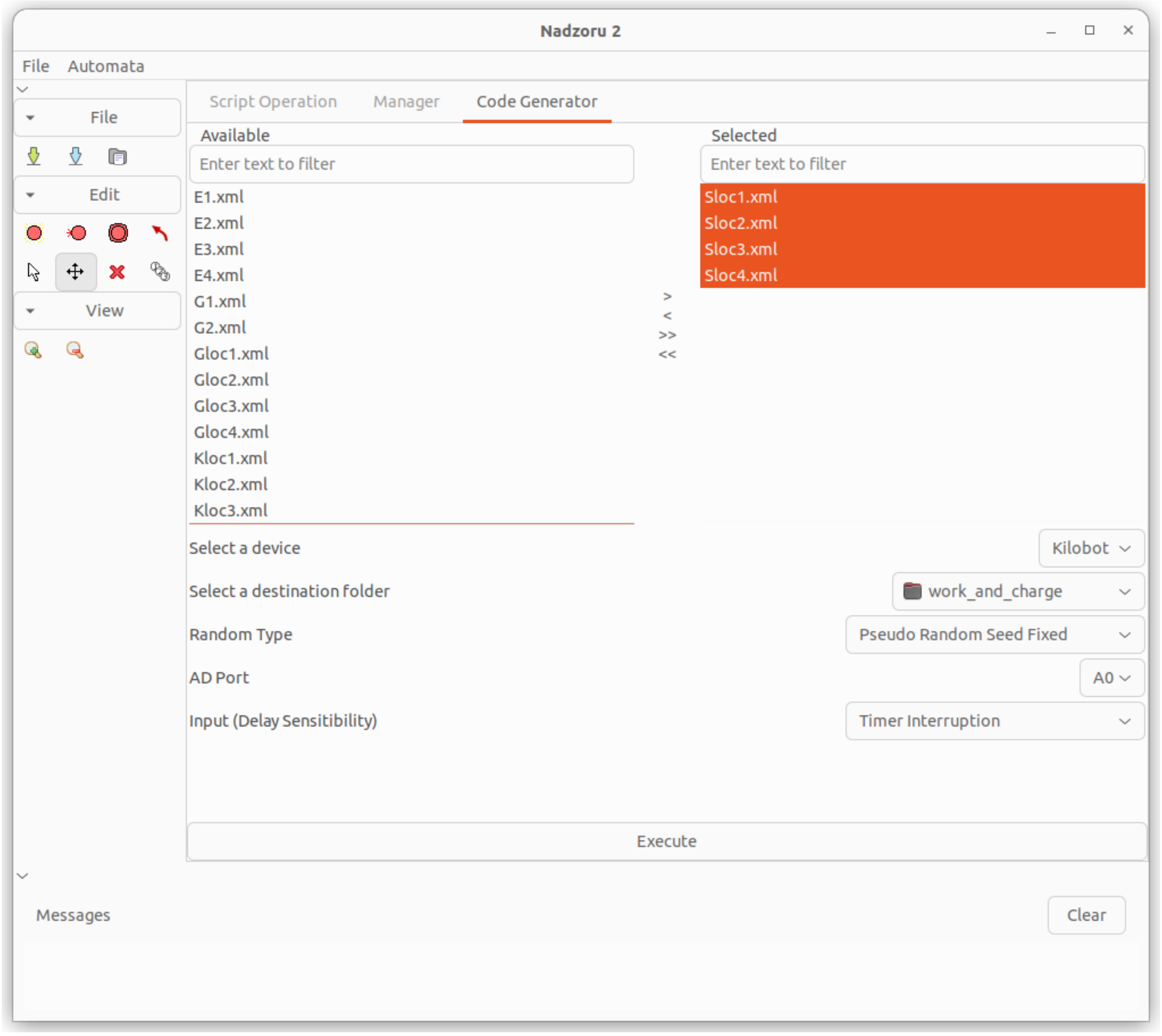
Once all free behaviour models and specifications have been defined, the models need to be synthesised together to produce the final control logic. Figure 10 shows the script operation screen, which consists of a text editor in the centre panel and a set of available functions listed in the left panel. The user makes use of the provided functions to build a script that instructs how to synthesise the models. Information such as the size of the final models can be printed as seen in the bottom panel.



```
# Local Modular

# Synchronize local G
Gloc1 = Sync(G1,G2)
Gloc2 = Sync(G1,G2)
Gloc3 = Sync(G1,G2)
Gloc4 = Sync(G1,G2)

# Synchronize local K
Kloc1 = Sync(Gloc1,E1)
Kloc2 = Sync(Gloc2,E2)
Kloc3 = Sync(Gloc3,E3)
Kloc4 = Sync(Gloc4,E4)

# Create local supervisors
Sloc1 = SupC(Gloc1, Kloc1)
Sloc2 = SupC(Gloc2, Kloc2)
Sloc3 = SupC(Gloc3, Kloc3)
Sloc4 = SupC(Gloc4, Kloc4)

Kloc1 = Minimize(Kloc1)
Kloc2 = Minimize(Kloc2)
Kloc3 = Minimize(Kloc3)
Kloc4 = Minimize(Kloc4)

Sloc1 = Minimize(Sloc1)
Sloc2 = Minimize(Sloc2)
Sloc3 = Minimize(Sloc3)
Sloc4 = Minimize(Sloc4)

```

Figure 10. The Nadzoru 2 interface showing the model synthesis process. The user can build a script in the built-in text editor to specify how the models should be synthesised.

Finally, the user selects the synthesised models to generate source code (see Figure 11). The user can choose from a number of template source code that can be used to program the robot, or prepare a custom template that fits their own needs.



E1.xml
E2.xml
E3.xml
E4.xml
G1.xml
G2.xml
Gloc1.xml
Gloc2.xml
Gloc3.xml
Gloc4.xml
Kloc1.xml
Kloc2.xml
Kloc3.xml

Sloc1.xml
Sloc2.xml
Sloc3.xml
Sloc4.xml

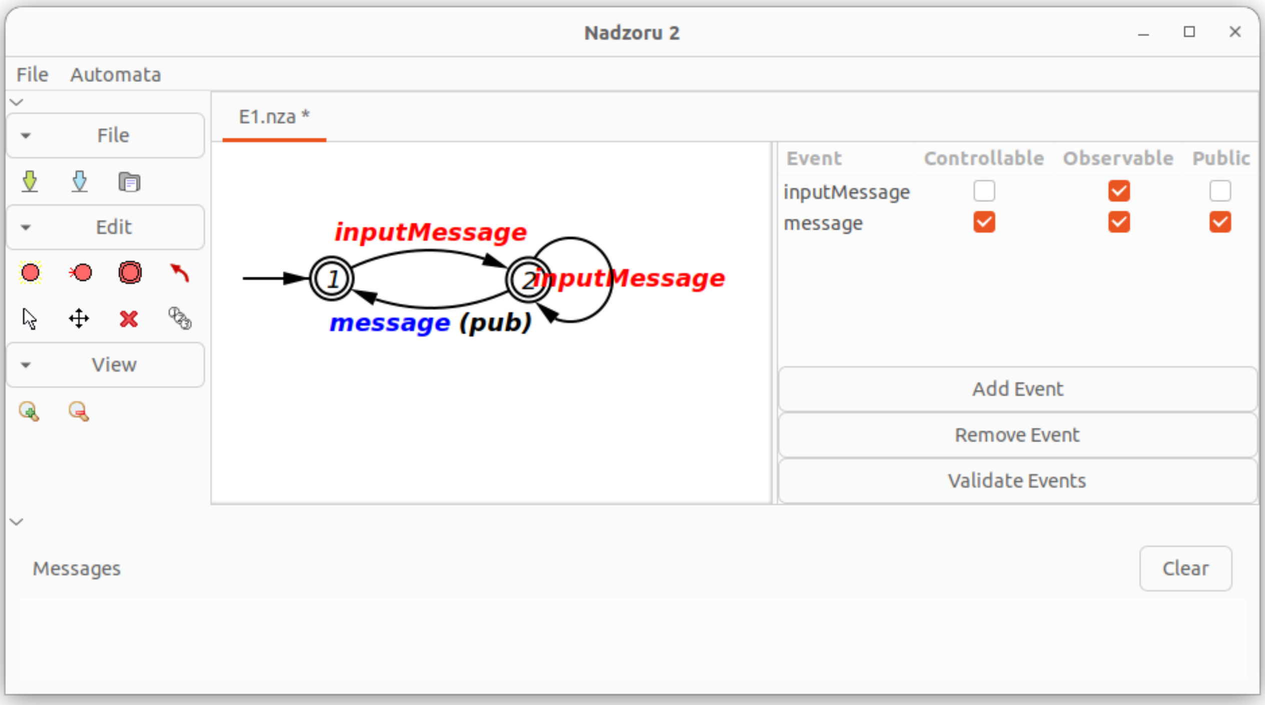
Select a device	Kilobot
Select a destination folder	work_and_charge
Random Type	Pseudo Random Seed Fixed
AD Port	A0
Input (Delay Sensitibility)	Timer Interruption

Figure 11. The Nadzoru 2 interface to generate the source code of the synthesised models for the selected device.

The template source code includes the virtual machine that is necessary to execute the generated control logic (e.g. in C programming language). The user is required to implement each callback function, which usually requires using robot-specific APIs. For uncontrollable events, the user must implement code that can evaluate whether the event has been triggered. For controllable events, the user must implement code that executes the action when the event is triggered.

Public Event Extension

The original SCT framework can be extended using public events, which share the occurrence of events across multiple devices. This allows communication among devices to be modelled using SCT, which is particularly useful for a robot swarm that may need to communicate with neighbouring robots. Figure 12 shows a user defining a public event from the toggle button in the right panel (event `message`). Here, the event `message` is a controllable event that is triggered when the event `inputMessage` is received, which may be a condition to send a certain message to other robots. Since event `message` is also a public event, the occurrence of the event is broadcasted to other robots. If they have a corresponding public uncontrollable event (e.g. event `_message`) that triggers upon receiving the occurrence of the event, it will also be triggered in each individual robot. The synthesised controllers also indicate which events are public so that they can be handled correctly by the virtual machine that executes the controllers on the robots.

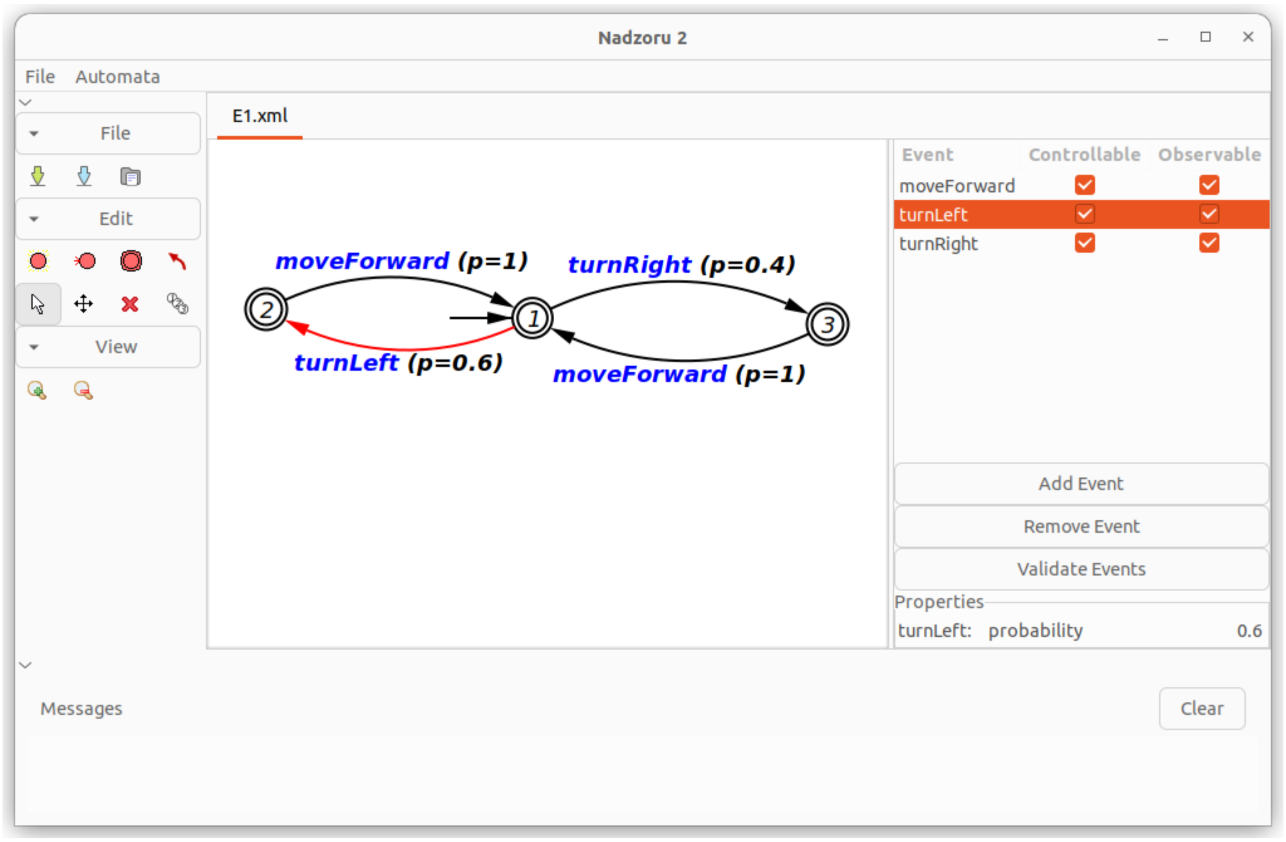


Event	Controllable	Observable	Public
inputMessage	☐	✓	☐
message	✓	✓	✓

Figure 12. The Nadzoru 2 interface showing a user defining a model that consists of public events, which are indicated by the label appended to its event name.

Probabilistic Event Extension

Another extension to the original SCT framework is the use of probabilistic events. When multiple controllable events can be triggered from a given state, the original SCT framework picks one event at random with equal probability. This can become an issue since the user cannot define a preference for some events to be chosen more often than others, even if they are aware that some events lead to better outcomes. Figure 13 shows a user defining probabilities to the controllable events in the model that was defined in Figure 7. Each event transition indicates the probability at which it will be chosen among other controllable events, if there are any. At state 1, two outgoing events `turnLeft` and `turnRight` can be triggered at a probability of 0.6 and 0.4, respectively. The probability can be modified by first selecting the transition (highlighted in red in Figure 13) and then entering a new probability in the Properties box found in the right panel. States 2 and 3 only have one outgoing controllable event `moveForward`, hence the probability is set to 1. Note that, it is possible for a user to enter probabilities values where the sum exceeds 1, in which case, the probabilities will be normalised across all controllable events by the virtual machine that executes the control logic at run time.



