Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Decentralized runtime enforcement for robotic swarms*

Chi HU†1, Wei DONG†‡1, Yong-hui YANG2, Hao SHI†1, Fei DENG2

[1]College of Computer Science, National University of Defense Technology, Changsha 410073, China

[2]Institute of Computer Application, China Academy of Engineering Physics, Mianyang 621999, China

†E-mail: huchi16@nudt.edu.cn; wdong@nudt.edu.cn; shihao14@nudt.edu.cn

Received Apr. 30, 2020; Revision accepted Sept. 20, 2020; Crosschecked Oct. 10, 2020

Abstract: Robotic swarms are usually designed in a bottom-up way, which can make robotic swarms vulnerable to environmental impact. It is particularly true for the widely used control mode of robotic swarms, where it is often the case that neither the correctness of the swarming tasks at the macro level nor the safety of the interaction among agents at the micro level can be guaranteed. To ensure that the behaviors are safe at runtime, it is necessary to take into account the property guard approaches for robotic swarms in uncertain environments. Runtime enforcement is an approach which can guarantee the given properties in system execution and has no scalability issue. Although some runtime enforcement methods have been studied and applied in different domains, they cannot effectively solve the problem of property enforcement on robotic swarm tasks at present. In this paper, an enforcement method is proposed on swarms which should satisfy multi-level properties in uncertain environments. We introduce a macro-micro property enforcing framework with the notion of agent shields and a discrete-time enforcing mechanism called $\mathcal{D}$-time enforcing. To realize this method, a domain specification language and the corresponding enforcer synthesis algorithms are developed. We then apply the approach to enforce the properties of the simulated robotic swarm in the robotflocksim platform. We evaluate and show the effectiveness of the method with experiments on specific unmanned aerial vehicle swarm tasks.

Key words: Runtime enforcement; Multi-level property; $\mathcal{D}$-time enforcement; Robotic swarm
https://doi.org/10.1631/FITEE.2000203 **CLC number:** TP311

1 Introduction

Nowadays, robotic systems are attracting considerable attention in many important areas (Wong et al., 2017). Inspired by the biology of flocks of birds, herds, or other animals, many modeling approaches and control algorithms for robotic swarms have been studied. Nevertheless, the previous swarm modeling approaches can neither ensure the correctness of the swarm task scheduling nor mimic the collision avoidance behavior of agents in biological swarms under dynamic perturbation. In a macro view, the robotic swarm task should be ensured with a given timing sequence. Each sequence consists of a series of swarming states that are abstracted from the individual agent states. When the macro task property is violated, we enforce the correctness of the swarm state sequence by adjusting the associated error agents. The agent members in the swarm are likely to be disturbed by our enforcement operations. Thus, our approach correspondingly considers the micro view. We enforce the single agent state, which often affects the established collision-free stable state in swarming dynamic change. In sum, we expect to construct shields (to output actions that correct a property violation) on the two-level properties at runtime.

‡ Corresponding author
* Project supported by the National Natural Science Foundation of China (Nos. 62032019 and 61690203)
ORCID: Chi HU, https://orcid.org/0000-0002-4314-5862; Wei DONG, https://orcid.org/0000-0002-8033-7943; Hao SHI, https://orcid.org/0000-0002-6318-9918; Fei DENG, https://orcid.org/0000-0002-0391-1569
© Zhejiang University and Springer-Verlag GmbH Germany, part of Springer Nature 2020

Sinhuber et al. (2019) stated that the swarm would first make an initial startle response after a sudden perturbation (property violation), and then relax into a steady state. Therefore, we believe that enforcing behaviors on the robotic swarm (Parr, 2013) should be a continuous driving force in a real multi-agent swarm system. On this basis, we propose a $\mathcal{D}$-time (discrete-time) enforcement mechanism on robotic systems.

To enforce the continuous-time robotic state sequences at the two levels, we need to describe periodic and multi-agent specifications. In our previous work (Shi et al., 2017), we used a periodic time interval to discretize the real-time property. The swarm states can be discretized into regular intervals, each separated from the next by a discrete-time interval.

In this study, we propose a runtime enforcement framework for a robotic swarm and implement the corresponding platform. The method includes a series of retrieval interfaces for monitoring the swarm states, macro-micro property monitor generation, $\mathcal{D}$-time enforcer generation, discrete-time temporal logic, and a hierarchical specification language for describing the two-level property.

Our main contributions include:

1. A macro-micro (two-level) swarm runtime monitoring approach, which observes sequence violations in the overall swarm tasks and single agent violations under sudden perturbations, is implemented by monitoring hierarchical interfaces that can retrieve the needed states of a swarm. The interfaces are designed as an extendable way to customize the monitoring requirements.

2. A property enforcement mechanism called $\mathcal{D}$-time enforcement is proposed for the continuous-time state changing characteristics of swarms. We argue that since $\mathcal{D}$-time enforcement works flexibly in time steps and requires no knowledge of the upcoming dynamic environmental states, it is also suitable for other continuous-time cyber-physical systems (CPSs) (Rajkumar et al., 2010).

3. Furthermore, a monitor specification language is designed that can specify safety properties and enforcement behaviors for robotic swarms with temporal and timed requirements.

4. We implement this runtime verification approach on robotflocksim, a swarm simulation tool (Vásárhelyi et al., 2018). The effect is demonstrated in experiments by a significant safety improvement.

Finally, we outline the industrial application challenges and discuss their potential solutions.

2 Related works

Early on, microscopic-agent-based swarm models were proposed (Reynolds, 1987), followed by the typical flocking models in collective robotics (Brambilla et al., 2013; Floreano and Wood, 2015). Then, there were a few works that included uncertainties from environmental impacts. In Turgut et al. (2008) and Gökçe and Şahin (2009), the flocking models including motion constraints and collision avoidance were proposed. Overall, based on the works of these predecessors, robot flocking models under self-organized algorithms have been greatly improved. For example, Vásárhelyi et al. (2018) carried out a self-organized approach with a real swarm of 30 drones. It is the largest outdoor aerial system without central control reported to date exhibiting flocking with uncertainty under environment impacts (e.g., collective collision and object avoidance).

The self-organized model of robotic swarms has attracted a significant number of research groups currently contributing to the field. From Şahin et al. (2008), an incomplete list of such groups includes Caltech (Williams and Burdick, 2006; Chung et al., 2018), Carnegie Mellon (Khosla et al., 2002; Nagavalli et al., 2015), Ecole Polytechnique Lausanne (Mondada et al., 2005; Pugh and Martinoli, 2006), Georgia Tech (Egerstedt et al., 2020), and MIT (Schwager et al., 2006). Nevertheless, the collective tasks of swarm robotics systems still face a series of challenges: robustness properties under extreme environments, safety under unexpected changes in complex swarming tasks, and a scalability problem under limited sensing, communication, and computation capabilities. Specifically, nowadays, as Fine and Shell (2013) remarked, "there is no consensus on the precise details of the motions needed to produce rich flocking motions under realistic sensing models, actuation, and dynamics constraints." As Vásárhelyi et al. (2018) showed, most works lack completeness and precision in terms of control and validation under uncertain environments, only a few have included motion constraints and collision avoidance (Reynolds, 1987; Turgut et al., 2008; Gökçe and Şahin, 2009), and none have handled motion constraints explicitly.

In recent years, an increasing amount of research has been done on runtime enforcement in robotics. For example, Könighofer et al. (2017) presented the first shield synthesis solution for reactive systems and reported experimental results. Raju et al. (2019) presented the first approach for synthesizing shields at runtime on multi-agent systems. However, the previous works cannot solve the problem of swarm enforcement on swarming tasks. Moreover, the runtime construction of discrete enforcers with the multi-layer properties of robotic swarms is an unsolved problem. Thus, the work in this paper is unprecedented.

3 Preliminaries and motivations

3.1 Runtime enforcement and shields

Runtime enforcement (RE) is a technique to monitor the execution of a system at runtime and ensure compliance against a set of formal requirements (Pinisetty et al., 2017). Application of the RE method on robotic swarms is rare at present. In this study, we use a domain specification to define the macro-micro properties and the corresponding initial values of the variables. Our swarm RE aims to ensure that the execution sequence satisfies the given property with the runtime monitor U and an enforcer generator.

As shown in Fig. 1, the given property φ is defined in the specification. When the property monitor takes the sequence σ of swarm events as input and finds a violation of the given property, then the enforcer can be generated at runtime, and the enforcing action will be added to the execution sequence.

