Successful Swarms: Operator Situational Awareness with Modelling and Verification at Runtime

Yue Gu[1], William Hunt[2], Blair Archibald[1], Mengwei Xu[1], Michele Sevegnani[1] and Mohammad D. Soorati[2]

***Abstract*— Robot swarms, through redundancy, offer fault-tolerant distributed sensing and actuation, but can lack complex mission-level decision making. Pairing a human operator with the swarm can improve decision making but only if the operator maintains situational awareness—knowledge of the current state of the swarm—as well as being able to anticipate future states. We show how formal methods, in the form of probabilistic models, executed and verified at runtime alongside the system can aid situational awareness by providing valuable insight into both current and future situations. Two models, for determining task and mission success probabilities, are given, and we show that statistical model checking allows timely approximate predictions that take no more than $1s$ while staying within 2% of the exact solution. We highlight and implement approaches to display this information to an operator, and show how models can be used to try what-if scenarios before decisions are made.**

***Index Terms*— Human-swarm interaction, probabilistic modelling, runtime verification, digital twin.**

I. INTRODUCTION

Resilience is an important feature of robot swarms as no single point of failure exists in the system and missions can still succeed, although with different non-functional properties, when robots are removed from the system. Despite this advantage, the increased complexity from multiple devices means that when deploying robot swarms in critical missions it remains difficult to reason about and guarantee properties of the overall system.

An essential property for a swarm system is *feasibility*, such as the feasibility of the mission, that is, how *likely* a mission is to succeed based on the current situation. Given the complexity and stochasticity involved in swarms, operators are not always capable of reaching a conclusion that a swarm may or may not reach a certain goal by merely observing the status of individual agents, as shown in [1]. In earlier work [2], we interviewed experienced Uncrewed Aerial Vehicles (UAV) operators and asked for the key requirements of a successful swarm operation. In their answers, the terms 'System monitoring' and 'UAV status' are frequently used to refer to the information that the operator has about the status of the mission and the swarm. Motivated by this demand in practice, in this study, we take a step further in implementing this requirement and use formal methods to inform the operator about the feasibility of a given task. We apply formal models in the form of Continuous-Time Markov Chains (CTMCs) and Continuous Stochastic Logic (CSL) [3] to *reason*[1] about swarm systems.

We focus on the specific scenario where a swarm of UAVs is applied to the task of Multi-Agent Pickup and Delivery (MAPD) [4], [5]. As our focus is on how runtime models can be used in swarm environments, not to gain insight into a specific physical environment, we evaluate our approach using the *H*uman *A*nd *R*obot *I*nteractive *S*warm (HARIS) Simulation platform [6] as a proxy for a real setup. An important aspect is that the simulator (or real environment) provides runtime data that we use to parameterise the formal models. Additionally the simulator can be seen to act as a *control interface* like those you might see in human-in-the-loop deployments [7]. It was carefully designed to reduce operator workload, and we extend this to show how the improved situational awareness enabled by the models can be visualised in practice.

We examine two types of feasibility: *task feasibility* that determines the chance a specific parcel delivery task is successful, and *mission feasibility* that determines the chance all delivery tasks are eventually completed. *Feasibility* is represented as a probabilistic property, *e.g.* the probability that the task/mission is accomplished. The property analysis is conducted through probabilistic model checking using PRISM [8].

Formal methods are most commonly deployed at design time, *e.g.* to provide certification of systems. Instead we use formal methods at runtime [9] to improve human decision making by increasing situational awareness during human-swarm interaction. To achieve this, we propose a framework that integrates formal methods/verification, the swarm system, and the human interactions together. Our framework is shown in Fig. 1, and consists of three components: dynamically created PRISM models to reason over the swarm system at *runtime* and provide quantitative predictions via model checking. The model checking results are presented through an interface (part of the simulator). The operator can utilise this new information to perform decision making, *e.g.* determining if new UAVs should be added. These interactions, in turn, update the PRISM models. By closing

This work is supported by the UK Engineering and Physical Sciences Research Council, under PETRAS SRF grants MAGIC and FARM (EP/S035362/1) and "Explainable Human-Swarm Systems" as part of the Trustworthy Autonomous Systems Hub (EP/V00784X/1), as well as an Amazon Research Award on Automated Reasoning.

[1]School of Computing Science, University of Glasgow, Glasgow, G12 8QQ, UK; {yue.gu, blair.archibald, mengwei.xu, michele.sevegnani}@glasgow.ac.uk

[2]School of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, UK; {w.hunt, m.soorati}@soton.ac.uk

[1]In the context of model checking *reason* means proving that a system meets some logical specifications.

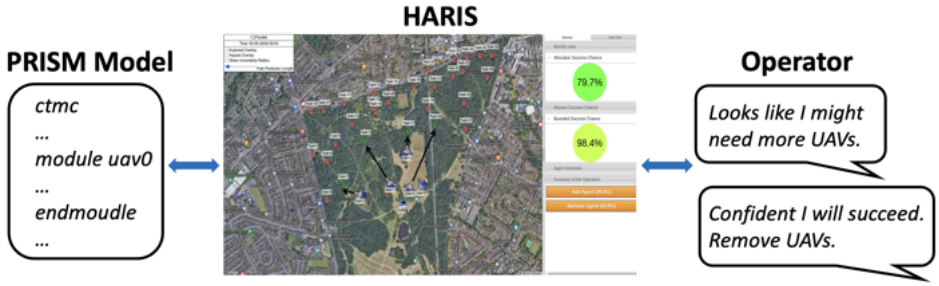


Fig. 1. Integration of formal (PRISM) model, HARIS simulation and control environment, and operator decision making. The right hand panel of HARIS interface is extended with the feasibility information to assist operator in decision making.

the loop the framework forms a digital-twin [10].