Event	Controllable	Observable
moveForward	✓	✓
turnLeft	✓	✓
turnRight	✓	✓

Properties
turnLeft: probability 0.6

Figure 13. The Nadzoru 2 interface showing a user defining a model where each state transition has a probability associated with them. The probability of each controllable event being chosen is indicated by the probability label appended to its event name. The probability can be modified by selecting the transition (highlighted as a red arrow) and then entering a new probability shown in the right panel under the Properties box.

Swarm Compiler Use Cases

This section presents two use cases where the swarm compiler and SCT were used to design and partially implement the behaviour of robot swarms that can (i) satisfy global performance metrics and (ii) engage with a human operator to perform tasks that require human-swarm interaction.

Swarms with global performance requirements

Many applications require a large group of machines, hereafter a swarm of robots, to perform tasks across expansive environments, with the overall missions often lasting between a few hours to many months. Examples include autonomous farming [24], mining [25], and planetary exploration [26].

We are interested in scenarios where access to global infrastructure is severely limited. Consequently, the robots may be unable to communicate with, or obtain position estimates from, a central entity. This is typically the case for underground, underwater, indoor, and remote missions. A potential solution is the formation of a mobile ad hoc network by the robots of the swarm. Such networks can be established through local interactions between the robots.

Most studies focus on the problem of establishing a network that simultaneously connects two locations of interest [27], [28], [29], [30], [31], [32]. In this case, minimal-length networks can be formed by arranging robots in a chain (formally, a *path* graph). Forming minimal-length networks among more than two locations may require more complex arrangements (i.e. *tree* graphs). Majcherczyk et al. [33] propose a decentralised approach to creating such tree graphs and evaluate it in scenarios using 2-4 locations of interest. The trees formed are *starlike*: they have a single root node that connects to all locations of interest via dedicated path graphs. For three locations, minimal-length networks need to be (optimal) *starlike* trees[3]. In the general case, however, minimal-length networks need to be *Steiner* trees [34]. Such networks not only reduce the time and energy used for information transfer and robot navigation along the network, but also the number of robots assigned to maintaining the network, thereby freeing the remaining robots for other tasks.

This paper presents a novel, decentralised approach for robot swarms to establish and maintain networks among two or more locations of interest. The robots operate in a fully decentralised manner and do not rely on global infrastructure for positioning or communication. Unlike prior works, the proposed approach does not restrict the network's topology to path graph [31], [32] or starlike trees [33]. We examine the efficiency of the proposed approach using embodied simulations and physical experiments, focusing on the length of established networks, and the number of robots therein, for up to six locations of interest. The networks are found to compare reasonably well with optimal, minimal-length networks (i.e. Steiner trees produced by a centralised solver). For scenarios with more than three locations of interest, their length is significantly less than that of optimal starlike topology networks, which are centrally computed to provide a lower bound for the length of starlike topology networks. Moreover, the approach succeeds to make available those robots not necessary for network formation for work at the locations of interest.

The following sections are organised as follows. First, we formulate the network formation problem. Next, we propose a decentralised approach for it and then detail the implementation for a robotic platform. We evaluate our approach using computer simulations compared to theoretical benchmarks and validate the findings using real-robot experiments.

[3] The root node coincides with the geometric median of the locations of interest. Path graphs extend from this node. Where the locations of interest form a triangle with an interior angle of $\frac{2}{3}$ or more, the starlike tree degenerates, becoming a path graph.

Problem Formulation

Consider a planar environment with n locations of interest and m robots. Similar to [33], we assume that the locations of interest are known to at least some of the robots. All robots can move. Each one can communicate with all other robots within communication range r_{com}. The robots represent the nodes of a *network*, hence, they control where the network is deployed. Moreover, they select which of their communication links to include in the network.

Formally, the set of possible networks is $\mathcal{G} = \{(V, E) | V \subset \mathbb{R}^2 : |V| = m, E \subseteq \{(u, v) \in V \times V | \|u - v\| \leq \mathrm{r}_{\text{com}}\}\}$. The robots' objective is to assume a network from $\mathcal{G}$ that

- is of minimal length;
- extends to the locations of interests. In other words, each location has an observer robot, that is, a robot no more than some constant ϵ away;
- for any pair of locations of interest, contains a path graph connecting the corresponding observer robots.

For $\epsilon = 0$, this problem could be considered an embodied adaptation of the Steiner tree problem with bounded edge length (r_{com}) and bounded number of nodes (m).

Decentralised Network Formation

The swarm of robots are initially positioned in a designated launch area, sufficiently close to exchange messages (assuming hop-to-hop communication). They are divided into n equal-sized teams, each assigned a specific location of interest, hereafter *target* location. Every team consists of a *lead* robot and any number of *followers*. The lead robots have the means to navigate to their respective target locations. Followers are instructed to remain close to their leader.

The swarm initially elects one follower robot to become the first connector (e.g. by choosing the robot of the lowest unique ID). We assume that this connector is in communication range to at least one member for every team. As the lead robots move towards their respective target locations (accompanied by the members of their respective teams), the swarm disperses. To prevent the teams from becoming disconnected, some followers leave their teams, joining the *network* as *connectors*: they extend the branch of the network that leads to the team they left. The logic of branch extension is described in Branch Extension. A connector continuously positions itself at the centre of the robots it maintains a connection with. The process of becoming a connector is reversible. Where a connector is no longer needed to ensure connectivity, it joins the closest team as a follower.

To enable topologies beyond starlike trees, we allow branches to change the node they are anchored to, where such change reduces the total network length. The logic of this branch repositioning is described in Branch Repositioning.

Branch Extension

In our approach, each robot follows local rules to update its position, role, and connections (i.e. the robots with whom it forms a network branch). At all times, every robot locally broadcast the list of robots it is currently connected to.

Starting from the initial connector, the network branches are extended using the approach in [31], which so far was only used to build path graphs in swarms of simulated robots.

While following a *lead* robot, some team members switch from the *follower* to the *connector* role to extend the branch of the network leading to the team. For follower i of team u to become a connector, the following conditions must all be met (see Figure 14):

- C_1: Its distance from connector j which directly connects to team u exceeds a safety limit, r_{safe}.
- C_2: It does not detect any other follower of team u that is closer to connector j than itself.
- C_3: The distance between the lead robot of team u and aforementioned connector j is greater than safety limit r_{safe}. This condition allows for the dynamic shrinking of the network, as explored in [31].

As multiple teams move towards their target locations, the aforementioned rule enables the establishment of a network of starlike topology, with the root node being the initial connector. Such networks may not be optimal in length for scenarios with more than three target locations.

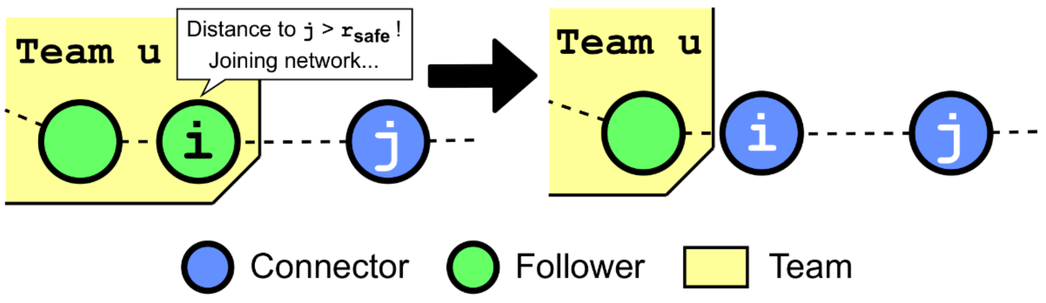


Figure 14. Branch extension. Dashed lines indicate the connections that are part of the network. The distance between follower i of team u and connector j is greater than r_{safe} (condition C_1), i does not detect any other follower of team u that is closer to connector j than itself (condition C_2), and does not detect the team's lead robot to be near connector j (condition C_3). In this case, follower i requests to become a connector, which is to be approved by connector j.

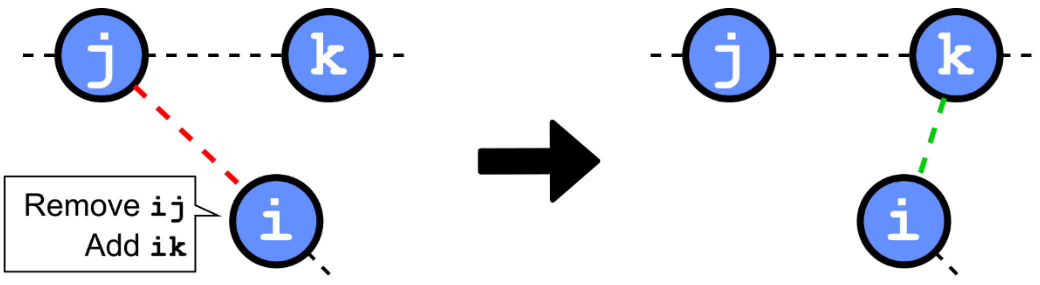
Branch Repositioning

To form minimal-length networks, we propose a local rule that enables branches to be repositioned. Consider connectors $\boldsymbol{i}$ and $\boldsymbol{j}$ and a third robot $\boldsymbol{k}$ that can either be a connector or a follower. Connector $\boldsymbol{i}$ modifies its connections when some of the following conditions are met (see Figure 15):

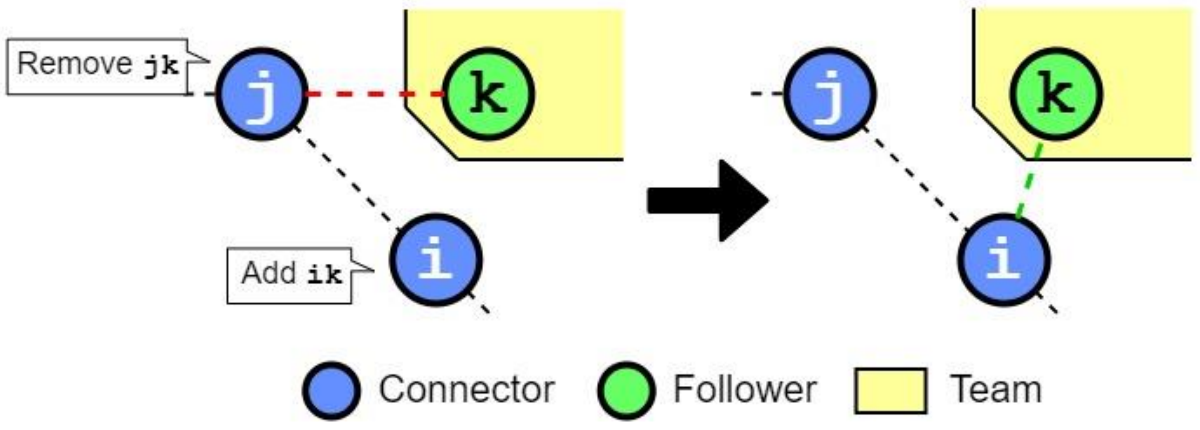
- RB_{1a}: k is a connector and the distance between i and k is shorter than the distance between i and j.
- RB_{1b}: k is a follower and the distance between i and k is shorter than the distance between j and k.
- RB_2: j and k are directly connected with each other.
- RB_3: j is connected to at least three other connectors or followers.

If conditions RB_{1a}, RB_2, and RB_3 are satisfied, i removes its existing connection with j and establishes a new connection with k (Figure 15a). However, if conditions RB_{1b}, RB_2, and RB_3 are satisfied, i establishes a connection with k and notifies j about this. Once notified, j removes its connection with robot j (Figure 15b).

As connections between robots are replaced only with shorter connections, following the change, the resulting network must be of shorter length than the original.



(a)



(b)

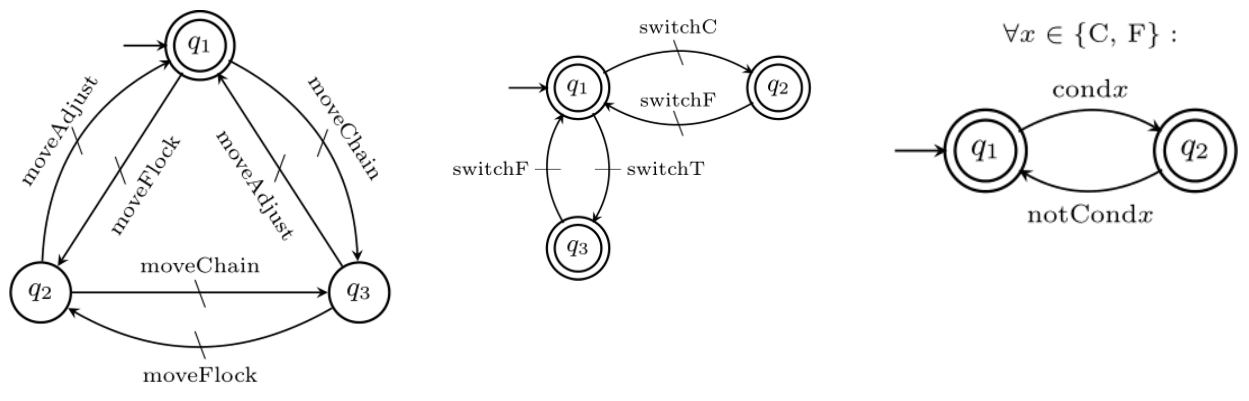
Figure 15. Branch repositioning between robots i, j, and k. Black, red, and green dashed lines indicate connections within the network that are unaffected, to be removed, or to be newly added, respectively. (a) k is a connector and the distance from i to k is shorter than the distance from i to j (condition RB_{1a}), j and k are directly connected with each other (condition RB_2), and j has at least three connections (condition RB_3). Connector i removes its existing connection with j and establishes a new connection with k. (b) k is a follower and the distance from i to k is shorter than the distance from j to k (condition RB_{1b}) and conditions RB_2 and RB_3 are satisfied.

Connector i notifies j that it will establish a connection with k. As a result, j removes its connection with k.

SCT-based Network Formation

We use SCT to formally model the behaviour of negotiating and applying the network changes among the robots. While the lead robot's behaviour are also modelled using SCT, their capabilities do not affect the behaviour of how the follower robots evolve the network. Therefore, we exclude their models from this document. Full details of the lead robot's models can be found in [35].

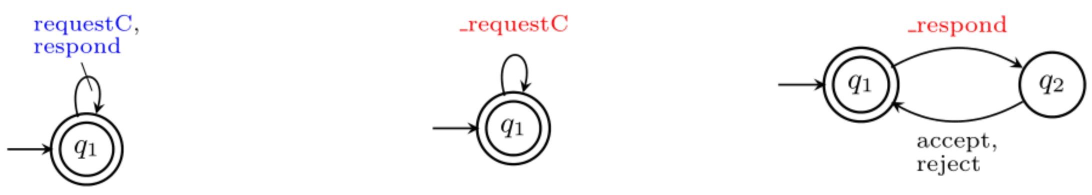
Figure 16 shows the free behaviour models related to the worker's movement and role-switching capabilities. G_1 represents the robot motion. It allows the robot to adjust its position in the network when it is a connector (state q_1), to flock with its team (state q_2) when it is a follower, or to switch teams by moving along the network (for further details, see [31]). G_2 represents its ability to switch its roles, e.g., switchC will switch to connector. G_3 represents the conditions for a robot to become a connector or a follower.