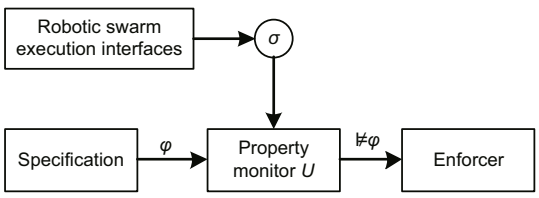

Fig. 1 Runtime enforcement monitor context for robotic swarms

In this paper, the robotic execution σ can be any sequence of events over the event set $\mathbb{E}$, i.e., $\sigma \in \mathbb{E}^*$ (Here, $\mathbb{E}^*$ is the set of finite length strings made up of the elements in $\mathbb{E}$). We consider the robot execution violation as a deviation between the observed execution event sequence σ and the given property φ that the robot system should satisfy.

An execution violation might lead to a swarm mission failure or swarm destruction. In our swarm RE approach, we deploy monitors that can automatically observe and detect the execution violation against the expected safety property defined in the specification and generate the enforcers. Generally, the monitoring process should bring a low overhead to the execution correctness of the current robotic execution.

Formally, let Σ denote the alphabet of the robotic system instead of $\mathbb{E}$ in the real system. We consider the execution of robotic systems to be a finite word, i.e., ω. Here, $\omega \in \Sigma^*$. The concatenation of two words ω and ω' is denoted as $\omega \cdot \omega'$. Word ω' is the prefix of word ω, denoted as $\omega' \leq \omega$, whenever there exists a word ω'' such that $\omega = \omega' \cdot \omega''$. Thus, the property violation/satisfaction problem of robotic executions can be abstracted to check whether the intercepted word ω is an element of the set of the finite words which satisfy the safety property φ (i.e., in a mathematical way, whether a given word ω is included in the language $\mathcal{L}(\varphi) \subseteq \Sigma^*$). Furthermore, the RE problem of a robotic swarm can be abstracted as $E(\sigma', t) = (\sigma, t)$, $\sigma \in \mathcal{L}(\varphi)$. Here, the pair (σ', t) denotes an actual monitored execution at the current time t. E is an enforcement operation.

The goal of RE is to guarantee the correctness of a small set of critical properties at runtime. These properties might occasionally be violated by an uncertain environment. Thus, some pioneers would like to automatically construct a component in the target systems, called a shield. The shield corrects mainly the erroneous output at runtime when necessary. Moreover, changes to the execution path of the shield should be minimal. The enforcement operation E is implemented by a set of shields.

In this study, our shield is applied to robotic systems. We regard the robotic swarm to be a reactive system as $\mathcal{D} = (Q, q_0, \Sigma_{\mathrm{I}}, \Sigma_{\mathrm{O}}, \delta, \lambda)$ is a Mealy machine, where Q is a finite set of swarm states, $q_0 \in Q$ the initial state, $\delta : Q \times \Sigma_{\mathrm{I}} \to Q$ the complete transition function, and $\lambda : Q \times \Sigma_{\mathrm{I}} \to \Sigma_{\mathrm{O}}$ the complete output function. Correspondingly, the decentralized agents are reactive subsystems $\mathcal{D}_i = (Q_i, q_{0i}, \Sigma_{\mathrm{I}i}, \Sigma_{\mathrm{O}i}, \delta_i, \lambda_i)$. The robotic swarm system designed in this study consists of agents, i.e., $\mathcal{D} \stackrel{\mathrm{def}}{=} \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{|\mathcal{U}|}\}$, where $\mathcal{U}$ represents

the set of all agents. The enforcers generate the decentralized shields on every agent. It is defined as $S \stackrel{\text{def}}{=} \{S_1, S_2, \ldots, S_{|\mathcal{U}|}\}$. Here, the property shield S_i is a set of similar reactive systems represented by a Mealy machine that will be described in detail in Section 3. Thus, the runtime enforcement on the swarm system is $\mathcal{D} \circ S = \{\mathcal{D}_1 \circ S_1, \mathcal{D}_2 \circ S_2, \ldots, \mathcal{D}_{|\mathcal{U}|} \circ S_{|\mathcal{U}|}\}$. Here, "$\circ$" denotes the joint behavior of the original system and the shields. The generating and synthesizing approach for the shields will be discussed in our framework. The problem that execution trajectories changed by the shield are correct and can deviate minimally is discussed in this paper.

3.2 An illustrative example

To enforce the critical properties automatically and flexibly, we prefer a shield synthesis approach (Könighofer et al., 2017) that automatically constructs two-level shields at runtime to avoid violation of these properties, instead of matching the current cognitive perturbation patterns or the wrong task sequence patterns and taking corresponding behavioral measurement. We give an illustrative example of the swarm enforcement here by referring to some main mission scenarios of unmanned aerial vehicles (UAVs) (Rasmussen et al., 2018). In addition, we introduce the runtime enforcement and shield synthesis foundations related to our work.

We consider a robotic swarm that can move among four regions, i.e., $\Sigma = \{A, B, B', C\}$, as shown in Fig. 2. We specify a macro-level property that the swarm should start from A and pass through B to region C under a self-organized modeling and control algorithm. The task states can be regarded as the position states of the overall swarm, which is shown as the region graph in Fig. 2. The moving behavior from B' to C immediately is not allowed in the specification. Intuitively, such a macro property involves the location state of the entire robotic swarm. However, different from the properties of agents, macro properties are occasionally uncertain. As shown in Fig. 2, Rmem.var.1 represents the state variable of a particular member of the robotic swarm with ID being 1. Corresponding, Swarm.var represents a particular swarming state. For example, we set the swarm location state $\text{Swarm.var}(A, t) = 1$ to represent that the swarm is in region A at time t. Since the individual robots in the swarm may not be in the same region, we define $\text{Swarm.var}(A, t) = -1$ if the swarm state is uncertain.

3.2.1 Safety properties

In this example, assume that there is an unexpected perturbation in C when the swarm is moving

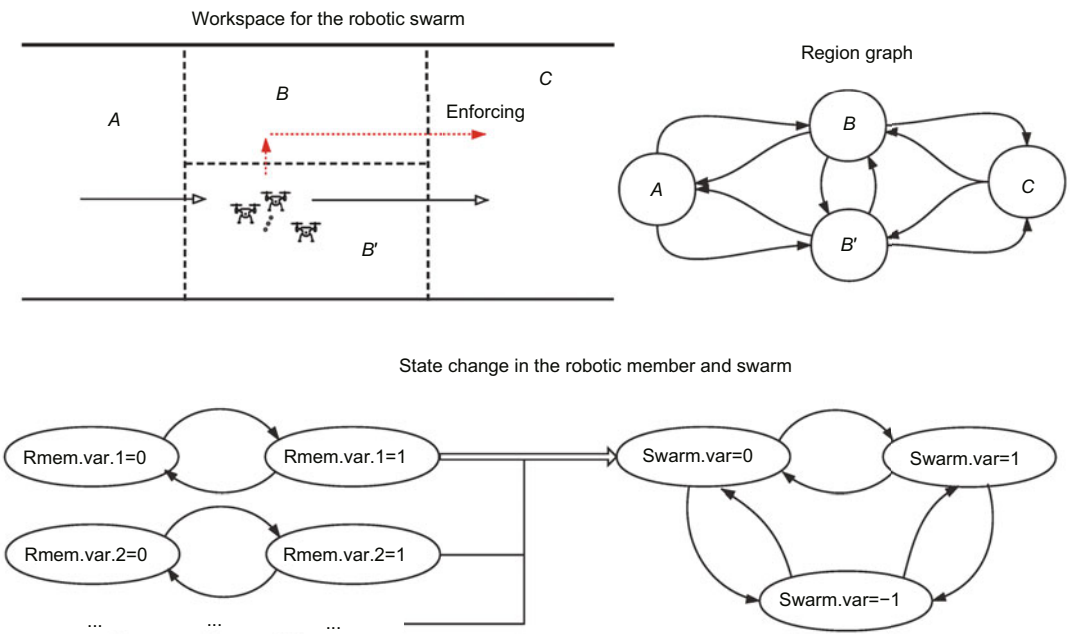

Fig. 2 An illustrative example in the robotic swarm safety moving task

from regions B to C. Specifically, one agent of the swarm should steer back to a wide space to avoid collisions with other neighbors when identifying the unexpected threats in region C. The micro-level flock safety enforcement operation should have the highest priority. Finally, the output agent behaviors from our shields should satisfy both the global task planning and the micro-level collision-free safety properties.

Thus, we declare the macro-micro safety requirements (REQs) in our illustrative example:

1. REQ1

The macro property for the task requirement of the robotic swarm in our example is that the swarm moves strictly in the order from A to B and then to C.

2. REQ2

The property for the local safety requirement of the robotic agents in our example is that the robotic sensors on the swarm cannot receive three consecutive collision alerts in 500 ms.

