2019 American Control Conference (ACC)
Philadelphia, PA, USA, July 10-12, 2019

# Synthesis of Minimum-Cost Shields for Multi-agent Systems

Suda Bharadwaj[1], Roderik Bloem[2], Rayna Dimitrova[3], Bettina Könighofer[2], and Ufuk Topcu[1]

*Abstract*— **In this paper, we propose a general approach to derive runtime enforcement implementations for multi-agent systems, called shields, from temporal logical specifications. Each agent of the multi-agent system is monitored, and if needed corrected, by the shield, such that a global specification is always satisfied. The different ways of how a shield can interfere with each agent in the system in case of an error introduces the need for quantitative objectives. This work is the first to discuss the shield synthesis problem with quantitative objectives. We provide several cost functions that are utilized in the multi-agent setting and provide methods for the synthesis of cost-optimal shields and fair shields, under the given assumptions on the multi-agent system. We demonstrate the applicability of our approach via a detailed case study on UAV mission planning for warehouse logistics and simulating the shielded multi-agent system on ROS/Gazebo.**

## I. Motivation and challenges

The use of multi-agent systems such as teams of unmanned aerial vehicles (UAVs) has been predicted to grow significantly in many areas of our life. The military has highlighted the need for coordinating teams of UAVs for use in surveillance and reconnaissance operations in urban environments [25], [28]. In the private sector, drones are being used in applications from package delivery to inventory management in warehouses [21].

Since multi-agent systems exhibit complex interactions with their environment and between the individual agents, they are often difficult to understand, and are notoriously hard to design correctly [1]. Individual agents have to not only fulfill their *local objectives* and meet their *local requirements*, but also abide by system-wide or *global safety requirements* such as avoiding collision with other agents. Distributed reactive synthesis is able to automatically transform a given correcness specification and a given architecture describing the individual agents' interaction into a correct-by-construction implementation. Unfortunately, except for a few restricted classes of architectures, the distributed synthesis problem is undecidable. Even the decidable versions of the problem lack practical solutions due to their nonelementary complexity [26].

To address this problem, there has been a large body of work in designing algorithms to perform agent coordination and task assignment for a wide array of applications [5], [29]. For example, software frameworks such as UxAS [24] provide mission-level autonomy for multi-agent systems and include capabilities from high-level task assignment to path planning for unmanned systems. Such frameworks often allow for dynamic task reallocation as missions change, but in doing so, cannot necessarily account for potential violations of global safety specifications. This necessitates *shielding* the agents at runtime from a possible task assignment that can cause a violation of a global safety specification.

One approach in this direction is to perform *runtime verification* [4] that allows checking whether a run of a system satisfies a given specification. An extension of this idea is to perform *runtime enforcement* [8], [27] of the specified property, by not only detecting property violations, but also altering the behaviour of the system in a way that maintains the desired property.

*Shield synthesis* [16] is a general method to automatically derive runtime enforcement implementations, called shields, from temporal logical specifications. A shield is attached to a reactive system, monitors the behaviour of the system (i.e., its inputs and outputs), and corrects erroneous outputs instantaneously, but only if necessary and as infrequently as possible.

In this paper, we introduce *shield synthesis for multi-agent systems*. A shield monitors, and if needed corrects the output of one or more agents in the system, such that a given global safety specification is always satisfied. The distributed nature of the problem gives rise to a number of considerations to be made during the shield synthesis procedure. In order to explore the design space of possible shields for multi-agent systems, we categorize shields based on three criteria according to: (1) the interference of the shield processes with the individual agents, (2) the assumptions on the behaviour of the agents the shield can rely on, and (3) the fairness of the shield w.r.t. the individual agents.

*1. Quantifying interference.* By construction, a shield is guaranteed to enforce correct operation of the shielded system. However, we might prefer one shield over another, based on how much the shield interferes with the system as a whole, or how it interferes with the individual agents in case of an error. In this paper, we introduce the notion of *interference cost* in order to quantify the quality of a shield and synthesize cost-optimal shields that minimize the interference cost for the worst-case behavior of the multi-agent system. We discuss different cost functions and provide algorithms

[1]Suda Bharadwaj and Ufuk Topcu are with the University of Texas at Austin, USA
[2] Roderik Bloem and Bettina Könighofer are with the Graz University of Technology, AT
[3] Rayna Dimitrova is with the University of Leicester, UK

to synthesize cost-optimal shields.

*2. Assumptions on the multi-agent system.* The shield synthesis procedure does not rely on the particular implementation of the system or specifications of each of the agents, which is the key to the practicability of the approach. Instead, a shield has to guarantee safety for any possible implementation. However, it is often realistic to make assumptions on the worst-case behavior of the system and synthesize optimal shields w.r.t. the chosen interference cost under these assumptions. A natural assumption is that wrong outputs occur rarely, i.e., the length of all sequences of wrong outputs is bounded. When such knowledge is available, we compute a cost-optimal shield considering the worst-case behavior of any system satisfying the assumptions.

*3. Fair shielding.* In the multi-agent setting, in which each individual agent might have to fulfill some individual goals, it is often important that a shield treats all agents fairly: in case of an error, a fair shield does not always interfere with the same agent repeatedly. In this paper, we define a fairness notion for shields, and discuss the corresponding synthesis procedure.

**Contributions.** We summarise our contributions as follows. To the best of our knowledge, this work is the first (1) to consider the automatic construction of runtime reinforcement modules (we call them shields) for multi-agent systems, (2) to discuss synthesis of shields for multi-agent systems with quantitative objectives, and (3) to construct shields under different assumptions on the behavior of the system. We show the universality and potential of our approach on several examples.

**Outline.** The remainder of this paper is organized as follows. We discuss related work in Section II, present a motivating case study in Section III and establish notation in Section IV. We formalize the shield synthesis problem for multi-agent systems in Section V, and discuss the synthesis procedure for the different interference constraints in Section VI. We present our experimental results in Section VII and, finally, give our conclusions in Section VIII.