(a) G_1 (b) G_2 (c) G_3

Figure 16. Free behaviour models regarding the worker's movement and ability to switch roles. (a) Motion capabilities; (b) role switching; (c) conditions for becoming a connector and a follower.

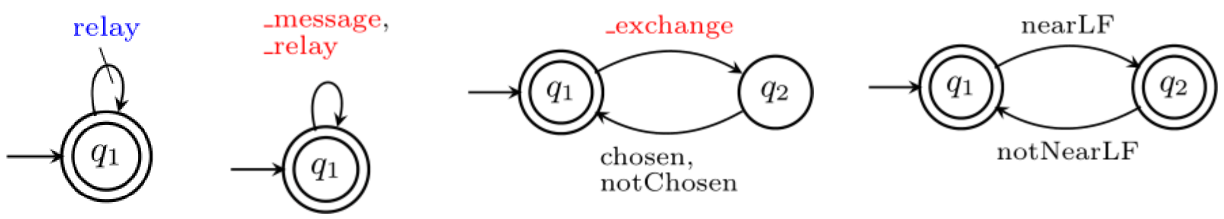
Figure 17 shows the free behaviour models related to the worker's ability to extend the network. $\boldsymbol{G_4}$, $\boldsymbol{G_5}$ and $\boldsymbol{G_6}$ represents the ability for robots to send, receive and respond to a request to extend the network by switching from a follower to a connector.



(a) G_4 (b) G_5 (c) G_6

Figure 17. Free behaviour models regarding the worker's ability to extend the network. (a) Request to a neighbouring tail connector or reply to a request from a follower; (b) receive the request; (c) receive the response.

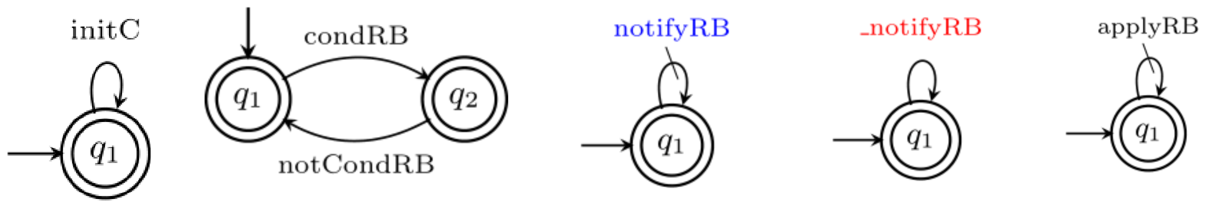
Figure 18 shows the free behaviour models related to the worker's ability to relay messages and exchange workers between teams.



(a) G_7 (b) G_8 (c) G_9 (d) G_{10}

Figure 18. Free behaviour models regarding the worker's ability to relay messages and exchange workers between teams. (a) Send a message to be relayed; (b) receive a message to be relayed from the lead agent or a worker; (c) receive a message from the lead agent to join another team; (d) determine whether it is near the team that it needs to join.

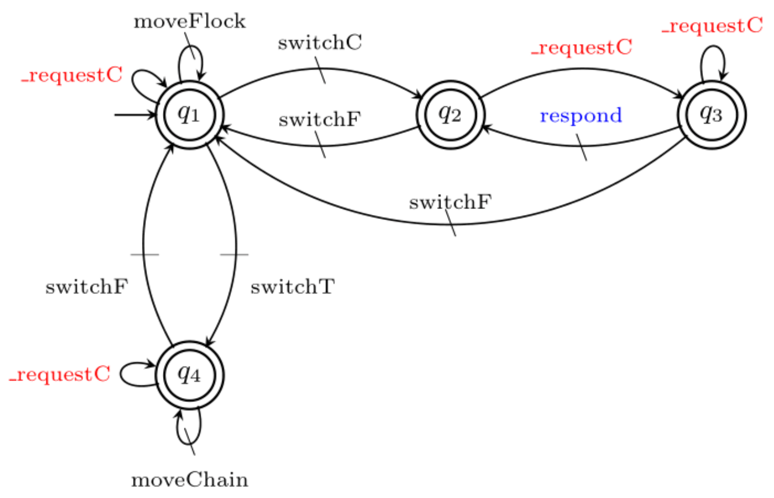
Figure 19 shows the free behaviour models related to branch repositioning. $\boldsymbol{G_1}$ represents the connector's ability to detect whether the aforementioned conditions to reposition the branch have been met (event `condRB`). $\boldsymbol{G_2}$ and $\boldsymbol{G_3}$ allow a connector to broadcast or receive a signal that informs relevant connectors to apply the modifications to their connections. Events `notifyRB` and `_notifyRB` are corresponding public events. This means if a robot triggers the controllable event `notifyRB`, it triggers the uncontrollable event `_notifyRB` for any relevant connectors within range. $\boldsymbol{G_4}$ defines the action to apply the network modification.



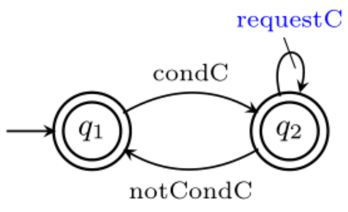
(a) G_{11} (b) G_{12} (c) G_{13} (d) G_{14} (e) G_{15}

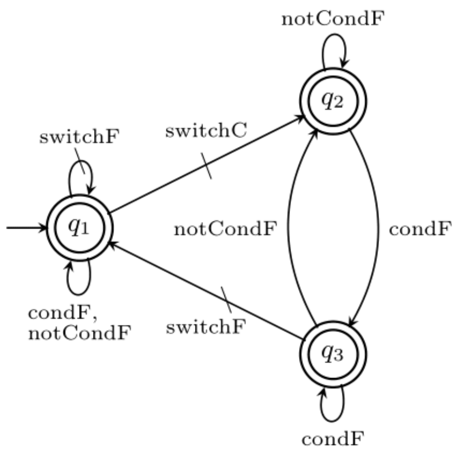
Figure 19. Free behaviour models representing the robot's ability (a) to determine whether the conditions to modify its connections have been satisfied, (b) to transmit or (c) receive messages informing the network modification to/from relevant connectors, and (d) to apply the network modification.

Figure 20 shows the control specifications regarding the worker's ability to switch its role. Specification $\boldsymbol{E_1}$ limits certain role-dependent controllable events to be enabled only in specific roles. Specifications $\boldsymbol{E_2}$ and $\boldsymbol{E_3}$ each represent the conditions that must be satisfied to switch roles between a follower and a connector.



(a) E_1

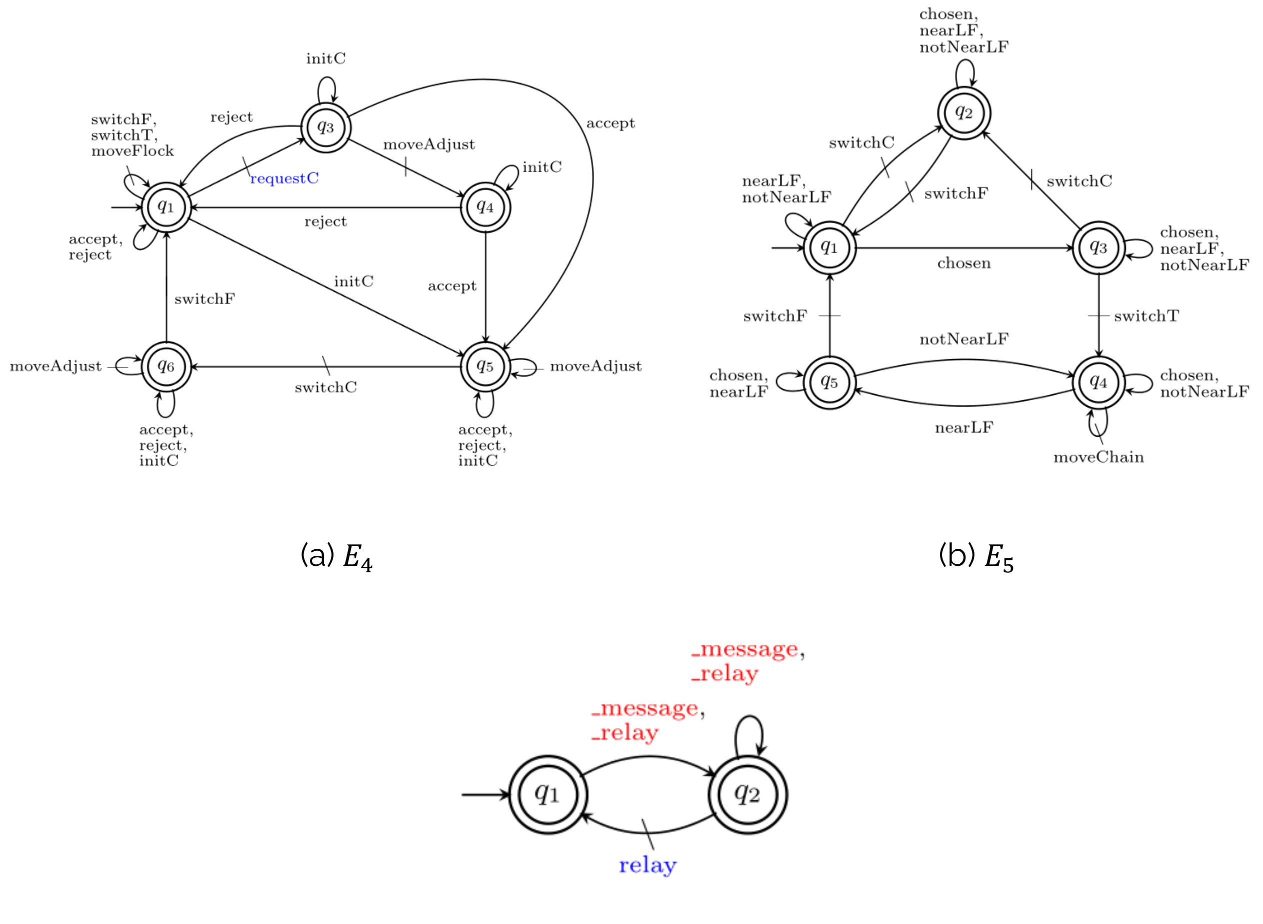




(b) E_2 (c) E_3

Figure 20. Specifications regarding (a) the worker's allowed actions in each role and (b, c) conditions that must be satisfied to switch roles.

Figure 21 shows the control specifications regarding the procedures to follow when switching roles. Specification $\boldsymbol{E_4}$ defines the steps for a follower to become a connector. Specification $\boldsymbol{E_5}$ defines the steps for a connector to become a follower. Specification $\boldsymbol{E_6}$ ensures a worker to relay the lead robot messages it has received.

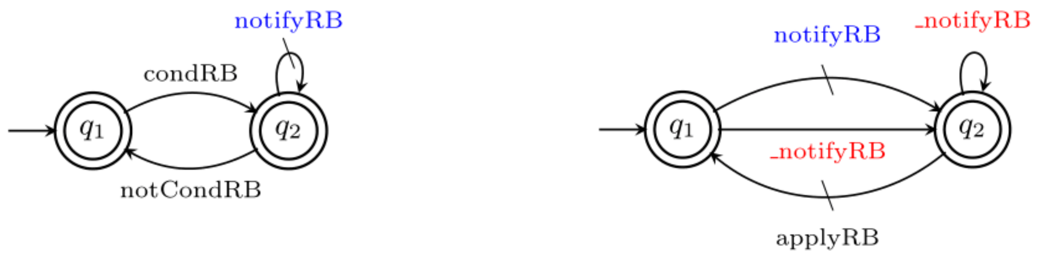


(a) E_4 (b) E_5

(c) E_6

Figure 21. Specifications regarding the steps for a follower (a) to switch to a connector, (b) to switch to a traveller, and (c) to relay messages it received.

Figure 22 shows the control specifications to apply the network modification. $\boldsymbol{E_7}$ defines that when a connector satisfies the branch repositioning conditions, it may broadcast a signal to the relevant connectors. $\boldsymbol{E_8}$ ensures that a connector may only reposition a branch in the network when it has either sent a signal (event `notifyRB`) or received a signal (event `_notifyRB`).



(a) E_7 (b) E_8

Figure 22. Specifications for a robot (a) to broadcast a message to relevant connectors when the conditions to modify its connections are satisfied and (b) to apply the network modification when it has either informed the neighbouring robots or was informed by a neighbouring connector.

The complete free behaviour models and control specifications can be found in the supplementary material [35]. The robot's controller source code is available online at [36].

Robot Motion

Each robot uses virtual forces to determine its direction of movement. Lead robots are attracted by the respective target locations while being repelled from nearby obstacles. Follower robots perform the flocking behaviour described in [31]. Connector robots move as described below.

We assume a range-and-bearing system for a robot to detect nearby robots relative to its position. Let $\boldsymbol{p}_{ij}$ denote robot $\boldsymbol{j}$'s position in robot $\boldsymbol{i}$'s local reference frame.

The virtual force acting on connector $\boldsymbol{i}$ is given by

$$\boldsymbol{u}_i = \boldsymbol{\alpha}_{c1}\boldsymbol{u}_i^{an} + \boldsymbol{\alpha}_{c2}\boldsymbol{u}_i^{at} + \boldsymbol{\beta}_c\boldsymbol{u}_i^{rn} + \boldsymbol{\gamma}_c\boldsymbol{u}_i^{ro}$$

where $\boldsymbol{\alpha}_{c1}$, $\boldsymbol{\alpha}_{c2}$, $\boldsymbol{\beta}_c$ and $\boldsymbol{\gamma}_c$ are positive scalars to weight the influence of the force components.

Force component $\boldsymbol{u}_i^{an}$ represents the attraction towards neighbouring robots within the network. It keeps the connectors and teams connected at all times, and results in individual connectors assuming positions that are right in between the nodes of their neighbourhood. It is defined as

$$\boldsymbol{u}_i^{an} = \frac{\mathbf{1}}{|\boldsymbol{\mathcal{N}}_i^{an}|} \sum_{j \in \boldsymbol{\mathcal{N}}_i^{an}} \boldsymbol{p}_{ij} \, \boldsymbol{\Phi}_j^{an}$$

$$\boldsymbol{\Phi}_j^{an} = \begin{cases} \dfrac{\mathbf{r}_{\text{com}} - \boldsymbol{r}_{\text{safe}}}{\mathbf{r}_{\text{com}} - ||\boldsymbol{p}_{ij}||}, & \textbf{if } ||\boldsymbol{p}_{ij}|| > \boldsymbol{r}_{\text{safe}} \\ \mathbf{1}, & \textbf{otherwise} \end{cases}$$

where $\boldsymbol{\mathcal{N}}_i^{an}$ denotes the set of neighbours of connector $\boldsymbol{i}$ within the network (which in the event of connector $\boldsymbol{i}$ being directly connected to the team includes the barycentre of visible team members), and $\boldsymbol{\Phi}_j^{an}$ is a modifier that controls the attraction to neighbour $\boldsymbol{j}$ from the default value (1) for distance $\boldsymbol{r}_{\text{safe}}$ to infinity for distance $\boldsymbol{r}_{\text{com}}$. This ensures that the connectivity within the network is maintained.