The specification for the above safety properties and the corresponding enforcement methods will be discussed later.

3.2.2 Robotic state enforcement

In Fig. 2, we model the region of operation for swarm agents as a hierarchical labeling system. For example, we assume that the sensor of a single robot receives the locating state $\mathrm{Rmem.loc}.i \in \{A, B, B', C\}$, which represents the signal of a robot state in the four regions. All states of the robot members make up the swarm states. The given property $\varphi(t)$ specifies $\mathrm{Swarm.loc}(A, t) = 1$, which means that all the locations of agents are in A at t. The property $\varphi(t)$ can be defined on agents in the system as

$$\begin{aligned}\varphi(t) &::= \mathrm{Swarm.loc}(A, t) = 1\\ &::= \wedge_{i\in\{1,\ldots,|\mathcal{U}|\}}\mathrm{Rmem.loc}.i(A, t) = 1.\end{aligned} \tag{1}$$

If the locating state of robot i violates the given property φ, we try to transfer the robot direction, which is denoted as $\mathrm{Rmem.dir}.i \in \{\mathrm{ff}, \mathrm{ll}, \mathrm{rr}, \mathrm{bb}\}$. Here, ff, ll, rr, and bb represent the moving forward, turning left, turning right, and steering back, respectively. If robot i is mounting a shield, the shield may make a correction in the output signals to change the moving direction output from ff to bb, i.e., $S_i(\mathrm{ff}) = \mathrm{bb}$. Shields play the role of transferring the violated input trajectory to the right output trajectory. Note that the moving states between regions do not happen instantaneously. The moving and enforcing behaviors output o is usually an action implemented by executing the (set of) underlying continuous controllers of a robot, e.g., an enforcing output of steering 90°. It takes some time to enforce the safety property.

3.3 Specification and temporal logic

In the RE specification for a robotic swarm, formally, a property φ is described with some temporal logic such as linear temporal logic (LTL) (Pnueli, 1977). To describe the periodic specification, in our previous work (Shi et al., 2017), we used the periodic nature of the embedded system to discretize the real-time property and replace the real number in the time constraint with the number of CPU clock cycles. The discrete-time metric temporal logic (DT-MTL) we used is obtained by adding temporal operators in LTL with discrete-time intervals. Compared with MTL, its computational complexity is reduced. DT-MTL has been applied to many important control systems. For example, it has been used to describe the real-time properties of NASA's unmanned robotics systems (Reinbacher et al., 2014).

We design our robot monitoring specification language based on DT-MTL. With the time interval $I = [t, t']$, where $t, t' \in \mathbb{N}$, the syntax of DT-MTL is similar to LTL (Reinbacher et al., 2014).

Definition 1 (Syntax of DT-MTL) Let AP be the set of atomic propositions, $p \in \mathrm{AP}$. The discrete-time MTL formula φ can be defined as

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1\, \mathcal{U}_I\, \varphi_2, \tag{2}$$

where $\bigcirc\varphi$ represents that the property φ should be satisfied at the next time point.

To be convenient, derived temporal operators are usually defined as follows:

$$\Diamond_I\varphi \triangleq \mathrm{true}\, \mathcal{U}_I\, \varphi, \quad \Box_I\varphi \triangleq \neg\Diamond_I\neg\varphi. \tag{3}$$

Definition 2 (Semantics of DT-MTL) Let $\pi = a_0\, a_1 \cdots \in \Sigma^\omega$ be an infinite word and $i \in \mathbb{N}$ be a time point. The semantics of DT-MTL is defined as

follows:

$$\pi(i) \models \begin{cases} P_{\text{Atom}}, & \text{if on } \pi(i),\ P_{\text{Atom}} \text{ is true}, \\ \neg\,\varphi, & \text{if } \pi(i) \not\models \varphi, \\ \varphi_1 \wedge \varphi_2, & \text{if } \pi(i) \models \varphi_1 \text{ and } \pi(i) \models \varphi_2, \\ \bigcirc\ \varphi, & \text{if } \pi(i+1) \models \varphi, \\ \varphi_1\, \mathcal{U}_I\, \varphi_2, & \text{if } \exists j \in I : \pi(i+j) \models \varphi_2, \\ & \text{and } \forall k, 0 \le k < j : \pi(i+k) \models \varphi_1, \\ \Box_I\, \varphi, & \text{if for all } j \in I,\ \pi(i+j) \models \varphi. \end{cases} \tag{4}$$

We use the monitor generation algorithm for MTL (Dreossi et al., 2019) and our previous work (Hu et al., 2020) to implement the runtime monitoring of DT-MTL properties.

4 Macro-micro runtime enforcement for a robotic swarm

In this study, conceptually, we wish to enforce the properties of a robotic swarm system organized as our example, with a robotic control model producing actions, and for every action produced, the underlying robotic execution system returns a result to the target control model for computing the next action. Results may be exceptions or void or unit values, so all actions can be considered to produce results. For simplication, we assume that all the robotic actions are synchronous; after the application produces an action a, it cannot produce another action until receiving a result for a. In contrast, the robotic action shield can be viewed as one monitor result, in which all agent actions are fully asynchronous (because shields can buffer without executing an unbounded number of actions).

4.1 Macro-micro properties

Fig. 3b shows how we think of a macro-micro monitor enforcing the swarm system in Fig. 2. In Fig. 3b, the monitor interposes on and generates the enforcers to transform actions and results. Enforcers ensure that the actions are actually executed, and that the results actually returned to the control model are valid (i.e., have the desired macro-micro properties). The monitor may be or may not be inlined into the target control algorithms.

The two-level enforcement mechanism capability for transforming the results of actions is novel among general runtime enforcement models, as far as we are aware of. However, this ability is crucial for enforcing many swarming task policies, which may require (trusted) mechanisms to sanitize the results of actions before a (an untrusted) control algorithm accesses those results. For example, policies may require the swarm to move in an ordered sequence to the regions when macro-level swarm organizing applications retrieve the to-do lists, but the robotic system might be impacted by malicious data or sudden environmental changes. Because existing swarm control frameworks do not monitor actions to allow changing of results, one cannot use existing models to specify or guarantee the correctness of a swarm task and the collision-free policies of agents without such an enforcement mechanism.

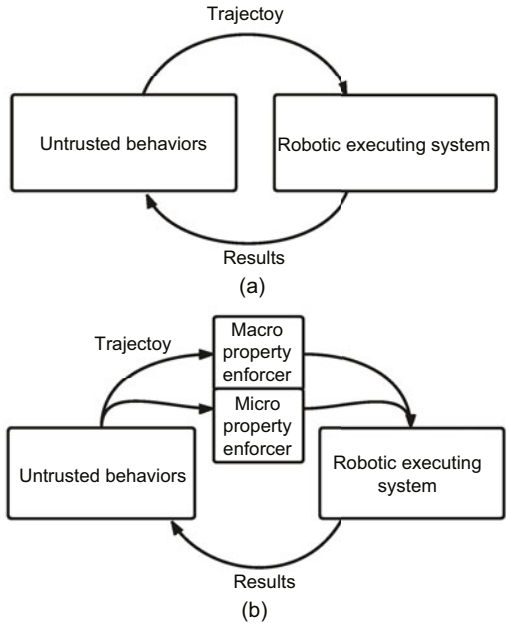

Fig. 3 An untrusted robotic control model that executes actions on a robotic swarm system and receives results for those actions (a) and a macro-micro monitor that interposes on and enforces the validity of the robotic actions executed and the results returned by synthesizing enforcers (b)

To enforce the safety properties (REQ) in our illustrative example, in the RE framework, we trace the robotic swarm execution sequence σ and construct two kinds of monitors to observe the truth values of the macro-micro properties. The macro property monitor U is to observe the truth values of swarming task properties given in the specification, i.e., REQ1. The micro property monitor C is to calculate the deviation of specified agent safety properties, i.e., REQ2. Here, our decentralized

monitor design is inspired by a similar work (Bauer and Falcone, 2016). If violations are monitored, the enforcer can be generated automatically (Fig. 4).

4.2 Property specification

In our RE specification, specification of all the swarm properties is not supported. Briefly, we focus on the swarming task requirements and individual robot requirements. At these two levels, we define a custom kind of property that considers the task sequence and individual robot properties as the macro-micro property of robotic swarms. For example, in a specification φ, we specify the moving sequence of a robotic swarm between regions and the immediate return of a single robot when its battery is low.