## II. Related Work

Monitoring distributed systems is an active area of research (see [12] for a survey). Significant effort has focused on providing efficient monitoring solutions by exploiting distribution [3], [10], [13]. While our work is not comparable to monitoring, exploring similar ideas for shielding is one avenue for future work.

In the context of enforcement, in case of an error the proposed solutions either stop system execution [27], or suppress, insert, or delay actions [9], [17]. The approach in [18] generates monitors that detect and prevent violations by one-step lookahead. Due to the limitation of one-step lookahead, this method can cause deadlocks. An alternative approach [22] circumvents this problem by using model checking to determine for each event whether it should be blocked, which is done online.

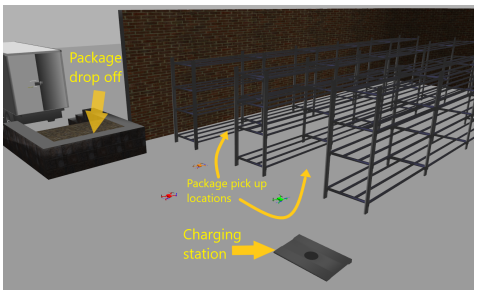



Fig. 1: Warehouse simulation environment in ROS.

In contrast, shields are synthesized offline, accounting for the effect of the shield on the future execution. None of these mentioned, nor previous works on shield synthesis [6], [16], considers quantitative specifications about how violations should be mitigated. In contrast, we synthesize shields with minimal interference cost, building on techniques for quantitative synthesis [14].

## III. Case Study

We present a case study used as a motivating example in this paper. Figure 1 is a snapshot of a warehouse environment where packages need to be moved from shelves to a loading area to be sent for delivery. UAVs also need to periodically visit a charging station.

The commands sent to move package(s) from the shelves to the loading dock are used to generate task assignments for the UAVs, and constitute the input to the system. The task assignment for the UAVs is sent either by a human operator who can command a particular UAV to perform a certain task, or by an automated system such as those in [15], [20]. To prevent congestion and collisions, global requirements for the multi-UAV system include not allowing more than one UAV in a given row at the same time, not allowing the UAVs to fly too close to each other, and not allowing UAVs to charge or drop of packages at the same time.

The methods proposed in this paper can be used to synthesize a shield that enforces such global safety properties and is agnostic to the nature of the system being protected. The shield takes the task assignment as an input and overwrites it with a new task assignment when necessary. This is then sent to a trajectory generator which generates a minimum snap trajectory for each UAV to accomplish its task. In the following sections, we present the formalization of the multi-agent shielding problem, motivated in this example.

The safety specifications for this case study, and the results of the shield synthesis procedure are described and discussed in more detail in Section VII-B.

## IV. Preliminaries

*1) Basic notations:* We consider reactive systems with a finite set $I$ ($O$) of Boolean *inputs (outputs)*. The input alphabet is $\Sigma_I = 2^I$, the output alphabet is $\Sigma_O = 2^O$, and $\Sigma = \Sigma_I \times \Sigma_O$. The set of finite (infinite) words over $\Sigma$ is denoted by $\Sigma^*$ ($\Sigma^\omega$), and we define $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. We

will also refer to words as *(execution) traces*. We write $|\overline{\sigma}|$ for the length of a trace $\overline{\sigma} \in \Sigma^*$. For an infinite trace $\overline{\sigma} \in \Sigma^\omega$ we define $|\overline{\sigma}| = \infty$. For $\overline{\sigma}_I = x_0 x_1 \ldots \in \Sigma_I^\infty$ and $\overline{\sigma}_O = y_0 y_1 \ldots \in \Sigma_O^\infty$, we write $\overline{\sigma}_I \parallel \overline{\sigma}_O$ for the composition $(x_0, y_0)(x_1, y_1) \ldots \in \Sigma^\infty$. For $i \in \mathbb{N}$ and a word $\overline{\sigma} = \sigma_0 \sigma_1 \ldots \in \Sigma^\infty$, we define $\overline{\sigma}[i] = \sigma_i$, and we define $\overline{\sigma}[i, j) = \sigma_i \sigma_{i+1} \ldots \sigma_{j-1}$ if $j \in \mathbb{N}$ and $\overline{\sigma}[i, j) = \sigma_i \sigma_{i+1} \ldots$ if $j = \infty$. A *language* is a set $L \subseteq \Sigma^\infty$ of words.

*2) Reactive systems:* Each agent in the multi-agent system, as well as the shield, is a *reactive system* which is defined by a 6-tuple $\mathcal{P} = (Q, q_0, \Sigma_I, \Sigma_O, \delta, \lambda)$, where $Q$ is a finite set of states, $q_0 \in Q$ is the initial state, $\Sigma_I$ is the input alphabet, $\Sigma_O$ is the output alphabet, $\delta : Q \times \Sigma_I \to Q$ is the complete transition function, and $\lambda : Q \times \Sigma_I \to \Sigma_O$ is the output function. Given an input trace $\overline{\sigma}_I = x_0 x_1 \ldots \in \Sigma_I^\infty$, a reactive system $\mathcal{P}$ produces an output trace $\overline{\sigma}_O = \mathcal{P}(\overline{\sigma}_I) = \lambda(q_0, x_0)\lambda(q_1, x_1) \ldots \in \Sigma_O^\infty$ with $q_{i+1} = \delta(q_i, x_i)$ for all $i \geq 0$. The set of words produced by $\mathcal{P}$ is denoted $L(\mathcal{P}) = \{\overline{\sigma}_I \parallel \overline{\sigma}_O \in \Sigma^\infty \mid \mathcal{P}(\overline{\sigma}_I) = \overline{\sigma}_O\}$.

