A Collective Adaptive Approach to Decentralised k-Coverage in Multi-robot Systems

A Collective Adaptive Approach to Decentralised k-Coverage in Multi-Robot Systems

DANILO PIANINI, Alma Mater Studiorum—Università di Bologna, Italy
FEDERICO PETTINARI, Alma Mater Studiorum—Università di Bologna
ROBERTO CASADEI, Alma Mater Studiorum—Università di Bologna
LUKAS ESTERLE, Aarhus University, Denmark

We focus on the online multi-object k-coverage problem (OMOkC), where mobile robots are required to sense a mobile target from k diverse points of view, coordinating themselves in a scalable and possibly decentralised way. There is active research on OMOkC, particularly in the design of decentralised algorithms for solving it. We propose a new take on the issue: rather than classically developing new algorithms; we apply a macro-level paradigm, called *aggregate computing*, specifically designed to directly program the global behaviour of a whole *ensemble* of devices at once. To understand the potential of the application of aggregate computing to OMOkC, we extend the Alchemist simulator (supporting aggregate computing natively) with a novel toolchain component supporting the simulation of mobile robots. This way, we build a software engineering toolchain comprising language and simulation tooling for addressing OMOkC. Finally, we exercise our approach and related toolchain by introducing new algorithms for OMOkC; we show that they can be expressed concisely, reuse existing software components, and perform better than the current state of the art in terms of coverage over time and number of objects covered overall.

CCS Concepts: • **Computer systems organization** → **Self-organizing autonomic computing**; *Robotic autonomy*; • **Theory of computation** → **Self-organization**; • **Computing methodologies** → *Distributed programming languages*; *Self-organization*; • **Software and its engineering** → *Application specific development environments.*

Additional Key Words and Phrases: Location based services Internet of things, online multi-object k-coverage, smart cameras, multi-robot, aggregate computing

ACM Reference Format:
Danilo Pianini, Federico Pettinari, Roberto Casadei, and Lukas Esterle. 2021. A Collective Adaptive Approach to Decentralised k-Coverage in Multi-Robot Systems . 1, 1 (July 2021), 39 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Recent technological trends foster a vision of large-scale, situated systems where devices sense and act upon their local environment to perform some joint task and coordinate with one another to provide global, system-wide benefits. However, as the scale and density of computational collectives increase, centralised solutions become impractical, whereas mobility and failure create a dynamicity that systems ought to partially address by themselves, i.e., autonomously [49]. In

Authors' addresses: Danilo Pianini, Alma Mater Studiorum—Università di Bologna, Via dell'Università, 50, Cesena (FC), Italy, danilo.pianini@unibo.it; Federico Pettinari, Alma Mater Studiorum—Università di Bologna, federico.pettinari2@studio.unibo.it; Roberto Casadei, Alma Mater Studiorum—Università di Bologna, roby.casadei@unibo.it; Lukas Esterle, Aarhus University, Nordre Ringgade 1, Aarhus, Denmark, lukas.esterle@eng.au.dk.

such pervasive computing scenarios, awareness of the local context and location is often leveraged to make appropriate decisions and coordinate activity in a decentralised fashion [60].

In this paper, we address the *Cooperative Multi-Robot Observation of Multiple Moving Targets (CMOMMT)* problem [59]: we consider multiple mobile robots (e.g., drones with vision sensors) able to observe or *cover* objects of interest (also known as *targets*) and interact with other robots in order to cooperate. More specifically, we focus on the *Online Multi-Object k-Coverage (OMOkC)* [29, 30] problem, where the number of cooperative robots and targets is unknown and possibly dynamic. Our goal is to operate the system to maximise the number of *k-covered* mobile targets (i.e., the number of targets *covered by at least k* robots) over time, while minimising the *cost* of doing it (where the definition of cost is application-specific—e.g., in terms of total movement or energy consumption). Importantly, the agents may not be able to achieve this goal optimally. This can be due to the fact that the agents do not know some of the objects or there are too many objects to cover all of them with k agents. The robots have to explore the area to discover targets; once they find one (or more), they have to choose whether to *follow* it or not (or which one). In other words, as the robots, tasked with covering targets know neither the area nor the number of targets or their location they are confronted with an *explore vs exploit dilemma*. However, the robots can communicate to cooperate towards the goal, which is essentially global in nature—i.e., the robots make up a *team* [44].

In the literature, several algorithms have been proposed to solve OMOkC [30], and evaluated through simulation. However, such algorithms typically use conventional techniques by which the global coordination logic is expressed according to a local viewpoint in terms of individual message-based communication acts. Since defining local behaviours to build a specific global behaviour (a.k.a. *local-to-global mapping* problem) from the bottom up tends to be difficult, in recent years, novel paradigms and abstractions are emerging that support the development of location-based services in a more top-down fashion [9]. These approaches internally deal with the inverse problem (a.k.a. *global-to-local mapping*) and let the programmer work at the macro-level perspective, generally at the expense of a more constrained programming model. Accordingly, in this work, we consider the latter approach and develop a method and practical framework for implementing and simulating networks of mobile robots with vision sensors through an aggregate perspective. We leverage the approach and the toolchain to realise and benchmark two novel algorithms: (i) one based on the idea of moving robots as if they were subject to virtual force fields generated by known targets and other robots, which has showed to be suitable for exploration; (ii) the other based on the idea of sharing the vision information among neighbouring robots and using these data to solve an optimisation problem locally, which has showed to improve over the state of the art when targets are spotted. Most specifically, our contribution is threefold.

(1) **An aggregate approach to OMOkC**: our *main* contribution is the application of an emerging paradigm (aggregate computing) to the problem of OMOkC. We apply, for the first time, aggregate computing [94] to OMOkC, thus modelling, engineering, and programming networks of mobile robots with vision sensors as *collective adaptive systems*;
(2) **Two novel OMOkC algorithms**: to showcase the applicability of the approach to the problem, we devised two novel algorithms for distributed OMOkC, and show they perform better than the pre-existing state of the art.
(3) **A simulation tool for aggregate programs in networks of robots with vision sensors**: the application of the technique required a toolchain for its evaluation, that we built by extending the Alchemist simulator [65] with new capabilities. These new features have been released and are currently part of the main distribution of the simulator: they are as such a by-contribution of this work.

The remainder of the paper is organised as follows: Section 2 provides a mathematical model of the problem; Section 3 describes the aggregate computing approach to designing software for systems of robots with vision sensors; Section 4 provides motivation and a description of the proposed toolchain; Section 5 describes the application of the approach and toolchain to OMOkC, presenting two novel algorithms and validating them against the state of the art; Section 6 discusses related work; Section 7 covers limitations and future work; finally, Section 8 concludes the paper with a wrap-up and an outline of research directions for the future.

2 MODEL AND PROBLEM DEFINITION

This section provides a mathematical model of the *Online Multi-Object k-Coverage (OMOkC)* problem, following the conceptualisation and notation introduced in [29, 30]. The problem extends the *Cooperative Multi-robot Observation of Multiple Moving Targets (CMOMMT)* problem [59]. Related problems are discussed in Section 6.1.

2.1 The Online Multi-Object k-Coverage (OMOkC) problem

Let $C = \{c_1, c_2, \ldots, c_n\}$ be a set of n autonomous *mobile robots with vision sensors*, capable of analysing their Field of View (FoV) and communicate with others in the environment. That is, we generally assume that robots are capable of communicating with other nearby robots, which may be captured by a (logical) neighbouring relationship (as covered in Section 3.1), and we abstract from the enabling actual networking mechanisms and protocols. An in-depth discussion of how the network topology can affect the collective response of decentralised systems [55] falls beyond the scope of this paper. Further, we consider $O = \{o_1, o_2, \ldots, o_m\}$ the set of m *mobile objects*, and $P = \{p_1, p_2, \ldots, p_l\} \subseteq O$ a set of l *important* objects. Objects can become important for various reasons such as specific suspicious behaviour, appearance, or simply because an operator selected them: the set of important objects is *dynamic*, i.e., it can change over time as elements become important and unimportant. In particular, targets may be identified according to some (possibly also dynamic) predicate $\mathcal{P}$, so that, e.g., o_i is a target p_j if predicate $\mathcal{P}(o_i) = 1$.

The *state* of each robot with vision sensors is modelled as a 4-tuple $c_i = \langle \vec{x}_i, \vec{v}_i, \omega_i, \mathcal{V}_i \rangle$ with *location* $\vec{x}_i = (x_i, y_i)$[1], *velocity* $\vec{v}_i = (v_i^X, v_i^Y) = (\frac{dx_i}{dt}, \frac{dy_i}{dt})$, *angular velocity* ω_i, and *field of view (FoV)* $\mathcal{V}_i$. We assume perfect localisation. Angular velocity is included in the 4-tuple despite the robot being modelled as point-wise to capture a rotating field of view in dynamic situations. The FoV $\mathcal{V}_i$ of robot's camera c_i is described as a triple $\langle \Theta_i, R_i, \frac{\beta_i}{2} \rangle$ where Θ_i models the *orientation* of the view with respect to some fixed reference system, R_i is the *range* of view (modelling the maximum range a camera can detect targets), and $\frac{\beta_i}{2}$ denotes half of the view angle (modelling the width of the FoV beyond which there are blind spots)—where we assume that the FoV is symmetric, i.e., both sides of the directrix for a given orientation have the same angle width and range.

An object o_a is *covered* at a given time t, if the object is geometrically within the field of view $\mathcal{V}_i$ of a camera c_i, as represented in Figure 1:

$$cov(o_a, c_i, t) = \begin{cases} 1, & \text{if } d_{i,a} \leq R_i \wedge |\alpha_{i,a}| \leq |\frac{\beta_i}{2}| \\ 0, & \text{otherwise,} \end{cases}$$

where $d_{i,a}$ and $\alpha_{i,a}$ denote, respectively, the Euclidean distance and the angle between the object o_a and the camera c_i: any object within any FoV is considered *covered*.

[1]For simplicity, we consider a two-dimensional environment, even though extensions are possible for three-dimensional scenarios.

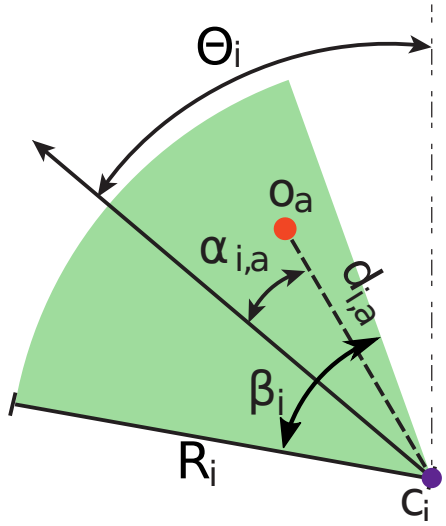

Fig. 1. Illustration of an object o_a inside the FoV $\mathcal{V}_i$ of camera c_i.

Since we are interested to cover each target with at least k robots at any time, we define the k-coverage at time t as follows:

$$kcov(o_a, k, t) = \begin{cases} 1, & if \ \sum_{i=1}^{n} cov(o_a, \mathcal{V}_i, t) \geq k \\ 0, & otherwise. \end{cases}$$

In order to measure how well k-coverage is achieved throughout an arbitrary period, we extend the normalised metric by Esterle and Lewis [29] considering a continuous-time flow beginning at T_0 and ending at T:

$$OMC_k = \frac{\int_{T_0}^{T} \frac{\sum_{a=1}^{m} kcov(o_a,k,t)}{max(1,|P_t|)} dt}{T - T_0} \tag{1}$$

for a given value of k. This value is normalised by the number of elements in the set of important objects P_t at time t. In short, the numerator of OMC_k represents the sum of the fraction of objects covered by k or more robot cameras over the total time $T - T_0$. We need this normalisation to keep results comparable even with changing numbers of important targets. We finally divide it for the time length to get the average coverage during the period of interest. In this work, we assume perfect localisation and detection to focus on the problem (demonstrated to be NP-hard [29, 59]) of coordinating mobile robots with vision sensors in a decentralised fashion in such a way that at least k of them are tracking each important mobile target (whose movement can not be controlled by robots), where the set of important targets is dynamic. Furthermore, we aim to maximise the number of important targets detected by the set of cameras. This makes coverage of each target with *exactly* k cameras the dominant strategy for the collective as cameras not tracking known targets are free the explore and detect new targets. However, when there are not enough agents n to cover all objects m this goal cannot be achieved, i.e., $m \times k > n$. In Figure 2, we show a sequence of snapshots exemplifying an instance of the problem[2].

We utilise the OMOkC problem as it brings about an interesting trade-off between exploration vs. exploitation. This dilemma requires decisions on the coordination to be considered continuously at runtime. Precisely, individual robots

[2]Snapshots are from the video publicly available at https://www.youtube.com/watch?v=yuaY_8Vr3oc

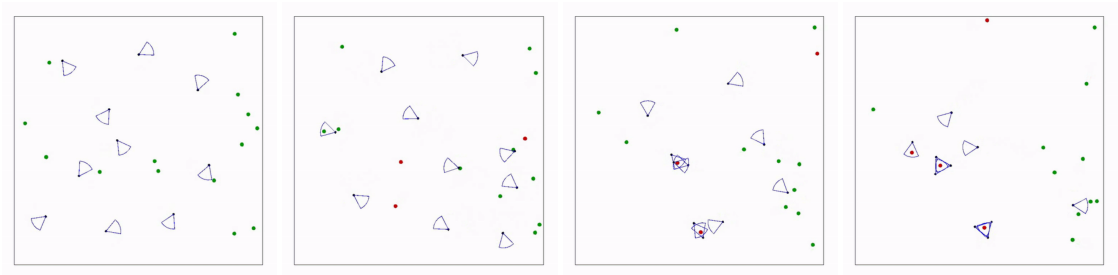

Fig. 2. Sequence of snapshots exemplifying an instance of the OMOkC problem. Green dots represent uninteresting targets, red dots are interesting targets, black dots are robots, and blue wedges their field of view. Targets move in the arena (black square), and they may randomly switch their "interesting" status. Robots must explore the area in search of interesting targets and, once some are found, they must organise (in a decentralised fashion) to follow the target from k points of view.

Symbol	Description
C	set of cameras
O	set of objects
$P \subseteq O$	set of important objects (targets)
n	number of robots with cameras
m	number of objects
l	number of targets
$c_i = \langle \vec{x}_i, \vec{v}_i, \omega_i, \mathcal{V}_i \rangle$	i-th camera/robot
o_i	i-th object
p_i	i-th object of interest
$\vec{x}_i = (x_i, y_i)$	i-th robot's location vector
$\vec{v}_i$	i-th robot's velocity vector
ω_i	i-th cameras's angular velocity
$\mathcal{V}_i = \langle \Theta_i, R_i, \frac{\beta_i}{2} \rangle$	i-th cameras's field of view
R_i	range of the i-th cameras's field of view
Θ_i	orientation of the i-th cameras's field of view
β_i	angle of the i-th cameras's field of view
α_{ij}	angle of the j-th object wrt the i-th camera's field of view
d_{ij}	distance of the j-th object wrt the i-th robot

Table 1. Summary of notation.

must decide whether to follow a specific target to improve the quality of its coverage or search for another to increase the total number of detected targets. As targets can change their state, becoming important or unimportant at random times, mobile robots have to re-evaluate their decisions continuously.

3 AN AGGREGATE APPROACH FOR OMOKC

All the methods tackling OMOkC in a decentralised fashion found in the literature (of which we provide an extensive review in Section 6.2) share a common trait: the solution is designed by focussing on the interaction among single robots, on the messages they should exchange, and on the ways they may form coalitions dynamically. In this work, we propose a different take: building on the idea on which aggregate computing is rooted, we advocate that the *ensemble* comprising all robots could and should be programmed as a *single, distributed computational entity*. To understand

how this can be done, we briefly introduce[3] aggregate computing (Section 3.1) and motivate its application to the decentralised OMOkC problem (Section 3.2).

3.1 Designing collective behaviours with Aggregate Computing

3.1.1 Approach overview. Aggregate computing is a paradigm and engineering approach for developing collective adaptive systems from a *global perspective*. The core functional language that formally founds aggregate computing is the *(computational) field calculus* [5]. As its name suggests, it is a calculus of *(computational) fields*, which are essentially (dynamic) maps from (a *domain* of) devices to computational values. In particular, a field can be seen as a distributed data structure that represents, over time, the result of a collective computation. Aggregate programming languages [94] provide the field calculus primitives and library functions to manipulate these distributed data structures. Following this approach, the designer does not need to focus on single devices or communication protocols but instead on how fields evolve and compose: it is up to the language's interpreter (or compiler) to determine the appropriate local interaction schema generating the desired global effect.

Most notably, the calculus (and, thus, the derived languages) provides the primary mechanisms for the predictable composition of emergent behaviour. Self-stabilising building blocks can be defined leveraging functional abstractions [93], and an entire library of collective behaviours [36] can get built upon them. The paradigm has been implemented in several languages: Protelis [67], a stand-alone, Java-interoperable, and JVM-hosted language; ScaFi [18], a domain-specific language (DSL) embedded in the Scala programming language; and FCPP [2], a lightweight native implementation designed to run on low-resource devices.

3.1.2 Aggregate computing model (structure, behaviour, interaction). Structurally, a logical[4] *aggregate system* consists of a set of *(uniquely identified) devices*; each device can communicate with other devices as per some *neighbouring relationship*. As a device moves in the environment, its set of neighbours might change. Notice that neighbourhoods are defined at a *logically* and independently of physical connectivity and spatial proximity (although it is natural to leverage those).

From the point of view of (global) *behaviour*, the aggregate system is instructed to continuously:

(1) update the context by sensing the environment and gathering coordination messages;
(2) interpret some *aggregate program* expressing the collective logic;
(3) *act* onto the environment as a consequence.

From a local, discrete perspective, every device works at asynchronous *rounds* of execution; in each round, an individual device gets data from its sensors and messages from neighbours, locally interprets the aggregate program against such input data, and triggers its actuators on the program local output (including data broadcasting to neighbours).

From the point of view of *interaction*, the devices continuously exchange coordination data with neighbour devices. The data to be exchanged results from the interpretation of the aggregate program.