To define the two-level property φ and enforcing actions in Fig. 4, we have proposed a novel hierarchical specification language in a previous work (Hu et al., 2020). First, our specification declares swarm sensor inputs, initial states, and enforcement information. We can describe the macro-micro property in DT-MTL logic. The syntax is shown as follows:

```
< Specification > SwarmID ::= {
  < Config parameters > ::= {
    Version: [string] | initBotNum: [int] | maxV: [float]
    ParaConf: [int] ···
    Data: [float] ···
    }
  < Atomic proposition > AP ::= {
    Rmem.var_1.i * Rmem.var_2.i | Rmem.var.i * a_val |
Rmem.loc.i = a_loc | Rmem.port.i= a_port
     SwarmID.var ::= x ∈ ∪_{i∈{1,...,|U|}} Rmem.var.i
     a.val ::= v ∈ Data      a.loc ::= v ∈ Data[]
    }
  < Macro property > φ_s ::= {
    AP | ¬φ | φ_1 ∧ φ_2 | ◯φ | φ_1 U_I φ_2
    } @ {<text Flocking enforcing action >}
  < Micro property > φ_a ::= {
    AP | ¬φ | φ_1 ∧ φ_2 | ◯φ | φ_1 U_I φ_2
    } @ {<text Agent enforcing action >}
  }
```

Our algorithms can parse the specification and generate the monitors. The enforcers are automatically generated in this process. In parallel threads, the monitoring interfaces obtain the multi-robot data from the execution and convert them to the truth value of {0, 1}.

Generally, the atomic propositions in the property are made up of robotic sensor events. For instance, an atomic proposition of the property can express a state of the last executed port, a speed value of a robot member, or the current location of a robot member. Formally, an event of the swarm can be defined as a state formula over the atomic propositions of the robot members expressed in the syntax of the specification (where $* \in \{<, \leq, =, \neq, \geq, >\}$).

The enforcing action declared in the specification contains actions that we can take on swarms and individual agents. For example, the partial enforcing actions in our illustrative example are:

(1) flocking enforcing actions (turning 90°, scattering, and stop);

(2) agent enforcing action (turning 90°).

Take "turning 90°" as an example. By calculating how many turning operations are inserted, the enforcement in four directions is realized. The directions are expressed as ff, ll, rr, and bb. The enforcing action generation is implemented through the enforcing automaton that will be covered next.

4.3 Enforcing automaton

We form two edit automatons (Ligatti et al., 2005) (macro and micro) for enforcing the swarming properties. Our machine is described by a five-tuple form $(Q, q_0, \delta, \gamma, \omega)$ defined with respect to the robotic swarm system $(\mathcal{A}, \Sigma)$. Here, $\mathcal{A}$ represents all actions in the robotic swarm, and Σ represents all states of the swarm. In the five-tuple form, Q specifies the possible automaton states, and q_0 is the initial state. The partial function $\delta : \mathcal{A} \times Q \rightarrow Q$ specifies the transition function for the automaton. γ specifies the insertion of a finite sequence of actions into the swarm's action sequence. The partial function ω indicates whether the action in question is to be inserted or not. The partial functions δ and ω have the same domain, while δ and γ have disjoint domains.

To specify the execution of our macro-micro enforcing edit automaton, we use labeled operational semantics. The basic single-step judgment will have the form of $(\sigma, q) \xrightarrow{\tau} E\,(a', q')$, where σ denotes the sequence of the original actions of the target program, q denotes the current state of the automaton, a' and q' denote the action sequence and state after the automaton takes a single step respectively, and τ denotes the sequence of actions produced by the automaton. The input sequence σ is not observable to the outside world, whereas the output τ is observable (i.e., τ may be any element of $\mathcal{A}^*$).

In our approach, we use the single-step

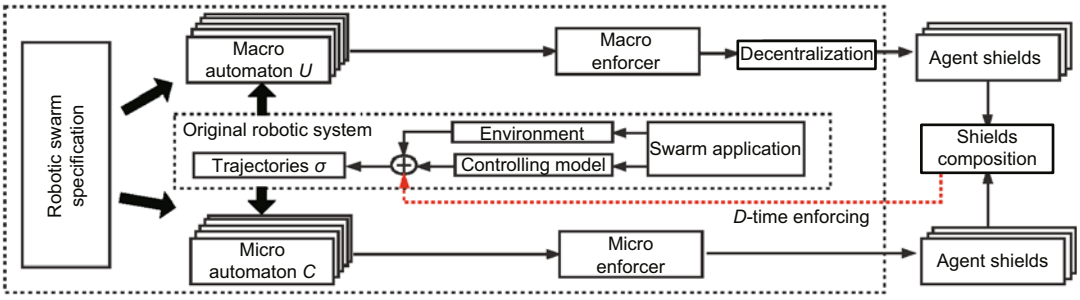

Fig. 4 Our runtime enforcement approach framework in a robotic swarm

enforcement semantics of edit automata as follows:

$$\begin{cases} (\sigma, q) \xrightarrow{\tau} E\left(a', q'\right), \text{ if } \sigma = a, a' \text{ and} \\ \qquad\qquad \gamma(a, q) = \tau,\ q' \text{ for E-INS}, \\ (\sigma, q) \xrightarrow{\cdot} E(\cdot, q), \text{ otherwise for E-STOP}, \end{cases} \tag{5}$$

where the symbol "·" denotes the empty sequence. Because of the continuous-time characteristic of robotic swarms, the suppression operation of actions in a control model is difficult to achieve. Thus, we believe that it is a more feasible approach to construct remedial enforcement operations. We define the rule E-INS as an insert operation (rather than accepting an action a, the automaton then suppresses the original a), and its presence allows acceptance of action in only one step. This simplifies the specification of the automaton and decreases its running time. Similarly, the effect of rule E-STOP can be accomplished by stopping any further robotic sequence input. It is obvious that different violations of properties will construct different enforcers that will construct corresponding decentralized shields on agents. The construction approach is discussed in detail in the next section.

In the illustrative example in Section 3, we discuss the macro-micro property. The property is that a swarm moves strictly in the order from A to B and then to C, and cannot receive the collision threats in three consecutive cycles. The single robot of a swarm system takes the collision warning (cw) of the robot sensor as an input. The swarm location variable is the value of the swarm location interface from all agents (loc). For simplication, we assume that the collision warning signals are Boolean numbers. The loc variable varies from $A_$loc to $C_$loc. The macro-micro requirements (REQ) and enforcing actions (ENF) are as follows:

1. REQ

The macro-micro properties are declared in the illustrative example in Section 3.

2. ENF

The enforcing action may be E-INS, scattering, or turning 90°.

As shown in Fig. 4, the construction of shields by the property enforcer needs the observed violation result and the deviation degree assessment. Thus, the deviation of this safety property can be regarded as a pathfinder which determines the right shield operation with the minimized deviation. For example, the swarming robots are threatened by numerous collision warning signals. Then the agent under the shield should rotate by a specific degree (abstracted as ff, ll, rr, and bb) by calculating how many enforcing actions need to be taken. The example should take into account the following specification with the formal property:

$$\begin{cases} \varphi_s ::= (\text{Swarm.loc}{=}A_\text{loc}\ \mathcal{U}\ \text{Swarm.loc}{=}?_\text{loc})\mathcal{U} \\ \quad ((\text{Swarm.loc} = B_\text{loc}\ \mathcal{U}\ \text{Swarm.loc} = ?_\text{loc}) \\ \quad \wedge\bigcirc(\text{Swarm.loc}{=}?_\text{loc}\mathcal{U}\ \text{Swarm.loc}{=}C_\text{loc})), \\ \varphi_a ::= \neg(\vee_{i=1}^{n}\text{Rmem.cw.}i \wedge \Diamond_{[0,5]} \vee_{i=1}^{n} \text{Rmem.cw.}i \\ \quad \wedge \Diamond_{[5,10]} \vee_{i=1}^{n} \text{Rmem.cw.}i). \end{cases} \tag{6}$$

Here, "?_loc" denotes the uncertain swarming state. "[0, 5]" denotes a time interval from 0 to 500 ms since our discrete-time period is 100 ms. Rmem.cw.i=1 denotes that the robot i receives the collision warning. We use the two properties as our macro-micro properties and construct the corresponding monitor (a DT-MTL logic parsing algorithm, non-automaton). Then, we can generate the $\mathcal{D}$-time enforcement shield by the enforcers of the two monitors. It is discussed below:

The safety property enforcement needs a corresponding deviation pathfinder. Thus, in this

example, if there is a violation of the macro property where the swarm attempts to enter C through B', we should rotate the direction of the swarm movement. The deviation variable is defined as the number of steps that can be returned to the normal task goal, i.e., the step number that the swarm takes to get back to region B. We denote the enforcing action as enforceAct. The deviationVal.min$(\cdot)$ function calculates the minimum deviation for different shield operations:

$$\text{enforceAct} ::= \text{deviationVal.min}\,(\,\text{ff}, \text{ll}, \text{rr}, \text{bb}\,). \quad (7)$$

4.4 $\mathcal{D}$-time enforcement

As shown in Fig. 5, we propose a domain discrete-time enforcement mechanism for robotic swarm systems. To ensure the global task of swarms under dangerous environments, we construct decentralized shields for execution only once each time a violation of a property is first monitored in the discrete-time interval. We denote this enforcement execution as $\delta'_{(1)}$. The $\mathcal{D}$-time enforcement is to synthesize the enforcement shields onto the decentralized agents. We use the enforcement shield output as the input of the agent control model, because the agents' micro-level control output ultimately determines the actual execution of the robotic system. From an external point of view, we think that the final action after $\mathcal{D}$-time enforcement is synchronized.