We make the following research contributions.

- We develop a probabilistic formal model of swarm systems for a delivery scenario. This model includes movement, battery effects, and failure rates.
- We extend the HARIS simulation environment [6] to communicate with the formal model dynamically at runtime to receive up-to-date estimates of mission success.
- We show runtime models give results in a timely manner, and describe how these results might be presented to an operator.

II. BACKGROUND

A. Human-Swarm Interaction

Humans can have multiple roles when interacting with swarms, *e.g.* supervisor, operator, mechanic, teammate, bystander, and analyser [11]. Here we focus on the *operator* role. An operator has to be aware of the state of the swarm, the dynamics of the environment, and the progress of the mission. Studies show strong awareness has a significant impact on improving the operator performance in terms of decision making [12], while poor awareness may cause problems in detecting and intervening during abnormal behaviours [13].

Situational awareness refers to the human operator's understanding of a system on three levels: *perception*, *comprehension* and *projection* [14]. For human-swarm interaction, these three levels are adapted into Situation awareness-based Agent Transparency (SAT) levels: (1) the swarm's immediate objectives and intentions; (2) the swarm's reasoning process; and (3) predictions of future expectations [15].

We focus on SAT level 3 and aim to provide advanced indicators to the operator that predict the future outcome for the entire swarm. This is similar to the approach of [3] where formal models were used to detect and mitigate *cognitive dissonance*: misalignment between the system status and how it is perceived by the operator[1].

Other factors, such as *workload* and *trust* also affect operator's performance [17]. Psychophysiological studies show correlations between the level of workload, and both swarm size [18] and situational uncertainty [19], suggesting this to be of substantial concern in real-life swarm domains with large numbers of agents in an uncertain environment. Human trust in swarm applications is believed to be equally important for understanding the human reliance on swarm. Multiple factors, including swarm transparency [20], influence the trust of human operators in the performance of the swarm [2].

[1]We use *cognitive dissonance* to describe the misalignment of the operator and system views. The term is also used in the psychology filed, where it generally means that a person holds two conflicting beliefs [16].

B. Swarm Verification and Modelling at Runtime

Existing approaches to verify swarm systems with formal methods use techniques like model checking to give guarantees on: swarm performance and robustness [21], security analysis [22], and verification of emergent behaviour [23]. Unlike these approaches, that give guarantees *before* deployment, we use modelling and verification at runtime to respond to a system as it exists.

Runtime monitoring [24], uses formal methods to dynamically analyse the execution traces that a system generates at runtime against formal specifications (*i.e.* properties that are mathematically expressed with well-defined syntax and semantics, often a variant of LTL [25]). Falcone *et al.* compare and classify 60 runtime monitoring tools accross multiple dimensions, including their specifications and deployment [26]. The main activities that current tools do when conducting runtime monitoring are: synthesising monitors (*i.e.* automatas) from specifications, receiving observations (of the current state [27]) from the monitored system, and performing trace equivalence against the specification. The monitors are static once created and observe finite system executions at runtime. These tools are broadly applied to verify agent-level specifications that are monitorable, *e.g.* safety and co-safety [28], but are challenging to analyse mission-level specifications, such as the success of human-swarm systems, where finite observations are usually not sufficient. Additionally, such systems can evolve based on human interactions, for example, the changing of the swarm size and topology. The monitor, thus, may not be capable to reflect the dynamics along with the system evolution.

Models@ run.time [29], instead, captures the changes of the system under investigation. A runtime model is a reflection layer that is causally connected with the associated system and serves as a self-representation to highlight the system structure, behaviour or objectives [30]. Runtime models have been successfully deployed in a range of

application domains, including software development [31], ongoing design [32], and system reasoning for unforeseen circumstances during execution [33].

We specifically focus on formal runtime models, where formal models are constructed and evolve at runtime taking account of swarm dynamics and human interactions. To our knowledge, this is the first instance integrating these two techniques for human-swarm interaction. Importantly, by modelling, we do not just monitor the system, but can predict what it might do to improve the situational awareness.

C. Probabilistic Model Checking

We use probabilistic model checking to analyse our swarm scenario and provide additional situational awareness not possible from a simulation directly. As swarms operate in continuous environments, our models are (labelled) Continuous-Time Markov Chains (CTMC). A CTMC consists of a set of states, with designated initial state(s), and a function that assigns a transition *rate* λ between any two states. A transition can occur between states s and s' only if $\lambda(s, s') > 0$, in which case the probability of this transition being triggered within t time units is $1-e^{-\lambda(s,s')\cdot t}$. Intuitively, this means transitions with a higher rate occur more often. We label states with atomic predicates to make it easier to write logical formulae, *e.g.* $s \mapsto \{\text{failure}\}$.