*3) Multi-agent reactive systems:* A *multi-agent reactive system* $\mathcal{D}$ is a tuple $(\mathcal{P}, \Sigma_I, \Sigma_O)$, where $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ is a set of agents, where each $\mathcal{P}_i = (Q_i, q_{0,i}, \Sigma_{I,i}, \Sigma_{O,i}, \delta_i, \lambda_i)$ is a reactive system. While multiple agents may be able to read the same input variables to indicate broadcast from the environment, the sets of outputs are pairwise disjoint: for $i \neq j$, we have $O_i \cap O_j = \emptyset$. Furthermore, agents cannot directly read each others outputs, that is, for all $i$ and $j$, we have $O_i \cap I_j = \emptyset$. The outputs of the multiagent system $\mathcal{D}$ are $O = \bigcup_{i=1}^n O_i$, and its inputs are $I = \bigcup_{i=1}^n I_i$. The joint behaviour of the multi-agent system is a reactive system $\mathcal{D} = (Q, q_0, \Sigma_I, \Sigma_O, \delta, \lambda)$ defined as follows: the set $Q = \bigotimes_i Q_i$ of states is formed by the product of the states of all agents $\mathcal{P}_i \in \mathcal{P}$. The initial state $q_0$ is formed by the initial states $q_{0,i}$ of all $\mathcal{P}_i \in \mathcal{P}$. The transition function $\delta$ updates, for each agent $\mathcal{P}_i \in \mathcal{P}$, the $Q_i$ part of the state in accordance with the transition function $\delta_i$, using the projection $\sigma(I_i)$ as input. The output function $\lambda$ labels each state with the union of the outputs of all $\mathcal{P}_i \in \mathcal{P}$ according to $\lambda_i$.

*4) Specifications:* A *specification* $\varphi$ defines a set $L(\varphi) \subseteq \Sigma^\infty$ of allowed traces. A reactive system $\mathcal{D}$ *realizes* $\varphi$, denoted by $\mathcal{D} \models \varphi$, iff $L(\mathcal{D}) \subseteq L(\varphi)$. Given a set of propositions $\mathsf{AP}$, a formula in *linear temporal logic* (LTL) describes a language in $(2^{\mathsf{AP}})^\omega$. LTL extends Boolean logic by the introduction of temporal operators such as $\bigcirc$ (next time), $\Box$ (globally), $\Diamond$ (eventually), and $\mathcal{U}$ (until) [23]. $\varphi$ is called a *safety specification* if every trace $\overline{\sigma}$ that is not in $L(\varphi)$ has a prefix $\tau$ such that all words starting with $\tau$ are also not in the language $L(\varphi)$. We represent a safety specification $\varphi$ by a safety automaton $\varphi = (Q, q_0, \Sigma, \delta, F)$, where $F \subseteq Q$ is a set of safe states.

*5) Games:* A game is a tuple $\mathcal{G} = (G, g_0, \Sigma, \delta, \mathsf{Acc}, \mathsf{Val})$, where $G$ is a finite set of states, $g_0 \in G$ is the initial state, $\delta : G \times \Sigma \to G$ is a complete transition function, $\mathsf{Acc} : (G \times \Sigma \times G)^\omega \to \mathbb{B}$ is a winning condition and defines the qualitative objective of the game, and $\mathsf{Val} : (G \times \Sigma \times G)^\omega \to \mathbb{R} \cup \{-\infty, \infty\}$ is a value function defining the quantitative objective of the game. A game can have a winning condition, a value function, or both. The game is played by two players: the system and the environment. In every state $g \in G$ (starting with $g_0$), the environment chooses an input $\sigma_I \in \Sigma_I$, and then the system chooses some output $\sigma_O \in \Sigma_O$. These choices define the next state $g' = \delta(g, (\sigma_I, \sigma_O))$, and so on. The resulting (infinite) sequence $\overline{\pi} = (g_0, \sigma_I, \sigma_O, g_1)(g_1, \sigma_I, \sigma_O, g_2) \ldots$ is called a *play*. A deterministic *strategy* for the environment is a function $\rho_e : G^* \to \Sigma_I$. A nondeterministic (deterministic) *strategy* for the system is a relation $\rho_s : G^* \times \Sigma_I \to 2^{\Sigma_O}$ (function $\rho_s : G^* \times \Sigma_I \to \Sigma_O$).

A play $\overline{\pi}$ is *won* by the system iff $\mathsf{Acc}(\overline{\pi}) = \top$. A strategy is *winning* for the system if all plays $\overline{\pi}$ that can be constructed when defining the outputs using the strategy result in $\mathsf{Acc}(\overline{\pi}) = \top$. The *winning region* $\mathsf{Win}$ is the set of states from which a winning strategy exists. A *permissive* winning strategy $\rho_s : G^* \times \Sigma_I \to 2^{\Sigma_O}$ is a strategy that is not only winning for the system, but also contains all deterministic winning strategies.

A *safety game* defines $\mathsf{Acc}$ via a set $F \subseteq G$ of safe states: $\mathsf{Acc}(\overline{\pi}) = \top$ iff $g_i \in F$ for all $i \geq 0$, i.e., if only safe states are visited in the play $\overline{\pi}$. Otherwise, $\mathsf{Acc}(\overline{\pi}) = \bot$. The *Büchi* winning condition is $\mathsf{Acc}(\overline{\pi}) = \top$ iff $\inf(\overline{\pi}) \cap F \neq \emptyset$, where $F \subseteq G$ is the set of accepting states and $\inf(\overline{\pi})$ is the set of states that occur infinitely often in $\overline{\pi}$. We abbreviate the Büchi condition as $\mathcal{B}(F)$. A *Generalized Reactivity 1* (GR(1)) acceptance condition is a predicate $\bigwedge_{i=1}^m \mathcal{B}(E_i) \to \bigwedge_{i=1}^n \mathcal{B}(F_i)$, with $E_i \subseteq G$ and $F_i \subseteq G$. A *Streett* acceptance condition is $\bigwedge_{i=1}^k \mathcal{B}(E_i) \to \mathcal{B}(F_i)$.