3.2 Networked robots as Aggregate Systems

Aggregate computing is a natural framework for expressing collective algorithms in a decentralised fashion [94]. The aggregate computing aspects and abstractions can be mapped to the problem considered in this paper as follows.

[3] A full treatise of the approach is beyond the scope of this work; the interested reader can refer to the dedicated literature [5, 10, 94].
[4] Namely, not related to an actual system implementation: it can be shown [17] that an aggregate system admits different kinds of deployments and execution architectures, ranging from purely decentralised (e.g., ad-hoc, peer-to-peer) to fully centralised (e.g., cloud-based).

- *Aggregate system.* The aggregate system logically consists in a network of mobile robots with vision sensors. Being in aggregate system, the set of interacting robots can be programmed as a whole, conceptually.
- *Individual node.* A mobile robot with vision sensors is an individual node of the aggregate system. It has identity, state, sensors, actuators, runs an aggregate program, and interacts with other robots by sending messages as prescribed by the semantical interpretation of the aggregate program.
- *Neighbouring relationship.* It depends on the particular application and deployment. It can merely mirror physical connectivity (e.g., to support programming of situated systems), or can be used to set up a logical overlay network [17] reflecting the spatial distribution of devices, or even purely logical relationships. For the scenario considered in Section 5, we consider robots connected if they are close by a certain threshold (hence simulating short-range radio communication).
- *Aggregate program.* It describes the behaviour of a network of mobile robots. The actual behaviour emerges from the combination of the environmental dynamics, the dynamics of the evaluation of the program by each robot against its context, and the dynamics of inter-robot communication.
- *Sensors.* The set of required sensors depends on the program. For the algorithms considered in the following (Section 5), a robot has a vision sensor, a sensor for estimating the distance to neighbours, and a sensor for estimating the direction towards neighbours.
- *Actuators.* The set of required actuators depends on the application. For the considered problem, a robot has movement actuators (for rotating and going forward).
- *State.* A robot, at a minimum, must have sensors and actuators. In principle, state and aggregate program computations can be offloaded to other machines [17]. The state of a robot would include the data implied by the local aggregate program execution, plus configuration data which could also be modelled via sensors.
- *Local computational behaviour.* The local computational behaviour of a robot consists of the application of the aggregate execution protocol as described in Section 3.1.2, which involves sensing the local context, running the aggregate program against the local context, and then acting on the local context by sending messages to neighbours and running actuations. The overall local behaviour, hence, emerges from the local computational behaviour and interaction with the environment (e.g., the detection of a target through the visual sensors).
- *Scheduling and execution details.* There is large flexibility regarding *when* computational rounds and communications are performed [17]. Typically, no synchronicity and message delivery guarantees are required: rounds and communications may be asynchronous, and the computation would tend to self-stabilise [93] once up-to-date data is available. As a rule of thumb, the frequency of computation and communications should be adequate to the dynamics of the phenomenon to be monitored or dealt with—in this case, the speed of targets. Of course, such details may significantly affect the overall performance. However, since the aggregate behaviour is emergent, it may not be easy to determine the optimal execution strategy, especially when also considering the costs in term of energy and bandwidth consumption. Such a detailed analysis of the performance is beyond the scope of this work, which instead focusses on the overall approach.

3.2.1 Benefits. Modelling networked robots as an aggregate system allows to enjoy the features that aggregate computing offers over other approaches, mainly:

(1) *abstraction from device-to-device communication*: in aggregate computing, communication protocols are a consequence of the structure of the program—the designer does not need to figure out messages to be exchanged, their order, and similar low-level details;

(2) *functional compositionality* [93]: aggregate programs are written in a functional language and can (and should) be encapsulated into reusable functions.

Thus, from a design point of view, describing the behaviour of the mobile robot network as an aggregate program introduces abstractions closer to the problem than messages and protocols. From an engineering point of view, the ability to encapsulate behaviour into functions in a reusable fashion opens the door to further simplification.

On the one hand, the designer can reuse an extensive API of collective behaviours [36] shipped as a standard library for the languages; the availability of aggregate library functions with proven guarantees [93] can reduce significantly the time and effort required to build and debug complex behaviours, as these reusable building blocks capture many low-level details. On the other hand, reusable blocks of specialised behaviour can be encapsulated into reusable functions, collected, and shared as blocks upon which more complex programs can be constructed, enabling guarantees and fine-grained control over growing complexity, ultimately promoting the creation of more and more refined behaviours.

4 A TOOLCHAIN FOR DEVELOPING SOLUTIONS TO OMOKC WITH AGGREGATE COMPUTING

Reaping the benefits of aggregate computing into the multi-robot coordination domain requires appropriate development tools. In particular, simulation platforms supporting aggregate computing *and* networks of robots with vision sensors are essential, as they enable evaluation and testing of the algorithms being developed in a low-cost and time-efficient fashion. We first analysed the state of the art and found that, to the best of our knowledge, no simulator supported both aggregate computing specifications and the simulation of multi-robot systems with a field of view. We thus took the subsequent step and extended an existing simulation platform, choosing between integrating aggregate programming into an existing simulator for networked robots with vision sensors or extending an existing aggregate computing simulator with the capabilities to support networked robots with vision sensors. This section first discusses the available options for a viable simulation tool (Sections 4.1 and 4.2), motivating our choice for the tool, and finally explaining how the extension has been realised in the selected product (Section 4.3).

4.1 Simulators for networked robots with vision sensors

Deploying and maintaining networks of mobile robots with vision sensors in the real world is generally cumbersome, time-intensive, and requires manual labour. Testing new approaches in real networks can be costly and problematic—as experiments often cannot be reproduced precisely. Various simulation tools for robotic systems have been developed over the past several years to overcome this problem [78]. These different simulators, however, often come with a trade-off between resource efficiency and fidelity [92]. Additionally, we identify three macro areas where a simulation tool can focus:

(1) interaction among physical objects and with the physical world in general;
(2) network evaluation and performance;
(3) robot behaviour and software.

Usually, tools focus on one of these areas. Consequently, multiple simulation tools may be used during development, depending on the current development stage: a tool focussing on behavioural and software aspects is necessary from the beginning to assess the functional correctness of the programs being designed and implemented; a network simulator is helpful to understand whether the designed software may induce excessive stress on the communication channels; a simulator specialised in physical interactions can be used to test and debug issues with the sensing and actuation before deployment.

In this work, we focus on simulators meant to be leveraged in the initial phase of software design, as we need a tool focussing on the behaviour of many devices, even if at the expense of simplified physical interactions. These kinds of tools allow for rapid prototyping, quick interception of possible mistakes, and streamlined acquisition of synthetic benchmarks: our goal is to understand the technical feasibility and convenience of aggregate computing as a means to tackle decentralised OMOkC. For the sake of completeness, we also review, in Section 6.4, simulators dedicated to measuring network performance and simulators focussing on a realistic reproduction of the physical world where robots work. These were not considered practical targets for our investigation, but they were considered, and we believe they could be leveraged in the future for more in-depth analyses.

The leading example of a simulator dedicated to quick prototyping and benchmarking of OMOkC algorithms is CamSim [31], focussing on the agent-based behaviour of networked robots with vision sensors. Initially developed for static cameras, CamSim has been extended towards Pan-Tilt-Zoom (PTZ) cameras and later even enabled networked mobile robots equipped with cameras to study coordination, individual self-adaption in collectives, and self-organisation properties. The simulator does not represent physical aspects of the real world except the vision sensors themselves; similarly, objects are represented as dots.

4.2 Simulators supporting aggregate programming

The ecosystem of simulators that natively support aggregate programming is still in its infancy. Typically, since an aggregate system with a single device is considered a degenerate case, languages rooted in the aggregate computing paradigm also feature a simulation system whose goal is to run code on a simulated network of devices. This is the case, for instance, for ScaFi [18] and FCPP [2], both of which ship with a lightweight simulation infrastructure for quick prototyping [2, 95]. This follows the tradition of MIT Proto [8], an early language for spatial computing (we review approaches similar to aggregate computing in Section 6.3), whose language interpreter and internal simulator were inextricably intertwined. However, these integrated simulators are not meant to be used for extensive benchmarking and generally do not provide means for extending them to simulate complex scenarios.

Consequently, most of the experiments with simulations that leveraged aggregate programming have been executed, so far, by integrating aggregate programming into an existing simulation platform[5], or by creating custom environments tailored to the specific analysis [11], or by leveraging the Alchemist simulator [65]. Thus, Alchemist is, to the best of our knowledge, the only stand-alone tool with first-class native support for aggregate programming (limited to the Protelis and ScaFi implementations).

Alchemist is a modular general-purpose meta-simulator for multi-agent systems. The core of Alchemist is an event-based engine derived from chemistry-oriented simulators, and its computational meta-model in part reflects these origins. The initial idea behind the simulator was to provide a lightweight core of abstractions, with few assumptions necessary to make the simulation engine work efficiently (a performance comparison with Repast is available in [65]), and providing a framework for easy extension in such a way that the meta-model entities could be refined differently depending on the case at hand.

4.3 Supporting multi-robot systems with vision sensors in Alchemist

After extensive evaluation of the possibilities, we were left with the choice between extending a simulator supporting aggregate computing with the tooling needed for robots with vision sensors, or extending a simulator supporting robots

[5] https://archive.ph/cI2QN

with the capabilities to run aggregate programs. In any case, it was paramount to execute the aggregate specification using the original interpreter: we did not want to perform paradigmatic conversions to fit aggregate computing into an alternative paradigm, as it would likely introduce errors and limit expressiveness.

We needed a system allowing for quick prototyping, and thus a tool focussing on behaviour and software, belonging to the first of the three categories identified in Section 4.1. Our choice was thus quickly restricted to (i) Alchemist, supporting aggregate computing on mobile nodes, but missing support for fields of view; and (ii) CamSim, with mature support for mobile robots with vision sensors, but lacking aggregate computing integration. Ultimately, we decided to tackle the creation of the toolchain by extending Alchemist; three dominant factors drove the choice:

(1) we deemed extending the Alchemist simulation model easier than integrating aggregate computing into CamSim;
(2) while Alchemist is actively developed, with the official repository[6] registering new commits at least weekly, CamSim appears to have been discontinued, as the latest commit on the official repository[7] being (at the time of writing) from 2017[8]; and
(3) we expected better performance, as Alchemist has been exercised in the past with tens of thousands of simulated devices [12], while all works leveraging CamSim used few dozen devices, and in no case (to the best of our knowledge) has ever been used with more than a hundred.

4.3.1 The Alchemist simulator meta-model. To better understand how we extended the original model of Alchemist, we briefly introduce its computational model. In Alchemist, every simulation is the event-driven evolution of an *environment*. An environment defines a coordinate system, the concept of *position*, and contains *nodes* and *obstacles*. We call N_t the set of nodes belonging to some environment at time t, and $\wp(N_t)$ its power set. Environments are programmed with a *network model*, a function $n : N_t \rightarrow \wp(N_t)$ such as:

$$y \in n(x) \Leftrightarrow x \in n(y) \; \forall \; x \in N_t, \; y \in N_t, \; x \neq y$$

defining, for each node in the environment, the set of nodes considered *neighbours*, with the restriction that if some node x is neighbour to a node y at time t, then node y must be neighbour of node x at the same time (neighbourhood relationships are symmetric). Every node in N_t is *situated*, i.e., it has a valid position in the environment. Nodes are containers of *reactions* and *molecules*. Both nodes and obstacles have a *shape*; the environment does not allow shapes belonging to diverse objects to overlap. Molecules are symbolic names that can be associated with a *concentration* (i.e., a value). Reactions are atomic events that can affect the environment. They are guarded by a set of *conditions*, namely boolean functions deciding whether the reaction can be executed or not. Every reaction is associated with a *time distribution*, providing putative execution times (or infinity, if conditions are unsatisfied). When a reaction is executed, it triggers a sequence of *actions*. Actions are arbitrary modifications of the environment. It has been proven that this abstract model allows for reuse of an extended version of the Gibson-Bruck stochastic Monte Carlo Algorithm [40], providing a performance edge over classic, agent-based engines [65], and consequently allowing better scaling with the count of simulated nodes. A so-called *incarnation* in Alchemist is a software component in charge of defining the actual type of data items being manipulated (the *concentration* type), possibly along with some other concrete Alchemist entities that manipulate it. This way, a precise trade-off can be achieved between generalisation and performance.

[6] https://github.com/AlchemistSimulator/Alchemist
[7] https://github.com/EPiCS/CamSim
[8] https://web.archive.org/web/20210908134925/https://github.com/EPiCS/CamSim

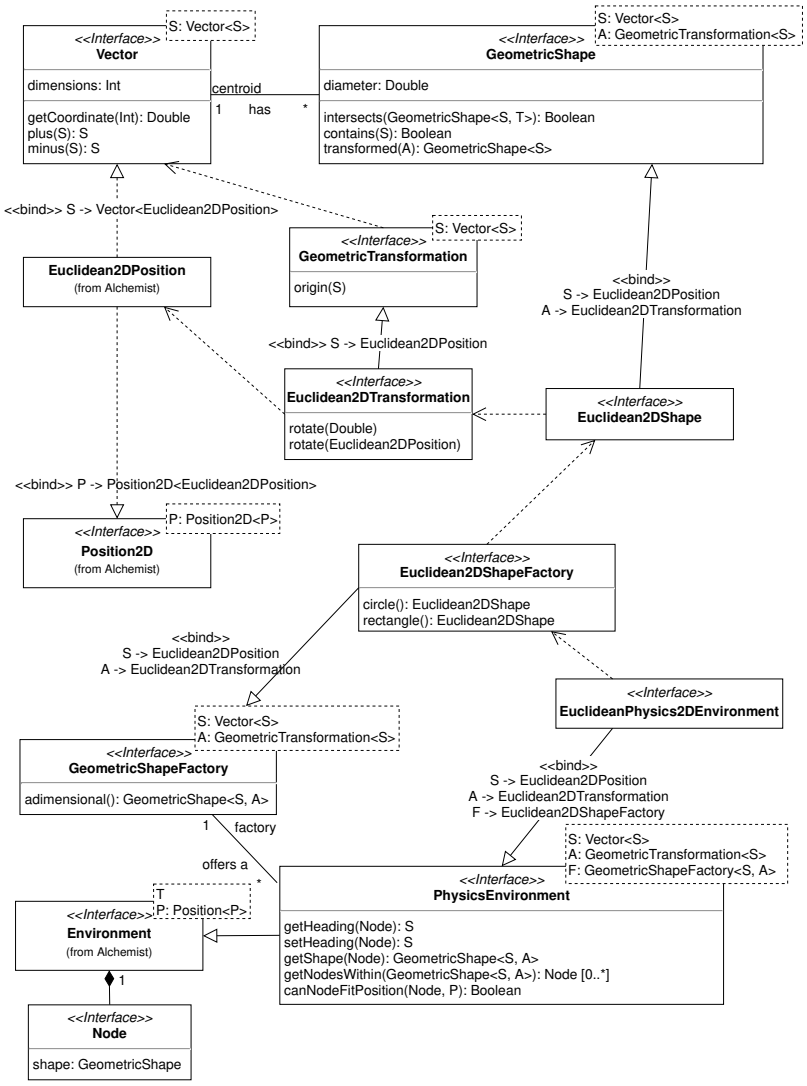

Fig. 3. Structural view of the implementation of the essential physical components into the existing simulator. Entities inherited from the original implementation are annotated with "(from Alchemist)". We associated `Node`s with a shape and a heading. To preserve a coherent view of the global coordinates system, this information is added to the `Environment` (as it originally was in charge of tracking the node position as well). Since the simulator is generic concerning the number of dimensions and the details of the manifold (as far as it is a Riemannian manifold), we had also to introduce means for rotation and translation of non-pointwise objects; hence we enriched the model with the possibility of expressing `GeometricTransformation`s, and we implemented the required machinery for these transformations to happen in bidimensional Euclidean spaces.

4.3.2 Contribution: two novel modules for Alchemist. In Alchemist, a *module* is a software component extending the capabilities of the simulator. The multi-robot with vision sensors support for Alchemist is composed of two such modules: (i) the `alchemist-influence-sphere`[9] module, introducing necessary physical interactions among

[9] http://bit.ly/alchemist-influence-sphere-maven-central

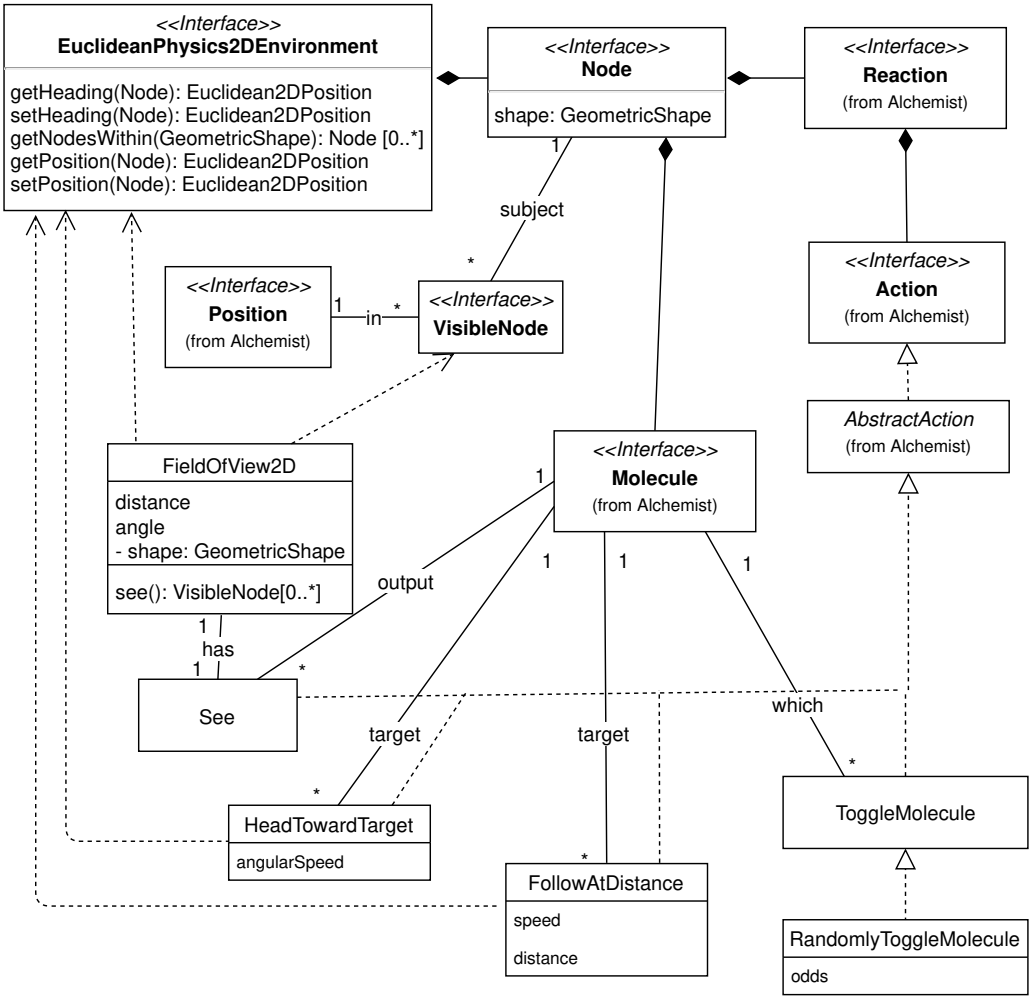

Fig. 4. Structural view of part of the implementation of the robots' vision and control system into the existing simulator. Entities inherited from the original implementation are annotated with "(from Alchemist)". The basic `Node` was extended with the concept of visibility, which enables some objects to be perceived by others (besides the neighbourhood relationship, which was built-in). We then introduced the sensing capabilities by modelling a `FieldOfView2D`; the field of view orientation is bound to the robot's heading (as heading and position are captured in the environment, see Figure 3). Consistently with the original model of Alchemist, access to the novel capabilities is modelled as a set of `Actions`—for the sake of conciseness, here we show some examples from the more extensive library: vision sensor reading (`See`) and target following (`FollowAtDistance`).

objects and enriching the concept of node with a perception area (de facto generalising the concept of field of view); and (ii) the `alchemist-smartcam`[10] module, which collects actions that allow for controlling, moving robots, and rotating their camera. For the sake of brevity, we do not delve into the implementation details of the simulator extension; however, to help the interested reader navigate the code, we provide a structural UML schema for the physical interactions in Figure 3 and a similar diagram for the robot controls in Figure 4.