Intuitively, enforcers start interfering only when they must, and the enforcing actions depend on the output of the deviation calculation algorithm. This shield is constructed on a discrete-time interval where the violation occurs and continues until the end of the time interval. This process is shown in Fig. 5.

The $\mathcal{D}$-time enforcement design is based on the

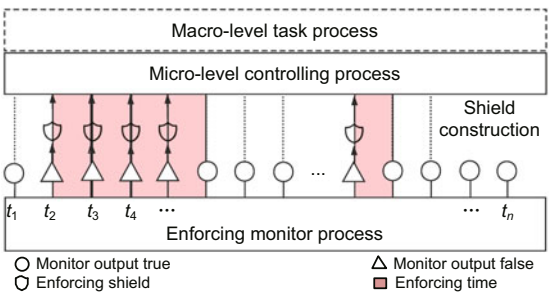

Fig. 5 $\mathcal{D}$-time enforcement approach based on the shield output and the original control output within a discrete-time interval

property shields, a reactive system represented by a Mealy machine $S = (Q, q_0, \Sigma_{\mathrm{I}}, \Sigma_{\mathrm{O}}, \delta, \lambda)$. Assume that our micro-level agents' designed control is also a reactive system, $\mathcal{D} = \left(Q, q_0, \Sigma, \Sigma_{\mathrm{O}}, \delta'_{(1)}, \lambda'\right)$, where $\Sigma = \Sigma_{\mathrm{I}} \cup \Sigma_{\mathrm{O}}$ is the set of new input symbols. The composition $\mathcal{D} \circ S$ is $\left(\hat{Q}, \hat{q}_0, \Sigma_{\mathrm{I}}, \Sigma_{\mathrm{O}}, \hat{\delta}, \hat{\lambda}\right)$, where $\hat{Q} = Q, \hat{q}_0 = q_0$, and $\hat{\delta}\,(q, \sigma_{\mathrm{I}}) = \delta'_{(1)}\,(\lambda\,(q, \sigma_{\mathrm{I}})\,, \sigma_{\mathrm{I}})$ is the transition function.

Thus, we observe the violation and execute the final enforcing action $\hat{\delta}$ by the shields once to ensure the swarming macro-micro property.

The $\mathcal{D}$-time enforcement method is proposed to extend the RE theory to robotic systems in a discrete-time way. Using $\mathcal{D}$-time enforcement, we can enforce robotic execution by referring to previous works in non-continuous systems.

5 Shield construction for enforcement

In this section, we formally define the enforcement shield construction problem. Enforcement of the swarm behaviors under threats must involve a reactive capability, i.e., a capability to correct the violation in the same control step. Furthermore, it must be constructed solely from the swarming safety properties denoted as φ, regardless of the implementation details of the design (the robotic system may be a black box). This ensures the simplicity of the enforcer as well as the scalability of the enforcement synthesizer.

From the macro-micro enforcing automaton in our approach framework, we can generate the single enforcing action E_φ. For example, we modify the swarming direction from B' to B (Fig. 6), steer the direction of robot 1 to avoid colliding with robot 2, and turn a robot to avoid colliding with the wall.

To monitor the property violation, we devise a monitoring algorithm from our previous work (Hu et al., 2020) and the Python library for MTL (Dreossi et al., 2019). It processes the runtime robotic states. In other words, the algorithm is an infinite loop that waits for input robotic states (letters of the alphabet Σ). If the monitored sequence of states following the current robotic state does not satisfy φ, then we hold this state and trigger the enforcer by implementing function E_φ. Otherwise, we output all events held by earlier iterations. The two-level property enforcing monitor algorithm is designed in Algorithm 1.

In the φ property monitor, we call a hierarchical two-level StateGo function. It is a runtime state transmission method which has been illustrated in our previous work (Hu et al., 2020). This method consists mainly of a series of retrieval interface functions and the logic operator algorithms of DT-MTL. When the enforcing monitor receives a violation, we build the enforcer by the trigger output E_φ_mac or E_φ_mic.

1. Decentralized shields

The agents in a swarm should satisfy a common macro task property. Thus, a macro enforcing action E_φ_mac can be defined as the finite agent actions as

$$E_\varphi_\mathrm{mac}(\mathrm{id}, t) \stackrel{\mathrm{def}}{=} \wedge_{i\in\{1,\ldots,|\mathcal{U}|\}} E_\varphi_\mathrm{mic}(i, t), \quad (8)$$

where i is the ID of the agent of the macro swarm. By recalling these agents in the swarm, we implement the macro enforcer as the specific subclass of the safety shield functions of agents. Here, E_φ_mic(i, t) might be decomposed by the macro enforcing trigger or the micro enforcing trigger. To show the difference, we define the shield $S_i(\varphi, t)$ generated by E_φ_mac as $S_i(\varphi, 1, t)$ and the shield directly generated by the micro enforcing trigger E_φ_mic as $S_i(\varphi, 2, t)$.

Thus, the decentralized shields generated by the enforcing trigger E_φ can be denoted as

$$\begin{cases} E_\varphi \stackrel{\mathrm{def}}{=} \wedge_{i\in\{1,\ldots,|\mathcal{U}|\}} S_i(\varphi, t), \\ S_i(\varphi, t) = \langle S_i(\varphi, 1, t),\ S_i(\varphi, 2, t)\rangle\,. \end{cases} \quad (9)$$

2. Shield synthesis

The set $\{S_i(\varphi, t) | i \in \mathcal{U}\}$ of synthesized shields (1) is correct, (2) deviates minimally, and (3) is bounded.

To satisfy φ and ensure a minimal deviation, the enforcer must not generate the output enforcing action arbitrarily. It must comply with the swarming goal and the degree of deviation of the enforcement trajectories. While correcting the erroneous behavior sequences, the deviation monitor returns the modified path with a minimized deviation.

This may be achieved by solving a two-player safety game as discussed in Standley and Korf (2011) and Zhang Y et al. (2014). Similarly, we propose a generic approach to obtain the deviation value of a specific agent by constructing the DeviationFinder (Fig. 6), where one player represents the design goal and the other player represents the alternative safety enforcement trajectories which are denoted as ff, ll, rr, and bb. Depending on the trajectories, our algorithm calculates an array of predicated deviations, i.e., DEVN. To be specific, the two tuples such as

Algorithm 1 Property enforcing monitor for φ

Input: a set of states STAT[id][σ] of the swarm
Output: an enforcing trigger E_φ

```
Function: enforcingMonitor(STAT, φ, σ)
loop
  update(σ)
  while macroStateGo(STAT, φ, σ) do
    while microStateGo(STAT, φ, σ) do
      if (microStateGo == false) then
        return E_φ_mic
      end if
    end while
    if (macroStateGo == false) then
      return E_φ_mac
    end if
  end while
end loop
```

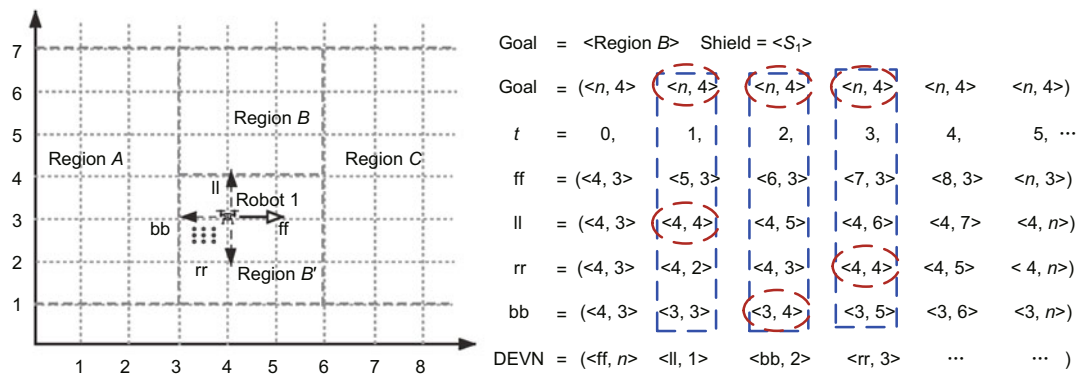

Fig. 6 Deviation degree constructed by our DeviationFinder for robot 1 in the example of Fig. 2 (the original moving direction, i.e., the input of shield S_1, is ff)

<ff, n> represent the enforcing action and the predicated deviation value. Using this general method to calculate the enforcing output, (1) the goal of enforcement construction makes sure that $\varphi(I, O')$ is always satisfied, (2) the enforcement trajectory output from shields is a minimal deviation modification, and (3) the path and enforcement shield actions are abstracted and defined on a discrete-time interval. Thus, the shields are bounded.