The quantitative objective of the system is to minimize $\mathsf{Val}(\overline{\pi})$, while the environment tries to maximize it.

*6) Properties of traces:* A finite trace $\overline{\sigma} \in \Sigma^*$ is *wrong* w.r.t. a specification $\varphi$, if the corresponding play cannot be won, i.e., if there is no way for the system to guarantee that any extension of $\overline{\sigma}$ satisfies $\varphi$. An output $\sigma_O$ is called *wrong* for a trace $\overline{\sigma}$ and input $\sigma_I$, if it makes the trace wrong, i.e. when $\overline{\sigma}$ is not wrong, but $\overline{\sigma} \cdot (\sigma_I, \sigma_O)$ is. Given a sequence $(\overline{\sigma}_I \parallel \overline{\sigma}_O \parallel \overline{\sigma}'_O) \in (\Sigma_I \times \Sigma_O \times \Sigma_O)^\infty$, we denote with $\mathsf{WIdx}(\overline{\sigma}_I \parallel \overline{\sigma}_O \parallel \overline{\sigma}'_O)$ the positions of occurrences of wrong outputs in $\overline{\sigma}_O$. Formally, $i \in \mathsf{WIdx}(\overline{\sigma}_I \parallel \overline{\sigma}_O \parallel \overline{\sigma}'_O)$ iff $\overline{\sigma}_O[i]$ is wrong for the trace $(\overline{\sigma}_I[0, i) \parallel \overline{\sigma}'_O[0, i))$ and the input $\overline{\sigma}_I[i]$.

We denote with $Agts = \{1, \ldots, n\}$ the set of agent ids of a multi-agent system $\mathcal{D}$. For a set $\Pi \subseteq Agts$, we define $O_\Pi = \bigcup_{i \in \Pi} O_i$ and $\Sigma_{O_\Pi} = 2^{O_\Pi}$. For $\sigma_O \in \Sigma_O$ and $i \in Agts$, we denote with $\sigma_O(O_i)$ the projection of $\sigma_O$ on $O_i$. For $\Pi \subseteq Agts$, we define $\sigma_O(O_\Pi)$ similarly.

For $\sigma_O, \sigma'_O \in \Sigma_O$, the set $Diff(\sigma_O, \sigma'_O) = \{i \in Agts \mid \sigma_O(O_i) \neq \sigma'_O(O_i)\}$ gives the set of agents whose outputs in $\sigma_O$ differ from those in $\sigma'_O$. Let $(\overline{\sigma}_O \parallel \overline{\sigma}'_O) \in (\Sigma_O \times \Sigma_O)^\infty$ be a sequence of output pairs. We call $(\overline{\sigma}_O \parallel \overline{\sigma}'_O)$ a *deviation period* if (1) $\overline{\sigma}_O[i] \neq \overline{\sigma}'_O[i]$ for every $i < |\overline{\sigma}_O|$ and (2) if $|\overline{\sigma}_O| < \infty$, then $\overline{\sigma}_O[|\overline{\sigma}_O|] = \overline{\sigma}'_O[|\overline{\sigma}_O|]$. Thus, a deviation period is either a finite sequence $(\overline{\sigma}_O \parallel \overline{\sigma}'_O)$ consisting of differing outputs followed by a last letter where the two outputs agree, or an infinite sequence

$(\overline{\sigma}_O \parallel \overline{\sigma}'_O)$ where the outputs always differ.

## V. Shields for Multi-Agent Systems

In this section, we first describe how to attach a shield to a multi-agent system. Then, we define shields formally, and discuss different interference requirements on shields for multi-agent systems.

### A. Attaching the Shield

A shield $\mathcal{S} = (Q, q_0, \Sigma_I \times \Sigma_O, \Sigma_O, \delta, \lambda)$ is a reactive system that is attached to a multi-agent system $\mathcal{D} = (\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, \Sigma_I, \Sigma_O)$. Since $\mathcal{S}$ has to enforce a *global* specification that can refer to all inputs and outputs of $\mathcal{D}$, $\mathcal{S}$ is attached to the whole multi-agent system $\mathcal{D}$: the shield $\mathcal{S}$ monitors the outputs of all the agents and corrects them if necessary. Thus, $\mathcal{S}$ is attached to $\mathcal{D}$ using serial composition, by feeding all inputs and outputs of the multi-agent system to the shield, which, in response produces a possibly corrected output for the system. Formally, given a multi-agent reactive system $\mathcal{D} = (\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, \Sigma_I, \Sigma_O)$, with $\mathcal{P}_i = (Q_i, q_{0,i}, \Sigma_{I,i}, \Sigma_{O,i}, \delta_i, \lambda_i)$, the *serial composition* of $\mathcal{D}$ and $\mathcal{S}$ is a reactive system $\mathcal{D} \circ \mathcal{S} = (\hat{Q}, \hat{q_0}, \Sigma_I, \Sigma_O, \hat{\delta}, \hat{\lambda})$, with states $\hat{Q} = \bigotimes_i Q_i \times Q$, $\hat{q_0} = (q_{0,1}, \ldots, q_{0,n}, q_0)$, transition function $\hat{\delta}((q_1, \ldots, q_n, q), \sigma_I) = (\delta_1(q_1, \sigma_{I1}), \ldots, \delta_n(q_n, \sigma_{In}), \delta(q, (\sigma_I, \sigma_O))$, where $\sigma_O$ is the output of all agents $\sigma_O = (\lambda_1(q_1, \sigma_{I1}), \ldots, \lambda_n(q_n, \sigma_{In}))$, and output function $\hat{\lambda}((q_1, \ldots, q_n, q), \sigma_I) = \lambda(q, (\sigma_I, \sigma_O))$.