Rather than directly build a CTMC, we use the PRISM modelling language [34], based on Reactive Modules [35], that supports high level specification of processes. Processes are modules consisting of non-deterministic choice over action-labelled guarded commands (that denote transitions).

A core component of PRISM is a model checker that allows *all* possible behaviours of a system to be quantified [36], *e.g.* asking the probability a system will fail. Properties of interest are expressed in an extension of the Continuous Stochastic Logic (CSL) [3], which is a temporal logic with probabilistic operators. In this work we only require the *eventually* $\mathbf{F}\,\varphi$ temporal operator, that asserts that, for all paths, we eventually reach a state labelled where φ is true. Quantitative properties are specified with an operator $\mathcal{P}_{=?}[\,\Psi\,]$ that determines, the likelihood a path exists where Ψ is true. Bounded variants are possible, *e.g.* property $\mathcal{P}_{=?}[\,\mathbf{F}^{\leq t}\,\Psi\,]$ requires Ψ to be true within t time units (domain dependent, *e.g.* hours, minutes or seconds). In addition, PRISM also allows *rewards* (*costs*) to be assigned to states. The $\mathcal{R}$ operator can quantify reward-based properties, *e.g.* $\mathcal{R}_{=?}[\,\mathbf{C}^{\leq t}\,]$, where $\mathbf{C}^{\leq t}$ calculates the cumulative reward within t time units.

For models with a large number of states, exhaustive model checking can be expensive. To overcome this, the built-in discrete-event simulator in PRISM allows us to apply *statistical model checking* (SMC) [37] to give approximately correct results. Statistical model checking effectively samples the model space through repeated simulation instead of exhaustive search [38], [39] resulting in higher performance at a cost of accuracy.

III. SWARM MODELS

To show how probabilistic model checking can be used to provide situation awareness in the form of feasibility information, *i.e.* probabilities of success (a domain-specific notion) to an operator, we use a set of UAV mission scenarios. Each scenario consists of a set of delivery *tasks* each with a given location. A UAV is assigned one task at a time following a certain allocation sequence. A task is completed when a UAV successfully moves from the hub to the delivery location. We count a task as completed when the UAV completes the delivery, without requiring the UAV to return to the hub.

We implement two feasibility models: a **task feasibility model** determines the probability a single UAV accomplishes a specific delivery task, and a **mission feasibility** model that determines the probability of success for a set of delivery tasks and a group of UAVs. Importantly, task feasibility considers a single allocation while mission feasibility must consider all future allocations. Full models are available in PRISM format online[1].

Models cannot capture all aspects of the system, and, as is typical in modelling, there are trade-offs between performance—getting useful results quickly—and accuracy—capturing more details of the system—, and, as we will see, it is often necessary to abstract specific details of the scenarios *e.g.* by discretising continuous values.

a) Task Feasibility Model: For the task feasibility model, we want to determine, for a UAV with given task allocation, the likelihood the UAV makes it from the hub to the task region without failure, *e.g.* battery critical, or background failure (collisions etc.). The main entities of the model are *UAVs* and spatial regions. Each UAV has a *battery* level that we discretise to four values: *high*, *mid*, *low* or *critical*, similar to the technique used in [40]. When UAVs are not at the hub, their batteries drain at a fixed rate based on the current battery level, *e.g.* batteries near critical levels might discharge faster than those fully charged. A UAV with *critical* battery cannot complete a task (task failure).

Next we model movement for the UAVs. Movement in HARIS is in a 2D environment. When a UAV is assigned a task, it rotates to adjust its heading (*i.e.* the yaw angle), moves towards the task location and then follows a similar route on its way back to the hub. Specific movement rates in HARIS are adjusted by added noise to better simulate real environments, *e.g.* wind variance.

As CTMCs are discrete state models, they are not well suited to modelling the continuous space found in the HARIS simulator and real-world scenarios. Therefore, to capture movement in our model, we discretise the two-dimensional Euclidean space by splitting it into a set of *spatial regions* (*i.e.* we define a topological space). UAVs are always in a specific *spatial region* r. We define regions based on a circular radius propagating from the hub as shown in Fig. 2 allowing each region to be identified using an integer with 0

[1]Full models are available at `https://doi.org/10.5281/zenodo.7649302`

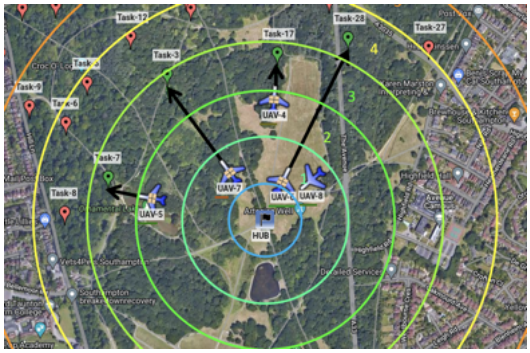


Fig. 2. Modelling continuous space as discrete radii.

TABLE I

MISSION FEASIBILITY SCALING FOR A FIXED CONFIGURATION OF 5 TASKS IN REGION 1.

UAVs	1	2	3	4	5
States	1.1×10^3	7.9×10^4	4.8×10^6	2.7×10^8	1.2×10^{10}
Transitions	2.5×10^3	2.8×10^5	2.4×10^7	1.8×10^9	9.7×10^{10}

TABLE II

MISSION FEASIBILITY MODEL SCALING FOR A SINGLE UAV AS WE INCREASE THE NUMBER OF TASKS.