[10] http://bit.ly/alchemist-smartcam-maven-central

The final result improves over the existing state of the art in several areas, in particular:

- *Environment model.* Alchemist supports a more detailed (yet still lightweight) model of the world, including support for indoor environments (by importing floor plans images) and modelling objects obstructing communication, movement, and view.
- *Programming abstractions.* Alchemist agents can be programmed with various approaches, including the aggregate computing languages Protelis [67] and Scafi [95]; moreover, the architecture allows for plugging in new languages in the future and using them for driving smart cameras with no case-specific code. The main benefit of this feature is the possibility of experimenting with a variety of different approaches and compare them.
- *Scalability.* Alchemist demonstrated to efficiently scale up to the order of tenths of thousands of devices on conventional consumer hardware [12]. By contrast, CamSim has never been exercised with more than a few dozen robots to the best of our knowledge.
- *Target behaviour.* Alchemist supports many advanced behaviours, such as movements accounting for cognitive, sociocultural, and emotional elements as defined by existing models in the literature [89].
- *Parallelism and distribution.* Alchemist supports parallel and distributed execution and statistical analysis [66].

Once the modules are available in the classpath, the simulation can be expressed declaratively in a YAML file[11]. YAML is a data serialisation format, superset of JSON, commonly used for non-trivial and human-readable configuration files. A commented example simulation descriptor is given in Figure 5, giving the reader an idea of the complexity of writing simulations. Alchemist is designed to allow third parties to extend the simulator and reuse the existing specification language. Due to space constraints, we will not unravel all the details of the specification language in this paper: the interested reader can refer to a recent tutorial [61]. By leveraging the pre-existing extension mechanisms of Alchemist, we were able to write our extension in terms of new *environments* and *nodes* (containing the physical properties, see Figure 3) and new *actions* (defining the behaviour of the camera sensors and exposing the actuators for its control, see Figure 4).

The software developed as part of this work has been integrated into the main Alchemist distribution and is available to the entire scientific community. Additional examples and a more extensive user guide are out of this work's scope: further (and up to date) details are provided on the Alchemist Simulator website[12].

5 AGGREGATE COMPUTING FOR ONLINE MULTI-OBJECT K-COVERAGE (OMOKC) IN ACTION

This section shows the potential of aggregate computing applied to OMOkC by exercising the proposed toolchain. We leverage aggregate computing capabilities to introduce two novel algorithmic solutions for the OMOkC problem that we show to improve over the state of the art. The first algorithm leverages for the first time the notion of computational field to build distributed data structures working as force fields, and then letting robots move according to them. This algorithm was a natural candidate for our initial investigation, as aggregate computing is particularly well-suited at expressing computations on field-like distributed data structures. We find that this algorithm is well-suited for the initial exploration of the environment (especially in the bootstrap phase), while it is not particularly effective in allocating cameras to targets. The algorithm, in fact, outputs a desired position for the robot, regardless of the existence of known targets or other robots. The second algorithm exploits aggregate computing to share the field of view among

[11] https://yaml.org/spec/1.2.1/
[12] https://alchemistsimulator.github.io/

```yaml
# Incarnation to be used. Dictates which data types can be used and how nodes can be programmed.
incarnation: protelis # Enables support for the Protelis programming language.

# Variables: parameters of the simulation. They can be used to reduce repetitions.
# They can be referenced (as per YAML specification) by their anchor name (starting with '&'),
# by prefixing a string with the special character '*'.
# When containing the 'formula: "code"' mapping, "code" is fed to a Groovy interpreter for evaluation.
variables:
  humans: &humans
    formula: "20" # Target count
  cameras: &cameras
    formula: 10 # Robot count
  size: &size
    formula: 400 # Arena side length, in meters
  origin: &origin
    formula: "-size/2" # Where the arena bottom-right corner should be for the origin to be (0, 0)
  FOVangle: &FOVangle
    formula: 60 # Field of view angle, in degrees
  FOVdistance: &FOVdistance
    formula: 20 # Field of view distance, in meters

# The kind of environment
environment:
  # The type/parameters syntax allows runtime loading of arbitrary simulation extensions: the
  # simulator searches the runtime classpath for a class that implements the necessary API, named
  # as the string passed for 'type', and whose constructor can produce a valid instance when fed
  # contextual information and the provided parameters.
  type: Rectangular2DEnvironment # an environment with an unpassable rectangular arena
  parameters: [*size, *size] # the horizontal and vertical dimensions of the arena, in meters

# Declaration of nodes together with their position and content
displacements:
  # Randomly place potential targets into the arena
  - type: Rectangle
    parameters: [*humans, *origin, *origin, *size, *size]
    nodes: # Defines the class of nodes representing potential targets within the simulation
      type: CircleNode # An alchemist node with a physical size (part of our extension)
      parameters: [1] # Dimension of the circular node (in meters)
  - type: Rectangle
    parameters: [*cameras, *origin, *origin, *size, *size]
    programs: # Robots get programmed here
      - time-distribution: 1 # Loads a negative exponential distribution with lambda=1
        type: Event
        actions:
          type: See # Custom action enabling the camera sensing (part of our extension).
          parameters: [*FOVdistance, *FOVangle, "inSight"]
      # The 'program' syntax is alternative to the type/parameter syntax for Alchemist's
      # reactions/events, the provided string is passed down to the incarnation for interpretation
      - program: "some:protelis:module" # Loads the omonym protelis program
      - program: send # Special action, enabling networking with the Protelis incarnation
```

Fig. 5. An Alchemist YAML simulation descriptor using the newly developed modules. It configures an environment with 20 potential targets and 10 robots with vision sensors in a 400mx400m square room. Robots are programmed (via the `See` action) to sense all the perceived nodes (humans and other robots) and write all the associated metadata to the `inSight` molecule.

neighbouring devices, allowing each one to "see" with multiple fields of view. This information is then leveraged to build a linear optimisation problem (describing only the system in the vicinity of the device) whose solution dictates the device's behaviour. The approach does not define an exploration strategy (as it outputs the position of the target assigned to the robot only if the robot is being assigned) and must thus be coupled with some other approach that does (including the previously presented force field-based algorithm). Data shows that this approach consistently improves over the state of the art in allocating robots to targets.

The proposed algorithms output positions, not orientations; selecting the latter can be done with an arbitrary strategy during exploration and keeping the target object at the centre of the field of view during tracking. Before exercising our new algorithms, in Section 5.3.1 we discuss our selected strategies for orientating the cameras.

5.1 Force Field Exploration ($\mathcal{A}_{\mathsf{ForceField}}$)

The $\mathcal{A}_{\text{ForceField}}$ algorithm is inspired by the idea of attraction and repulsion fields, notions widely used in force-directed graph drawing [7, 52]. Each robot generates a repulsive force field ϕ_c, whereas every known target (i.e., targets that are currently in the field of view of at least one robot) generates an attractive force field ϕ_o. Of course, targets outside all the fields of view are unknown to the robots, and since targets are not part of the aggregate computational systems, they cannot emit any field and are thus unknown to the robots. The direction of movement of a robot is given by the vector sum of the force fields involved. Moreover, to avoid the system from getting stuck into static situations, we also consider an additional notion, *willpower* (symbol: W), leveraged by robots to stick to the previous resolution despite the current force fields. The force fields are defined as functions of the *distance* (symbol: d) between entities, as follows:

$$\phi_c(d) = \frac{W}{2}\frac{(2\mathcal{V}_R)^2}{\max(1,d)^2} \tag{2}$$

$$\phi_o(d) = -k\frac{4\phi_c(d)}{\max(1,d)} \tag{3}$$

where $\mathcal{V}_R$ is the distance of the field of view, and k is the desired maximum coverage (namely, the k in k-coverage). This algorithm is a form of coordinated exploration that can be expressed directly as a collective field computation. The aggregate computing approach is particularly effective at expressing this kind of computation succinctly. As such, we attach a Protelis-written implementation in Figure 6. A complete implementation including code interacting with the simulated robot with vision sensors is available online[13].

5.2 Linear Programming-based Algorithm ($\mathcal{A}_{\mathsf{LinPro}}$)

$\mathcal{A}_{\text{LinPro}}$ is rooted in the idea of continuously solving multiple local linear programming problems defining the target selection strategy to minimise the robots' movements while attaining coverage. This approach is motivated by the idea that the problem could be broken down into smaller pieces (the neighbourhood of a robot, for each robot), and then a solution could be searched for each smaller problem. Although this kind of modelling does not preserve the possibility to reach a *globally* optimal solution, our intuition is that it should provide reasonably good *local* behaviour if the robots can access the fields of view of their neighbours. The approach we propose thus builds an aggregate view of the local system, sharing for each robot the fields of view of all neighbouring robots. The shared view is leveraged to build a classic optimisation problem that we solve locally for each device on every round (recall the local view of the behavioural description of aggregate computing introduced in Section 3.1).

This algorithm is different from a classic resolution of the *global* optimisation problem, as it works with *partial information* and needs to be continuously updated due to the intrinsic *dynamicity* of the system. In fact, the absence of a central leader means that the problem runs under partial information, and although the movement of robots can be controlled and programmed, no control can be exerted by the program over the target's behaviour. As such, the optimisation is somewhat aiming at a moving target; in other words, it is not just simple optimisation, but continuous optimisation towards an ever-changing optimum. Although techniques exist for building increasingly large alliances

[13] http://archive.is/wip/MxJ5h

```
def will() = ...
def repulsion(distance) = will()/2 *
  ((fovDistance() * 2)^2) / (max(1, distance)^2)

def attraction(distance) = if(distance >= fovDistance()/3) {
      -((desiredK() * repulsion(distance) * 4) / max(1, distance))
    } else { 0 }

def boundaries() { ... }

public def fieldExploration() {
  let cameraForces = repulsion(nbrRange()) * nbrVersor()
  let targetForces = foldUnion(nbr(vision()))
    .map { attraction(distanceFrom(it)) * versor(position(self) - position(it))
     }
    .reduce([0,0]) { a, b -> a + b }
  let sumOfForces = foldSum(cameraForces) + boundaries() + targetForces
  rep(myDirectionAngle <- randomAngle()){ // Start with a random angle
    let myDirection = angleToVersor(myDirectionAngle)
    let myForce = myDirection * will()
    let destination = position(self) + sumOfForces + myForce
    let newAngle = directionToAngle(destination - position(self))
    env.put("destination", destination +
      [step() * cos(newAngle), step() * sin(newAngle)])
    newAngle
  }
}
```

Fig. 6. Protelis code for $\mathcal{A}_{\mathsf{ForceField}}$ executed independently by each robot, stripped of low level details.

of robots with a central leader, up to the point where the whole network has a single leader where all information is centralised [63], this comes with several downsides:

- the communication time with the leader, using these techniques on opportunistic networks, grows linearly with the network diameter;
- data collection into a leader in mobile networks has its own sets of significant limitations that a growing body of literature is analysing [3, 4, 105];
- the leader robot must solve the global optimisation problem for all devices, which may introduce scaling problems (the more robots, the more difficult is the problem) and issues of asymmetric power consumption among robots.

We thus preferred to experiment with solving many simple problems, considering only the fields of view of neighbouring robots for each robot. Leveraging aggregate computing, we show that the algorithm can be expressed in few lines of code by relying on the interoperability with existing languages and platforms for the centralised component (the solver of the linear programming problem) and exploiting *field-of-view fields* (i.e. maps from neighbours to their fields of view) to gather the necessary data.

We formalise the mathematical model of the problem as follows:

$$\text{Minimise} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{i=1}^{n} q x_{i,m+1} \tag{4}$$

$$\text{Subject to} \quad \sum_{j=1}^{m+1} x_{ij} = 1 \qquad i = 1,\dots,n \tag{5}$$

$$\sum_{i=1}^{n} x_{ij} \le k \qquad j = 1,\dots,m \tag{6}$$

$$\sum_{i=1}^{n} x_{ij} \ge \min\left(1, \left\lfloor \frac{n}{m} \right\rfloor\right) \qquad j = 1,\dots,m \tag{7}$$

$$x_{ij} \in \{0,1\} \qquad i = 1,\dots,n \quad j = 1,\dots,m+1 \tag{8}$$

where:

- n is the number of known neighbouring robots;
- m is the number of targets (*important objects*) located within the field of view of at least one neighbouring robot;
- $m+1$ denotes a *fictitious target* that will be assigned to redundant cameras or when there are no targets; indeed, the second addend of Equation (4) has the goal to permit solutions to the problem where some cameras are left unassigned: robots assigned to that target are considered free and will adopt an exploratory behaviour;
- c_{ij} is the cost of assigning target j to robot i. In our case, the Euclidean distance between the two entities was used; however, the cost metric could be either a more elaborate notion of distance and/or could take into account additional costs (e.g. presumed additional network communication, energy consumption for enacting camera rotation, etc.);
- q is the constant cost associated with the fictitious target, always set to

$$q = \max_{\substack{i=1,\dots,n \\ j=1,\dots,m}} \{c_{ij}\} + 1 \tag{9}$$

  to ensure that keeping cameras unassigned when non-k-covered targets are known is never optimal;
- k is the desired k-coverage;
- x_{ij} are the unknown variables. Regarding the optimal solution, they will be 1 when target j is assigned to robot i, or 0 otherwise;
- $\lfloor x \rfloor$ is the flooring of x.

More informally, the objective is to minimise the overall cost for the cameras to reach their respective targets (4). Each robot must be assigned one and only one target (5). Each target but the fictitious one can be assigned to up to k robots: assigning more than k robots to a single target is a *cost* (9), as robots would be kept from exploration and possible discovery of other targets (6). Each target must be assigned to at least one robot if the number of robots is large enough. Otherwise, if the number of targets is greater than the number of robots, the result of the min function will be zero, and the constraint will have no effect. This particular constraint prioritises covering all the possible targets if all robots are already assigned and a new target is detected. (7). Finally, (8) is a non negativity constraint. Our model does not consider objects which are currently not interesting: it only focuses on targets (interesting objects).

```
rep(solver <- linProSolver()) {
  let targets = foldUnion(nbr(localTargets()))
  let cameras = nbr(getCenterOfFov())
  let myTarget = solver.solve(cameras, targets, desiredK(), false)
    .getOrDefault(getUID(), noTarget())
  followOrExplore(myTarget, fieldExploration)
  avoidCameraCollision(myTarget, localTargets)
  solver
}
```

Fig. 7. Protelis code for $\mathcal{A}_{\mathsf{LinPro}}$. This code polls the neighbouring robots for information about their position and the targets they have in sight. The information is collected and sent to a local process, in charge of solving the linear programming problem.

In case of multiple equivalent solutions, we sort them based on the matrix collecting the resulting x_{ij}'s, compare elements row-by-row and column-by-column, and pick the first one. This way, we properly deal with the case of environments with particular symmetries, which could otherwise lead to inconsistent behaviour: for instance, if two cameras have exactly the same distance from two shared targets, they may independently decide to move towards the same target. If ordering were not in place and nothing broke such symmetry even after the robots' movements (although extremely unlikely in the real world, as the slightest error would), this unwanted behaviour could persist as well.

The cases in which there are more robots than those required to achieve k-coverage for all targets (for instance, if there are no targets) are dealt with by exploiting the *fictitious target*, that will be assigned to all robots in excess.

This model is similar to the well-known "transportation problem"[21] in which robots are the sources and targets are the destinations. Moreover, the constraints matrix is totally unimodular, and the constant terms and the costs c_{ij} are integers; therefore, the solutions are integers, and integral constraints are not needed [71]. It has to be highlighted that each robot is supposed to solve the above problem with the pieces of information it knows, which are expected to be incomplete in relation to the entire network, and that we assume that robots can estimate and share the position of the targets in their field of view: the output of the algorithm thus includes the position that should be reached.