The enforcement shields will be executed only during a single interval, and the enforcing output depends on the DeviationFinder algorithm, as shown in Algorithm 2.

Algorithm 2 Property DeviationFinder for φ

Input: a set of signals VAR of the aimed sensors on each bot, the critical value CVAL for the swarming goal, and the enforcing action set actionSet

Output: the minimized deviation degree value DEVN_φ and the corresponding action ACT

```
Function: computeDevn(VAR, CVAL, actionSet)
n ← the number of elements in actionSet
for iter ← 1 to n do
    DEVN_iter ← CVAL−VAR_iter
    if DEVN_iter < min(DEVN_iter) then
        Update DEVN_φ = DEVN_iter
        Update ACT = actionSet[iter]
    end if
end for
Return DEVN_φ, ACT
```

The enforcement execution is a $\mathcal{D}$-time enforcement progress. The enforcing action outputs should be computed by the deviation degree DEVN_φ. Function computeDevn accumulates the violated properties φ and the corresponding enforcing trigger E_φ. If an enforcement requirement is received, then only the enforcement control signal is updated by calculating the minimal enforcing actions. If the enforcement requirement is not received, then it immediately loops the result logging event, which is sent out by the property monitor.

3. Shield composition

As shown above, the shield on agent i is a pair of partial functions $\langle S_i(\varphi, 1, t), S_i(\varphi, 2, t)\rangle$. Each agent's shield applies only to itself. This means that when multiple agents act in the same system, their trajectories are modified only by their respective shields. However, the individual shields of each agent act together to make the system safe. The joint behavior of the shields is captured by the functional composition.

Our synthesis algorithm builds upon the existing works (Bloem et al., 2014; Raju et al., 2019). We define the composition of shields for agents 1 and 2 at time t as

$$\begin{aligned} &S_1(\varphi, t) \circ S_2(\varphi, t) = S_2(\varphi, t) \circ S_1(\varphi, t) \\ &= \langle S_1(\varphi, 1, t) \circ S_2(\varphi, 1, t), S_1(\varphi, 2, t) \circ S_2(\varphi, 2, t)\rangle . \end{aligned} \tag{10}$$

Similarly, "$\circ$" denotes the joint behavior of the shields. This composition can be extended to an arbitrary number of shields by composing their constituent functions. We implement the algorithms in the C programming language. Our tool takes a specification file as input and reads the robotic states from the robotflocksim tool (Vásárhelyi et al., 2018). The specification describes the property and enforcement information. Our algorithms output the enforcement signal to the robotflocksim and drive the simulated robotic swarm movement as we wish. The tool cannot impact the global task-controlling behaviors in the robotflocksim, but we enforce the safety property in swarm tasks by the runtime enforcement mechanism. The enforcement progress can also be displayed in the robotflocksim tool. In this way, we perform experiments for robot control and enforcement on this platform.

6 Platform implementation and experiments

We implement the monitor specification and enforcement platform on robotflocksim, a simulation tool for robotic swarms. We apply shields in a scenario, in which a UAV swarm is controlled by a given control program. These shields maintain certain properties while performing a surveillance mission in a dynamic environment and ensuring collision-free movement among the UAV agents.

Note that a common UAV control model in the robotflocksim tool simulates a ground control station that communicates with an actuator onboard the UAV. The property monitor receives and displays updates from the UAV, including position, heading, wind speed, and battery state. Errors occur all the time if we use this type of purely adaptive control to ensure safety properties in dynamic swarming tasks. This is because a simple swarm control model might neglect some of the required safety properties due

to the previously unknown information, burst disturbance, illegal task instruction, high workload, fatigue, or an incomplete understanding of exactly how commands onboard UAV agents are executed.

Thus, we implement an enforcing monitor in the manner described earlier. It can also modify the actual execution trajectories to the UAV's actuator, such as waypoints to fly to. We assume that the UAV swarm routes have certain properties. For the UAV swarm simulated in the robotflocksim tool, its states are possibly modified by our enforcing monitor during mission execution to respond to the swarming task environment.

Consider the mission simulation platform in Fig. 7, which contains four huge open areas (already illustrated in Section 3), in which a UAV swarm contains 100 UAVs. In addition, we increase the number of UAVs to 150 or even 200 to experimentally compare the effect of our enforcement approach with that of the original system.

The experiments involve three fault injection approaches that set a risk identification on the regions, set the battery level, and construct unstable winds to cause collisions among the UAVs. The UAV enforcing monitor can collect all the related states on the UAV itself and observe the neighbors' loc.x, loc.y, dir.x, and dir.y from a specified nearby range. In this map, we identify region C as a target area that we have to enter in the correct sequence, and the boundary between two openly connected regions.

To reduce monitor overhead on the UAVs, we collect just the corresponding event and state traces of interest. The runtime monitors are generated and deployed on the robot agents automatically by parsing the specification. Furthermore, we release the idle monitors when a property specification does not require them.

6.1 Experimental settings

In our experiments, we specify the macro-micro property constructed by different robotic levels, e.g., the swarm task level and the collision-free agent level. These properties protect the robotic self-organized control algorithm from violation; the swarm task is illustrated as in the given specification. The properties are shown as follows:

(P1) The UAV swarm can fly only from area B to area C, but not from B' to C.

(P2) The UAV has to leave primary area A within 200 time steps.

(P3) The distance between any two UAVs is not less than 0.5 m to avoid collision.

(P4) Once the battery of a UAV is low, it should return to the designated landing site in zone A within

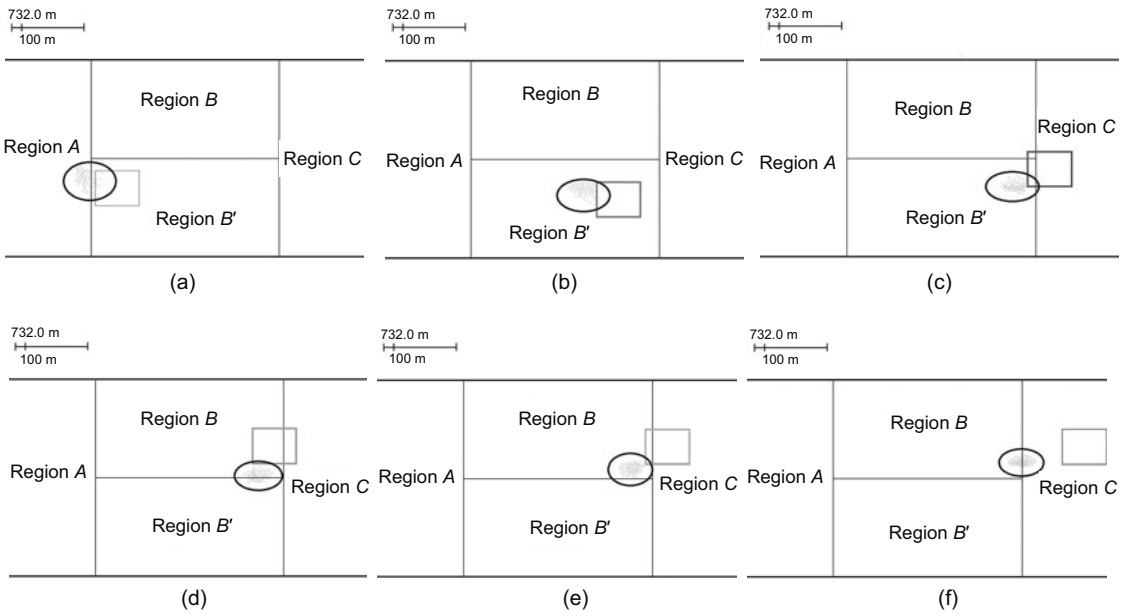

Fig. 7 Simulation platform containing four areas and a swarm of 100 UAVs: (a) from A to B'; (b) marching in B'; (c) from B' to C; (d) enforcing UAVs to B; (e) marching in B; (f) from B to C

The circles indicate the swarming locations and the boxes indicate the current target locations of the UAVs

5 time steps.