Force component $\boldsymbol{u}_i^{at}$ provides a connector $\boldsymbol{i}$ that is directly connected to a team with strong attraction towards it. This ensures the connectors follow the team (to the extent possible). A team moving away results in the connectors spreading out along the branch, helping to minimise the number of connectors. The force component is defined as

$$\boldsymbol{u}_i^{at} = \frac{\mathbf{1}}{|\boldsymbol{\mathcal{N}}_i^{at}|} \sum_{j \in \boldsymbol{\mathcal{N}}_i^{at}} \boldsymbol{p}_{ij} \, \boldsymbol{\Phi}^{at}$$

$$\Phi^{at} = \begin{cases} \max\left(0{,}1 - \dfrac{d_i^{max} - r_{\text{safe}}}{\Delta}\right) & \text{if } \mathrm{d}_\mathrm{i}^{\max} > \mathrm{r}_{\text{safe}} \\ 1, & \text{otherwise} \end{cases}$$

where $\mathcal{N}_i^{at}$ denotes the set of neighbours of connector i that belong to the team, and $d_i^{\max}$ denotes the distance to the furthest neighbouring connector. The attraction to the team neighbours diminishes as $d_i^{\max}$ approaches $r_{\text{safe}} + \Delta$.

Force component u_i^{rn} causes connector i to repel from all neighbouring robots. It is defined as

$$u_i^{rn} = -\frac{1}{|N_i^{an}|} \sum_{j \in N_i^{an}} \frac{\tau\sigma^4}{||p_{ij}||^5} \frac{p_{ij}}{||p_{ij}||}$$

where σ denotes a reference distance to maintain between the neighbours, and τ specifies the gain.

To enhance the robot's ability to avoid collisions with nearby obstacles, force component u_i^{ro} uses the reading values of proximity sensors. The e-puck [37] robots, which we use in our study, have a circular body of radius 3.5cm. The body is equipped with eight proximity sensors, which are distributed along the robot's circumference. Force component u_i^{ro} is defined as

$$u_i^{ro} = -\frac{1}{8d_{\max}}\sum_{j=1}^{8}(d_{\max} - d_j)\hat{v}_j$$

where $d_{\max} = 10\,\mathrm{cm}$ is the assumed range of the proximity sensors, d_j is the distance extracted from the j^{th} sensor and $\hat{v}_j$ is the unit vector pointing from the robot's centre to the j^{th} sensor. Where sensor j detects no object, we set $d_j = d_{\max}$.

Simulation Results

We conduct simulation trials using ARGoS [38]. In particular, we simulate the e-puck [37], which is a mobile differential-wheeled robot. We assume a range-and-bearing system with a range $r_{\text{com}} = 0.8m$ to communicate with neighbouring robots and a safety limit of $r_{\text{safe}} = 0.5m$. We use $\Delta = 0.1m$. The simulation physics and the robot control cycle are updated every 0.1s.

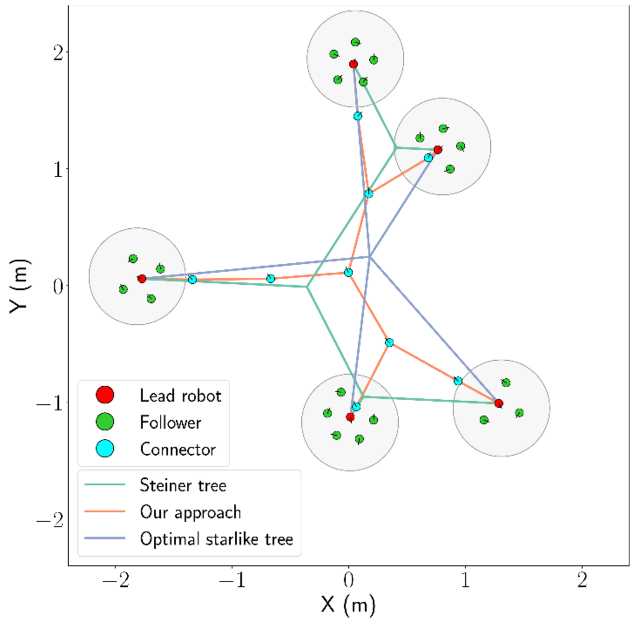
Network Length. To evaluate the length of the networks formed by our approach, we compare them against three alternative strategies:

- Centrally computed Steiner trees. These are guaranteed to be of minimal length [34]. To connect n points in the plane, they allow for additional points called *Steiner points*. The latter are nodes of degree three, which establish a Y-junction with the edges being $120°$ apart. We use the GeoSteiner solver [39], which requires exponential time, to compute the equivalent Steiner tree, using the final positions of the lead robots as the input.
- Centrally computed optimal starlike trees. These have a root node in the geometric median of the final positions of the lead robots, as computed by the iterative solver introduced in [40]. The lead robot's final positions are connected to the root node using line segments.
- Starlike trees produced by our approach when branch repositioning is disabled.

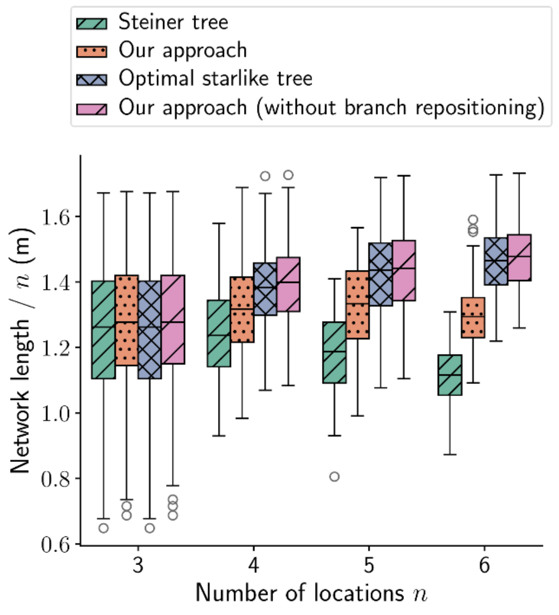
We investigate the performance for $n \in \{3,4,5,6\}$ target locations, which are uniformly randomly distributed in a circular arena of radius 2.8m. The swarm consists of n lead robots and $6n$ followers. Each lead robot was assigned a unique target location to visit. The simulation is terminated 40s after all lead robots reach their target locations or after 30 minutes, which ever occurred first. Trials were repeated 100 times for each n. To probe whether the recorded lengths for the different types of networks were significantly different, we conducted a repeated measures analysis of variance (ANOVA) with a p-value < 0.05.

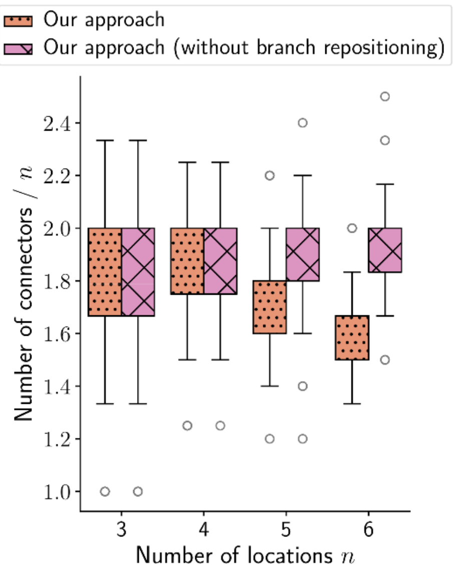
In all trials, the lead robots successfully reached the target locations while the robots formed a network connecting them. Figure 23a presents the network formed by our approach in a typical trial, and compares it to the idealised networks by the theoretical benchmarks (note that the latter disregard communication range constraints). One can observe that the optimal starlike topology features a dedicated branch per team, extending from the root node. Our approach and the Steiner tree create branches that are partially shared among certain teams, reducing total network length.

Figure 23b shows the average network length per team for different numbers of teams. For $n = 3$, the Steiner tree and starlike topology have the same network lengths as they produce identical networks. Our approach shows similar results. As n is increased, we see that our approach remains fairly consistent and is significantly better than the optimal starlike topology ($p < 0.001$). This can be attributed to the repositioning of branches, which allows branch segments to be shared by multiple teams. This advantage enables the Steiner trees to even reduce the network length per team as n becomes larger. By contrast, the per team length of the optimal starlike networks increases with n.



(a)





(b) (c)

Figure 23. Comparison of the lengths of networks. (a) Robots at the final timestep of a simulation trial. The grey circles represent target locations. The lines show the resulting network from our decentralised approach (orange), the optimal solution

found by a central GeoSteiner solver [39] (green) and the optimal starlike topology found by using as root node the geometric medians of lead robots (blue). (b) Average network length per team. Each team initially consisted of 7 robots (1 lead robot and 6 followers). Each number of target locations n was tested for 100 trials. (c) The number of connectors (i.e. robots used to maintain the network) with and without branch repositioning.

Number of Connectors. We evaluate the extent to which the repositioning of branches reduces the number of connectors. Figure 24 shows the average number of connectors per team when using our approach with and without the ability to form shared branches. Note that the latter approach results in starlike topologies. The number of connectors per location is identical for $n \leq 4$. For $n > 4$, our approach reduces the number of connectors per location. By reducing the number of connectors in the network, more robots become available at the target locations.

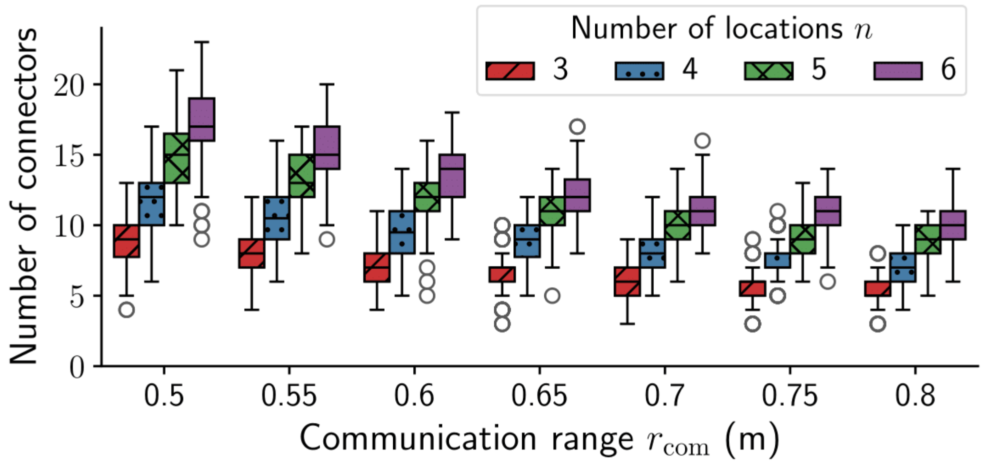


Figure 24. Number of connectors maintaining the network for different communication ranges. Each team initially consisted of 13 robots (1 lead robot and 12 followers). Each configuration was tested for 100 trials.

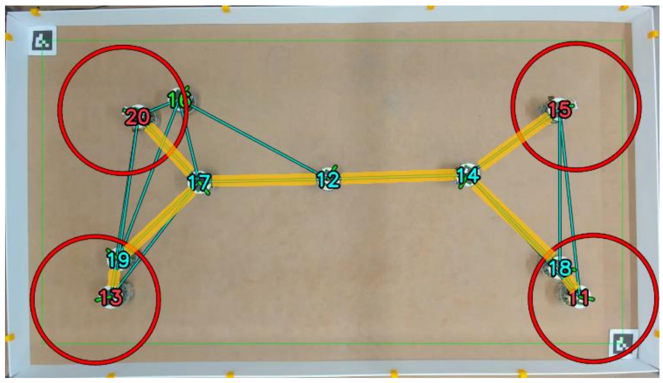
Impact of Communication Range. When the lead robots arrive at the target locations, only the necessary number of connectors should form the network to allow as many as possible robots to be available for other activities. We further examined different communication ranges $r_{\text{com}} = \{0.5, 0.55, \ldots, 0.7\}$ for n lead robots and $12n$ followers to quantify the effect on the number of connectors. Figure 24 shows that as r_{com} increases, the number of connectors decreases, showing that the swarm was able to take advantage of the longer communication range and maximise the number of robots at the target locations.

Real-Robot Validation

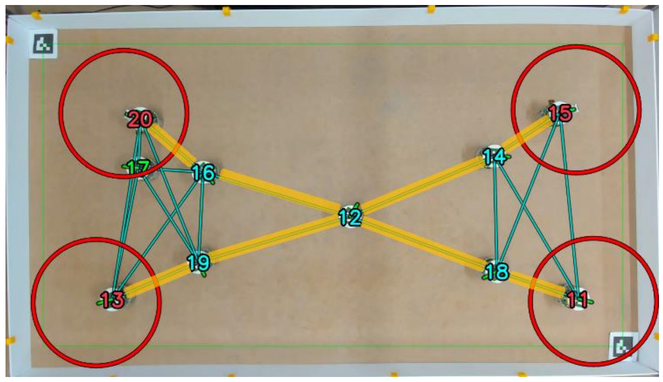
We conduct experiments using 10 Pi-pucks [41] in a bounded 2m×1m arena to validate our approach in the real world. The robots have a virtual range-and-bearing sensor with range $\boldsymbol{r}_{\mathrm{com}}$ = 0.6m. We randomly placed two targets in $[\mathbf{-1},\ \mathbf{-0.5}]\times[\mathbf{-0.5},\ \mathbf{0.5}]$ and a further two in $[\mathbf{0.5},\ \mathbf{1}]\times[\mathbf{-0.5},\ \mathbf{0.5}]$. Four lead robots moved towards unique task locations, while the remaining robots formed the network. An overhead camera was used to track the position and orientation of the robots through unique ArUco markers attached to the top of each robot [42]. We tested 10 trials with and without branch repositioning. Trials were terminated after 30s.