### B. Shield Definition

Now we define the basic requirements that a shield must satisfy: namely it should enforce correctness without deviating from the system's output unnecessarily.

*1) Correctness:* We say that a reactive system $\mathcal{S} = (Q, q_0, \Sigma_I \times \Sigma_O, \Sigma_O, \delta, \lambda)$ *ensures correctness* with respect to a safety specification $\varphi$ if for any multi-agent system $\mathcal{D} = (\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, \Sigma_I, \Sigma_O)$ it holds that $(\mathcal{D} \circ \mathcal{S}) \models \varphi$.

*2) No unnecessary interference:* A shield is only allowed to interfere when the output of the multi-agent system is *wrong*. Formally, given a safety specification $\varphi$, a reactive system $\mathcal{S} = (Q, q_0, \Sigma_I \times \Sigma_O, \Sigma_O, \delta, \lambda)$ *does not interfere unnecessarily* if for any multi-agent system $\mathcal{D} = (\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, \Sigma_I, \Sigma_O)$ and any trace $(\overline{\sigma}_I \parallel \overline{\sigma}_O) \in (\Sigma_I \times \Sigma_O)^\infty$ of $\mathcal{D}$ that is not wrong, we have that $\mathcal{S}(\overline{\sigma}_I \parallel \overline{\sigma}_O) = \overline{\sigma}_O$.

*Definition 1:* A *shield* for a given safety specification $\varphi$ is a reactive system $\mathcal{S} = (Q, q_0, \Sigma_I \times \Sigma_O, \Sigma_O, \delta, \lambda)$ such that for any multi-agent system $\mathcal{D} = (\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}, \Sigma_I, \Sigma_O)$ it holds that $(\mathcal{D} \circ \mathcal{S}) \models \varphi$ and $\mathcal{S}$ does not deviate from $\mathcal{D}$ unnecessarily.

### C. Interference Costs and Shield Optimality

In Section V-B we defined the qualitative requirements that a shield must meet. In many applications, there are additional quantitative properties a shield should optimize, expressing *in what way* it should interfere with the system. Now we define several different interference cost functions and optimization objective.

*1) Interference cost functions:* We formalize quantitative requirements on shields by introducing the notion of an *interference cost function*, which quantifies the deviation of the shield's output from the system's output.

*Definition 2:* An *interference cost function* $c : \Sigma_O \times \Sigma_O \to \mathbb{N}$ assigns a cost $c(\sigma_O, \sigma'_O)$ to each pair of system output $\sigma_O \in \Sigma_O$ and shield output $\sigma'_O \in \Sigma_O$, such that

$$c(\sigma_O, \sigma'_O) = \begin{cases} 0 & \text{if } \sigma_O = \sigma'_O \\ f(\sigma_O, \sigma'_O) > 0 & \text{if } \sigma_O \neq \sigma'_O, \end{cases}$$

where $f$ is a function chosen by the system designer. The higher the cost, the more undesirable is the corresponding way of interference by the shield. Thus, the designer can assign different costs to interference with different agents, expressing preference of one over another. We propose two concrete cost functions.

The *boolean cost function* $c_{\mathbb{B}} : \Sigma_O \times \Sigma_O \to \{0, 1\}$ considers the multi-agent system as a monolithic system:

$$c_{\mathbb{B}}(\sigma_O, \sigma'_O) = \begin{cases} 0 & \text{if } \sigma_O = \sigma'_O \\ 1 & \text{if } \sigma_O \neq \sigma'_O. \end{cases}$$

Thus, the cost for any interference is 1, no matter with how many or with which agents the shield interferes.

The *counting cost function* $c_{\#} : \Sigma_O \times \Sigma_O \to \mathbb{N}$, on the other hand, takes into account the number of agents whose output is modified:

$$c_{\#}(\sigma_O, \sigma'_O) = |\{p \in Agts \mid \sigma_O(O_p) \neq \sigma'_O(O_p)\}|.$$

Intuitively, the smaller the number of agents whose output is altered by the shield, the better.

*Definition 3:* Let $c : \Sigma_O \times \Sigma_O \to \mathbb{N}$ be a cost function. We define the *accumulated interference cost function* $c_{\mathsf{acc}} : \Sigma_O^\infty \times \Sigma_O^\infty \to \mathbb{N}$ based on the given cost function $c$ by

$$c_{\mathsf{acc}}(\overline{\sigma}_O, \overline{\sigma}'_O) = \sum_{i=0}^{|\overline{\sigma}_O|} c(\overline{\sigma}_O[i], \overline{\sigma}'_O[i]).$$

*2) Shield optimization objective:* In this work we consider the cost-optimization objective for infinite traces that requires minimizing the cost per deviation period.

In this objective, the task of the shield is to minimize the worst-case accumulated cost for ending the deviation period. To formalize this, we first define the set of maximal deviation periods resulting from the execution of a shield $\mathcal{S}$ under all possible behaviours of multi-agent system and the environment.

*Definition 4:* Let $\mathcal{S}$ be a shield for a safety specification $\varphi$, and let $\overline{\sigma} \in \Sigma^*$ be a finite trace. We define $Dev(\overline{\sigma}, \mathcal{S})$ to be the set of all deviation periods that extend the trace $(\overline{\sigma} \parallel \mathcal{S}(\overline{\sigma}))$, and result from outputs of $\mathcal{S}$. Formally, if $(\overline{\sigma}_O \parallel \overline{\sigma}'_O) \in (\Sigma_O \times \Sigma_O)^\infty$, then $(\overline{\sigma}_O \parallel \overline{\sigma}'_O) \in Dev(\overline{\sigma}, \mathcal{S})$ iff it satisfies the following conditions:

- $(\overline{\sigma}_O \parallel \overline{\sigma}'_O)$ is a deviation period, and
- there exists $\overline{\sigma}_I \in \Sigma_I^\infty$ such that $\overline{\sigma}'_O = \mathcal{S}(\overline{\sigma}_I \parallel \overline{\sigma}_O)$ and the trace $(\overline{\sigma} \parallel \mathcal{S}(\overline{\sigma}))$ is a prefix of $(\overline{\sigma}_I \parallel \overline{\sigma}_O \parallel \overline{\sigma}'_O)$.