In our implementation, each robot executes a 1-hop broadcast in its communication range, communicating its position and the positions of the targets it detects. With the information received from its neighbourhood, a robot can determine local values for n, m, c_{ij}, and q, solve the above linear programming problem, and then follow the target indicated by its optimal solution, or explore if the result yields the fictitious target. Of course, the single problems solved by each robot individually *do not* represent valid solution for the *global* optimisation problem, (unless the network is fully connected). Our idea is to exploit these local (and *globally sub-optimal*, in general) solutions to select the local robot behaviour. This may cause situations in which some target attracts more attention than it should, and gets followed by too many robots; however, as soon as they can communicate, some will be either allocated to other targets or freed and set in exploration mode. Our bet is that even though the algorithm executed by each robot is not globally optimal, its re-evaluation in face of changes (as promoted by the aggregate computing rounds) leads to a high degree of adaptation.

We implemented this algorithm with a mixture of Kotlin[14] (in order to reuse the simplex solver included in the Apache Commons Math[15] library) and Protelis. The Kotlin part deals with solving the simplex, while the Protelis part is

[14] https://kotlinlang.org/
[15] http://archive.ph/wip/HVZ7O

responsible for the coordination of devices. We report the Protelis part in Figure 7, without imports and ancillary code. The complete implementation is available online[16].

5.2.1 Fair version ($\mathcal{A}_{LinProF}$). One shortcoming of $\mathcal{A}_{\text{LinPro}}$, as presented in the previous section, is that it does not try to balance out the load among different targets, possibly leading to a situation where k robots follow the same target at the cost of other targets having inadequate coverage. A simple modification to the problem definition, however, can lead to higher "fairness". The idea is to detect the ratio between the count of robots and targets and use it as preferential over k in situations where the k coverage for all targets cannot be achieved. More formally, this leads to the following mathematical model (inheriting the notation of the previous section):

$$\text{Minimise} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} + \sum_{i=1}^{n} q x_{i,m+1} \tag{10}$$

$$\text{Subject to} \quad \sum_{j=1}^{m+1} x_{ij} = 1 \qquad i = 1,\dots,n \tag{11}$$

$$\sum_{i=1}^{n} x_{ij} \geq \min\left(k, \left\lfloor \frac{n}{m} \right\rfloor\right) \qquad j = 1,\dots,m \tag{12}$$

$$\sum_{i=1}^{n} x_{ij} \leq \min\left(k, \left\lceil \frac{n}{m} \right\rceil\right) \qquad j = 1,\dots,m \tag{13}$$

$$x_{ij} \in \{0,1\} \qquad i = 1,\dots,n \quad j = 1,\dots,m+1 \tag{14}$$

Where $\lceil x \rceil$ is ceiling of x. Constraints (12) and (13) serve the purpose to limit the number of robots assigned to a target between $\left\lfloor \frac{n}{m} \right\rfloor$ and $\left\lceil \frac{n}{m} \right\rceil$ but not greater than k. All the other equations are the same of $\mathcal{A}_{\text{LinPro}}$.

Using $\mathcal{A}_{\text{LinProF}}$ over $\mathcal{A}_{\text{LinPro}}$ may be preferable when achieving a balanced cover is deemed more important than reaching full k coverage for a smaller number of targets.

5.3 Evaluation: experimental setup

With reference to Table 2, a set of m of objects and n robots are randomly scattered in a square arena with edge length s situated within a Euclidean bidimensional manifold. We simulate the k-coverage problem in a dynamic setting, where objects move continuously within the arena using Lévy walks[19] [104] at an average speed of $\vec{v}_o$. Every object can either be *important* or *unimportant*, depending on the last evaluation of a predicate:

$$\mathcal{P}(o) = o \in O \ \oplus \ \mathbf{x} < \mathbf{P} \mid \mathbf{x} \in \mathcal{U}(\mathbf{0},\mathbf{1}) \ \wedge \ \mathbf{0} < \mathbf{P} < \mathbf{1}$$

namely, the object changes its importance ($\oplus$ indicates a logical exclusive disjunction operation, or xor) if a sample of the uniform distribution in $[0,1]$ is lower than a number P. Predicate $\mathcal{P}$ is evaluated with a Poisson process with rate λ: every time an event of the process happens. The Poisson process has been chosen due to its memory-less behaviour, highlighting the system's response to unpredictability. Robots move at an average speed of $\vec{v}_c$ and can rotate at a maximum angular velocity of ω, their field of view has depth $\mathcal{V}_R$ and angle $\mathcal{V}_\beta$. Robots are programmed to achieve

[16] http://archive.ph/wip/fyfES
[17] It approximates pedestrians' preferred walking speed [14].
[18] This is a conservative assumption based on the performance of modern commercial flying drones see http://archive.is/LhWCk.
[19] We used Lévy walks as they reasonably approximate walking patterns of human beings [72].

Name	Description	Values
m	objects count	100
n	robot count	10, 20, …, 200
s	arena edge length	500 m
$\vec{v}_o$	object average speed[17]	1.4 m/s
$\vec{v}_c$	robot linear velocity[18]	3 m/s
ω	robot's camera angular velocity[18]	$\pi/5$ rad/s
λ	evaluation rate of predicate $\mathcal{P}$	0.05 Hz
P	probablity of switching importance at $\mathcal{P}$ evaluation	0.05
m/n	objects/robots ratio	-
$\mathcal{V}_R$	FOV depth[18]	30 m
$\mathcal{V}_\beta$	FOV angle[18]	$2\pi/3$ rad
k	desired maximum coverage	3
r	robots' communication range	25, 50, …, 200 m
f	round frequency	1 Hz
$\mathcal{A}$	coordination algorithm	see section 5.3.1
T	simulation end time	600 s
W	Willpower for $\mathcal{A}_{\text{ForceField}}$	40

Table 2. List of the variables and their values for the simulations.

k-coverage by running an aggregate algorithm $\mathcal{A}$ with round frequency f. We captured a rendering of the simulated dynamics of the scenarios and produced a video, which has been shared and is freely visible online[20]. The network infrastructure is programmed to allow communication among robots whose distance is within communication range r. Variables and their values are summarised in Table 2.

Once initialised, the simulation is executed for a simulated time $T = 600s$. For each combination of variable values (namely, for each member of the set representing the Cartesian product of the possible values of each variable), 100 simulation runs were executed. Perfect localisation and communication are assumed, no errors are introduced. For all experiments, we measure the average normalised k-coverage as per Equation (1). Data generated by the simulator has been analysed using xarray [45]; visual reports of the data have been created via matplotlib [48].

For the sake of detailed understanding, reproducibility, and reuse, the experiment is public[21], it has been documented for exact reproduction of the results and charts reported in this manuscript, released as open-source, and assigned a permanent DOI reference [35] for archival purposes.

5.3.1 Algorithms. The robot coordination algorithms compared in this work can be classified using three parameters:

(1) *exploration strategy*: defines the behaviour of the robot when the response model can not determine a target to follow (for instance, in case no target is in sight and no information has been received yet from other robots);
(2) *communication strategy*: determines the subset of neighbours each robot communicates with;
(3) *response model*: determines the strategy applied by a robot in response to the available information.

In this paper, we compare the aggregate computing-based algorithms introduced in Section 5 with the state-of-the-art algorithms analysed in [30]. This work represents the current state of the art on the problem at hand, being the online multi-object k-coverage still a relatively new and unexplored problem. As exploration strategies, we compare $\mathcal{A}_{\text{ForceField}}$

[20] https://www.youtube.com/watch?v=yuaY_8Vr3oc
[21] https://github.com/DanySK/Experiment-2019-Smartcam/

(FF) introduced in Section 5.1 and "ZigZag exploration" (ZZ), corresponding to the *Random movement* strategy introduced in [30]. In ZZ, robots generate a random vector and follow it, bouncing off from the arena boundaries; once they reach their destination, they generate a new random vector. In both FF and ZZ, robots are programmed to rotate at maximum angular velocity ω in order to increase the probability of intercepting an interesting object. We consider three communication strategies:

- *no communication* (NoComm), as the name suggests, allows for no information exchange among robots and each robot operates in isolation, this serves as baseline;
- *neighbourhood broadcast* (BC) allows communication with all robots within communication range;
- *smooth* (SM) limits communication based on a "spatio-temporal closeness" metric measuring how long robots within communication range have been close to each other for long periods. Robots learn that they are close if they observe the same objects at the same time. According to the metric mentioned above, the longer they observe the same space, the closer they are. Over time, when robots move and do not observe the same objects any more, they progressively *forget* their previous relationships and reduce their spatio-temporal closeness. We use this measure as a probability to communicate with another robot; over time and space, this value tends to zero [30, 33].

Additionally, we compare the following response models indicating the behaviour of the robot as a reaction to receiving a request:

- *Available (AV).* A robot, if and only if it is not already busy following an object, attempts to cover the most recently requested object from another robot; if multiple requests are present, the nearest is chosen (newest-nearest approach) [30];
- *Received calls (RE).* A robot currently not following an object will provision the object with the least number of requests, as this corresponds to a small number of robots currently observing it [30];
- $\mathcal{A}_{LinPro}$ *(LinPro).* It is a linear programming-based local problem solution, as described in Section 5.2;
- $\mathcal{A}_{LinProF}$ *(LinProF).* Fair version of $\mathcal{A}_{\text{LinPro}}$ introduced in Section 5.2.1.

Since all the response models imply communication, no response model is adopted in the NoComm communication strategy. Finally, we adopted a common and straightforward control strategy for the robots to follow targets. Once a robot decides which target to follow based on its response model, it calculates the coordinates where it should go in order to keep the target at the centre of its FoV. All the infinite points of a circumference centred in the target with radius proportional to the depth of the FoV satisfy this condition; if the robot is the only known observer, it picks the closest point of such circle. In case multiple robots have been assigned to the same target, the devices compete based on their device id (as assigned by the aggregate program execution platform); the device with the lowest id selects the position first, and others, in order, occupy the positions on the circumference maximising the distance among each other: the turn angle (2π) is divided by the number of assigned observers, and each robot position itself at a $2\pi/k$ angle relative to the previous robot. Then velocities are calculated to be the highest possible ones (up to $\vec{v}_c$ for movement and ω for rotation) to reach the position but without going past it. Acceleration and inertia are not simulated. Table 3 summarises the algorithms for this comparison.

5.4 Evaluation: results

The charts in Figure 8 show the average levels of k-coverage achieved for $k = 1$ and $k = 3$ during the simulations, respectively 1-cov and 3-cov. Note that 3 was set as the maximum desired value for k. We chose $k = 3$ deliberately as it

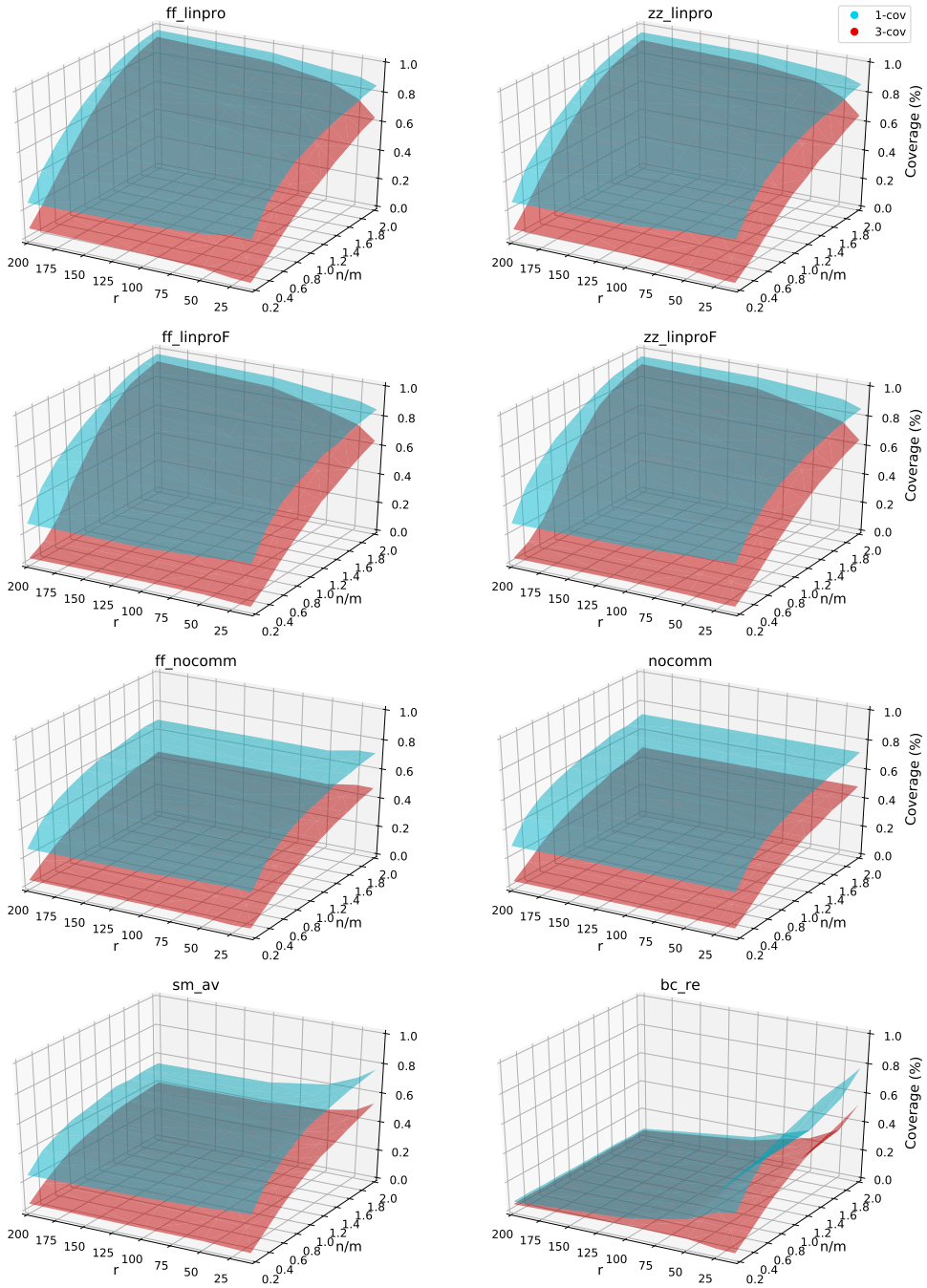

Fig. 8. Compact representation of the performance of the algorithms under test varying the robot/object ratio $\frac{n}{m}$ and the communication radius r. Blue surfaces are 1-coverage levels, red surfaces are 3-coverage. Linear programming-based approaches outperform in most cases the current state of the art. Curiously, BC-RE (bottom right) performance degrade with a higher radius. This is most likely due to increasingly large groups of robots simultaneously called in help once an interesting target is found.

Name	Exploration	Communication	Response
FF-LinPro	ForceField	Neighbourhood Broadcast	LinPro
ZZ-LinPro	ZigZag	Neighbourhood Broadcast	LinPro
FF-LinProF	ForceField	Neighbourhood Broadcast	Fair LinPro
ZZ-LinProF	ZigZag	Neighbourhood Broadcast	Fair LinPro
FF-NoComm	ForceField	Neighbourhood Broadcast	None
NoComm	ZigZag	None	None
SM-AV [30]	ZigZag	Smooth	Available
BC-RE [30]	ZigZag	Neighbourhood Broadcast	Received Calls

Table 3. Algorithms considered in our evaluation, described by component.

allows observation of objects from all angles. A higher value for k would only be necessary if the horizontal visual angle is very tight or if a higher redundancy is required. An approach to calculate a feasible number for k is using the following formula: $k = \frac{360}{\beta} \cdot \varrho$ where ϱ represents the desired redundancy (i.e., how many robots should observe the same area at the same time at a minimum).

In our experiments, $\mathcal{A}_{\text{LinPro}}$ and $\mathcal{A}_{\text{LinProF}}$ show a clear improvement over previous methods found in the literature for the scenario under test. Data shows that these algorithms are susceptible to the communication range: the more accurate the information about the surrounding world, the closer the mathematical model is to the actual problem at hand, the higher the chance that the adopted strategy is close to optimality. The "fair" version of LinPro differs from the other one by attaining a higher 1- and 2-coverage at the expense of a lower of 3-coverage, matching the initial expectation.

Figure 9 depicts detailed results for 1-coverage. Linear programming based algorithms show much better use of a high number of robots, compared to SM-AV and BC-RE, that is, *smooth* (SM) communication in combination with *available* (AV) response and *broadcast* (BC) communication and *received calls* (RE) response (cp. 5). The latter two, and BR-RE in particular, also show a curious behaviour: larger communication ranges do not improve performance but degrade it. This is most likely due to large groups of robots being called for help when a new target is discovered, reducing the ability to discover untracked targets. For large robot/object ratios and very short communication ranges, SM-AV and BC-RE are competitive with $\mathcal{A}_{\text{LinPro}}$ and $\mathcal{A}_{\text{LinProF}}$. Detailed results for the 2-coverage presented in fig. 10 show an appreciable improvement of $\mathcal{A}_{\text{LinProF}}$ over pure $\mathcal{A}_{\text{LinPro}}$. Response to higher communication ranges is similar to those discussed for 1-coverage. Finally, Figure 11 shows detailed data on 3-coverage, which was the target coverage for our experiment. In this case, the relation between $\mathcal{A}_{\text{LinProF}}$ and $\mathcal{A}_{\text{LinPro}}$ predictably reverses: $\mathcal{A}_{\text{LinPro}}$, focussing on actual k-coverage, actually achieves better k-coverage. $\mathcal{A}_{\text{LinProF}}$, on the other hand, tries to balance the coverage over as many targets as possible, preferring lower coverage for many targets over higher coverage for fewer.

Results depicted in Figures 8 to 11 show very similar performance across the proposed variants. To better show the difference, we summarised the data for a fixed range $r = 100$m in Figure 12, where we also tested for a robots/objects ratio $\frac{n}{m} > 1$. The proposed algorithms can better scale with a larger number of robots compared to the baseline. Data also shows how the "fair" version achieves higher 2-coverage at the cost of lower 3-coverage.