Results

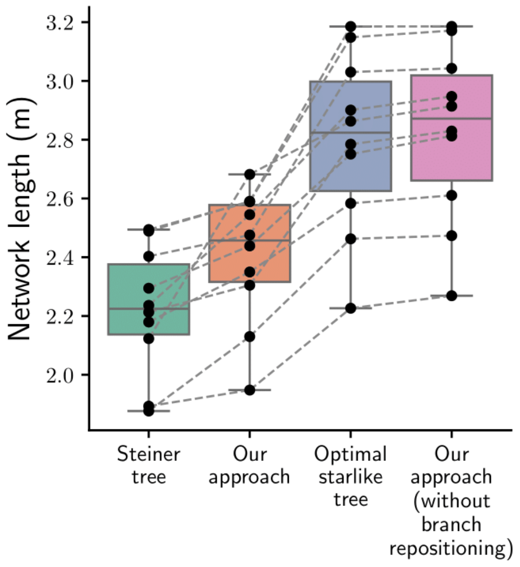
Figure 25a—c show the networks formed by the robots where repositioning of branches was enabled or disabled, respectively. The overall network length using our approach was 2.48m, while the starlike topology without repositioning the branches was 3.04m. Figure 25c shows a summary of all trials. Our approach formed networks that were significantly shorter than optimal starlike topologies, and on average only 10.5% longer than minimal-length networks as derived from the centrally computed, idealist Steiner trees.



(a) Our approach



(b) Starlike tree



(c)

Figure 25. Real robot validation results. (a--b) Robots at the final timestep where the repositioning of branches is enabled (a) and disabled (b), respectively. The numbers represent robot IDs, and the red, green and cyan colours represent their roles: lead robot, follower, and connector. Red circles represent locations of interest. Orange lines are overlaid between robots to indicate the connections that robots are maintaining. (c) Total network length for our approach with (orange) and without (pink) branch repositioning, the Steiner tree (green), and the optimal starlike tree (blue). Each configuration was tested for 10 trials. Paired results are joined by a dashed line. Video recordings of all the trials are provided in the supplementary material [36].

Summary

We presented a decentralised controller for a swarm of robots to autonomously establish minimal-length networks among two or more locations of interest. The controller creates networks of tree topologies, and can reposition individual branches within the tree where this results in an overall reduced network length. We validated our approach using embodied simulations and real robot experiments. The robots formed networks that were significantly shorter in length than centrally computed optimal starlike trees and compared reasonably well against centrally computed Steiner trees, which are known to be of optimal length. Future work will explore lead robots that adapt to changing locations of interest, and consider environments that comprise obstacles.

Swarms with human-in-the-loop

Robot swarms are large groups of loosely coupled robots that accomplish tasks using local interaction [43]. With their inherent robustness and scalability, robot swarms are a promising technology for assisting in complex missions from search-and-rescue to infrastructure inspection and repair. As the push to bring robot swarms into the real world continues, it is important to consider how we leverage the autonomy of swarms to support humans. Recent advances in human-swarm interaction have shown that it is possible for a single human operator to control a swarm of robots using methods such as teleoperation [44], proximal interactions via gestures [45], [46], [47], augmented reality [48], and virtual reality systems [49].

When a robot swarm splits into smaller sub-swarms, it becomes challenging for a single operator to control the sub-swarms without exceeding their own cognitive capacity [50]. Furthermore, if working alongside the robots, a single operator will not always be able to supervise all sub-swarms through proximal interaction, as each sub-swarm could be working at a different location. These problems could be addressed by having multiple operators who each interact with a different sub-swarm. This would allow the operators to share the required cognitive load to potentially achieve more complex tasks and improve productivity.

Introducing multiple operators requires careful design of the system. One factor affecting system performance is the relationship between humans and robots [51]. Prior research on multi-operator control of robot swarms can often be categorized into either (1) statically allocated robots (i.e. a subset of robots are assigned to each operator) or (2) shared robots (i.e. all operators can interact with all robots at any time) [52]. Statically allocated robots are beneficial when the tasks can be completed by the operators independently [53], [54], but do not have the flexibility to swap robots between the initially allocated teams. Shared robots give individual operators some flexibility on how many robots to use on a given task [55], [56], but operators may struggle from ``diffusion of responsibility'' due to all the robots being shared [57].

Another factor that affects the performance is communication, which is often considered a prerequisite for operators to cooperate effectively [58]. Poor communication quality can hinder this cooperation as it becomes difficult to maintain a shared mental model of the situation. In addition, excessive communication can become an overhead for the operators, resulting in increased cognitive workload [59]. Implicit or indirect communication (e.g. via a computer interface) can be effective when undertaking highly interdependent tasks, provided that the members have a sufficient understanding of the situation [60].

A further factor is the user interface. Achieving a high level of transparency is important, especially when operators are using the swarm as a shared resource [61]. Information such as the robots' position and internal states can be helpful in understanding the state of the swarm. However, it may not be possible to obtain such information from all the robots of the swarm, and in practical scenarios, the operators may have to work with the information that is available locally on a single robot. Having such restricted situational awareness can have a detrimental effect on the operators' ability to interact with a swarm [62], making it challenging for them to collaborate.

In [31], we proposed a framework based on supervisory control theory [4], [19], [21] whereby a swarm of robots solves spatially distributed tasks while maintaining overall connectivity. The swarm was accompanied by two leader agents who directed the robots towards specific task areas. However, the behaviour of the leader agents was hard-coded: no human operators were involved.

In this section, we conduct a user study to examine the ability of multiple human operators to dynamically share the control of robot swarms. To understand if the sharing of robots can be used effectively, we compare the performance of a pair of operators in completing a set of spatially distributed tasks with and without the ability to share robots. As sharing robots requires some level of coordination between the operators, we also investigate how either direct or indirect communication affects performance (see Figure 26). To conduct the user study, we extend our framework [31] with a novel user interface that allows each operator to drive the respective leader agent of their team while being provided with its local camera feed, and request from or send robots to the other team. Our study is in contrast to previous work on multi-operator control of robot swarms as it (i) considers the case that the operators have dedicated teams and can exchange robots between them and (ii) restricts each operator's situational awareness by providing only a local view of the environment.

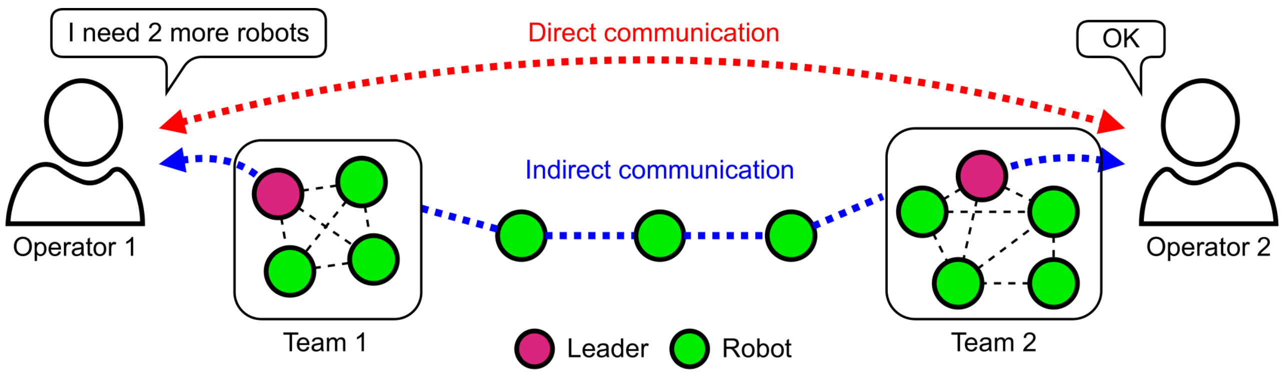


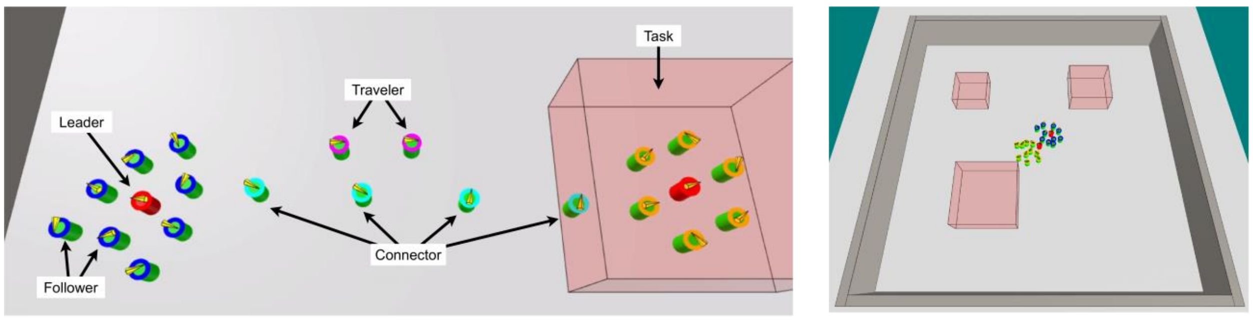
Figure 26. Illustration of two human operators each controlling a subset of the robot swarm (e.g. a team) via a leader agent. Other robots maintain connectivity between the two teams. An operator can request robots from the other operator either directly (i.e. verbally communicating) or indirectly (i.e. sending a message through the swarm).

First, we describe the system setup, user interface, and experimental design. Next, we present the results obtained from the experiments, followed by a discussion.

Scenario

Our study investigates the ability of two human operators to control a swarm of robots to complete a set of tasks scattered across a bounded environment with no obstacles. We employ the robot behaviours from our previous work on connectivity-preserving robot swarms [31]. In this framework, each robot chooses its actions based on a formal model designed using supervisory control theory [4], [19], [21]. We consider two types of agents: leader and worker. Each operator controls a *leader*, allowing them to move through the environment and provide instructions to the robots. In a real-world scenario, the leader agent could either be the operator themself, a portable device carried by the operator, or a robot controlled remotely by the operator.

The *workers* are robots that are capable of completing tasks. At any moment in time, each worker is assigned to at most one leader. Throughout the mission, the workers autonomously maintain connectivity between the two teams by forming a robot chain. The workers can either assume the roles of *follower*, *connector*, or *traveller* as seen in Figure 27a. Here, we summarize the robots' behaviour in each role.



(a) (b)

Figure 27. Overview of simulation scenario. (a) Leaders and workers are represented as red and green cylinders respectively; yellow arrows above the robots represent their headings. The workers can assume three roles (indicated by the colour of their LED ring): followers (blue or orange) are part of a team, connectors (cyan) maintain connectivity among teams, and travellers (magenta) switch from one team to another. The task to complete is shown as a red transparent box. (b) Overhead view of an example arena containing three tasks. Tasks with a larger area require more

followers to complete. Once a task is completed, it is removed from the arena and a new task appears at a random position.

Follower Behaviour. A worker assigned to a leader is a follower. A follower performs a flocking motion based on virtual forces. It is attracted towards the leader and other followers within the same team and repulsed from any other leaders or workers. When inside the same task area as the leader, the follower is considered to be working on that task. When the distance to the other team increases, followers will leave the team, one at a time, to become a connector to maintain connectivity between the two teams by forming a robot chain.

Connector behaviour. A worker maintaining the robot chain is a connector. They allow the two teams to exchange information with each other. Unlike in [31], the connectors minimize the length of the chain by maintaining a straight line between the two teams. This is done by each connector finding the distance and angle to its two adjacent neighbours in the chain and moving to the middle point between these two neighbours. If the two teams move toward each other, the robot chain becomes shorter; the connector closest to the team leaves the chain to join the team.

Traveler behaviour. The leaders are able to send a specified number of their followers to the other leader. When a follower receives a signal from its leader to join the other team, it becomes a traveller. A traveller moves along the robot chain until it reaches the other team to become a follower of that team. This allows the operators to rebalance the number of followers in each team.

Each task requires a specific number of followers to complete, which is visible to the operator when inside the task area. When outside, the operator can make a guess, as the task areas have lengths and widths of 0.4m, 0.5m, 0.6m, 0.8m, or 1.0m, for tasks requiring 1, 3, 6, 9, or 12 followers, respectively. This resembles a search-and-rescue scenario in the real world; a rescuer might be able to predict the size of a task, but may only find out the exact number of robots or resources needed to accomplish it after arriving at the mission site. To complete a task, the leader and its followers must remain inside the task area for a certain duration. The time required to remain inside the task area increases linearly to the number of followers needed to complete it, and having an excess number of followers does not make the task complete faster. In the extreme cases considered---single-robot and 12-robot tasks---the followers need to remain for 5s and 30s, respectively. Completing a task awards the pair of operators a score equal to the number of followers required to complete it. Once a task is completed, it is removed and a new task appears at a uniformly random location without overlapping with existing tasks.

The operators' joint objective is to score as many points as possible during the trial. The operators must guide their followers to the task areas, while also monitoring the number of followers in their team to ensure they have enough to complete the tasks. The full robot controller models can be found in the supplementary material [63].

Robot and Simulation Platform

The operators interact with the simulated robots running in the ARGoS simulator [38]. The simulation consists of 2 leaders and 20 workers in a 4m×4m arena (see Figure 27b). We use the e-puck [37] for our study, which is a mobile differential-wheeled robot with a diameter of 0.07m. We use a maximum speed of 8cm/s. The e-puck has eight proximity sensors with a range of 0.1m distributed around its body. We assume a range-and-bearing system with a range of 0.8m to communicate with neighbouring robots.

User Interface

The operators interact with the simulated robot swarm through a custom graphical user interface based on Webviz [64], a web interface plugin for ARGoS [38]. Webviz uses a client-server architecture, allowing multiple users to simultaneously interact with the same simulation from different devices (see Figure 28).