Tasks (Region 1)	5	6	7	8	9
States	1070	1190	1310	1430	1550
Transitions	2458	2707	2956	3205	3454

being the hub, and increasing as distance increases. When a UAV is assigned a task in a specific region r (or 0 if no task is available), the goal of the UAV is to move through all regions $[0 \dots r]$ to deliver a package. As we are in a topological space we do not have a notion of rotation/heading like in HARIS. Instead we introduce an extra movement rate between the regions when a rotation is needed to account for the extra time a UAV needs to turn. For example, when a UAV moves $r_1 \rightarrow r_0$, a rotation is introduced as an extra state at r_1 if needed, in which case the UAV takes more time than moving $r_1 \rightarrow r_0$ directly.

To each region, except the hub, we associate a background failure rate that captures unknown failures associated with that region, *e.g.* hazards, failed collision avoidance; and a rate of movement, *e.g.* how quickly it moves between adjacent regions. All UAVs operate independently so the failure/movement rates are not influenced by the presence of multiple UAVs in the same region[1].

An example task feasibility scenario is in Fig. 2, where UAV-5 has been assigned Task-7 in region 3. UAV-5 leaves the hub (region 0) and moves through the two regions all while being affected by battery drain and background failures. If UAV-5 reaches region 3 without failure, the delivery task is accomplished and UAV-5 performs the same movement in reverse to return to the hub. The model scales sub-linearly for *single* UAV assigned with a task in different regions, starting at 71 states (134 transitions) for a task in region 1 with a maximum of 113 states (212 transitions) for a task in region 7. As expected, more states are required for further tasks but overall the number of states remains low.

b) Mission Feasibility Model: Unlike the task feasibility, which reasons if a single UAV can complete an assigned task, mission feasibility accounts for all UAVs and all tasks. All uncompleted tasks form a configuration $c = [n_1, n_2, \dots, n_7]$, where n_r represents the number of uncompleted tasks in region r. Alongside UAVs, the mission feasibility model has another important entity: the *allocator*, that decides which task is assigned to a UAV. Currently, allocation starts from the closest region and radiates outwards to further regions. Other allocation strategies are possible, such as max-sum task allocation [41]. A partial allocation is in Fig. 2 where UAV-5, UAV-7, UAV-4 and UAV-6 have been assigned Task-7, Task-3, Task-17 and Task-28 respectively. They leave the hub and move through regions, being affected by battery drain and background failures. UAVs return to the hub once they have completed their tasks, *e.g.* UAV-8, where they are re-assigned a new task by the allocator. Once there are no more tasks to complete the mission is accomplished.

To support missions with a high number of tasks and a low number of UAVs, we provide an option for UAVs with *critical* battery to be recharged in the hub (at a fixed recharge rate) allowing them to rejoin the mission once their battery is *high*. When *recharging* is enabled, UAVs return to the hub when their battery becomes *critical* regardless which region they are currently in. That is, we assume critical means just enough battery to return home (regardless of current location), rather than catastrophic failure. Any currently assigned tasks are reallocated rather than waiting for that specific UAV to recover.

Table I shows how the model scales as we increase the number of UAVs for a fixed task configuration $c = [5, 0, \dots, 0]$, on the other hand, Table II shows how the model scales to as we vary the task configuration by increasing the number of tasks in region 1 for a single UAV[2]. As can be seen, the mission feasibility model is sensitive to the number of UAVs and suffers a state explosion due to the combinatorial nature of interleavings when more UAVs are involved, and this causes a challenge for real-time implementations. The model is much less sensitive to the number of tasks, and these results suggest the model is best utilised for missions with low numbers of UAVs and higher numbers of tasks. Given the cost of UAVs we expect this to be the more common scenario in practice.

[1]UAVs do not cooperate to improve performance, *e.g.* through slipstreaming, or degrade performance, *e.g.* through increase collision chances. We do not directly model collisions between UAVs, but a collision chance can be included in the region failure rates if required.

[2]Model construction is done automatically by PRISM when performing model checking. PRISM computes the set of reachable states from the initial state and builds the corresponding transition matrix.

A. Parameters

The reality gap is a well-known problem in robotics: behaviours that perform well in models/simulations do not always match the real-world implementations [42]. This is due to the fact that simulators are based on simplified assumptions and are not able to fully capture all features of the real world. To stay as close to reality as possible, we derive the model parameters directly from the HARIS platform which has been studied and shown to be able to cross the reality gap [43]. In CTMCs, parameters, or rates, affect the *probability* a transition is taken rather than forcing a transition. This means that UAVs behave stochastically, and is enough to capture uncertainties applied in HARIS, *e.g.* movement differences due to wind variance.

Parameters for the UAVs include the battery drain rates λ^{drain}, the recharging rate $\lambda^{recharge}$ and the rotation (movement speed adjustment) rate λ^{rotate}. The swarm is homogeneous, meaning all UAVs are identical and fly at the same speed and with the same battery consumption, although it is possible to vary this, *e.g.* to model faster UAVs.