Table 4 compares the average coverage achieved across the board. Both the novel algorithms outperform SM-AV and BC-RE under most conditions. SM-AV shows poor performances in every case but with the shortest communication range. This is likely due to two leading causes. First, the algorithm does not consider the maximum desired value for k, assigning too many robots to each target. Second, the P_{smooth} formula [30], which computes the probability that

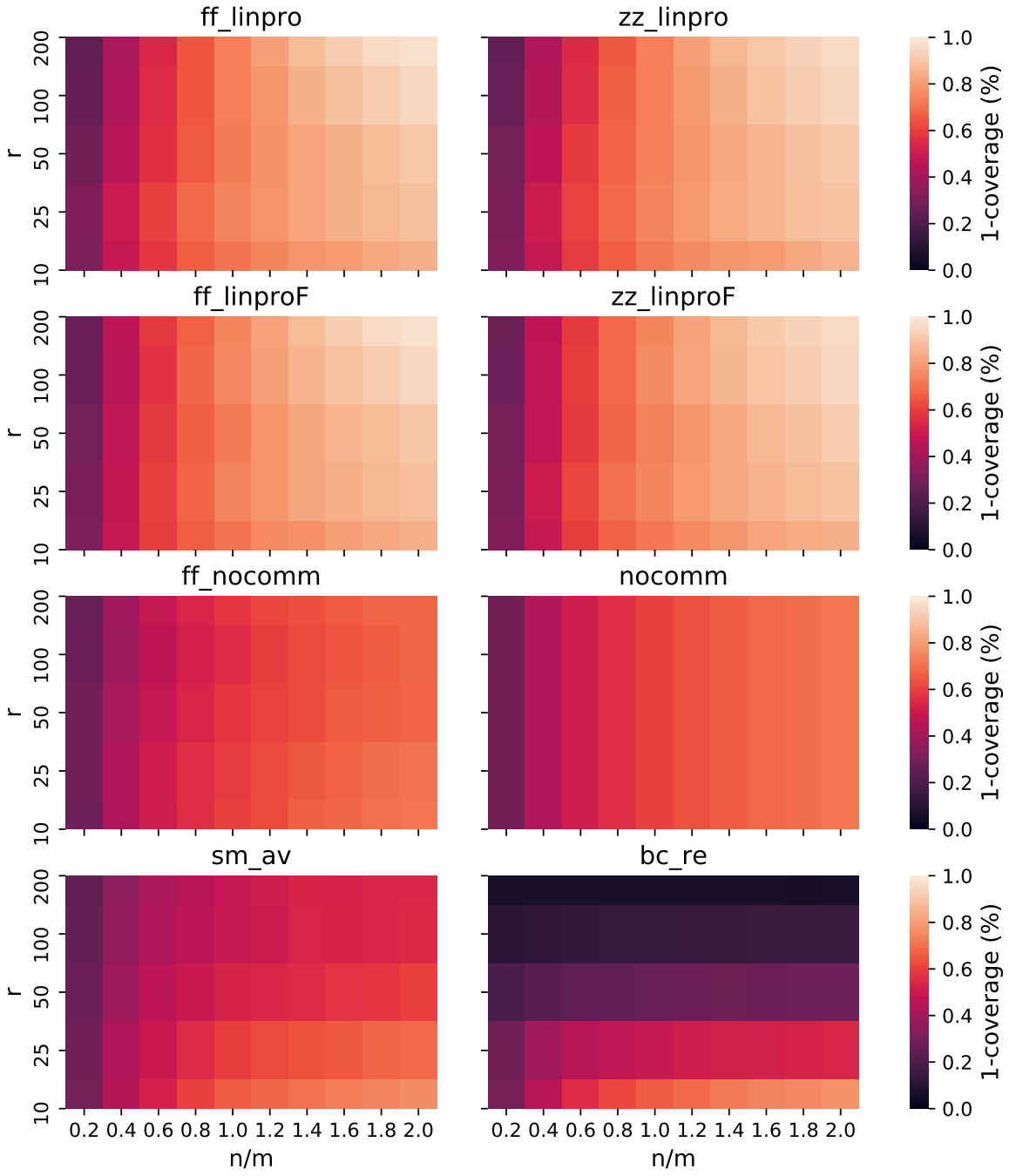

Fig. 9. Detailed 1-coverage performance for the algorithms under testing. $\mathcal{A}_{\mathsf{LinPro}}$ and $\mathcal{A}_{\mathsf{LinProF}}$ primarily benefit from greater communication ranges, while both BC-RE and SM-AV begin to suffer in case too many devices must coordinate at once. As expected, $\mathcal{A}_{\mathsf{LinProF}}$ shows (marginally) better performance for 1-coverage than pure $\mathcal{A}_{\mathsf{LinPro}}$ in some cases. Force field-based exploration does not impact results.

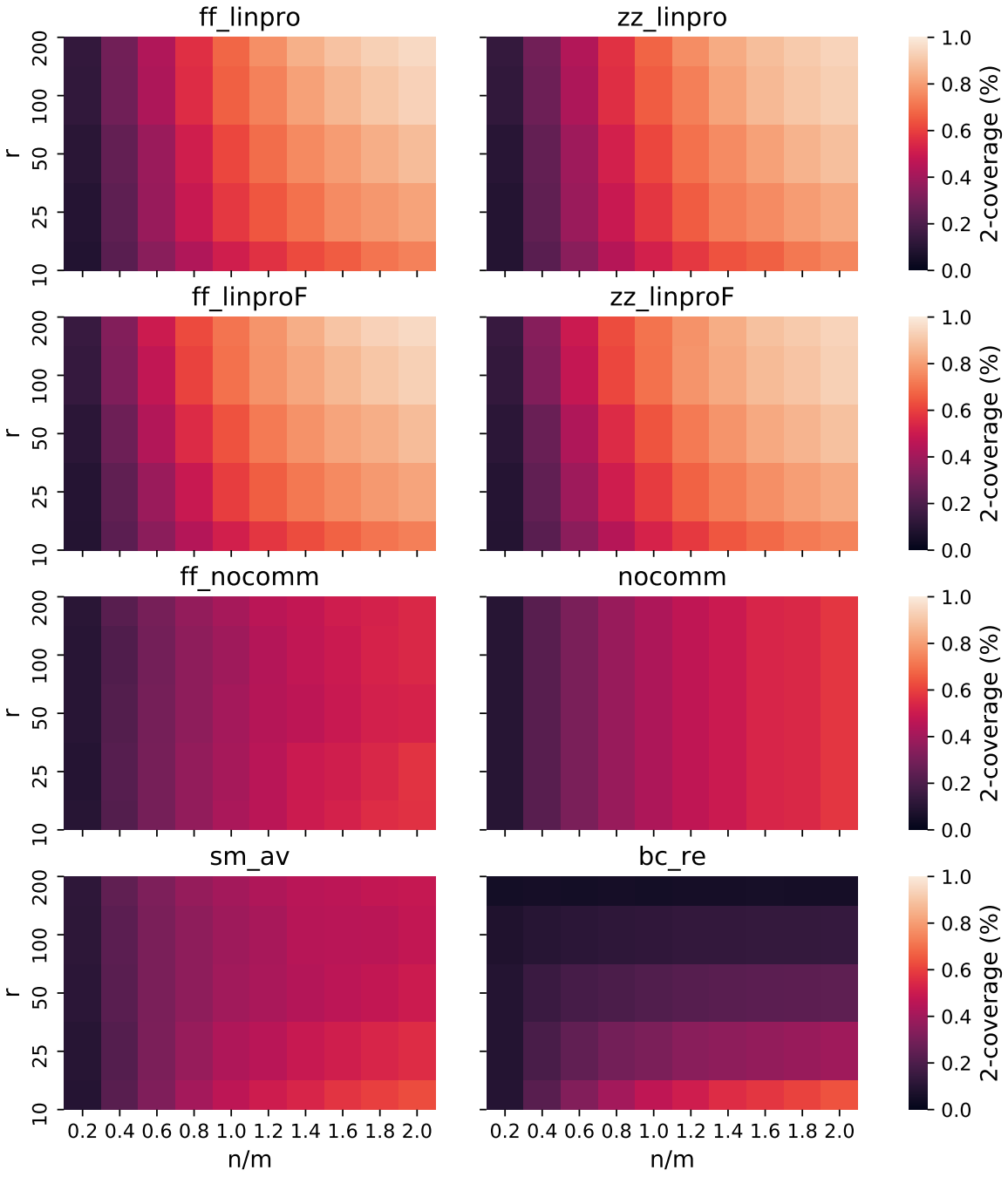

Fig. 10. Detailed 2-coverage performance for the algorithms under testing. $\mathcal{A}_{\mathsf{LinPro}}$ and $\mathcal{A}_{\mathsf{LinProF}}$ show better performance than baselines in almost all conditions. Both benefit from larger communication ranges, while on the contrary BC-RE and SM-AV suffer this condition. As expected, $\mathcal{A}_{\mathsf{LinProF}}$ performance for 2-coverage is superior to $\mathcal{A}_{\mathsf{LinPro}}$. Force field-based exploration does not impact results perceptibly.

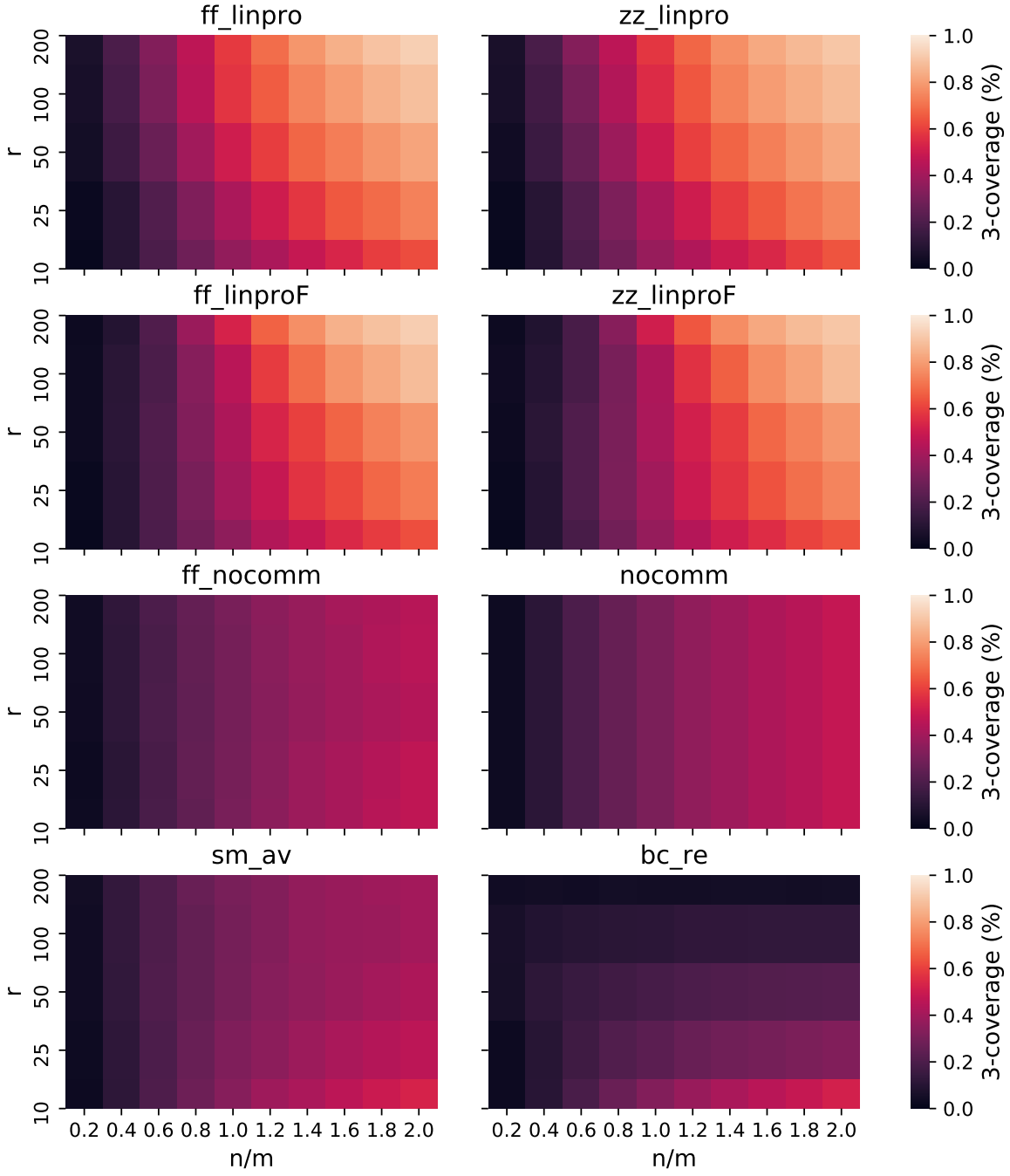

Fig. 11. Detailed 3-coverage performance for the algorithms under testing. $\mathcal{A}_{\mathsf{LinPro}}$ shows better performance in all conditions. $\mathcal{A}_{\mathsf{LinProF}}$ still outperforms the baseline algorithms, but obtains lower 3-coverage w.r.t. plain $\mathcal{A}_{\mathsf{LinPro}}$ due to its "fair" nature favouring some coverage for most targets over actual k-coverage over few. Force field-based exploration does not impact results perceptibly.

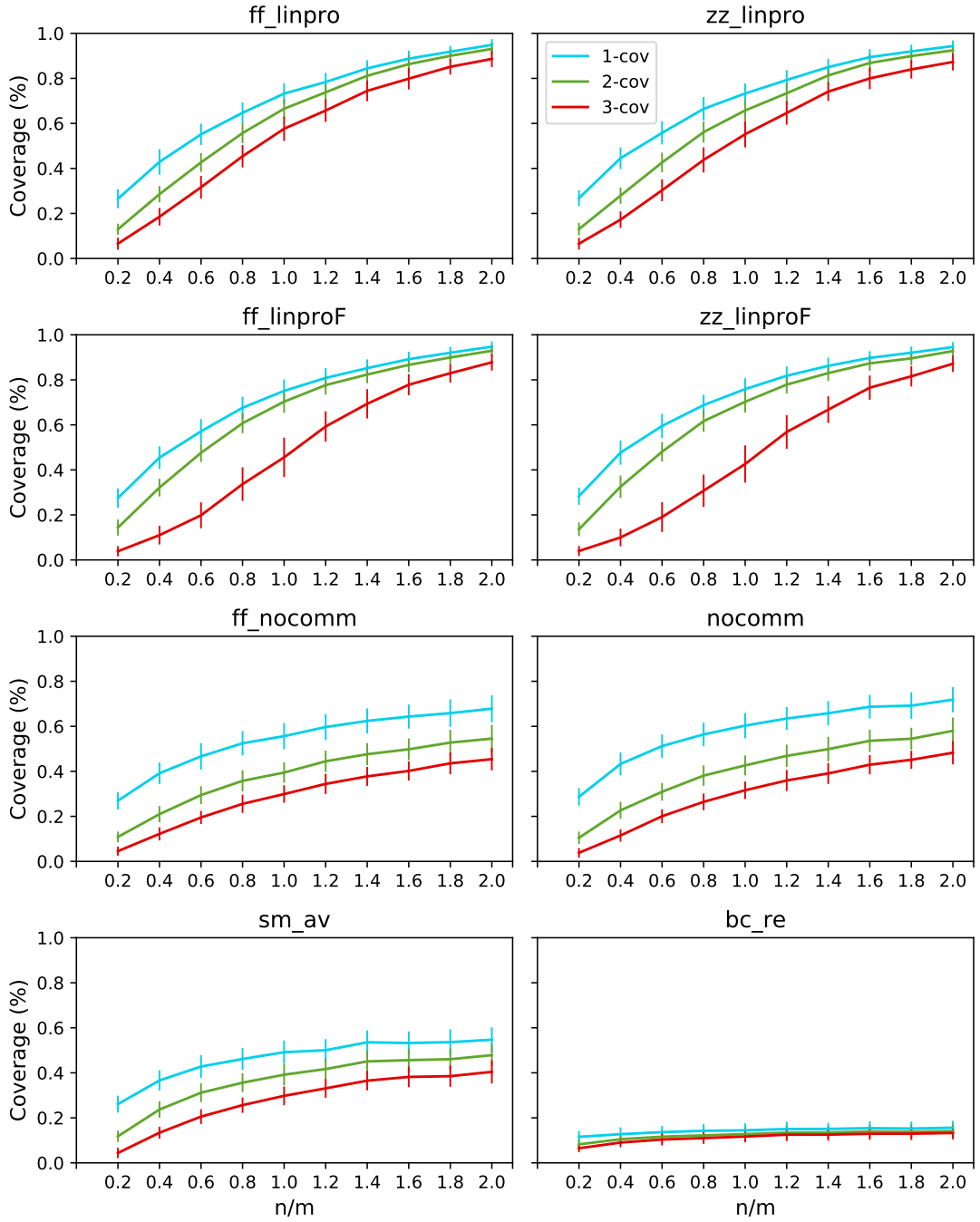

Fig. 12. Mean OMC_k by varying the ratio between the number of robots and objects, with a fixed communication range of 100m. Linear programming based algorithms can deal much better than alternatives when the ratio between robots and targets grows. The "fair" version of these algorithms obtains similar 1-coverage, outperforms the base one for 2-coverage, but achieves worse results for 3-coverage.