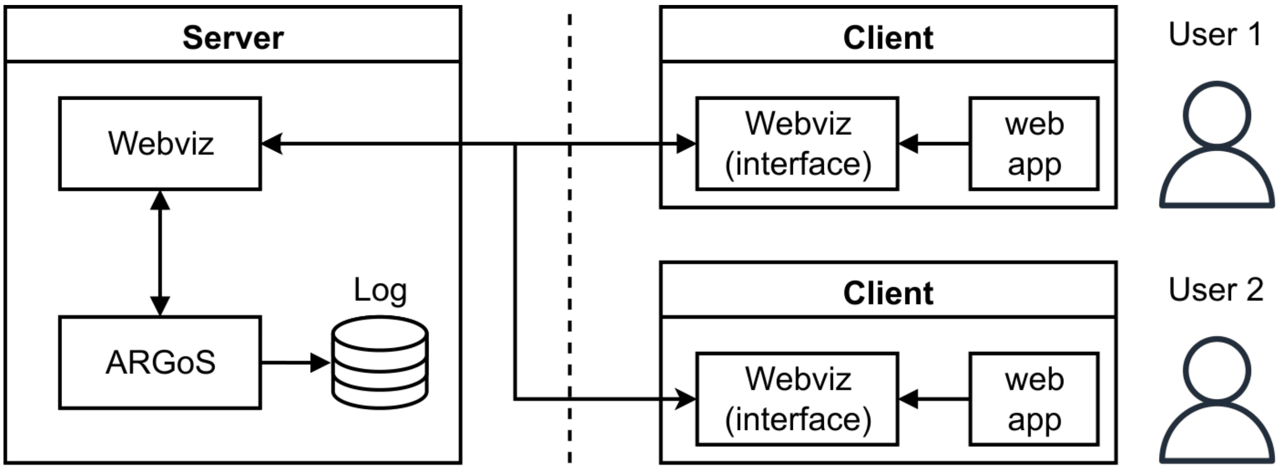
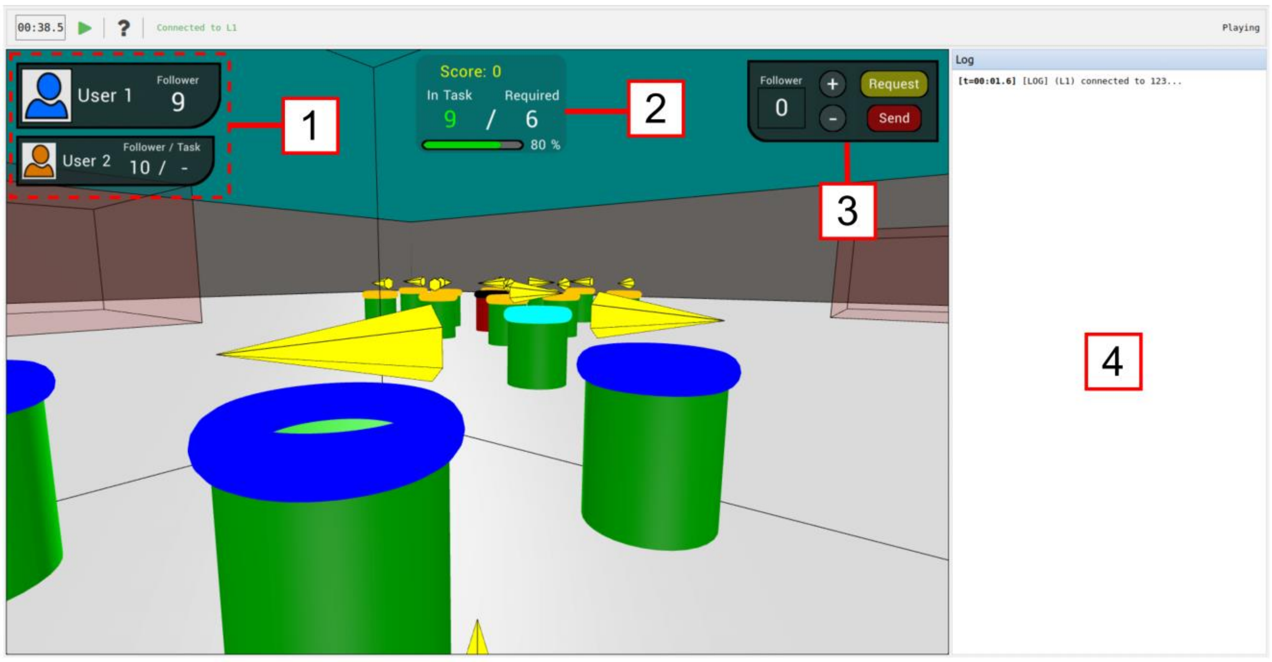


Figure 28. System overview. A user accesses the shared swarm simulation through a simple web-based app. When the user interacts with the interface, the inputs are sent to the server and reflected in the ARGoS simulation. The client-server architecture is realised using Webviz.



```
[t=00:01.6] [LOG] (L1) connected to 123...
```

Figure 29. The interface used for the experiment. The operator is shown a first-person perspective of the leader situated in the simulated arena. The interface displays (1) the team information panel, (2) the task information panel, (3) the request-and-send panel, and (4) the log panel. No overhead view or mini-map of the arena is provided to the user.

Figure 29 shows the interface presented to a user. It shows a first-person view of the environment from the leader's perspective. The interface consists of four panels that support the user in completing the tasks; *team information panel*, *task information panel*, *request-and-send panel*, and *log panel*. No overhead view of the environment is provided. The users can drive the leader inside the simulation using a keyboard.

Team information panel. This panel provides information for the user to determine whether it or its partner has enough followers to complete their respective tasks. It is placed top-left in the interface and shows information about the two teams. The panel consists of two parts, which display the number of followers in the user and partner's team, respectively. In addition, if the partner is inside a task area, it also displays the size of that task (i.e. the number of followers needed) next to the partner's follower count.

Task information panel. This panel provides information about the current task and is placed top-centre in the interface. When a user is inside a task area, it displays the size of the task, the number of followers currently inside the task area, and a progress bar that fills up when the followers are performing the task. The team score is also displayed at the top, which updates when either of the users completes a task. If the user moves outside of the task area, information about that task will no longer be displayed until the user re-enters the task area. Any progress made towards a task will remain even if a user exits the task area before it is completed.

Request-and-send panel. This panel is used to request or send followers to the partner and is placed top-right in the interface. A user can specify the number of followers using the plus and minus buttons and then press either the request or send buttons to confirm the action. If the `Request` button is selected, a message containing the specified number of followers will be sent to the partner via the robot chain. If the `Send` button is selected, the specified number of followers from the user's team will begin to travel to the partner's team. This allows the users to explicitly share their followers without needing to communicate with each other verbally.

Log panel. This panel displays the request and send messages received from the partner as well as any messages sent by the user, in chronological order. It is placed on the right side of the interface. By monitoring the log panel, a user can keep track of their partner's request or confirm that their partner has sent followers to their team.

Experiment Design

Our experiment followed a 2x2 mixed factorial design [65], where the robot-sharing condition (robot-sharing [RS] vs. no-robot-sharing [NRS]) was the within-subjects factor and the communication type (direct [DIR] vs. indirect [IND]) was the between-subjects factor. Within-subject evaluations assessed whether the dynamic sharing of robots affected the performance and human factors. Between-subject evaluations compared whether differences in communication style affected the performance and human factors.

In the RS condition, the operators were able to request or send robots to each other using the interface. If an operator found that it did not have enough followers to execute the task, they could request followers from their partner. In the NRS condition, the operators could not share robots. This meant some tasks required more robots than were available in a single team. In such cases, the task could still be completed if both teams entered the task area and the sum of followers reached the required number of followers for the task.

In DIR communication, the operators verbally communicated with their partners throughout the trial. They were seated nearby but were unable to see each other's computer screens. To share robots, an operator verbally requested followers from their partner. Only the `Send` button was displayed as the operators directly asked their partner for followers. In IND communication, the operators were not aware of who their partners were (and could not see their respective screens) because they were randomly paired. They were not allowed to verbally communicate: they could only request a specific number of followers via the interface. The operator that received a request, either verbally or through the interface, then decided how many followers to send.

Performance Measures. Performance measures include *task score*, *distance travelled*, *team separation*, and *robots shared*. These data were obtained from the simulation logs, which recorded the position and state of every robot and task, and all inputs received from both operators. Task score is the points obtained during the trial. If a task was being performed when the trial ended, we awarded partial points for the proportion of the task that was completed. Distance travelled is the total distance moved by the leaders and workers during a trial. It is used here as a proxy for the total energy consumed by the swarm. Team separation is the average distance between the two teams throughout the trial. The separation was calculated by finding the geometric centre of both teams and taking the distance between the two points. This tells us how spread out the two teams were during the trial. For the RS condition, we recorded the number of followers shared between the teams.

Subjective Questionnaires. Subjective data were obtained individually through a paper-based questionnaire after each trial. The participant's subjective workload was obtained using the NASA-TLX [66] on a 7-point Likert scale. Subjective situational awareness was obtained using the Situational Awareness Rating Technique (SART) [67] on a 7-point Likert scale. We also asked participants about how well they understood their partner or the robots' actions, the usability of the interface, and whether they found the ability to share robots useful on a 7-point Likert scale. The questionnaire concluded with open-ended questions asking about what their strategy was in completing the tasks and any additional comments they had about the experiment. The questionnaires used in the experiment can be found in the supplementary material [63].

Participants. The study received ethical approval from The University of Sheffield. All participants were staff and students within the university. A total of 52 participants (34 males, 18 females) completed the experiment. All participants were adults over 18 and belonged to one of the age groups below (*Under 20*: 1, *20-29*: 28, *30-39*: 14, *40-49*: 5, *50-59*: 3, *60 & Over*: 1).

Procedure. First, the participants completed a preliminary questionnaire that asked about their previous gaming experience. Next, the experimenter gave an introduction on how to use the interface and explained the participant's goal in the trials. This was followed by a short training session for the participants to familiarize themselves with the robots' behaviours and the functionality of the interface.

For the main trials, participants were randomly paired to work together. The pair of participants each interacted with the robot swarm through the interface to score as many points as possible until the trials ended. The robot-sharing condition was controlled by enabling or disabling the request-and-send panel in the interface. The communication type was controlled by assigning the pair to either DIR or IND communication. Participants completed two trials to experience both robot-sharing conditions (RS and NRS) within their assigned communication type (DIR or IND).

The order of the robot-sharing conditions were counterbalanced across participants to minimize any learning effect. Each trial lasted for 10 minutes. After each trial, the participants individually completed a post-trial questionnaire. Participants took a short break before moving on to the second trial. The overall experiment took around 75 minutes. An accompanying video illustrating the user study, the simulation, and the user interface can be found here: https://youtu.be/hY2q7QB9Dgw.

Results

We performed quantitative and qualitative analyses to examine the effect of the robot-sharing capability and communication type between two operators. Quantitative analyses were based on the simulation logs, which recorded the states and positions of robots and tasks, the points scored, and the user inputs to the interface at each time step. Qualitative analyses were based on questionnaire responses, which asked the participants to rate their experience on a set of scales. Table 1 summarizes the results. Two-way mixed analysis of variance (ANOVA) were conducted to examine the effect of the two factors. Results are reported as significant when $p < 0.05$ and marginally significant when $p < 0.08$.

Table 1. Summary of performance measure and questionnaire results. N is the number of samples and SD is the standard deviation.

Measure	Cond.	Comm.	N	Mean	SD
Task Score	RS	DIR	14	100.39	16.84
		IND	12	92.25	16.71
	NRS	DIR	14	106.31	13.97
		IND	12	99.20	16.19
Distance Traveled (m)	RS	DIR	14	10.28	1.44
		IND	12	10.51	1.22
	NRS	DIR	14	10.79	1.59
		IND	12	11.54	1.29
Team Separation (m)	RS	DIR	14	1.42	0.21
		IND	12	1.48	0.14
	NRS	DIR	14	0.96	0.17
		IND	12	1.00	0.18
Robots Shared	RS	DIR	14	22.79	11.19
		IND	12	25.33	7.23
	NRS	DIR	—	—	—
		IND	—	—	—
Workload	RS	DIR	27	3.62	1.10
		IND	24	3.56	1.04
	NRS	DIR	28	3.42	1.00
		IND	24	3.18	1.09
Situational Awareness	RS	DIR	28	4.44	0.78
		IND	23	4.49	0.76
	NRS	DIR	28	4.36	0.65
		IND	24	4.14	0.72

A comparison between the communication types indicates that there was a significant difference in task score ($F(1,23) = 5.36, p = 0.030$). Figure 30 shows that participants in the DIR communication scored more points ($M = 103.35, SD = 15.48$) than those in the IND communication ($M = 95.73, SD = 16.48$). No significant difference was found in the points scored between the robot-sharing conditions ($F(1,23) = 2.72, p = 0.112$).

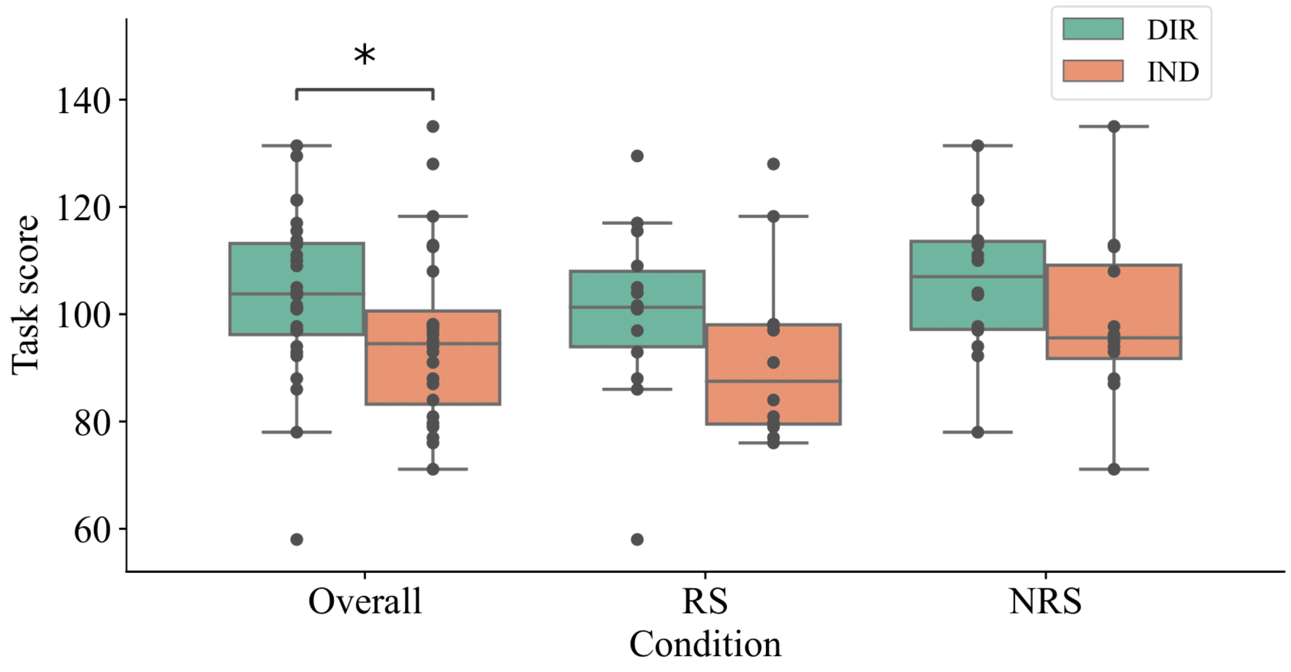
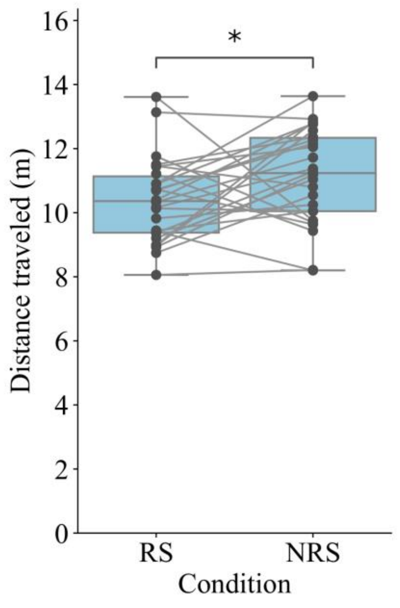
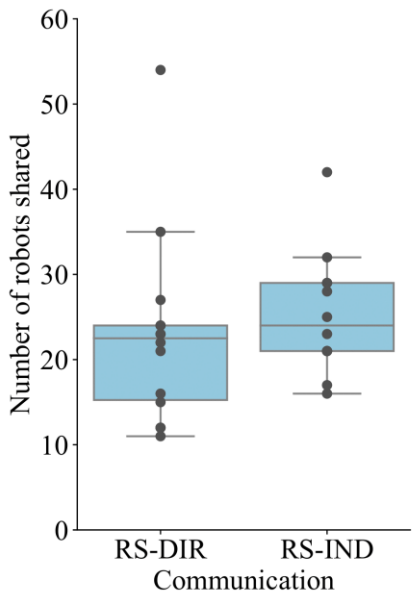


Figure 30. Comparisons of task score by robot-sharing condition and communication type. Statistically significant results ($p < 0.05$) are indicated with an asterisk ($*$).