Parameters for the regions include the background failure rates λ_r^{fail} and the movement rates λ_r^{move} for each region r. Currently all regions are identical so they have the same failure rate and movement rate. Different rates could be used to model regions with specific hazards. We assume the movement rates are symmetrical regardless of what direction the UAV is moving, although it is possible to vary this, *e.g.* to model flying into a headwind.

B. Properties

In PRISM we specify properties using CSL (see Section II-C). Within the formulae we are allowed to directly access model variables, *e.g.* states of the UAVs. We introduce these informally as we give the properties.

We consider two types of quantitative properties. *Time-unbounded* properties describe if a formula is true on an infinite horizon, while *time-bounded* properties check if a formula is true within a limited (bounded) period of time. There are two main properties of interest: 1. **expected task feasibility**: the probability that a UAV, with a given battery, can accomplish a task in region r; 2. **expected mission feasibility** the probability that a group of UAVs, each with an initial battery level, can accomplish a given set of tasks.

The task feasibility property is expressed as:

$$\mathcal{P}_{=?}\,[\,\mathbf{F}\; delivered_i\,] \tag{1}$$

Task feasibility for a UAV $i \in N$ (where N is the total number of UAVs in the swarm) with an allocated task is given as the probability ($\mathcal{P}$) that, from the initial state, *eventually* ($\mathbf{F}$) it reaches the task location (*e.g.* $delivered_i$ is true). Unlike simulation, these results are *exact* based on analysis of the underlying CTMC of the *task feasibility* model.

The mission feasibility property determines if all tasks are completed and is expressed as:

$$\mathcal{P}_{=?}\,[\,\mathbf{F}\; c = [0,\ldots,0]\,] \tag{2}$$

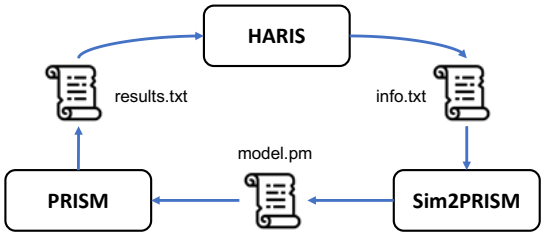


Fig. 3. Closed loop runtime verification: system information moves between HARIS and PRISM through a program, *Sim2PRISM*, that dynamically constructs a corresponding model for PRISM model checker. Model checking results are sent back and presented in the simulator allowing operators to respond if required.

where c is the configuration of all uncompleted tasks and 0 means there is no task pending in the corresponding region. For all regions $r \in [1,7]$, this requests the probability ($\mathcal{P}$) that, given a swarm of UAVs, *eventually* ($\mathbf{F}$) we reach a state where there are no more tasks to complete ($n_r = 0$).

This property can naturally be expressed in a bounded time as:

$$\mathcal{P}_{=?}\,[\,\mathbf{F}^{\leq t}\; c = [0,\ldots,0]\,] \tag{3}$$

In this case we check *all* tasks complete within time t.

C. Integration with HARIS Environment

We integrate our PRISM models with the HARIS simulator in two separate ways.

For *task feasibility*, we precompute a lookup table of success probabilities based on a single UAV assigned a task in different regions ($r \in [1,7]$) and three initial battery levels (*high*, *mid* and *low*). The UAVs are generally quite successful at completing their tasks so long as they have enough battery. As the task feasibility model is of relatively small size, these values are generated through a numerical (not statistical) model checking approach, *i.e.* the answers are an *exact* probability.

To calculate *mission* feasibility, we integrate our model with the simulator to provide verification at runtime as illustrated in Fig. 3. For runtime verification, HARIS can, at any point, provide all known information, including the configuration of tasks (how many tasks to complete in each region), the number of UAVs and their status (*e.g.* locations, battery levels and allocations), allowing a specific model *instantiation* to be constructed for analysis. To implement the process, we use a Python program (*Sim2PRISM*) as a middleware to read the information from HARIS, construct a model accordingly and run the analysis in parallel. Due to the state explosion of the mission feasibility model (Table I), we use statistical model checking (SMC), instead of an exact model checking approach, to get approximate results with some significance (*e.g.* $\alpha = 0.01$).

IV. IMPLEMENTATION AND ANALYSIS

For the models to be useful in practice we must show 1. the time to compute results is low enough relative to the system under test, and 2. methods to effectively communicate the results to operators.

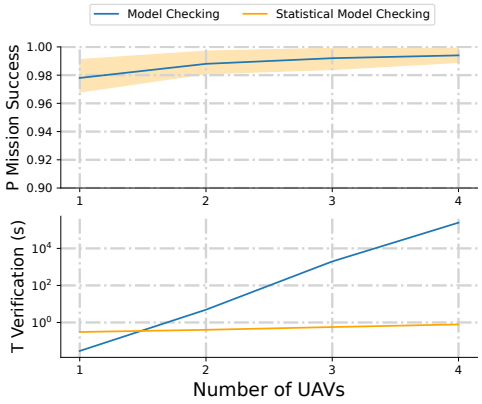


Fig. 4. Comparing statistical model checking (SMC) and exact model checking: solution (top) and time (bottom). Fixed task configuration of 5 tasks in region 1 and up to 4 UAVs. Range of SMC results is max–min approximate solution over 100 trials.