The set $Dev(\overline{\sigma}, \mathcal{S})$ consists of all finite or infinite deviation periods resulting from possible future behaviours of the multi-agent system and the environment. Our goal is to synthesize a shield with minimal worst-case accumulated cost over the traces in the corresponding *Dev-* sets, as formalized in the next definition.

*Definition 5:* Let $c : \Sigma_O \times \Sigma_O \to \mathbb{N}$ be a cost function. A shield $\mathcal{S}$ is *locally optimal* w.r.t. $c$, if for every shield $\mathcal{S}'$ and every trace $\overline{\sigma} = (\overline{\sigma}_I \parallel \overline{\sigma}_O) \in \Sigma^*$ it holds that

$$\sup_{(\overline{\sigma}_O \| \overline{\sigma}'_O) \in Dev(\overline{\sigma}, \mathcal{S})} c_{\mathsf{acc}}(\overline{\sigma}_O, \overline{\sigma}'_O) \leq \sup_{(\overline{\sigma}_O \| \overline{\sigma}'_O) \in Dev(\overline{\sigma}, \mathcal{S}')} c_{\mathsf{acc}}(\overline{\sigma}_O, \overline{\sigma}'_O).$$

With this objective, a shield plans optimally in the short term, without considering the possibility of further wrong outputs once the deviation period ends. Therefore, this optimality criterion is useful when wrong outputs of the system are expected to be rare.

### D. Assumptions on the Occurrences of Faults

A shield has to enforce the global safety specification for any possible implementation of the multi-agent system. However, often we have some knowledge about the the system which can be used to make assumptions about its worst-case behavior and synthesize cost-optimal shields under these assumptions.

Pervious work on shield synthesis [6], [16] assumed that the system $\mathcal{D}$ is correct, but that through external faults an arbitrary number of correct outputs are replaced by wrong ones. To make use of this optimistic assumption, Bloem et al. [6] used a subset construction: when $\mathcal{D}$ emits a wrong output, the shield begins to track all possible correct behaviours of $\mathcal{D}$ under the current input, with the rationale that $\mathcal{D}$ meant to give a correct output, but which one is unknown.

In this paper, we take a new view, in which the system is supposed to have a real safety bug, manifested by the interaction between the agents. This case is especially important in the multi-agent setting. When developing multi-agent systems, the individual agents are often implemented separately, with individual goals in mind, and then combined to obtain the final system. Since each agent has to satisfy its own objective, the design of a single agent often neglects some global safety requirements. However, it is often realistic to assume that the length of all sequences of wrong outputs is bounded. This assumption can be made e.g., if the agents interact only rarely, and when they do, they interact only for a bounded period of time.

We define an assumption $\mathsf{Assumption} \subseteq (\Sigma \times \Sigma_O \times \Sigma_O)^\infty$ on the occurrences of system faults to be a set of *allowed finite traces*, represented as a finite automaton.

In particular, we define $\mathsf{Assumption}(b)$ to be the set of traces in which the length of each sequence of wrong outputs does not exceed a given bound $b$.

*Definition 6:* Let $b \in \mathbb{N}$. The set $\mathsf{Assumption}(b) \subseteq (\Sigma \times \Sigma_O \times \Sigma_O)^\infty$ consists of all traces $\overline{\sigma}$ for which all sub-traces $\overline{\sigma}' = \overline{\sigma}[i, j]$ it holds that if $|\overline{\sigma}'| > b$, then there exists an index $k$ with $i \leq k \leq j$, such that $k \notin \mathsf{Wldx}(\overline{\sigma})$.

We incorporate assumptions on the system in the definition of cost-optimal shields. We modify Definition 5 (locally-optimal shields) by restricting the sets of deviation periods to those corresponding to traces in $\mathsf{Assumption} \subseteq (\Sigma \times \Sigma_O \times \Sigma_O)^\infty$. More precisely, in the supremum for $\mathcal{S}$ we replace $Dev(\overline{\sigma}, \mathcal{S})$ by $Dev(\overline{\sigma}, \mathcal{S}) \cap \{(\overline{\sigma}_O \parallel \overline{\sigma}'_O) \mid \exists\ \overline{\sigma}_I : (\overline{\sigma}_I \parallel \overline{\sigma}_O \parallel \overline{\sigma}'_O) \in \mathsf{Assumption}\}$ and analogously for $\mathcal{S}'$.

### E. Fair Shielding

Now we define one more constraint on the shield's interference: we require that a shield is *fair* with respect to the different agents. The definition of fair shields uses the notion of *minimal correcting sets*: a minimal correcting set for a wrong trace is the minimal set of agents such that modifying some of the outputs of these agents results in correct output.

Let $\overline{\sigma} = (\overline{\sigma}_I \parallel \overline{\sigma}_O) \cdot (\sigma_I, \sigma_O) \in (\Sigma_I \times \Sigma_O)^*$ be a wrong trace. A set $\Pi \subseteq Agts$ of agents is a *minimal correcting set* for $\overline{\sigma}$ if and only if the following conditions are satisfied:

- there exists $\sigma'_O \in \Sigma_O$ such that $Diff(\sigma_O, \sigma'_O) = \Pi$ and the trace $(\overline{\sigma}_I \parallel \overline{\sigma}_O) \cdot (\sigma_I, \sigma'_O)$ is not wrong,
- for all $\sigma'_O \in \Sigma_O$ such that $Diff(\sigma_O, \sigma'_O) \subsetneq \Pi$, it holds that the trace $(\overline{\sigma}_I \parallel \overline{\sigma}_O) \cdot (\sigma_I, \sigma'_O)$ is wrong.