The ability to share robots had a significant effect on the total distance travelled by the robots ($F(1{,}23) = 4.53, p = 0.044$). Figure 31a shows the total distance travelled in the RS and NRS conditions. Since each pair of participants experienced both conditions, a line is drawn to show the paired data. On average, participants in the RS condition ($M = 10.39, SD = 1.32$) traveled less than in the NRS condition ($M = 11.14, SD = 1.48$), and therefore consumed less energy. No significant effect was found on the distance traveled between communication types ($F(1{,}23) = 0.62, p = 0.441$).



(a)



(b)

Figure 31. Comparisons of (a) the total distance travelled by the robots against the robot-sharing condition and (b) the total number of robots shared between the two teams in the RS condition against the communication type. Paired samples are joined by a line. Statistically significant results ($p < 0.05$) are indicated with an asterisk ($*$).

The ability to share robots also had a significant effect on team separation ($F(1,23) = 86.2, p < 0.0001$). Figure 32 shows that participants in the RS condition ($M = 1.45, SD = 0.18$) were more likely to stay apart from the other team than in the NRS condition ($M = 0.98, SD = 0.18$) throughout the trial. There was no reportable difference in team separation between communication types ($F(1,23) = 0.98, p = 0.333$).

Within the RS condition, we examined if the communication type affected the number of robots shared between the participants. Figure 31b shows that there was no significant difference in the number of robots shared $(F(1,24) = 0.46, p = 0.505)$.

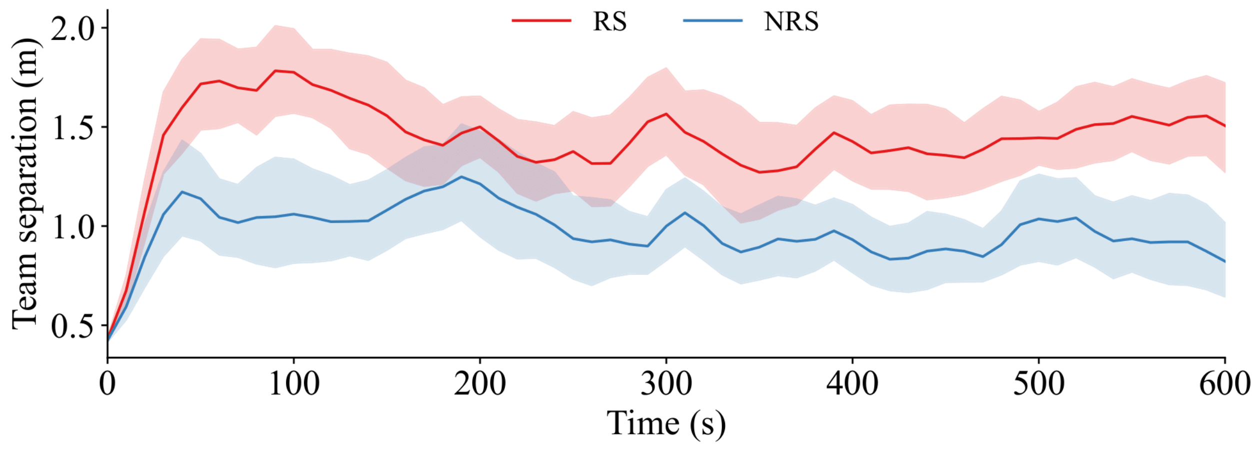


Figure 32. Average separation between the two teams under different robot-sharing conditions. The solid lines each represent the mean across 26 trials and the transparent regions represent the 95% confidence intervals.

To compare the overall workload, we calculated the mean of the raw scores from the six categories in the NASA-TLX questionnaire. The ability to share robots had a marginally significant effect on the global workload ($F(1{,}49) = 3.82, p = 0.056$). Participants in the RS condition ($M = 3.59, SD = 1.06$) reported a slightly higher global workload than in the NRS condition ($M = 3.31, SD = 1.04$). The overall situational awareness was also calculated using the mean of the raw scores from the nine categories in SART. The participants' global situational awareness was not affected by either communication type ($F(1{,}49) = 0.18, p = 0.673$) or the ability to share robots ($F(1{,}49) = 3.09, p = 0.085$).

Discussion

In all trials, every pair of participants were able to complete tasks and dynamically share robots, despite undergoing only a short training period (i.e. up to 10 minutes). Figure 33 shows that the lack of direct communication caused the percentage of participants understanding their partner's actions to drop from 89.3% to 81.3%. Despite the lack of direct communication and understanding of who their partners were (of the people in the room), on average, participants in the IND communication were able to score 92.6% of the points achieved by those using DIR communication. Overall, these results are in line with previous work, where the operators benefit from direct communication for coordinating their actions.

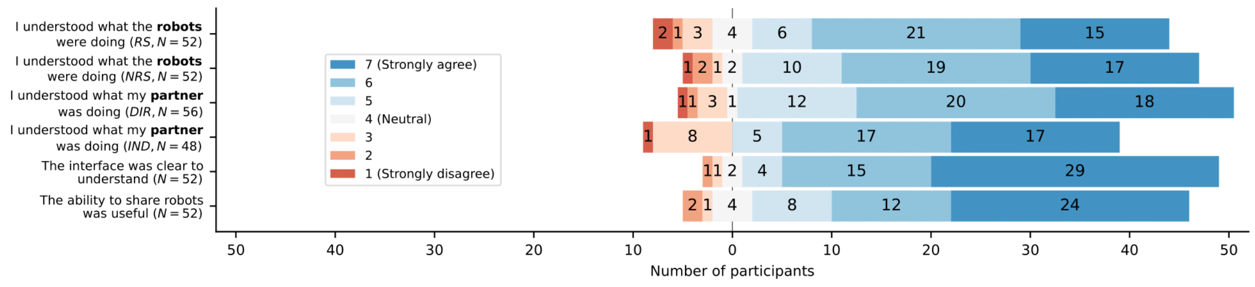


Figure 33. Post-trial questionnaire responses. Participants were asked to rate how well they understood their partner and the robots' actions, the interface, and whether they found the ability to share robots useful after the trial.

The ability to share robots (i.e. RS condition) decreased the total distance travelled by the robots, and hence the energy consumed by the swarm. This could be attributed to participants more likely focusing on different tasks at the same time. If the partner requested some followers, the participant could simply send the requested amount of followers, or even more if they choose to do so. By contrast, when participants were unable to share robots (i.e. NRS condition), they had to bring their teams into the same area if the task required many followers. This resulted in increasing the average distance travelled by each robot in the NRS condition.

Participants in the NRS condition were also hesitant to tackle larger tasks compared to those in the RS condition. Some participants reported that when they did not have the required number of followers to complete a task, they preferred to move on to another task when the other team was far away. This meant the teams in the RS condition were often exploring new task areas earlier on, as their ability to share robots provided added flexibility to meet the task requirements. This could be crucial during search-and-rescue missions.

Although the teams were faster at reaching the tasks in the RS condition, the task scores were slightly better in the NRS condition. This could be explained by the larger team separation seen in the RS condition. The larger team separation meant there were fewer followers in each team. This resulted in some participants spending more time waiting for the requested followers to arrive instead of working on the tasks. The best performing participants were those who prioritized tasks that required the same or slightly fewer followers than their current number of followers in the team. These participants were able to minimize the waiting time and obtain higher scores. The limiting factor in this study was the number of workers available to the participants compared to the size of the tasks. Increasing the number of workers may have increased the performance in the RS condition.

It was expected that participants in the RS condition would experience a higher workload than those in the NRS condition due to the addition of the robot-sharing capability. While this trend was observed, results indicated only a marginal significance, where the RS condition had a slightly higher workload. This suggests that the dynamic sharing of robots between operators could be incorporated into human-swarm systems without having a significant effect on operator workload. According to the questionnaire responses in Figure 33, most participants found the ability to share robots useful, even among those that scored fewer points in the RS condition. Several participants expressed that they found the ability to share robots useful not only because they could adjust the number of followers in the two teams, but because it allowed them to work more independently on the tasks and potentially increase their performance by executing tasks in parallel. This behaviour to work independently matches with the increased team separation observed in the RS condition.

Summary

We conducted a user study to investigate the effects of dynamically sharing robots between two human operators on task-related performance measures and human factors. The operators were each provided only a local, first-person view of the simulated environment, and were accomplishing tasks, starting with preassigned teams of robots. They were able to request and share robots with each other either by communicating directly (i.e. verbally) or indirectly via simple messages that passed through the robot swarm. Our findings show that, despite the short training received, the operators were able to dynamically share robots under limited situational awareness. Although sharing robots did not necessarily increase task scores, it provided the operators the flexibility to work independently or collaboratively, reduced the energy consumed by the swarm (i.e. the total distance travelled), and was considered useful by the operators. Regarding the communication type, a decrease in task scores was noted when verbal communication was prohibited. Further training could help operators understand when are effective moments to request or send robots to further improve task scores.

For future work, we plan on conducting the user study with physical robots to understand whether these findings generalize to more realistic scenarios. In many scenarios, the robots will have to navigate through environments that are densely populated by humans, or other dynamic obstacles, for example, when operating in shopping malls and plazas, factory floors or warehouses. To further our understanding of the underlying challenges, we conducted a study investigating the benefits of robots to organise into logic sub-groups – platoon formations – for navigating such crowded environments. This includes adaptive strategies that can switch between cooperative and independent strategies.

Conclusion

This document describes the work carried out in the context of Task 4.3, which has focused on the development of compilers that produce correct-by-construction adaptive swarm controllers and their application in practice. By using a swarm compiler, the capabilities and specifications of each device in the swarm were automatically synthesised to generate source code that could be executed directly on different types of robots. Two use cases involving heterogeneous swarm of robots that ran control logics produced by the swarm compilers were presented. The first use case showcased a swarm of robots split into teams performing work at distinct task locations. Some of the robots dynamically formed a mobile ad-hoc network ensuring the connectivity between all teams at all times. The length of these networks was shown to be near optimal. The second use case showcased the robot swarm dynamically engaging with human operators who guided the teams to task locations. Moreover, the robots responded to requests by the operators to exchange robots between teams. In both use cases, the compiled swarm controllers were executed on virtual machines which are explained in more detail in Deliverable 4.4.

Appendices

- Supplementary material for swarms with global performance requirements
  - https://doi.org/10.15131/shef.data.25382908
- Supplementary material for swarms with human-in-the-loop
  - https://doi.org/10.15131/shef.data.21088627

Bibliography

[1] H. Hamann, *Swarm robotics: A formal approach*. Springer, 2018.

[2] D. Bozhinoski and M. Birattari, ‘Enhancing the technological maturity of robot swarms’, in *2023 IEEE/ACM 5th International Workshop on Robotics Software Engineering (RoSE)*, May 2023, pp. 5–8. doi: 10.1109/RoSE59155.2023.00006.

[3] M. Dorigo, G. Theraulaz, and V. Trianni, ‘Reflections on the future of swarm robotics’, *Sci. Robot.*, vol. 5, no. 49, Dec. 2020, doi: 10.1126/scirobotics.abe4385.

[4] Y. K. Lopes, S. M. Trenkwalder, A. B. Leal, T. J. Dodd, and R. Groß, ‘Supervisory control theory applied to swarm robotics’, *Swarm Intell.*, vol. 10, no. 1, Art. no. 1, Mar. 2016, doi: 10.1007/s11721-016-0119-0.

[5] M. Webster *et al.*, ‘A corroborative approach to verification and validation of human–robot teams’, *Int. J. Robot. Res.*, vol. 39, no. 1, pp. 73–99, Jan. 2020, doi: 10.1177/0278364919883338.

[6] E. M. Clarke, *Model checking*. Cambridge, MA ; London: MIT Press, 1999.

[7] J. Chen, H. Wang, M. Rubenstein, and H. Kress-Gazit, ‘Automatic Control Synthesis for Swarm Robots from Formation and Location-Based High-Level Specifications’, *2020 IEEERSJ Int. Conf. Intell. Robots Syst. IROS*, vol. 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 8027–8034, 2020.

[8] M. Brambilla, A. Brutschy, M. Dorigo, and M. Birattari, ‘Property-Driven Design for Robot Swarms: A Design Method Based on Prescriptive Modeling and Model Checking’, *ACM Trans. Auton. Adapt. Syst.*, vol. 9, no. 4, p. 17:1-17:28, Dec. 2014, doi: 10.1145/2700318.

[9] G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, and M. Birattari, ‘AutoMoDe: A novel approach to the automatic design of control software for robot swarms’, *Swarm Intell.*, vol. 8, no. 2, pp. 89–112, Jun. 2014, doi: 10.1007/s11721-014-0092-4.

[10] G. Francesca *et al.*, ‘AutoMoDe-Chocolate: automatic design of control software for robot swarms’, *Swarm Intell.*, vol. 9, no. 2, pp. 125–152, Sep. 2015, doi: 10.1007/s11721-015-0107-9.

[11] D. Garzón Ramos and M. Birattari, 'Automatic Design of Collective Behaviors for Robots that Can Display and Perceive Colors', *Appl. Sci.*, vol. 10, no. 13, Art. no. 13, Jan. 2020, doi: 10.3390/app10134654.

[12] K. Hasselmann and M. Birattari, 'Modular automatic design of collective behaviors for robots endowed with local communication capabilities', *PeerJ Comput. Sci.*, vol. 6, p. e291, Aug. 2020, doi: 10.7717/peerj-cs.291.

[13] J. Kuckling, K. Ubeda Arriaza, and M. Birattari, 'AutoMoDe-IcePop: Automatic Modular Design of Control Software for Robot Swarms Using Simulated Annealing', in *Artificial Intelligence and Machine Learning*, B. Bogaerts, G. Bontempi, P. Geurts, N. Harley, B. Lebichot, T. Lenaerts, and G. Louppe, Eds., in Communications in Computer and Information Science. Cham: Springer International Publishing, 2020, pp. 3–17. doi: 10.1007/978-3-030-65154-1_1.

[14] A. Ligot, K. Hasselmann, and M. Birattari, 'AutoMoDe-Arlequin: Neural Networks as Behavioral Modules for the Automatic Design of Probabilistic Finite-State Machines', in *Swarm Intelligence*, M. Dorigo, T. Stützle, M. J. Blesa, C. Blum, H. Hamann, M. K. Heinrich, and V. Strobel, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 271–281. doi: 10.1007/978-3-030-60376-2_21.