We use a modified version of the HARIS multi-agent simulation engine that allows calling our verification backend and displaying the results. HARIS is a Java-based web app that allows creation and deployment of swarm robotics scenarios while providing simulation of robot dynamics and operator control. In realistic setups we expect the system dynamics, *e.g.* sensor data, and operator control, *e.g.* control messages, to be decoupled.

A. Verification Performance

The models provide two pieces of information to the operator: success percentage that the *current* allocation (of UAVs to tasks) is completed, and success percentage that the *mission* will be completed (forecasting all future allocations). To be useful, this data must be available to an operator in a timely manner.

As we assume all UAVs are independent, we can compute the current allocation success for a swarm as the *product* of the success probabilities of individual UAVs[1].

For mission success percentage, we use SMC to perform the calculation at runtime. For a fixed task configuration $c = [5, 0, \ldots, 0]$, Fig. 4 compares SMC and explicit model checking in terms of both approximation accuracy and the verification time. We tested both methods with up to 4 UAVs in our mission feasibility model. As expected, increasing the number of UAVs dramatically increases the time for exact model checking (*e.g.* almost 3 days to fully verify the model with 4 UAVs) making it impractical for use at runtime[2]. However, SMC is able to provide results within 1 second for the 4-UAV model, and can be applied to an even larger models, making it practical for runtime verification. We performed 100 trials of SMC for the each number of UAVs and the approximate solution was, in the worst case, no more than 2% from the exact solution.

[1]The model is fast enough to also re-compute these each time, but for such a small number of values caching is more practical.

[2]Verification of a 5-UAV model ran out of memory.

B. Displaying Results and Decision Making

We expand the existing interface of HARIS to dynamically display the computed predictions to the operator. Fig. 1 shows our interface in a typical scenario. The operator can observe the agents delivering packages to the task locations in the central panel on a satellite view. To the right is a panel showing the allocation and mission success chances. These are colour-coded based on a gradient from green for high probability of success, to red for a lower probability of success.

In future we are planning to determine how operators respond to both the predictions, *e.g.* does it cause them to change the number of UAVs as supported by the interface, and the interface elements, *e.g.* do we need percentages at all or is colour adequate?

C. What-if Scenarios

A key benefit of having a formal model, that you do not get with runtime monitoring, is the ability to perform analysis of what-if scenarios to give the operator extra information before making a decision. For example, an operator might query the effect of adding/removing a UAV before taking the decision, or check the effect of turning off recharging etc. This could be displayed to an operator graphically, for example, Fig. 5(a) shows how the success probability changes with number of UAVs and recharging on/off within a bounded simulation time. Likewise, an operator might consider using faster UAVs to improve the success likelihood but be concerned of increasing energy cost. Fig. 5(b) shows how the probability and the battery cost (a model *reward*) change as more fast UAVs are deployed.

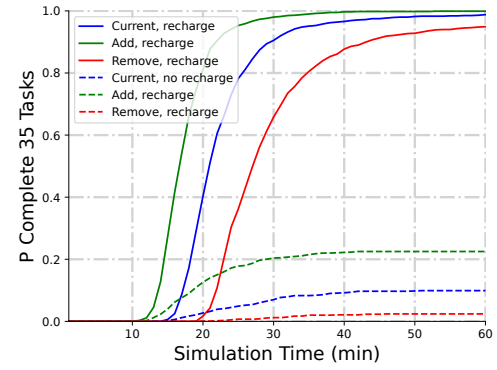
Large sets of parameterised properties can be abstracted by their upper and lower probability bounds, *i.e.* their *envelopes of behaviour* [44], to improve reasoning scalability and ease the operator's cognitive load.

V. DISCUSSION AND CONCLUSION

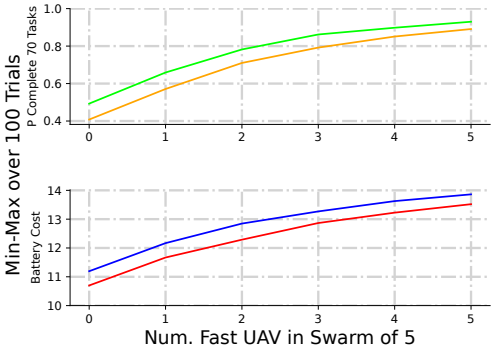
Formal methods can help reason about the behaviour of swarm systems, providing mission-level analysis, *e.g.* probability of successfully completing a task, that is not captured by raw data alone, *e.g.* individual robot speeds/headings. This allows an operator to understand system complexity and stochasticity, and anticipate future events to perform effective decision making.

We have shown how to combine formal modelling and verification, (simulated) swarm system data, and human control. This approach is fully dynamic and, unlike the common static use of formal verification, the reasoning is performed at runtime with new models being instantiated according to the current swarm status. A benefit of this approach is that any non-determinism, *e.g.* the exact task allocation, is already resolved by the system so we do not need to support it. By reflecting system data in the model the framework acts as a digital-twin.