$Mcs(\overline{\sigma})$ is the set of all minimal correcting sets for $\overline{\sigma}$.

In order to be fair, a shield should guarantee that each agent is treated fairly when choosing agents whose outputs should be modified. More precisely, it should not choose the same agent all the time when it is possible to alternatively choose a different agent.

First, given a trace $\overline{\sigma} = (\overline{\sigma}_I \parallel \overline{\sigma}_O \parallel \overline{\sigma}'_O) \in (\Sigma_I \times \Sigma_O \times \Sigma_O)^\infty$ and an agent $p$, we define $\mathsf{Alt}(\overline{\sigma}, p)$ to be the set of positions in $\overline{\sigma}$ where there exist both a minimal correcting set containing $p$ and a minimal correcting set that does not contain $p$. Formally, $i \in \mathsf{Alt}(\overline{\sigma}, p)$ if and only if $i \in \mathsf{Wldx}(\overline{\sigma})$ and there exist two minimally correcting sets $\Pi, \overline{\Pi} \in Mcs((\overline{\sigma}_I[0, i-1] \parallel \overline{\sigma}'_O[0, i-1]) \cdot (\overline{\sigma}_I[i], \overline{\sigma}_O[i]))$ at position $i$ such that $p \in \Pi$ and $p \notin \overline{\Pi}$. Now, using the sets $\mathsf{Alt}(\overline{\sigma}, p)$ we formally define fair shields as follows.

*Definition 7:* A shield $\mathcal{S}$ is *fair* if for every agent $p \in Agts$ and trace $\overline{\sigma} = (\overline{\sigma}_I \parallel \overline{\sigma}_O \parallel \overline{\sigma}'_O) \in L(\mathcal{S})$ it holds that, if the set $\mathsf{Alt}(\overline{\sigma}, p)$ is infinite, then there exist infinitely many indices $i \in \mathsf{Alt}(\overline{\sigma}, p)$ in which $p \notin Diff(\overline{\sigma}_O[i], \overline{\sigma}'_O[i])$ (i.e., the output of $p$ is not altered by $\mathcal{S}$ in step $i$).

## VI. Synthesis of Shields for Multi-Agent Systems

In this section, we study the synthesis of shields for multi-agent systems with respect to the interference requirements defined in Section V. First, we discuss how the knowledge that all sequences of wrong outputs are bounded can be incorporated in the synthesis approach. Then, we propose a synthesis approach to construct *locally-optimal* shields. The synthesis procedure consists of the following three steps:

1) We construct a safety game $\mathcal{G}^s$ and compute its permissive winning strategy $\rho^s$, such that any shield $\mathcal{S}$

that implements $\rho^s$ ensures *correctness* ($\mathcal{D} \circ \mathcal{S} \models \varphi$) and $\mathcal{S}$ *does not interfere with* $\mathcal{D}$ *unnecessarily*. This construction is similar to the one in [16].

2) We augment the game graph with the assumptions on the occurrence of system errors.
3) We compute deterministic strategy that implements $\rho^s$ and satisfies the interference constraints.

We also present a synthesis algorithm for *fair* shields. The first step above is common for both algorithms, the other two depend on the interference requirements.

### A. Constructing and Solving the Safety Game

Let $\varphi$ be a safety specification represented as a safety automaton $\varphi = (Q, q_0, \Sigma, \delta, F)$. Let $W \subseteq F$ be the winning region of $\varphi$ when considered as a safety game.

We construct a safety game $\mathcal{G}^s$ such that its most permissive strategy subsumes all possible shields that are correct w.r.t. $\varphi$ and that do not interfere unnecessarily. The state space of $\mathcal{G}^s$ is constructed by augmenting the states $Q$ of $\varphi$ with two Boolean variables: (1) the variable $u$ is a flag that indicates wrong outputs by the system, and (2) the variable $t$ tracks deviations between the outputs of the system and the shield.

We construct a safety game $\mathcal{G}^s = (G^s, g_0^s, \Sigma, \Sigma_O, \delta^s, F^s)$ such that $G^s = \{(g, u, t) \mid g \in Q, u, t \in \{\top, \bot\}\}$ is the state space, $g_0^s = (g_0, \bot, \bot)$ is the initial state, $\delta^s$ is the next-state function, and $F^s$ is the set of safe states, such that $\delta^s((g, u, t), (\sigma_I, \sigma_O), \sigma'_O) = (\delta(g, \sigma_I, \sigma'_O), u', t')$ with

(1) $u' = \top$ if $\delta(g, \sigma_I, \sigma_O) \notin W$, and $u' = \bot$ otherwise,
(2) $t' = \top$ if $\sigma_O \neq \sigma'_O$, and $t' = \bot$ otherwise;

and $F^s = \{(g, u, t) \in G^s \mid (g \in W) \wedge (u = \bot \rightarrow t = \bot)\}$. We use standard algorithms for safety games (e.g. [19]) to compute the winning region $W^s$ and the most permissive winning strategy $\rho^s : G \times \Sigma_I \rightarrow 2^{\Sigma_O}$ of $\mathcal{G}^s$.