To evaluate the effectiveness when deploying P1–P4 enforcing monitor, we run the robotic swarm flocking simulation on the robotflocksim tool and test whether it correctly avoids the uncertain threats we have injected. The settings of this experiment are shown as follows:

(1) EXP_1. Command robotic swarm to forward from region A to region C without passing through region B.

(2) ErrorLevel. The property to be enforced is at the macro level.

(3) Violation_1. Swarm motion routes violate task assignment.

(4) Enforcing_1. The action of the swarm entering region C is suppressed.

In this experiment, we model a task where the swarm moves from A to C bypassing B'. After the swarming location value changes to C_loc from B'_loc, the macro monitor captures the violation of the swarm task policy. Relying on the Deviation-Finder calculation, the monitor executes the shields on agents to redirect the UAV swarm to B correctly (The experimental process is shown in Fig. 7).

(1) EXP_2. Command robotic swarm to keep moving in region A.

(2) ErrorLevel. The property to be enforced is at the macro level.

(3) Violation_2. Swarm motion routes violate task assignment.

(4) Enforcing_2. The action of the swarm entering other regions is inserted.

After 200 time steps, we test whether the swarm is still in region A as scheduled. The observed result is that, as expected, the shields force the UAVs to move toward the target region C through Deviation-Finder's calculation result.

We do another micro enforcing experiment on the robotflocksim to test the local agents' collision-free enforcing ability with the micro monitor. The settings of this experiment are shown as follows:

(1) EXP_3. Configure the UAV swarm to move at different speeds.

(2) ErrorLevel. The property to be enforced is at the micro level.

(3) Violation_3. Location oscillation of agents violates the collision-free property.

(4) Enforcing_3. The action of agents is inserted. Agents tend to reduce the movement speed of the swarm.

In this experiment, we increase the wind speed to 4 m/s and constantly adjust the swarming speed of the UAVs. It tests whether our micro enforcing monitors can reduce the collisions between the UAVs that occur in the original control system. We experiment with different numbers of UAVs, and the results are shown in Fig. 8.

In the enforcement experiment where the battery is low, we add the virtual battery power flag for each UAV in the robotflocksim tool and set different simulated power conditions. Then, we observe that the UAVs would return to the base under a low battery level as expected. The settings of this experiment are shown as follows:

(1) EXP_4. Set a specific UAV to low battery power status.

(2) ErrorLevel. The property to be enforced is at the micro level.

(3) Violation_4. The battery level of a UAV is too low to navigate.

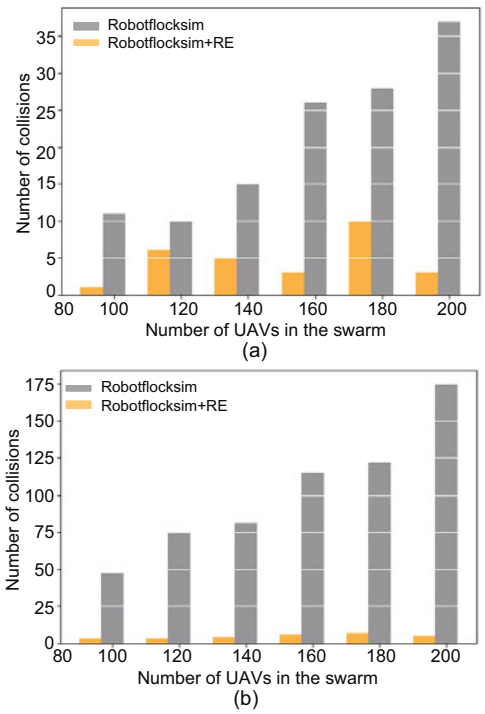

Fig. 8 Enforcement results of the collision-free property of P3 simulated on a swarm in size ranging from 100 to 200 UAVs with different swarming speeds: (a) collisions at swarm speed of 6.0 m/s; (b) collisions at swarm speed of 8.0 m/s

The orange cylinder in robotflocksim+RE indicates the number of collisions on the robotflocksim platform with the RE approach. References to color refer to the online version of this figure

(4) Enforcing_4. The action of the agent is inserted. It is enforced to move back into region A.

6.2 Evaluation

Table 1 shows the overhead time of the enforcing monitors under the simulation task with 100 UAVs, i.e., Time.Overhead. The cardinality of the state space $|Q|$ is on a specific property that the robotic simulation platform monitors. Shield size $|S|$ is the number of shields that are generated only by the enforcing monitors. Time.Task is the task execution time of the monitored robotic system in our experiments. For example, the whole time cost of the UAV task process specified in P1 is 80 time steps. A time step contains 30 clock cycles and each clock cycle set in the experiment is 100 ms. Similarly, we observe the task processes for property P2 in 200 time steps, i.e., 600 s.

All experiments were performed on a computer running Ubuntu 16.04 with 3.1 GHz CPU and 4 GB RAM. Results showed that our macro-micro enforcement approach scales well for the robotic systems on the robotflocksim simulation platform. A summary of our experimental results is shown as follows:

1. Many previous methods can construct a self-organized control model for robots in a stable environment. However, they cannot cope with a sudden change of the environment or guarantee the correctness of a swarming mission. There are few studies on a runtime enforcement method for the hierarchical macro-micro levels of a robotic swarm. Therefore, it is not easy to compare our work with others directly. However, in a comparison with the classic swarm control algorithms without enforcement (Turgut et al., 2008; Gökçe and Şahin, 2009; Brambilla et al., 2013; Floreano and Wood, 2015), our approach has the effect of correcting mission planning in uncertain external environments as shown in Fig. 7. In addition, by taking the collision-free property as an example, we conduct a micro property enforcement experiment, through which we can see that our enforcement method mitigates the collision effect under different swarming speed conditions and different numbers of UAVs (Fig. 9).

Table 1 Runtime enforcement overhead evaluation of the UAV mission properties P1–P4 developed on the robotflocksim simulation tool

Property	$\|Q\|$	$\|S\|$	Time.Task (s)	Time.Overhead (s)
P1	202	100	240	0.32
P2	202	100	600	0.01
P1+P3	302	112	240	1.05
P1+P4	302	101	240	0.12
P1+P3+P4	402	121	240	1.21

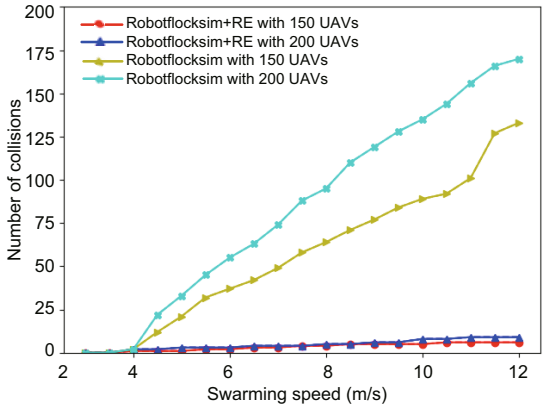

Fig. 9 Number of collisions under different swarming speed conditions and different numbers of UAVs

2. Enforcement from the experiments includes multiple properties, P1–P4. The monitoring and enforcing overhead on the Linux operating system is negligible because we send only part of the robotic data by calling the bus. An important part of our approach's impact on the original system is the communication overhead and the shield generation overhead on the swarm simulation platform. The overhead in our approach in experiments is shown as the results of Time.Overhead in Table 1. Obviously, property P3 has high overhead because of the continuous generation and the release of the enforcement shields in the process of enforcing the collision-free property. Even so, the correct task sequence of P1 can be guaranteed.

7 Conclusions and future work

In this study, we have proposed a runtime enforcement (RE) approach for robotic swarm systems, and discussed a formal specification to describe the macro-micro property and the enforcement information. To monitor the property using traditional semantics on infinite traces for temporal logic and to consider the robotic swarm characteristics, we have used the discrete-time metric temporal logic and proposed the $\mathcal{D}$-time enforcement mechanism for robotic swarm systems. To build the enforcers automatically, we have generated shields from macro monitors

for task sequence properties and the micro monitors for safety properties of a single agent. The corresponding enforcement shield construction algorithm has been illustrated.

We have implemented the above enforcement mechanism on the robotflocksim robotic simulation tool and demonstrated it with a moving swarm task from region A to region C. Furthermore, we have considered the safety property of being collision-free and having adequate battery. Experiments demonstrated that our approach can automatically generate shields at runtime. The experiments also showed that our approach is feasible and that the overhead is low.