We validated the approach using on a Multi-Agent Pickup and Delivery task and, as a proxy for a real system, utilised



(a)



(b)

Fig. 5. What-if scenarios of the mission feasibility. (a) Probabilities of success against time. The probability bounds give information for operators to decide to add/remove UAVs or turn on/off recharging. Current scenario is 5 UAVs. (b) Probabilities of success and the battery cost of deploying faster UAVs. The battery cost is total number of recharges needed.

the HARIS swarm simulation platform to evaluate our framework. Two models are shown, for task and mission feasibility, both make use of probabilistic model checking using PRISM. Feasibility information is presented by extending the HARIS interface to increase the situation awareness and aid the operator in their decision making. Our models also support *rewards* (*i.e.* costs) associated with states or transitions allowing, *e.g.* reasoning about energy consumption.

Future work will validate our approach with real UAVs and support more what-if scenarios that are not possible when only monitoring the system. For example:

- How does environment change feasibility, *e.g.* different non-radial topologies, or how the probability changes if there is a hazard, *e.g.* a fire, in a specific region.
- Effect of different task allocation methods *e.g.* is success probability increased for furthest-first allocation?

The HARIS interface will need to be extended to support operator interactions in these scenarios. For example, buttons can be dynamically displayed to show options for the operator at different stages. When reasoning about real systems, the reality gap becomes even more crucial. To maintain model accuracy, parameters could be updated dynamically along with the real system [45].

Our approach is not restricted to this scenario or the formal methods presented, and instead serves as a template for verification of swarm systems at runtime, and complex systems more generally. Although we currently assume central access to data, *e.g.* a digital twin, incomplete data could be replaced by suitable probabilities. A key piece of future work involves evaluating how operators interact with the information provided by models: do they find it useful? and applying the framework to more scenarios including physical ones.

References

[1] A. Kolling, K. Sycara, S. Nunnally, and M. Lewis, "Human swarm interaction: An experimental study of two types of interaction with foraging swarms," *Journal of Human-Robot Interaction*, vol. 2, no. 2, 2013.

[2] K. J. Parnell, J. Fischer, J. Clarck, A. Bodenman, M. J. Galvez Trigo, M. P. Brito, M. Divband Soorati, K. Plant, S. Ramchurn, *et al.*, "Trustworthy UAV relationships: Applying the Schema Action World taxonomy to UAVs and UAV swarm operations," *International Journal of Human–Computer Interaction*, 2022.

[3] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, "Model-checking algorithms for continuous-time Markov chains," *IEEE Transactions on software engineering*, vol. 29, no. 6, pp. 524–541, 2003.

[4] M. Liu, H. Ma, J. Li, and S. Koenig, "Task and path planning for multi-agent pickup and delivery," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2019.

[5] O. Salzman and R. Stern, "Research challenges and opportunities in multi-agent path finding and multi-agent pickup and delivery problems," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 2020, pp. 1711–1715.

[6] W. Hunt, J. Ryan, A. O. Abioye, S. D. Ramchurn, and M. D. Soorati, "Demonstrating performance benefits of human-swarm teaming," *arXiv preprint arXiv:2303.12390*, 2023.

[7] L. C. W. Kong, S. Harper, D. Mitchell, J. Blanche, T. Lim, and D. Flynn, "Interactive digital twins framework for asset management through internet," in *2020 IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT)*. IEEE, 2020, pp. 1–7.

[8] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: Probabilistic symbolic model checker," in *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. Springer, 2002, pp. 200–204.

[9] R. Calinescu and S. Kikuchi, "Formal methods @ runtime," in *Monterey Workshop*. Springer, 2010, pp. 122–135.

[10] A. El Saddik, "Digital twins: The convergence of multimedia technologies," *IEEE multimedia*, vol. 25, no. 2, pp. 87–92, 2018.

[11] A. Kolling, P. Walker, N. Chakraborty, K. Sycara, and M. Lewis, "Human interaction with robot swarms: A survey," *IEEE Transactions on Human-Machine Systems*, vol. 46, no. 1, pp. 9–26, 2015.

[12] M. R. Endsley, "Situation awareness global assessment technique (SAGAT)," in *Proceedings of the IEEE 1988 national aerospace and electronics conference*. IEEE, 1988, pp. 789–795.

[13] J. M. Riley, L. D. Strater, S. L. Chappell, E. S. Connors, and M. R. Endsley, "Situation awareness in human-robot interaction: Challenges and user interface requirements," *Human-Robot Interactions in Future Military Operations*, pp. 171–192, 2010.

[14] M. Endsley and W. Jones, "Situation awareness, information warfare and information dominance," *Endsley Consulting, Belmont, MA, Tech. Rep*, pp. 97–01, 1997.

[15] J. Y. Chen, K. Procci, M. Boyce, J. Wright, A. Garcia, and M. Barnes, "Situation awareness-based agent transparency," *ARMY RESEARCH LAB ABERDEEN PROVING GROUND MDUNIVERSITY OF CENTRAL FLORIDA*, 2014.

[16] L. Festinger, *A theory of cognitive dissonance*. Stanford university press, 1962, vol. 2.

[17] A. Hussein and H. Abbass, "Mixed initiative systems for human-swarm interaction: Opportunities and challenges," in *2018 2nd Annual Systems Modelling Conference (SMC)*. IEEE, 2018, pp. 1–8.

[18] G. Podevijn, R. O'grady, N. Mathews, A. Gilles, C. Fantini-Hauwel, and M. Dorigo, "Investigating the effect of increasing robot group sizes on the human psychophysiological state in the context of human–swarm interaction," *Swarm Intelligence*, vol. 10, no. 3, pp. 193–210, 2016.