r	Approach	Ratio n/m					
		0.2	0.6	1.0	1.2	1.6	2.0
25	ff_linpro	0.03 (0.02)	0.21 (0.04)	0.43 (0.05)	0.50 (0.05)	0.65 (0.05)	0.74 (0.05)
	zz_linpro	0.03 (0.02)	0.21 (0.04)	0.42 (0.04)	0.51 (0.05)	0.65 (0.05)	0.75 (0.04)
	ff_linproF	0.03 (0.02)	0.20 (0.04)	0.41 (0.05)	0.49 (0.04)	0.62 (0.05)	0.72 (0.05)
	zz_linproF	0.02 (0.02)	0.21 (0.04)	0.40 (0.05)	0.50 (0.06)	0.64 (0.05)	0.74 (0.05)
	ff_nocomm	0.03 (0.02)	0.19 (0.04)	0.30 (0.04)	0.35 (0.04)	0.41 (0.05)	0.47 (0.05)
	nocomm	0.04 (0.02)	0.20 (0.03)	0.32 (0.04)	0.36 (0.05)	0.43 (0.04)	0.48 (0.05)
	sm_av	0.04 (0.02)	0.20 (0.03)	0.32 (0.04)	0.35 (0.04)	0.42 (0.05)	0.46 (0.05)
	bc_re	0.04 (0.02)	0.16 (0.03)	0.24 (0.04)	0.27 (0.04)	0.30 (0.05)	0.33 (0.05)
50	ff_linpro	0.05 (0.02)	0.27 (0.04)	0.51 (0.05)	0.60 (0.05)	0.73 (0.05)	0.82 (0.04)
	zz_linpro	0.05 (0.02)	0.26 (0.04)	0.50 (0.06)	0.60 (0.05)	0.73 (0.05)	0.83 (0.04)
	ff_linproF	0.04 (0.02)	0.21 (0.05)	0.43 (0.06)	0.53 (0.06)	0.68 (0.06)	0.78 (0.05)
	zz_linproF	0.03 (0.02)	0.21 (0.05)	0.42 (0.06)	0.52 (0.06)	0.68 (0.06)	0.79 (0.05)
	ff_nocomm	0.04 (0.02)	0.20 (0.03)	0.30 (0.04)	0.34 (0.04)	0.40 (0.05)	0.44 (0.05)
	nocomm	0.04 (0.02)	0.20 (0.03)	0.32 (0.04)	0.36 (0.05)	0.43 (0.04)	0.48 (0.05)
	sm_av	0.04 (0.02)	0.21 (0.04)	0.30 (0.04)	0.34 (0.04)	0.39 (0.04)	0.43 (0.04)
	bc_re	0.06 (0.02)	0.15 (0.03)	0.19 (0.03)	0.20 (0.03)	0.22 (0.04)	0.23 (0.04)
100	ff_linpro	0.07 (0.03)	0.32 (0.05)	0.58 (0.05)	0.66 (0.05)	0.80 (0.05)	0.89 (0.04)
	zz_linpro	0.07 (0.03)	0.30 (0.05)	0.55 (0.06)	0.65 (0.05)	0.80 (0.05)	0.87 (0.04)
	ff_linproF	0.04 (0.02)	0.20 (0.06)	0.46 (0.09)	0.59 (0.07)	0.78 (0.05)	0.88 (0.04)
	zz_linproF	0.04 (0.02)	0.19 (0.07)	0.43 (0.08)	0.57 (0.07)	0.77 (0.05)	0.87 (0.04)
	ff_nocomm	0.05 (0.02)	0.20 (0.03)	0.30 (0.04)	0.34 (0.05)	0.40 (0.04)	0.45 (0.05)
	nocomm	0.04 (0.02)	0.20 (0.03)	0.32 (0.04)	0.36 (0.05)	0.43 (0.04)	0.48 (0.05)
	sm_av	0.04 (0.02)	0.21 (0.03)	0.30 (0.04)	0.33 (0.04)	0.38 (0.05)	0.40 (0.05)
	bc_re	0.07 (0.02)	0.10 (0.03)	0.12 (0.03)	0.13 (0.03)	0.13 (0.03)	0.13 (0.03)

Table 4. Comparison of mean OMC_k achieved by different approaches with different communications ranges r and different ratios for objects/cameras, the standard deviation is indicated in brackets.

a robot will call another one for help to follow a target, asymptotically converges to zero. Consequently, the longer the simulation runs and the more the robots encounter each other, the higher is the number of notifications sent; moreover, algorithms are executed with a frequency of 1Hz, thus generating a high number of notifications. Lower frequencies might allow improvement by preventing calling for help too often. BC-RE shows the same problems as SM-AV, considerably worsened by the fact that it performs broadcasts. Despite these problems, BC-RE remains a simple approach and still works better than NoComm for short communication ranges.

Force field-based exploration deserves some discussion as well. Apparently, it does not show any tangible effect on the coverage along the whole experiment. The main reason is that it is used as an exploration strategy in the initial phase, then replaced by other algorithms for most of the time. As such, its impact is lesser and lesser with the experiment length. To better understand if there is any benefit for the initial exploration, we isolated in Figure 13 the first 100 seconds of simulation. Data shows that force field-based exploration outperforms the baseline ZZ algorithm during the bootstrap phase, however, this edge gets lower and lower with time. Data shows that force field exploration is a valid companion for any response model compared to the baseline: this is most likely due to robots repelling each other from the beginning, and thus covering a larger area in the attempt to maximise the distance from each other.

6 RELATED WORK

6.1 Problems related to OMOkC

This section briefly mentions well-known problems related to OMOkC and CMOMMT, providing corresponding references for the reader to be acquainted with the current state of the art. The first problem is the *coverage maximisation problem*, addressed when deploying camera networks and deciding where to position and orient each camera to maximise the observed area. This problem, also known as the *Art Gallery problem*, has been researched quite intensively [47, 68, 79, 83]. To cover an area with a defined number of cameras, Fusco and Gupta [38] utilise a simple greedy algorithm. Dieber et al. [25] utilise an evolutionary algorithm to identify the optimal location and orientation for PTZ cameras. They further combine this with market-based approaches to assign moving targets to static cameras [74]. Rudolph et al. [76]

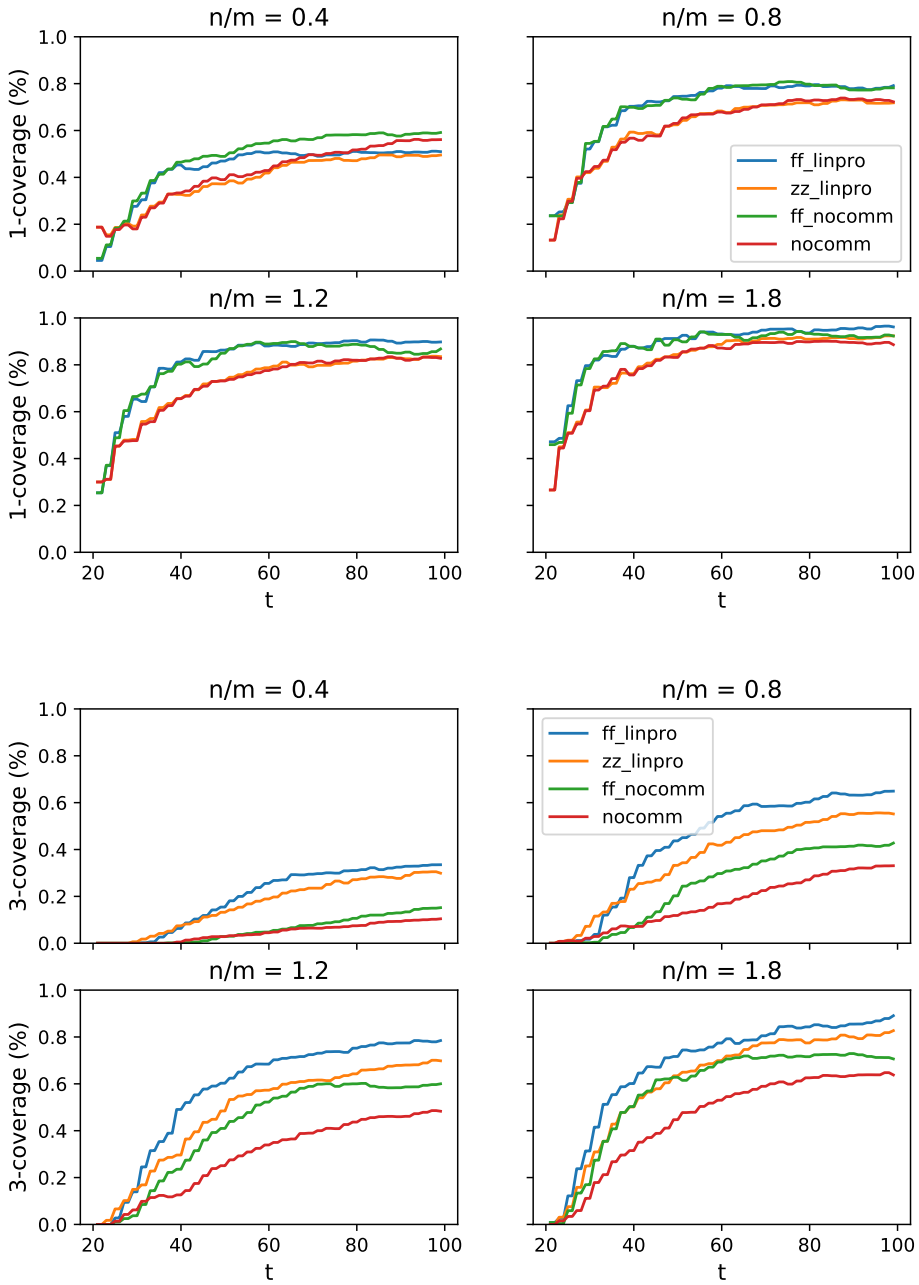

Fig. 13. Mean OMC_k observed during the first 100s of simulation with a fixed communication range of 100m. Results show that force-field-based exploration (blue and green lines) perform better than zig-zag based exploration (yellow and red lines) during the initial phase of the simulation, when exploration algorithms are more exercised.

enable individual PTZ cameras, able to change their orientation, to learn their local performance using W-Learning. By exchanging information about their current state, they can optimise their orientation over time. Arslan et al. [1] propose novel conic Voronoi diagrams based on the visual quality of cameras. They utilise the information from all cameras to determine the optimal orientation to maximise the coverage of a given area. Using a density estimation, Hatanaka et al. [42] utilise a distributed gradient descent algorithm to define the optimal orientation of cameras to cover all targets in the area. To optimally place and orient a set of cameras, several approaches rely on Particle Swarm Optimisation in a centralised as well as distributed fashion [26, 56, 100–102].

An extension to the coverage maximisation problem is the k-coverage problem. Here a set of sensors needs to be placed to cover specific points in the environment with k sensors [46]. This not only allows to turn off individual sensors without leaving the area uncovered, but also increases the amount of gathered information and, therefore, accuracy and precision when multiple sensors are operational simultaneously. Hefeeda and Bagheri [43] propose a distributed approximate algorithm for omnidirectional sensors allowing close to optimal sensor placement. Li and Kao [53] utilise Voronoi diagrams to estimate the location of individual sensors and hence adjust their location accordingly. Similarly, Stergiopoulos and Tzes [85] use Voronoi-alike distance measures to guide mobile, non-uniform but omnidirectional, sensors for optimal coverage of an area.

The last related problems are *search-and-rescue* operations, also known as the *detect-and-track problem*. Here, a set of agents is tasked to find objects or targets in a given area. Targets might be stationary (search-and-rescue) [41] or mobile (detect-and-track) [39]. Stormont [86] presents different types of robots and how to employ them as a swarm to quickly cover an area and find potential victims in a disaster scenario. Waharte and Trigoni [97] explicitly use unmanned aerial vehicles (UAVs) to support ground robots and cover a defined area faster. Using the RSSI value of individual UAVs, Ruetten et al. [77] enable UAVs to find optimal locations to cover a given area. Path planning is at the core of the work of Macwan et al. [54] to optimise the movement of all UAVs in the area and ensure the entire environment is covered at the end of the operation. Scherer et al. [80] propose a hybrid approach between centralised and decentralised decisions for different tasks in search-and-rescue operations, while Yanmaz et al. [103] focus on generating ad-hoc networks for localised coordination and decision-making for subsets of UAVs.

6.2 State of the art in decentralised OMOkC

There is a wide range of coordination and control algorithms for multi-camera and multi-robot systems [57, 75]. Usually, they differ according to the task to be accomplished and whether the approach relies on a central component, gathering information and coordinating individual robots, or is purely distributed and self-organised. In this article, we focus on the online multi-object k-coverage problem (OMOkC). While closely related to the cooperative multi-robot observation of multiple moving targets (CMOMMT), whose state-of-the-art solutions are surveyed by Khan et al. [50], it differs significantly: the number of cooperative robots is unknown, and the number of objects is neither constant nor known to the robots. Furthermore, OMOkC requires *multiple* cameras to observe the same target simultaneously. While this is positive for the observation of the object as targets can be observed from different angles, the robots need to be coordinated to ensure over-provisioning, that is, the case in which too many robots observe a single target is avoided. A related sub-problem is autonomous search and rescue (ASR) operations, often tackled by swarms of (collaborative) robots [6, 58, 77, 86]. However, in ASR, the targets are often stationary and do not require multiple robots to attend them simultaneously. Nevertheless, to initially cover and observe the area, swarming techniques can be utilised.

In [32], Esterle and Lewis rely on purely distributed approaches. They enable the individual robots to learn about their local environment, including other robots and analyse the potential of the topological neighbourhood of interaction.

Later on, Esterle and Lewis also compare the performance of the distributed approaches against a naive centralised approach, gathering all information of all robots to coordinate them [29]. While the centralised approach dominates the distributed approaches when it comes to achieved coverage, the centralisation generates an additional overhead in communication.

Another distributed approach incorporates the observed behaviour of other robots in the network into their decision processes [34], using ideas of networked self-awareness [28]. King et al. [51] use entropy to attract robots towards individual objects. To avoid over-compensation, they also introduce a suppression signal. Rather than attracting robots towards individual objects, Frashieri et al. [37] enable robots to join a coalition for each object based on their individual willingness to interact. While this approach generates good results on the OMC_k metric, the coalition formation requires additional communication.

When all robots in a network operate towards the common goal of covering all objects with k cameras, they can quickly cluster in specific areas. This makes objects appearing in the remaining environment prone to remain undetected and missed by the network. To overcome this, dynamic team formation can be used, where each team has a different goal, i.e. following objects or covering the remaining area to ensure a majority of appearing objects are detected [27].

6.3 Similar and competing programming models

The aggregate computing paradigm adopted in this paper has its roots in *spatial computing* and *collective adaptive systems* research, surveyed in [9] and more recently, from the point of view of coordination, in [94]. Research fields recognising the importance of the spatial and collective aspect for computing and interaction include *multi-agent systems* [99], where various organisational paradigms [44] have emerged to take into account the *social* dimension, as well as *mobile ad-hoc networks (MANETs)* and *wireless sensor networks (WSNs)* [87], where it is common to program the collective behaviour of large networks of devices producing and collecting information. Such an amount of related work can be classified along multiple dimensions.

First, there are extensions to traditional approaches that aim to simplify the development of networked applications through proper abstractions. For instance, *Abstract Regions* [98] provides a collective communication interface for region- and neighbourhood-oriented data propagation and collection.

At a step further, some approaches address so-called *ensembles*, i.e., dynamic formations of devices. Examples include *DEECo (Distributed Emergent Ensembles of Components)* [15], where components can only communicate by dynamically binding together through ensembles (formed according to a membership condition), and *SCEL (Service Component Ensemble Language)* [24], which leverages attribute-based communication.

Finally, there are so-called *macro-programming approaches*, which consider an entire network of devices as the programming target. Examples of this family include *Chronus* [96], a spatio-temporal DSL for data gathering and event detection in WSNs, and *Sense2P* [20], a logic macro-programming system for solving queries in WSNs.

6.4 Simulators for network performance and physical interactions

Once the software reaches reasonable maturity, interaction among devices must be validated as compatible with the available networking infrastructure. A network-focussed simulator is the right tool for the job. An example is Mobile MultiMedia Wireless Sensor Network (M3WSN) [106], which focuses on the network-level simulation of image transmissions, and can easily be adapted as a camera network simulator by using real-world video streams to mimic the simulated cameras. Similarly, WiSE-Mnet++ [78] combines these ideas of real-world and synthetic videos with

improved network simulation. This is done by employing the dedicated network-simulator OMNet++ [90] for discrete events and Castalia [13] for wireless networks and modelling radio channels.

Finally, simulators with higher fidelity to the real-world can help produce and inspect corner cases before deployment and provide a platform for developing software closer to the hardware (e.g., object detection from a video feed). However, generating a complex virtual world is usually resource expensive. Several tools for camera networks simulation leverage recent developments in rendering synthetic worlds realistically in three dimensions [81, 84, 88]. We also remark that there is an ongoing *e-robotics* trend in (swarm) robotics research where increasingly sophisticated 3D-graphical and physics-rich simulators (e.g., Gazebo [70], ARGoS [69], AirSim [82]) – sometimes building on game engines such as Unreal Engine or Unity 3D [23] – are exploited to develop simulations with a certain degree of physical fidelity. However, to keep computational expenses low, we perform simulations in 2D. An extension to 3D can be achieved by incorporating the third dimension in the location and velocity vectors for objects and robots with vision sensors and adding a vertical angle to the field of view.

7 LIMITATIONS AND FUTURE WORK

In this article, we focus our contribution on the following aspects:

(1) demonstrating the feasibility of engineering distributed solutions for the OMOkC problem within the aggregate computing framework;
(2) provide evidence that solutions built in this way are competitive with the current state of the art;
(3) making the tools for developing and evaluating solutions available to other researchers.

Naturally, some issues are not considered in the evaluation presented in this work. In this section, we aim to state such limitations clearly and outline potential future work.

7.1 Evaluation

7.1.1 Robot simulation. Our evaluation does not consider energy consumption even though mobile devices (e.g., drones) rely on energy stored in batteries to operate, thus their working time is generally limited. While we do not focus on energy consumption in this work, taking the energy consumption and limits into account could lead to an extended version of LinPro where these costs are factored in and considered in the solution. Under the point of view of the proposed simulation framework, we note that the level of abstraction proposed abstracts away the realistic modelling of the robot's hardware (electric engines, control electronics, and so on). Depending on the degree of realism that is required, the following strategies can be pursued:

- estimation of energy cost via proxy metrics; or
- extension of the simulation model.

In the former case, power consumption may be estimated by relying on data already available in the simulator, such as the travelled length. This strategy allows using the currently provided toolchain at the price of realism. In the latter case, a detailed model of the robots, including a model of the energy consumption, is required. How detailed depends on the required level of realism, up to the point that the proposed simulation platform is not equipped to provide support to. For instance, realistically modelling the engines' work, or physically challenging and possibly evolving conditions (such as wind for flying devices or terrain asperities for ground vehicles), fall outside the kind of details the simulator has been designed to support. In these cases, it is probably worth extending a different simulator, with a realistic hardware model in place, with the required capabilities. We note, however, that the more detailed is a model, the harder it is to

scale up. Our goal in this work is to demonstrate that it is possible (and practical and convenient indeed) to consider the ensemble of robots as an aggregate system. To this end, we believe it is more valuable to show that it is indeed possible to reach coordination among a high number of mobile devices rather than precisely measuring their power drain.