[15] M. Salman, A. Ligot, and M. Birattari, 'Concurrent design of control software and configuration of hardware for robot swarms under economic constraints', *PeerJ Comput. Sci.*, vol. 5, p. e221, Sep. 2019, doi: 10.7717/peerj-cs.221.

[16] S. Jones, M. Studley, S. Hauert, and A. Winfield, 'Evolving Behaviour Trees for Swarm Robotics', in *Distributed Autonomous Robotic Systems*, vol. 6, R. Groß, A. Kolling, S. Berman, E. Frazzoli, A. Martinoli, F. Matsuno, and M. Gauci, Eds., in Springer Proceedings in Advanced Robotics, vol. 6. , Cham: Springer International Publishing, 2018, pp. 487–501. doi: 10.1007/978-3-319-73008-0_34.

[17] E. Hogg, S. Hauert, D. Harvey, and A. Richards, 'Evolving behaviour trees for supervisory control of robot swarms', *Artif. Life Robot.*, vol. 25, no. 4, pp. 569–577, Nov. 2020, doi: 10.1007/s10015-020-00650-2.

[18] J. Kuckling, V. van Pelt, and M. Birattari, 'Automatic Modular Design of Behavior Trees for Robot Swarms with Communication Capabilites', in *Applications of Evolutionary Computation*, P.

A. Castillo and J. L. Jiménez Laredo, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 130–145. doi: 10.1007/978-3-030-72699-7_9.

[19] P. J. G. Ramadge and W. M. Wonham, 'Supervisory Control of a Class of Discrete Event Processes', *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, Jan. 1987, doi: 10.1137/0325013.

[20] Y. K. Lopes, S. M. Trenkwalder, A. B. Leal, T. J. Dodd, and R. Groß, 'Probabilistic Supervisory Control Theory (pSCT) Applied to Swarm Robotics', in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, in AAMAS '17. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, May 2017, pp. 1395–1403.

[21] Y. K. Lopes, S. M. Trenkwalder, A. B. Leal, T. J. Dodd, and R. Groß, 'Supervisory Control of Robot Swarms Using Public Events', in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 7193–7199. doi: 10.1109/ICRA40945.2020.9197418.

[22] Y. K. Lopes, 'Supervisory Control Theory for Controlling Swarm Robotics Systems', phd, University of Sheffield, 2016. Accessed: Dec. 15, 2020. [Online]. Available: http://etheses.whiterose.ac.uk/16765/

[23] L. P. Pinheiro, Y. K. Lopes, A. B. Leal, and R. S. U. Rosso, 'Nadzoru: A Software Tool for Supervisory Control of Discrete Event Systems', *IFAC-Pap.*, vol. 48, no. 7, pp. 182–187, Jan. 2015, doi: 10.1016/j.ifacol.2015.06.491.

[24] S. Pearson *et al.*, 'Robotics and Autonomous Systems for Net Zero Agriculture', *Curr. Robot. Rep.*, vol. 3, no. 2, pp. 57–64, Jun. 2022, doi: 10.1007/s43154-022-00077-6.

[25] Ł. Bołoz and W. Biały, 'Automation and Robotization of Underground Mining in Poland', *Appl. Sci.*, vol. 10, no. 20, Art. no. 20, Jan. 2020, doi: 10.3390/app10207221.

[26] R. Bogue, 'Robots for space exploration', *Ind. Robot Int. J.*, vol. 39, no. 4, pp. 323–328, Jun. 2012, doi: 10.1108/01439911211227872.

[27] S. Nouyan, R. Groß, M. Bonani, F. Mondada, and M. Dorigo, 'Teamwork in Self-Organized Robot Colonies', *IEEE Trans. Evol. Comput.*, vol. 13, no. 4, pp. 695–711, Aug. 2009, doi: 10.1109/TEVC.2008.2011746.

[28] V. S. Varadharajan, D. St-Onge, B. Adams, and G. Beltrame, ‘Swarm Relays: Distributed Self-Healing Ground-and-Air Connectivity Chains’, *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 5347–5354, Oct. 2020, doi: 10.1109/LRA.2020.3006793.

[29] S. Yi, W. Luo, and K. Sycara, ‘Distributed Topology Correction for Flexible Connectivity Maintenance in Multi-Robot Systems’, in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, May 2021, pp. 8874–8880. doi: 10.1109/ICRA48506.2021.9561721.

[30] K. Kobayashi, T. Higuchi, and S. Ueno, ‘Connectivity maintenance for robotic swarms by distributed role switching algorithm’, *Artif. Life Robot.*, vol. 27, no. 4, pp. 876–884, Nov. 2022, doi: 10.1007/s10015-022-00803-5.

[31] G. Miyauchi, Y. K. Lopes, and R. Groß, ‘Multi-Operator Control of Connectivity-Preserving Robot Swarms Using Supervisory Control Theory’, in *2022 International Conference on Robotics and Automation (ICRA)*, May 2022, pp. 6889–6895. doi: 10.1109/ICRA46639.2022.9812242.

[32] G. Miyauchi, Y. K. Lopes, and R. Groß, ‘Sharing the Control of Robot Swarms Among Multiple Human Operators: A User Study’, in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2023, pp. 8847–8853. doi: 10.1109/IROS55552.2023.10342457.

[33] N. Majcherczyk, A. Jayabalan, G. Beltrame, and C. Pinciroli, ‘Decentralized Connectivity-Preserving Deployment of Large-Scale Robot Swarms’, in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2018, pp. 4295–4302. doi: 10.1109/IROS.2018.8594422.

[34] H. J. Prömel and A. Steger, *The Steiner Tree Problem*. in Advanced Lectures in Mathematics. Wiesbaden: Vieweg+Teubner Verlag, 2002. doi: 10.1007/978-3-322-80291-0.

[35] G. Miyauchi, M. S. Talamali, and R. Groß, ‘Online supplementary material’. The University of Sheffield, 2024. [Online]. Available: https://doi.org/10.15131/shef.data.25382908

[36] G. Miyauchi, M. S. Talamali, and R. Groß, ‘Robot controller source code.’ 2024. [Online]. Available: https://github.com/openswarm-eu/minimal-length-swarm-networks

[37] F. Mondada *et al.*, ‘The e-puck, a robot designed for education in engineering’, *Proc. 9th Conf. Auton. Robot Syst. Compet.*, vol. 1, no. 1, pp. 59–65, 2009.

[38] C. Pinciroli *et al.*, ‘ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems’, *Swarm Intell.*, vol. 6, no. 4, pp. 271–295, 2012.

[39] D. Juhl, D. M. Warme, P. Winter, and M. Zachariasen, ‘The GeoSteiner software package for computing Steiner trees in the plane: an updated computational study’, *Math. Program. Comput.*, vol. 10, no. 4, pp. 487–532, Dec. 2018, doi: 10.1007/s12532-018-0135-8.

[40] Y. Vardi and C.-H. Zhang, ‘The multivariate L1-median and associated data depth’, *Proc. Natl. Acad. Sci.*, vol. 97, no. 4, pp. 1423–1426, Feb. 2000, doi: 10.1073/pnas.97.4.1423.

[41] A. G. Millard *et al.*, ‘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)*, Sep. 2017, pp. 741–748. doi: 10.1109/IROS.2017.8202233.

[42] A. G. Millard *et al.*, ‘ARDebug: An Augmented Reality Tool for Analysing and Debugging Swarm Robotic Systems’, *Front. Robot. AI*, vol. 5, Jul. 2018, doi: 10.3389/frobt.2018.00087.

[43] M. Birattari *et al.*, ‘Automatic Off-Line Design of Robot Swarms: A Manifesto’, *Front. Robot. AI*, vol. 6, p. 59, 2019, doi: 10.3389/frobt.2019.00059.

[44] P. Walker, S. A. Amraii, N. Chakraborty, M. Lewis, and K. Sycara, ‘Human control of robot swarms with dynamic leaders’, in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 1108–1113. doi: 10.1109/IROS.2014.6942696.

[45] J. Nagi, A. Giusti, L. M. Gambardella, and G. A. Di Caro, ‘Human-swarm interaction using spatial gestures’, in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2014, pp. 3834–3841. doi: 10.1109/IROS.2014.6943101.

[46] J. Alonso-Mora, S. Haegeli Lohaus, P. Leemann, R. Siegwart, and P. Beardsley, ‘Gesture based human - Multi-robot swarm interaction and its application to an interactive display’, in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 5948–5953. doi: 10.1109/ICRA.2015.7140033.

[47] B. Gromov, G. Abbate, L. M. Gambardella, and A. Giusti, ‘Proximity Human-Robot Interaction Using Pointing Gestures and a Wrist-mounted IMU’, in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 8084–8091. doi: 10.1109/ICRA.2019.8794399.

[48] S. O. Sachidanandam, S. Honarvar, and Y. Diaz-Mercado, ‘Effectiveness of Augmented Reality for Human Swarm Interactions’, 2022, p. 7.

[49] I. Jang, J. Hu, F. Arvin, J. Carrasco, and B. Lennox, ‘Omnipotent Virtual Giant for Remote Human–Swarm Interaction’, in *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*, Aug. 2021, pp. 488–494. doi: 10.1109/RO-MAN50785.2021.9515542.

[50] A. Kolling, P. Walker, N. Chakraborty, K. Sycara, and M. Lewis, ‘Human Interaction With Robot Swarms: A Survey’, *IEEE Trans. Hum.-Mach. Syst.*, vol. 46, no. 1, pp. 9–26, 2016, doi: 10.1109/THMS.2015.2480801.

[51] H. A. Yanco and J. Drury, ‘Classifying human-robot interaction: an updated taxonomy’, in *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, Oct. 2004, pp. 2841–2846 vol.3. doi: 10.1109/ICSMC.2004.1400763.

[52] S. Nagavalli, M. Chandarana, K. Sycara, and M. Lewis, ‘Multi-Operator Gesture Control of Robotic Swarms Using Wearable Devices’, in *Proceedings of the Tenth International Conference on Advances in Computer-Human Interactions*, IARIA, Mar. 2017. Accessed: Dec. 04, 2019. [Online]. Available: http://d-scholarship.pitt.edu/34583/

[53] T. D. Fincannon, A. W. Evans, F. Jentsch, E. Phillips, and J. Keebler, ‘Effects of Sharing Control of Unmanned Vehicles on Backup Behavior and Workload in Distributed Operator Teams’, *Proc. Hum. Factors Ergon. Soc. Annu. Meet.*, vol. 53, no. 18, pp. 1300–1303, Oct. 2009, doi: 10.1177/154193120905301829.

[54] J. R. Grosh and M. A. Goodrich, ‘Multi-human Management of Robotic Swarms’, in *Human-Computer Interaction. Multimodal and Natural Interaction*, M. Kurosu, Ed., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 603–619. doi: 10.1007/978-3-030-49062-1_41.

[55] F. Gao, M. L. Cummings, and E. T. Solovey, ‘Modeling Teamwork in Supervisory Control of Multiple Robots’, *IEEE Trans. Hum.-Mach. Syst.*, vol. 44, no. 4, pp. 441–453, Aug. 2014, doi: 10.1109/THMS.2014.2312391.

[56] J. Patel and C. Pinciroli, ‘Improving Human Performance Using Mixed Granularity of Control in Multi-Human Multi-Robot Interaction’, in *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, Aug. 2020, pp. 1135–1142. doi: 10.1109/RO-MAN47096.2020.9223553.

[57] M. Lewis *et al.*, ‘Teams organization and performance in multi-human/multi-robot teams’, in *2010 IEEE International Conference on Systems, Man and Cybernetics*, IEEE, 2010, pp. 1617–1623. doi: 10.1109/ICSMC.2010.5642379.

[58] S. L. Marlow, C. N. Lacerenza, J. Paoletti, C. S. Burke, and E. Salas, ‘Does team communication represent a one-size-fits-all approach?: A meta-analysis of team communication and performance’, *Organ. Behav. Hum. Decis. Process.*, vol. 144, pp. 145–170, Jan. 2018, doi: 10.1016/j.obhdp.2017.08.001.

[59] J. MacMillan, E. E. Entin, and D. Serfaty, ‘Communication overhead: The hidden cost of team cognition.’, in *Team cognition: Understanding the factors that drive process and performance.*, E. Salas and S. M. Fiore, Eds., Washington: American Psychological Association, 2004, pp. 61–82. doi: 10.1037/10690-004.

[60] R. Rico, M. Sánchez-Manzanares, F. Gil, and C. Gibson, ‘Team implicit coordination processes: A team knowledge-based approach’, *Acad. Manage. Rev.*, vol. 33, no. 1, pp. 163–184, Jan. 2008, doi: 10.5465/AMR.2008.27751276.

[61] K. A. Roundtree, M. A. Goodrich, and J. A. Adams, ‘Transparency: Transitioning From Human–Machine Systems to Human-Swarm Systems’, *J. Cogn. Eng. Decis. Mak.*, vol. 13, no. 3, pp. 171–195, Apr. 2019, doi: 10.1177/1555343419842776.

[62] G. Kapellmann-Zafra, N. Salomons, A. Kolling, and R. Groß, ‘Human-Robot Swarm Interaction with Limited Situational Awareness’, in *Swarm Intelligence*, M. Dorigo, M. Birattari, X. Li, M. López-Ibáñez, K. Ohkura, C. Pinciroli, and T. Stützle, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 125–136. doi: 10.1007/978-3-319-44427-7_11.

[63] G. Miyauchi, Y. K. Lopes, and R. Groß, ‘Online supplementary material’. The University of Sheffield, 2022. doi: 10.15131/shef.data.21088627.

[64] J. Patel, P. Sonar, and C. Pinciroli, ‘On multi-human multi-robot remote interaction: a study of transparency, inter-human communication, and information loss in remote interaction’, *Swarm Intell.*, vol. 16, no. 2, pp. 107–142, Jun. 2022, doi: https://doi.org/10.1007/s11721-021-00209-2.

[65] C. D. Wickens, S. E. Gordon, Y. Liu, and J. Lee, *An introduction to human factors engineering*, vol. 2. Pearson Prentice Hall Upper Saddle River, NJ, 2004.

[66] S. G. Hart and L. E. Staveland, ‘Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research’, in *Advances in Psychology*, vol. 52, Elsevier, 1988, pp. 139–183. doi: 10.1016/S0166-4115(08)62386-9.

[67] R. M. Taylor, ‘Situation awareness rating technique (SART): the development of a tool for aircrew systems design’, in *Situational Awareness in Aerospace Operations (AGARD-CP-478)*, Neuilly Sur Seine, France: NATO-AGARD, 1990, p. 3/1-3/17.

Glossary

ANOVA	Analysis of Variance
DIR	Direct (communication)
IND	Indirect (communication)
NRS	No Robot Sharing
RS	Robot Sharing
SART	Situational Awareness Rating Technique
SCT	Supervisory Control Theory
TLX	Task Load Index