[19] D. St-Onge, M. Kaufmann, J. Panerati, B. Ramtoula, Y. Cao, E. B. Coffey, and G. Beltrame, "Planetary exploration with robot teams: Implementing higher autonomy with swarm intelligence," *IEEE Robotics & Automation Magazine*, vol. 27, no. 2, pp. 159–168, 2019.

[20] J. A. Adams, J. Y. Chen, and M. A. Goodrich, "Swarm transparency," in *Companion of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, 2018, pp. 45–46.

[21] A. Lomuscio and E. Pirovano, "Verifying fault-tolerance in probabilistic swarm systems," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 2021, pp. 325–331.

[22] I. Boureanu, P. Kouvaros, and A. Lomuscio, "Verifying security properties in unbounded multiagent systems," in *Proceedings of the 2016 international conference on autonomous agents & multiagent systems*, 2016, pp. 1209–1217.

[23] P. Kouvaros and A. Lomuscio, "Verifying emergent properties of swarms," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[24] E. Bartocci, Y. Falcone, A. Francalanza, and G. Reger, "Introduction to runtime verification," *Lectures on Runtime Verification: Introductory and Advanced Topics*, pp. 1–33, 2018.

[25] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. ieee, 1977, pp. 46–57.

[26] Y. Falcone, S. Krstić, G. Reger, and D. Traytel, "A taxonomy for classifying runtime verification tools," *International Journal on Software Tools for Technology Transfer*, vol. 23, no. 2, pp. 255–284, 2021.

[27] C. Sánchez, G. Schneider, W. Ahrendt, E. Bartocci, D. Bianculli, C. Colombo, Y. Falcone, A. Francalanza, S. Krstić, J. M. Lourenço, *et al.*, "A survey of challenges for runtime verification from advanced application domains (beyond software)," *Formal Methods in System Design*, vol. 54, pp. 279–335, 2019.

[28] M. Leucker and C. Schallhart, "A brief account of runtime verification," *The Journal of Logic and Algebraic Programming*, vol. 78, no. 5, pp. 293–303, 2009.

[29] G. Blair, N. Bencomo, and R. B. France, "Models@run.time," *Computer*, vol. 42, no. 10, pp. 22–27, 2009.

[30] P. Maes, "Concepts and experiments in computational reflection," *ACM Sigplan Notices*, vol. 22, no. 12, pp. 147–155, 1987.

[31] L. Baresi and C. Ghezzi, "The disappearing boundary between development-time and run-time," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*, 2010, pp. 17–22.

[32] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Future of Software Engineering (FOSE'07)*. IEEE, 2007, pp. 37–54.

[33] N. Bencomo, S. Hallsteinsen, and E. S. De Almeida, "A view of the dynamic software product line landscape," *Computer*, vol. 45, no. 10, pp. 36–41, 2012.

[34] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems," in *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23*. Springer, 2011, pp. 585–591.

[35] R. Alur and T. A. Henzinger, "Reactive modules," *Formal methods in system design*, vol. 15, no. 1, pp. 7–48, 1999.

[36] A. Legay, B. Delahaye, and S. Bensalem, "Statistical model checking: An overview," in *International conference on runtime verification*. Springer, 2010, pp. 122–135.

[37] H. L. Younes and R. G. Simmons, "Probabilistic verification of discrete event systems using acceptance sampling," in *International Conference on Computer Aided Verification*. Springer, 2002, pp. 223–235.

[38] Y. Butkova, A. Hartmanns, and H. Hermanns, "A Modest approach to modelling and checking Markov automata," in *International Conference on Quantitative Evaluation of Systems*. Springer, 2019, pp. 52–69.

[39] C. Hensel, S. Junges, J.-P. Katoen, T. Quatmann, and M. Volk, "The probabilistic model checker storm," *International Journal on Software Tools for Technology Transfer*, pp. 1–22, 2021.

[40] X. Zhao, M. Osborne, J. Lantair, V. Robu, D. Flynn, X. Huang, M. Fisher, F. Papacchini, and A. Ferrando, "Towards integrating formal verification of autonomous robots with battery prognostics and health management," in *International Conference on Software Engineering and Formal Methods*. Springer, 2019, pp. 105–124.

[41] F. M. Delle Fave, A. Rogers, Z. Xu, S. Sukkarieh, and N. R. Jennings, "Deploying the max-sum algorithm for decentralised coordination and task allocation of unmanned aerial vehicles for live aerial imagery collection," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 469–476.

[42] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Advances in Artificial Life: Third European Conference on Artificial Life Granada, Spain, June 4–6, 1995 Proceedings 3*. Springer, 1995, pp. 704–720.

[43] S. D. Ramchurn, J. E. Fischer, Y. Ikuno, F. Wu, J. Flann, and A. Waldock, "A study of human-agent collaboration for multi-UAV task allocation in dynamic environments," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[44] M. Calder and M. Sevegnani, "Stochastic model checking for predicting component failures and service availability," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 1, pp. 174–187, 2019.

[45] X. Xin, S. L. Keoh, M. Sevegnani, and M. Saerbeck, "Run-time probabilistic model checking for failure prediction: A smart lift case study," *IEEE 8th World Forum on Internet of Things*, 2022.