In the future, we will consider the approach of anticipatory active monitoring (Dong et al., 2012) on robotic systems and generating shields without definite enforcement information. Moreover, the current RE method on robotics has not adopted a prediction ability to avoid property violations. We will introduce predictive semantics (Zhang X et al., 2012) for enforcement in robotic swarm systems.

Contributors

Chi HU, Wei DONG, and Yong-hui YANG designed the research. Chi HU, Hao SHI, and Fei DENG processed the data. Chi HU drafted the manuscript. Wei DONG helped organize the manuscript. Chi HU and Wei DONG revised and finalized the paper.

Compliance with ethics guidelines

Chi HU, Wei DONG, Yong-hui YANG, Hao SHI, and Fei DENG declare that they have no conflict of interest.

References

Bauer A, Falcone Y, 2016. Decentralised LTL monitoring. *Form Meth Syst Des*, 48(1-2):46-93. https://doi.org/10.1007/s10703-016-0253-8

Bloem R, Chatterjee K, Greimel K, et al., 2014. Synthesizing robust systems. *Acta Inform*, 51(3-4):193-220. https://doi.org/10.1007/s00236-013-0191-5

Brambilla M, Ferrante E, Birattari M, et al., 2013. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intell*, 7(1):1-41. https://doi.org/10.1007/s11721-012-0075-2

Chung SJ, Paranjape AA, Dames P, et al., 2018. A survey on aerial swarm robotics. *IEEE Trans Robot*, 34(4):837-855. https://doi.org/10.1109/TRO.2018.2857475

Dong W, Zhao CZ, Shu SX, et al., 2012. Anticipatory active monitoring for safety- and security-critical software. *Sci China Inform Sci*, 55(12):2723-2737. https://doi.org/10.1007/s11432-012-4739-8

Dreossi T, Fremont DJ, Ghosh S, et al., 2019. VERIFAI: a toolkit for the formal design and analysis of artificial intelligence-based systems. Proc 31st Int Conf on Computer Aided Verification, p.432-442. https://doi.org/10.1007/978-3-030-25540-4_25

Egerstedt M, Lee SG, Diaz-Mercado Y, et al., 2020. Control of Swarming Robots. US Patent 10 537 996.

Fine BT, Shell DA, 2013. Unifying microscopic flocking motion models for virtual, robotic, and biological flock members. *Auton Robot*, 35(2-3):195-219. https://doi.org/10.1007/s10514-013-9338-z

Floreano D, Wood RJ, 2015. Science, technology and the future of small autonomous drones. *Nature*, 521(7553):460-466. https://doi.org/10.1038/nature14542

Gökçe F, Şahin E, 2009. To flock or not to flock: the pros and cons of flocking in long-range "migration" of mobile robot swarms. Proc 8th Int Joint Conf on Autonomous Agents and Multiagent Systems, p.65-72.

Hu C, Dong W, Yang YH, et al., 2020. Runtime verification on hierarchical properties of ROS-based robot swarms. *IEEE Trans Reliab*, 69(2):674-689. https://doi.org/10.1109/TR.2019.2923681

Khosla P, Brown B, Paredis C, et al., 2002. Millibot Report. Report on Millibot Project, DARPA Contract DABT63-97-1-0003, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA.

Könighofer B, Alshiekh M, Bloem R, et al., 2017. Shield synthesis. *Form Meth Syst Des*, 51(2):332-361. https://doi.org/10.1007/s10703-017-0276-9

Ligatti J, Bauer L, Walker D, 2005. Edit automata: enforcement mechanisms for run-time security policies. *Int J Inform Sec*, 4(1-2):2-16. https://doi.org/10.1007/s10207-004-0046-8

Mondada F, Gambardella LM, Floreano D, et al., 2005. The cooperation of swarm-bots: physical interactions in collective robotics. *IEEE Robot Autom Mag*, 12(2):21-28. https://doi.org/10.1109/MRA.2005.1458313

Nagavalli S, Chien SY, Lewis M, et al., 2015. Bounds of neglect benevolence in input timing for human interaction with robotic swarms. Proc 10th Annual ACM/IEEE Int Conf on Human-Robot Interaction, p.197-204. https://doi.org/10.1145/2696454.2696470

Parr T, 2013. The Definitive ANTLR 4 Reference. Pragmatic Bookshelf, Dallas, USA, p.21-33.

Pinisetty S, Preoteasa V, Tripakis S, et al., 2017. Predictive runtime enforcement. *Form Meth Syst Des*, 51(1):154-199. https://doi.org/10.1007/s10703-017-0271-1

Pnueli A, 1977. The temporal logic of programs. Proc 18th Annual Symp Foundations of Computer Science, p.46-57. https://doi.org/10.1109/SFCS.1977.32

Pugh J, Martinoli A, 2006. Multi-robot learning with particle swarm optimization. Proc 5th Int joint Conf on Autonomous Agents and Multiagent Systems, p.441-448. https://doi.org/10.1145/1160633.1160715

Rajkumar R, Lee I, Sha L, et al., 2010. Cyber-physical systems: the next computing revolution. Proc 47th Design Automation Conf, p.731-736. https://doi.org/10.1145/1837274.1837461

Raju D, Bharadwaj S, Topcu U, 2019. Online synthesis for runtime enforcement of safety in multi-agent systems. https://arxiv.org/abs/1910.10380

Rasmussen S, Kingston D, Humphrey L, 2018. A brief introduction to unmanned systems autonomy services (UxAS). Int Conf on Unmanned Aircraft Systems, p.257-268. https://doi.org/10.1109/icuas.2018.8453287

Reinbacher T, Rozier KY, Schumann J, 2014. Temporal-logic based runtime observer pairs for system health management of real-time systems. Proc 20th Int Conf on Tools and Algorithms for the Construction and Analysis of Systems, p.357-372. https://doi.org/10.1007/978-3-642-54862-8_24

Reynolds CW, 1987. Flocks, herds and schools: a distributed behavioral model. *ACM SIGGRAPH Comput Graph*, 21(4):25-34. https://doi.org/10.1145/37401.37406

Şahin E, Girgin S, Bayindir L, et al., 2008. Swarm robotics. In: Blum C, Merkle D (Eds.), Swarm Intelligence. Natural Computing Series. Springer Berlin Heidelberg, p.87-100. https://doi.org/10.1007/978-3-540-74089-6_3

Schwager M, McLurkin J, Rus D, 2006. Distributed coverage control with sensory feedback for networked robots. Proc Robotics: Science and Systems, p.117-132. https://doi.org/10.15607/RSS.2006.II.007

Shi H, Dong W, Zhou G, et al., 2017. Monitor synthesis for parametric MTL properties in discrete control software. IEEE Int Conf on Software Quality, Reliability and Security Companion, p.355-362. https://doi.org/10.1109/QRS-C.2017.66

Sinhuber M, van der Vaart K, Ouellette NT, 2019. Response of insect swarms to dynamic illumination perturbations. *J Roy Soc Interf*, 16(150):20180739. https://doi.org/10.1098/rsif.2018.0739

Standley T, Korf R, 2011. Complete algorithms for cooperative pathfinding problems. Proc 22nd Int Joint Conf on Artificial Intelligence, p.668-673. https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-118

Turgut AE, Çelikkanat H, Gökçe F, et al., 2008. Self-organized flocking in mobile robot swarms. *Swarm Intell*, 2(2-4):97-120. https://doi.org/10.1007/s11721-008-0016-2

Vásárhelyi G, Virágh C, Somorjai G, et al., 2018. Optimized flocking of autonomous drones in confined environments. *Sci Robot*, 3(20):eaat3536. https://doi.org/10.1126/scirobotics.aat3536

Williams K, Burdick JW, 2006. Multi-robot boundary coverage with plan revision. IEEE Int Conf on Robotics and Automation, p.1716-1723. https://doi.org/10.1109/ROBOT.2006.1641954

Wong C, Yang EF, Yan XT, et al., 2017. An overview of robotics and autonomous systems for harsh environments. Proc 23rd Int Conf on Automation and Computing, p.1-6. https://doi.org/10.23919/IConAC.2017.8082020

Zhang X, Leucker M, Dong W, 2012. Runtime verification with predictive semantics. Proc 4th Int Symp on NASA Formal Methods, p.418-432. https://doi.org/10.1007/978-3-642-28891-3_37

Zhang Y, Kim K, Fainekos G, 2014. DisCoF: cooperative pathfinding in distributed systems with limited sensing and communication range. In: Chong NY, Cho YJ (Eds.), Distributed Autonomous Robotic Systems, p.325-340. https://doi.org/10.1007/978-4-431-55879-8_23