In this work, we consider the same FOV and attributes for all robots. However, a real system may be composed of several different device types, differing e.g., for mobility (static, predefined paths, angular or translational mobility), field of view (depth, width, single or multiple), and several miscellaneous factors (power usage and source, zooming capabilities, processing power, etc.). Different mobility capabilities and field of view changes can be simulated within the proposed toolchain, preserving the ability to scale up to thousands of (different) devices. Despite that, the evaluation of this paper is intended to provide insights on the feasibility of an aggregate computing-based approach to OMOkC, and as such, we did not include several different device types, which can be targeted in future works. Furthermore, some investigations involving more realistic modelling of the world are outside of what is readily reproducible in the proposed toolchain. Considerations similar to those previously made for realistically modelling how the hardware works for power use apply to several other features, for instance, image recognition capabilities: in this work, we consider devices to be able to tell whether a target is interesting with precision, and to be able to locate and recognise it. While accounting for an error with some well-known distribution would be feasible within the proposed framework, an in-depth analysis including authentic imagery and on-the-fly recognition is beyond the scope of the proposed tools.

We simulate on a fixed-sized arena, and we change density by changing the device count. Further investigation could be devoted to analysing the impact of different device speeds and arena sizes. This would explore the impact of different arena sizes and the relation of different device movement speeds to the OMOkC problem.

7.1.2 Environment simulation.

Physical environment. In future work, it would be interesting to run experiments using more realistic arenas. The simulator is already equipped to import floor maps and model static obstacles (a capability already exploited in other works, see, e.g., [95]), which should allow for collecting evidence of how the system can perform in an actual deployment.

Network. The current simulation infrastructure abstracts from realistic modelling of the underlying network. A possible extension to this work includes integrating Alchemist with a dedicated network simulator such as NS3 [73] or Omnet++ [91]. This would produce a hybrid environment that provides insights both for large-scale, highly dynamic experiments (focussing on algorithmic evaluation) and for smaller-scale evaluations with realistic networking (simulation oriented to predict after-deployment performance).

7.2 Software evolution

Approaching device coordination at the aggregate level simplifies coordination by hiding details under-the-hood, thus promoting the development of richer software. However, such development requires the correct abstractions in terms of mechanisms and whole libraries providing easy access to advanced coordination mechanisms [36]. Potential future work is thus the development of a domain-specific API of aggregate behaviour, designed explicitly for coordinating networks of robots with vision sensors. In particular, it would be interesting to leverage the notion of *aggregate process* [19] to regulate the formation of dynamic coalitions of robots and consider adopting a full-fledged version of the *self-organising coordination regions* pattern [63] to organise the coordination and decision-making at larger scales.

This evolution may include an evolution of the proposed LinPro algorithm. As discussed in Section 7, the algorithm could be extended to capture costs related to moving robots (e.g., battery power consumption). Also, different techniques could be used for the optimisation phase: since the problem is solved using partial information, the capability of the optimisation algorithm to provide suitable solutions with limited information is critical. Examples of different possible heuristics are particle swarm optimisation and simulated annealing. Finally, the proposed version $\mathcal{A}_{\text{LinPro}}$ does not consider objects that are marked as not important, even though they may become targets in the future. This information could be exploited by future works, improving the performance.

7.3 Safety and security

Aggregate computing provides basic support for resiliency, based on abstraction from low-level details of device distribution and networking [12], and on a continuous execution model where changes in context automatically trigger local (and, consequently, global) adaptation. However, future work is required to verify the actual robustness of the proposed algorithms in front of unpredicted failures. Little work is instead available on security, namely, detecting, isolating, and counteracting proactive malicious behaviour, such as hijacked robots. Some preliminary work has been proposed based on *computational trust* [16] at the application level or by delegating most of the security to the underlying platform [64]; however, further work is necessary to establish solid security practices [62]. This is especially true in case the robot system is deployed to perform collective surveillance [22].

8 CONCLUSION

In this paper, we address the online multi-object k-coverage problem and accordingly provide a contribution in terms of (i) an aggregate computing solution to decentralised multi-robots with vision sensors coordination; (ii) a toolchain for experimentation and development, including a publicly available extension to an existing simulator for large-scale systems of multi-robots with vision sensors; and (iii) two novel k-coverage algorithms that improve over the state of the art. Systems situated in the real-world environment often have to perform actions related to their physical location. In this paper, we use a novel paradigm called *aggregate computing* to implement the behaviour of entire *ensembles* instead of individual devices. We validate our approach via simulation; to this end, we extend the Alchemist simulator with features specific to the simulation of robots with vision sensors, enabling large-scale simulations of mobile vision sensor networks. By gathering information of the robot proximity and modelling it as an optimisation problem, we leverage a linear programming-based heuristic to enable the set of autonomous robots to outperform previously proposed approaches in covering objects over a period of time with k robots.

9 ACKNOWLEDGEMENT

The idea and initial effort behind this work originated from the discussion during the GI-Dagstuhl Seminar 18343 “Software Engineering for Intelligent and Autonomous Systems (SEfIAS)”. This work has been partially supported by the Italian PRIN project N. 2017KRC7KT “Fluidware”.

REFERENCES

[1] Omur Arslan, Hancheng Min, and Daniel E Koditschek. 2018. Voronoi-based coverage control of pan/tilt/zoom camera networks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1–8.

[2] Giorgio Audrito. 2020. FCPP: an efficient and extensible Field Calculus framework. In *IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2020, Washington, DC, USA, August 17-21, 2020*. IEEE, 153–159. DOI: http://dx.doi.org/10.1109/ACSOS49614.2020.00037

[3] Giorgio Audrito, Sergio Bergamini, Ferruccio Damiani, and Mirko Viroli. 2020. Resilient Distributed Collection Through Information Speed Thresholds. In *Coordination Models and Languages - 22nd IFIP WG 6.1 International Conference, COORDINATION 2020, Held as Part of the 15th International Federated Conference on Distributed Computing Techniques, DisCoTec 2020, Valletta, Malta, June 15-19, 2020, Proceedings (Lecture Notes in Computer Science)*, Simon Bliudze and Laura Bocchi (Eds.), Vol. 12134. Springer, 211–229. DOI: http://dx.doi.org/10.1007/978-3-030-50029-0_14

[4] Giorgio Audrito, Roberto Casadei, Ferruccio Damiani, Danilo Pianini, and Mirko Viroli. 2021. Optimal resilient distributed data collection in mobile edge environments. *Comput. Electr. Eng.* 96, Part (2021), 107580. DOI: http://dx.doi.org/10.1016/j.compeleceng.2021.107580

[5] Giorgio Audrito, Mirko Viroli, Ferruccio Damiani, Danilo Pianini, and Jacob Beal. 2019. A Higher-Order Calculus of Computational Fields. *ACM Trans. Comput. Log.* 20, 1 (2019), 5:1–5:55. DOI: http://dx.doi.org/10.1145/3285956

[6] M. Bakhshipour, M. [Jabbari Ghadi], and F. Namdari. 2017. Swarm robotics search & rescue: A novel artificial intelligence-inspired optimization approach. *Applied Soft Computing* 57 (2017), 708 – 726. DOI: http://dx.doi.org/https://doi.org/10.1016/j.asoc.2017.02.028

[7] Michael J. Bannister, David Eppstein, Michael T. Goodrich, and Lowell Trott. 2012. Force-Directed Graph Drawing Using Social Gravity and Scaling. In *Proceedings of the 20th International Conference on Graph Drawing (GD'12)*. Springer-Verlag, Berlin, Heidelberg, 414–425. DOI: http://dx.doi.org/10.1007/978-3-642-36763-2_37

[8] Jacob Beal and Jonathan Bachrach. 2006. Infrastructure for Engineered Emergence on Sensor/Actuator Networks. *IEEE Intell. Syst.* 21, 2 (2006), 10–19. DOI: http://dx.doi.org/10.1109/MIS.2006.29

[9] Jacob Beal, Stefan Dulman, Kyle Usbeck, Mirko Viroli, and Nikolaus Correll. 2012. Organizing the Aggregate: Languages for Spatial Computing. Vol. abs/1202.5509. http://arxiv.org/abs/1202.5509

[10] Jacob Beal, Danilo Pianini, and Mirko Viroli. 2015. Aggregate Programming for the Internet of Things. *IEEE Computer* 48, 9 (2015), 22–30. DOI: http://dx.doi.org/10.1109/MC.2015.261

[11] Jacob Beal, Kyle Usbeck, Joseph P. Loyall, and James M. Metzler. 2016. Opportunistic Sharing of Airborne Sensors. In *International Conference on Distributed Computing in Sensor Systems, DCOSS 2016, Washington, DC, USA, May 26-28, 2016*. IEEE Computer Society, 25–32. DOI: http://dx.doi.org/10.1109/DCOSS.2016.43

[12] Jacob Beal, Mirko Viroli, Danilo Pianini, and Ferruccio Damiani. 2017. Self-Adaptation to Device Distribution in the Internet of Things. *ACM Trans. Auton. Adapt. Syst.* 12, 3, Article 12 (Sept. 2017), 29 pages. DOI: http://dx.doi.org/10.1145/3105758

[13] Athanassios Boulis. 2007. Castalia: revealing pitfalls in designing distributed algorithms in WSN. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems, SenSys 2007, Sydney, NSW, Australia, November 6-9, 2007*. 407–408. DOI: http://dx.doi.org/10.1145/1322263.1322318

[14] Raymond C. Browning, Emily A. Baker, Jessica A. Herron, and Rodger Kram. 2006. Effects of obesity and sex on the energetic cost and preferred speed of walking. *Journal of Applied Physiology* 100, 2 (Feb. 2006), 390–398. DOI: http://dx.doi.org/10.1152/japplphysiol.00767.2005

[15] Tomás Bures, Ilias Gerostathopoulos, Petr Hnetynka, Jaroslav Keznikl, Michal Kit, and Frantisek Plasil. 2013. DEECO: an ensemble-based component system. In *CBSE'13, Proceedings of the 16th ACM SIGSOFT Symposium on Component Based Software Engineering, part of Comparch '13, Vancouver, BC, Canada, June 17-21, 2013*, Philippe Kruchten, Dimitra Giannakopoulou, and Massimo Tivoli (Eds.). ACM, 81–90. DOI: http://dx.doi.org/10.1145/2465449.2465462

[16] Roberto Casadei, Alessandro Aldini, and Mirko Viroli. 2018. Towards attack-resistant Aggregate Computing using trust mechanisms. *Sci. Comput. Program.* 167 (2018), 114–137. DOI: http://dx.doi.org/10.1016/j.scico.2018.07.006

[17] Roberto Casadei, Danilo Pianini, Andrea Placuzzi, Mirko Viroli, and Danny Weyns. 2020a. Pulverization in Cyber-Physical Systems: Engineering the Self-Organizing Logic Separated from Deployment. *Future Internet* 12, 11 (2020), 203. DOI: http://dx.doi.org/10.3390/fi12110203

[18] Roberto Casadei, Mirko Viroli, Giorgio Audrito, and Ferruccio Damiani. 2020b. FScaFi : A Core Calculus for Collective Adaptive Systems Programming. In *Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part II (Lecture Notes in Computer Science)*, Tiziana Margaria and Bernhard Steffen (Eds.), Vol. 12477. Springer, 344–360. DOI: http://dx.doi.org/10.1007/978-3-030-61470-6_21

[19] Roberto Casadei, Mirko Viroli, Giorgio Audrito, Danilo Pianini, and Ferruccio Damiani. 2019. Aggregate Processes in Field Calculus. In *Coordination Models and Languages - 21st IFIP WG 6.1 International Conference, COORDINATION 2019, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17-21, 2019, Proceedings (Lecture Notes in Computer Science)*, Hanne Riis Nielson and Emilio Tuosto (Eds.), Vol. 11533. Springer, 200–217. DOI: http://dx.doi.org/10.1007/978-3-030-22397-7_12

[20] Supasate Choochaisri, Nuttanart Pornprasitsakul, and Chalermek Intanagonwiwat. 2012. Logic Macroprogramming for Wireless Sensor Networks. *IJDSN* 8 (2012). DOI: http://dx.doi.org/10.1155/2012/171738

[21] George B. Dantzig and Mukund N. Thapa. 1997. *Linear Programming 1: Introduction (Springer Series in Operations Research and Financial Engineering) (v. 1)*. Springer.

[22] Rustem Dautov, Salvatore Distefano, Dario Bruneo, Francesco Longo, Giovanni Merlino, Antonio Puliafito, and Rajkumar Buyya. 2018. Metropolitan intelligent surveillance systems for urban areas by harnessing IoT and edge computing paradigms. *Software: Practice and Experience* 48, 8 (may 2018), 1475–1492. DOI: http://dx.doi.org/10.1002/spe.2586

[23] Mirella Santos Pessoa de Melo, José Gomes da Silva Neto, Pedro Jorge Lima da Silva, João Marcelo Xavier Natario Teixeira, and Veronica Teichrieb. 2019. Analysis and Comparison of Robotics 3D Simulators. In *2019 21st Symposium on Virtual and Augmented Reality (SVR)*. IEEE, 242–251.

[24] Rocco De Nicola, Michele Loreti, Rosario Pugliese, and Francesco Tiezzi. 2014. A Formal Approach to Autonomic Systems Programming: The SCEL Language. *TAAS* 9, 2 (2014), 7:1–7:29. DOI: http://dx.doi.org/10.1145/2619998

[25] B. Dieber, C. Micheloni, and B. Rinner. 2011. Resource-Aware Coverage and Task Assignment in Visual Sensor Networks. *IEEE Transactions on Circuits and Systems for Video Technology* 21, 10 (2011), 1424–1437.

[26] L. Esterle. 2017. Centralised, Decentralised, and Self-Organised Coverage Maximisation in Smart Camera Networks. In *2017 IEEE 11th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. 1–10.

[27] Lukas Esterle. 2018. Goal-Aware Team Affiliation in Collectives of Autonomous Robots. In *12th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2018, Trento, Italy, September 3-7, 2018*. IEEE, 90–99. DOI: http://dx.doi.org/10.1109/SASO.2018.00020

[28] Lukas Esterle and John N A Brown. 2020. I Think Therefore You Are: Models for Interaction in Collectives of Self-Aware Cyber-physical Systems. *ACM Transactions on Cyber-Physical Systems* (2020), 1–24. In Press.

[29] Lukas Esterle and Peter R. Lewis. Distributed autonomy and trade-offs in online multiobject k-coverage. *Computational Intelligence* n/a, n/a (????). DOI: http://dx.doi.org/10.1111/coin.12264

[30] Lukas Esterle and Peter R. Lewis. 2017. Online Multi-object k-coverage with Mobile Smart Cameras. In *Proceedings of the 11th International Conference on Distributed Smart Cameras, Stanford, CA, USA, September 5-7, 2017*. 107–112. DOI: http://dx.doi.org/10.1145/3131885.3131909

[31] Lukas Esterle, Peter R. Lewis, Horatio Caine, Xin Yao, and Bernhard Rinner. 2013. CamSim: A Distributed Smart Camera Network Simulator. In *7th IEEE International Conference on Self-Adaptation and Self-Organizing Systems Workshops, SASOW, 2013, Philadelphia, PA, USA, September 9-13, 2013*. 19–20. DOI: http://dx.doi.org/10.1109/SASOW.2013.11

[32] Lukas Esterle, Peter R. Lewis, Richie McBride, and Xin Yao. 2017. The Future of Camera Networks: Staying Smart in a Chaotic World. In *Proceedings of the 11th International Conference on Distributed Smart Cameras, Stanford, CA, USA, September 5-7, 2017*, Miguel O. Arias-Estrada, Christian Micheloni, Hamid K. Aghajan, Octavia I. Camps, and Victor M. Brea (Eds.). ACM, 163–168. DOI: http://dx.doi.org/10.1145/3131885.3131931

[33] Lukas Esterle, Peter R. Lewis, Xin Yao, and Bernhard Rinner. 2014. Socio-economic vision graph generation and handover in distributed smart camera networks. *TOSN* 10, 2 (2014), 20:1–20:24. DOI: http://dx.doi.org/10.1145/2530001

[34] Lukas Esterle and Bernhard Rinner. 2018. An Architecture for Self -Aware IOT Applications. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*. IEEE, 6588–6592. DOI: http://dx.doi.org/10.1109/ICASSP.2018.8462005

[35] Fedpet and Danilo Pianini. 2021. DanySK/Experiment-2019-Smartcam: 1.0.1. (2021). DOI: http://dx.doi.org/10.5281/ZENODO.5506710

[36] Matteo Francia, Danilo Pianini, Jacob Beal, and Mirko Viroli. 2017. Towards a Foundational API for Resilient Distributed Systems Design. In *2nd IEEE International Workshops on Foundations and Applications of Self* Systems, FAS*W@SASO/ICCAC 2017, Tucson, AZ, USA, September 18-22, 2017*. 27–32. DOI: http://dx.doi.org/10.1109/FAS-W.2017.116

[37] Mirgita Frasheri, Lukas Esterle, and Alessandro Vittorio Papadopoulos. 2020. Modeling the Willingness to Interact in Cooperative Multi-robot Systems. In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence, ICAART 2020, Volume 1, Valletta, Malta, February 22-24, 2020*. 62–72. DOI: http://dx.doi.org/10.5220/0008951900620072

[38] G. Fusco and H. Gupta. 2009. Selection and Orientation of Directional Sensors for Coverage Maximization. In *Proc. of the Conf. on Sensor, Mesh and Ad Hoc Communications and Networks*. 1–9. DOI: http://dx.doi.org/10.1109/SAHCN.2009.5168968

[39] José M. Gascueña and Antonio Fernández-Caballero. 2009. Agent-Based Modeling of a Mobile Robot to Detect and Follow Humans. In *Agent and Multi-Agent Systems: Technologies and Applications*, Anne Håkansson, Ngoc Thanh Nguyen, Ronald L. Hartung, Robert J. Howlett, and Lakhmi C. Jain (Eds.). Springer Berlin Heidelberg, 80–89.

[40] Michael A. Gibson and Jehoshua Bruck. 2000. Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels. *The Journal of Physical Chemistry A* 104, 9 (March 2000), 1876–1889. DOI: http://dx.doi.org/10.1021/jp993732q

[41] M. Guarnieri, R. Debenest, T. Inoh, E. Fukushima, and S. Hirose. 2004. Development of Helios VII: an arm-equipped tracked vehicle for search and rescue operations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 39–45.

[42] Takeshi Hatanaka, Riku Funada, and Masayuki Fujita. 2019. Visual Surveillance of Human Activities via Gradient-Based Coverage Control on Matrix Manifolds. *IEEE Transactions on Control Systems Technology* (2019).

[43] Mohamed Hefeeda and Majid Bagheri. 2007. Randomized k-Coverage Algorithms For Dense Sensor Networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 6-12 May 2007, Anchorage, Alaska, USA*. 2376–2380. DOI: http://dx.doi.org/10.1109/INFCOM.2007.284

[44] Bryan Horling and Victor R. Lesser. 2004. A survey of multi-agent organizational paradigms. *Knowledge Eng. Review* 19, 4 (2004), 281–316. DOI: http://dx.doi.org/10.1017/S0269888905000317

[45] S. Hoyer and J. Hamman. 2017. xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software* 5, 1 (2017). DOI: http://dx.doi.org/10.5334/jors.148

[46] Chi-Fu Huang and Yu-Chee Tseng. 2005. The coverage problem in a wireless sensor network. *Mobile Networks and Applications* 10, 4 (2005), 519–528.

[47] S. Huang, R. S. H. Teo, and W. L. Leong. 2017. Review of coverage control of multi unmanned aerial vehicles. In *2017 11th Asian Control Conference (ASCC)*. 228–232.

[48] J. D. Hunter. 2007. Matplotlib: A 2D Graphics Environment. *Computing in Science Engineering* 9, 3 (May 2007), 90–95. DOI: http://dx.doi.org/10.1109/MCSE.2007.55

[49] Jeffrey O. Kephart and David M. Chess. 2003. The Vision of Autonomic Computing. *IEEE Computer* 36, 1 (2003), 41–50. DOI: http://dx.doi.org/10.1109/MC.2003.1160055

[50] Asif Khan, Bernhard Rinner, and Andrea Cavallaro. 2018. Cooperative Robots to Observe Moving Targets: Review. *IEEE Trans. Cybernetics* 48, 1

(2018), 187–198. DOI: http://dx.doi.org/10.1109/TCYB.2016.2628161

[51] David W. King, Lukas Esterle, and Gilbert L. Peterson. 2019. Entropy-Based Team Self-Organization with Signal Suppression. (2019). DOI: http://dx.doi.org/10.1162/isal_a_00154

[52] Stephen G. Kobourov. 2012. Spring Embedders and Force Directed Graph Drawing Algorithms. *CoRR* abs/1201.3011 (2012). http://arxiv.org/abs/1201.3011

[53] J. S. Li and H. C. Kao. 2010. Distributed K-coverage self-location estimation scheme based on Voronoi diagram. *IET Communications* 4, 2 (2010), 167–177.

[54] A. Macwan, J. Vilela, G. Nejat, and B. Benhabib. 2015. A Multirobot Path-Planning Strategy for Autonomous Wilderness Search and Rescue. *IEEE Transactions on Cybernetics* 45, 9 (2015), 1784–1797.

[55] David Mateo, Nikolaj Horsevad, Vahid Hassani, Mohammadreza Chamanbaz, and Roland Bouffanais. 2019. Optimal network topology for responsive collective behavior. *Science advances* 5, 4 (2019), eaau0999.

[56] Yacine Morsly, Nabil Aouf, Mohand Said Djouadi, and Mark Richardson. 2011. Particle swarm optimization inspired probability algorithm for optimal camera network placement. *IEEE Sensors Journal* 12, 5 (2011), 1402–1412.

[57] Prabhu Natarajan, Pradeep K. Atrey, and Mohan S. Kankanhalli. 2015. Multi-Camera Coordination and Control in Surveillance Systems: A Survey. *TOMM* 11, 4 (2015), 57:1–57:30. DOI: http://dx.doi.org/10.1145/2710128

[58] John Page, Robert Armstrong, and Faqihza Mukhlish. 2019. Simulating Search and Rescue Operations Using Swarm Technology to Determine How Many Searchers Are Needed to Locate Missing Persons/Objects in the Shortest Time. In *Intersections in Simulation and Gaming: Disruption and Balance*, Anjum Naweed, Lorelle Bowditch, and Cyle Sprick (Eds.). Springer Singapore, Singapore, 106–112.

[59] Lynne E. Parker and Brad A. Emmons. 1997. Cooperative multi-robot observation of multiple moving targets. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation, Albuquerque, New Mexico, USA, April 20-25, 1997*. 2082–2089. DOI: http://dx.doi.org/10.1109/ROBOT.1997.619270

[60] Veljko Pejovic and Mirco Musolesi. 2015. Anticipatory Mobile Computing: A Survey of the State of the Art and Research Challenges. *ACM Comput. Surv.* 47, 3 (2015), 47:1–47:29. DOI: http://dx.doi.org/10.1145/2693843

[61] Danilo Pianini. 2021. Simulation of Large Scale Computational Ecosystems with Alchemist: A Tutorial. In *Distributed Applications and Interoperable Systems - 21st IFIP WG 6.1 International Conference, DAIS 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June 14-18, 2021, Proceedings (Lecture Notes in Computer Science)*, Miguel Matos and Fabíola Greve (Eds.), Vol. 12718. Springer, 145–161. DOI: http://dx.doi.org/10.1007/978-3-030-78198-9_10

[62] Danilo Pianini, Roberto Casadei, and Mirko Viroli. 2019. Security in Collective Adaptive Systems: A Roadmap. In *IEEE 4th International Workshops on Foundations and Applications of Self* Systems, FAS*W@SASO/ICCAC 2019, Umea, Sweden, June 16-20, 2019*. IEEE, 86–91. DOI: http://dx.doi.org/10.1109/FAS-W.2019.00034

[63] Danilo Pianini, Roberto Casadei, Mirko Viroli, and Antonio Natali. 2021. Partitioned integration and coordination via the self-organising coordination regions pattern. *Future Generation Computer Systems* 114 (2021), 44 – 68. DOI: http://dx.doi.org/https://doi.org/10.1016/j.future.2020.07.032

[64] Danilo Pianini, Giovanni Ciatto, Roberto Casadei, Stefano Mariani, Mirko Viroli, and Andrea Omicini. 2018. Transparent Protection of Aggregate Computations from Byzantine Behaviours via Blockchain. In *Proceedings of the 4th EAI International Conference on Smart Objects and Technologies for Social Good, GOODTECHS 2018, Bologna, Italy, November 28-30, 2018*. 271–276. DOI: http://dx.doi.org/10.1145/3284869.3284870

[65] Danilo Pianini, Sara Montagna, and Mirko Viroli. 2013. Chemical-oriented simulation of computational systems with ALCHEMIST. *J. Simulation* 7, 3 (2013), 202–215. DOI: http://dx.doi.org/10.1057/jos.2012.27

[66] Danilo Pianini, Stefano Sebastio, and Andrea Vandin. 2014. Distributed statistical analysis of complex systems modeled through a chemical metaphor. In *International Conference on High Performance Computing & Simulation, HPCS 2014, Bologna, Italy, 21-25 July, 2014*. 416–423. DOI: http://dx.doi.org/10.1109/HPCSim.2014.6903715

[67] Danilo Pianini, Mirko Viroli, and Jacob Beal. 2015. Protelis: practical aggregate programming. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015*. 1846–1853. DOI: http://dx.doi.org/10.1145/2695664.2695913

[68] C. Piciarelli, L. Esterle, A. Khan, B. Rinner, and G. L. Foresti. 2016. Dynamic Reconfiguration in Camera Networks: A Short Survey. *IEEE Trans. Circuits and Systems for Video Technology* 26, 5 (2016), 965–977.

[69] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Mauro Birattari, Luca Maria Gambardella, and Marco Dorigo. 2012. ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence* 6, 4 (2012), 271–295.

[70] Lenka Pitonakova, Manuel Giuliani, Anthony G. Pipe, and Alan F. T. Winfield. 2018. Feature and Performance Comparison of the V-REP, Gazebo and ARGoS Robot Simulators. In *TAROS (Lecture Notes in Computer Science)*, Vol. 10965. Springer, 357–368.

[71] Kenneth R. Rebman. 1974. Total unimodularity and the transportation problem: a generalization. *Linear Algebra Appl.* 8, 1 (Feb. 1974), 11–24. DOI: http://dx.doi.org/10.1016/0024-3795(74)90003-2

[72] Injong Rhee, Minsu Shin, Seongik Hong, Kyunghan Lee, Seong Joon Kim, and Song Chong. 2011. On the Levy-Walk Nature of Human Mobility. *IEEE/ACM Transactions on Networking* 19, 3 (June 2011), 630–643. DOI: http://dx.doi.org/10.1109/tnet.2011.2120618

[73] George F. Riley and Thomas R. Henderson. 2010. The ns-3 Network Simulator. In *Modeling and Tools for Network Simulation*. Springer Berlin Heidelberg, 15–34. DOI: http://dx.doi.org/10.1007/978-3-642-12331-3_2

[74] B. Rinner, B. Dieber, L. Esterle, P. R. Lewis, and X. Yao. 2012. Resource-aware configuration in smart camera networks. In *2012 IEEE Computer*

Society Conference on Computer Vision and Pattern Recognition Workshops. 58–65.

[75] Cyril Robin and Simon Lacroix. 2016. Multi-robot target detection and tracking: taxonomy and survey. *Auton. Robots* 40, 4 (2016), 729–760. DOI: http://dx.doi.org/10.1007/s10514-015-9491-7

[76] Stefan Rudolph, Sarah Edenhofer, Sven Tomforde, and Jörg Hähner. 2014. Reinforcement Learning for Coverage Optimization Through PTZ Camera Alignment in Highly Dynamic Environments. In *Proc. of the Int. Conf. on Distributed Smart Cameras*. Article 19, 6 pages. DOI: http://dx.doi.org/10.1145/2659021.2659052

[77] L. Ruetten, P. A. Regis, D. Feil-Seifer, and S. Sengupta. 2020. Area-Optimized UAV Swarm Network for Search and Rescue Operations. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. 0613–0618.

[78] Juan C. SanMiguel and Andrea Cavallaro. 2017. Networked Computer Vision: The Importance of a Holistic Simulator. *IEEE Computer* 50, 7 (2017), 35–43. DOI: http://dx.doi.org/10.1109/MC.2017.213

[79] J. C. SanMiguel, C. Micheloni, K. Shoop, G. L. Foresti, and A. Cavallaro. 2014. Self-Reconfigurable Smart Camera Networks. *Computer* 47, 5 (2014), 67–73.

[80] Jürgen Scherer, Saeed Yahyanejad, Samira Hayat, Evsen Yanmaz, Torsten Andre, Asif Khan, Vladimir Vukadinovic, Christian Bettstetter, Hermann Hellwagner, and Bernhard Rinner. 2015. An Autonomous Multi-UAV System for Search and Rescue. In *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*. Association for Computing Machinery, New York, NY, USA, 33–38. DOI: http://dx.doi.org/10.1145/2750675.2750683

[81] Melanie Schranz and Bernhard Rinner. 2014. Demo: VSNsim - A Simulator for Control and Coordination in Visual Sensor Networks. In *Proceedings of the International Conference on Distributed Smart Cameras, ICDSC '14, Venezia Mestre, Italy, November 4-7, 2014*, Andrea Prati and Niki Martinel (Eds.). ACM, 44:1–44:3. DOI: http://dx.doi.org/10.1145/2659021.2669475

[82] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. 2018. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*. Springer, 621–635.

[83] Stanislava Soro and Wendi Heinzelman. 2009. A survey of visual sensor networks. *Advances in Multimedia* 2009 (2009).

[84] Wiktor Starzyk and Faisal Z. Qureshi. 2013. Software Laboratory for Camera Networks Research. *IEEE J. Emerg. Sel. Topics Circuits Syst.* 3, 2 (2013), 284–293. DOI: http://dx.doi.org/10.1109/JETCAS.2013.2256827

[85] Yiannis Stergiopoulos and Anthony Tzes. 2014. Cooperative positioning/orientation control of mobile heterogeneous anisotropic sensor networks for area coverage. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1106–1111.

[86] D. P. Stormont. 2005. Autonomous rescue robot swarms for first responders. In *Proceedings of the 2005 IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety, 2005*. 151–157.

[87] Ryo Sugihara and Rajesh K. Gupta. 2008. Programming models for sensor networks: A survey. *TOSN* 4, 2 (2008), 8:1–8:29. DOI: http://dx.doi.org/10.1145/1340771.1340774

[88] Geoffrey R. Taylor, Andrew J. Chosak, and Paul C. Brewer. 2007. OVVV: Using Virtual Worlds to Design and Evaluate Surveillance Systems. In *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), 18-23 June 2007, Minneapolis, Minnesota, USA*. IEEE Computer Society. DOI: http://dx.doi.org/10.1109/CVPR.2007.383518

[89] C. Natalie van der Wal, Daniel Formolo, Mark A. Robinson, Michael Minkov, and Tibor Bosse. 2017. Simulating Crowd Evacuation with Socio-Cultural, Cognitive, and Emotional Elements. *Trans. Computational Collective Intelligence* 27 (2017), 139–177. DOI: http://dx.doi.org/10.1007/978-3-319-70647-4_11

[90] András Varga and Rudolf Hornig. 2008a. An overview of the OMNeT++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, SimuTools 2008, Marseille, France, March 3-7, 2008*. 60. DOI: http://dx.doi.org/10.4108/ICST.SIMUTOOLS2008.3027

[91] András Varga and Rudolf Hornig. 2008b. An overview of the OMNeT++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, SimuTools 2008, Marseille, France, March 3-7, 2008*, Sándor Molnár, John R. Heath, Olivier Dalle, and Gabriel A. Wainer (Eds.). ICST/ACM, 60. DOI: http://dx.doi.org/10.4108/ICST.SIMUTOOLS2008.3027

[92] Arezoo Vejdanparast, Peter R. Lewis, and Lukas Esterle. 2018. Online Zoom Selection Approaches for Coverage Redundancy in Visual Sensor Networks. In *Proceedings of the 12th International Conference on Distributed Smart Cameras, ICDSC 2018, Eindhoven, The Netherlands, September 3-4, 2018*. 15:1–15:6. DOI: http://dx.doi.org/10.1145/3243394.3243697

[93] Mirko Viroli, Giorgio Audrito, Jacob Beal, Ferruccio Damiani, and Danilo Pianini. 2018a. Engineering Resilient Collective Adaptive Systems by Self-Stabilisation. *ACM Trans. Model. Comput. Simul.* 28, 2 (2018), 16:1–16:28. DOI: http://dx.doi.org/10.1145/3177774

[94] Mirko Viroli, Jacob Beal, Ferruccio Damiani, Giorgio Audrito, Roberto Casadei, and Danilo Pianini. 2018b. From Field-Based Coordination to Aggregate Computing. In *Coordination Models and Languages - 20th IFIP WG 6.1 International Conference, COORDINATION 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018. Proceedings*. 252–279. DOI: http://dx.doi.org/10.1007/978-3-319-92408-3_12

[95] Mirko Viroli, Roberto Casadei, and Danilo Pianini. 2016. Simulating Large-scale Aggregate MASs with Alchemist and Scala. In *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems, FedCSIS 2016, Gdańsk, Poland, September 11-14, 2016*. 1495–1504. DOI: http://dx.doi.org/10.15439/2016F407

[96] Hiroshi Wadaa, Pruet Boonmab, and Junichi Suzukic. 2010. Chronus: A spatiotemporal macroprogramming language for autonomic wireless sensor networks. *Autonomic Network Management Principles: From Concepts to Applications* (2010), 167.

[97] S. Waharte and N. Trigoni. 2010. Supporting Search and Rescue Operations with UAVs. In *2010 International Conference on Emerging Security Technologies*. 142–147.

[98] Matt Welsh and Geoffrey Mainland. 2004. Programming Sensor Networks Using Abstract Regions. In *1st Symposium on Networked Systems Design and Implementation (NSDI 2004), March 29-31, 2004, San Francisco, California, USA, Proceedings*. 29–42. http://www.usenix.org/events/nsdi04/tech/welsh.html

[99] Michael J. Wooldridge. 2009. *An Introduction to MultiAgent Systems, Second Edition*. Wiley.

[100] Yichun Xu, Bangjun Lei, Shuifa Sun, Fangmin Dong, and Chilan Chai. 2010. Three particle swarm algorithms to improve coverage of camera networks with mobile nodes. In *2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA)*. IEEE, 816–820.

[101] Yi-Chun Xu, Bangjun Lei, and Emile A Hendriks. 2011. Camera network coverage improving by particle swarm optimization. *EURASIP Journal on Image and Video Processing* 2011, 1 (2011), 458283.

[102] Yi-Chun Xu, Bangjun Lei, and Emile A Hendriks. 2013. Constrained particle swarm algorithms for optimizing coverage of large-scale camera networks with mobile nodes. *Soft computing* 17, 6 (2013), 1047–1057.

[103] Evşen Yanmaz, Saeed Yahyanejad, Bernhard Rinner, Hermann Hellwagner, and Christian Bettstetter. 2018. Drone networks: Communications, coordination, and sensing. *Ad Hoc Networks* 68 (2018), 1–15.

[104] V. Zaburdaev, S. Denisov, and J. Klafter. 2015. Lévy walks. *Reviews of Modern Physics* 87, 2 (Jun 2015), 483–530. DOI: http://dx.doi.org/10.1103/revmodphys.87.483

[105] Hunza Zainab, Giorgio Audrito, Soura Dasgupta, and Jacob Beal. 2020. Improving Collection Dynamics by Monotonic Filtering. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2020, Companion Volume, Washington, DC, USA, August 17-21, 2020*. IEEE, 127–132. DOI: http://dx.doi.org/10.1109/ACSOS-C51401.2020.00043

[106] Zhongliang Zhao, Denis Rosario, Torsten Braun, and Eduardo Cerqueira. 2015. A Tutorial of the Mobile Multimedia Wireless Sensor Network OMNeT++ Framework. (2015).

Received January 2021