### B. Synthesis with Assumptions on the Occurrences of Faults

Our goal is to synthesize cost-optimal shields under the assumption that the length of sequences of wrong outputs is bounded by some constant $b$. Therefore, we construct a new game graph, that incorporates this knowledge and that can be used to synthesize cost-optimal shields in the next subsections. We start from the safety game $\mathcal{G}^s = (G^s, g_0^s, \Sigma, \Sigma_O, \delta^s, F^s)$ with winning region $W^s$ and permissive winning strategy $\rho^s$, and a bound $b \in \mathbb{N}$ on the maximal length of sequences of wrong outputs. We construct a new game $\mathcal{G}^a = (G^a, g_0^a, \Sigma, \Sigma_O, \delta^a, \mathsf{Acc}^a)$ where $G^a = W^s \times \{0, \ldots, b+1\}$ is the set of states, $g_0^a = (g_0^s, 0)$ is the initial state, $\delta^a$ is the next-state function, such that $\delta^a((g^s, v), \sigma, \sigma'_O) = (\delta^s(g^s, \sigma, \sigma'_O) \cap \rho^s(g^s, \sigma), v')$ such that

- if $v \leq b$ and $u' = \top$, then $v' = v + 1$,
- if $v \leq b$ and $u' = \bot$, then $v' = 0$, and
- if $v = b + 1$, then $v' = b + 1$;

and $\mathsf{Acc}^a$ is such that $\mathsf{Acc}^a(\overline{\pi}) = \top$ iff

- $\exists i \geq 0 \,.\, g_i^a = (g_i^s, v_i)$ with $v_i = b + 1$, or
- there are inf. many $g_i^a = (g_i, u_i, t_i, v_i)$ with $t_i = \bot$.

Intuitively, the counter $v$ tracks the length of the current sequence of wrong outputs by the system, and is reset to 0 when the output of the system is correct. If $v$ exceeds the bound $b$, it remains $b+1$ forever. Using this counter $v$, we encode the assumption on the system. Thus, the set $\mathsf{Acc}^a$ of wining plays in $\mathcal{G}^a$ consists of all infinite plays that violate the assumption plus the infinite plays that visit infinitely often a state in which the output of the shield does not deviate from the system's output. Hence, $\mathsf{Acc}^a$ is a GR(1) condition.

### C. Synthesis of Locally-Optimal Shields

Next, we propose a procedure to synthesize shields that minimize the cost per deviation period assuming that all sequences of wrong outputs are bounded.

We start with the augmented game graph $\mathcal{G}^a = (G^a, g_0^a, \Sigma, \Sigma_O, \delta^a, \mathsf{Acc}^a)$ and construct a new game $\mathcal{G}^{opt} = (G^a, g_0^a, \Sigma, \Sigma_O, \delta^a, \mathsf{Acc}^a, \mathsf{Val}^{opt})$ with value function $\mathsf{Val}^{opt}(\overline{\pi})$ which is an *accumulated cost objective* using $c$ as edge labeling: $cost^{opt}(g^a, (\sigma_I, \sigma_O), \sigma'_O) = c(\sigma_O, \sigma'_O)$.

Using the procedure described in [14], we synthesize shields, that are winning according to $\mathsf{Acc}^a$ (i.e., either the assumption on the system that any sequence of wrong outputs has a length of at most $b$ is violated, or infinitely often the shield does not interfere) and optimize $\mathsf{Val}^{opt}$ (i.e., the worst-case accumulated cost for reaching the end of the deviation per deviation period).

### D. Synthesis of Fair Shields

We now turn to the synthesis of fair shields. For this, we augment the states of $\mathcal{G}^s$ with Boolean variables that track information about the minimal correcting sets for each transition. We use these flags to encode the fairness requirements for the agents.

Let $Mcs(g, \sigma_I, \sigma_O, W)$ be the set of all minimal sets of agents such that correcting the output of these agents results in a successor state of $g$ that is in $W$. Formally, for $\Pi \subseteq Agts$ it holds that $\Pi \in Mcs(g, \sigma_I, \sigma_O, W)$ iff (1) there exists $\sigma'_O$ such that $\delta(g, \sigma_I, \sigma'_O) \in W$ and $Diff(\sigma_O, \sigma'_O) = \Pi$, and (2) $\Pi$ is minimal (i.e., for all $\sigma'_O \in \Sigma_O$ with $Diff(\sigma_O, \sigma'_O) \subsetneq \Pi$ it holds that $\delta(g, \sigma_I, \sigma'_O) \notin W$).

Given $\mathcal{G}^s = (G^s, g_0^s, \Sigma, \Sigma_O, \delta^s, F^s)$ with $W^s$ and $\rho^s$, we construct a game $\mathcal{G}^f = (G^f, g_0^f, \Sigma, \Sigma_O, \delta^f, \mathsf{Acc}^f)$ with set of states $G^f = W^s \times \{\bot, \top\}^n \times \{\bot, \top\}^n$, where $n = |Agts|$ is the number of agents. The initial state is $g_0^f = (g_0^s, \bot^n, \bot^n)$, and the transition relation $\delta^f$ is such that $\delta^a((g, u, t, m_1, \ldots, m_n, c_1, \ldots, c_n), \sigma, \sigma'_O) = (\delta^s((g, u, t), \sigma, \sigma'_O) \cap \rho^s((g, u, t), \sigma), m'_1, \ldots, m'_n, c'_1, \ldots, c'_n)$ where for each agent $p \in Agts$ it holds that

- $m'_p = \top$ iff there exist minimal correcting sets $\Pi, \overline{\Pi} \in Mcs(g, \sigma_I, \sigma_O, W)$ with $p \in \Pi$ and $p \notin \overline{\Pi}$,
- $c'_p = \top$ iff $p \in Diff(\sigma_O, \sigma'_O)$.

Intuitively, for each agent $p \in Agts$, the flag $m_p$ is set to $\top$ whenever the set of minimally correcting sets at that step contains a minimally correcting set which does not

contain $p$, and one that does. The flag $c_p$ is set to $\top$ whenever the output of $p$ is corrected by the shield.

The acceptance condition $\mathsf{Acc}^f$ encodes the fairness requirement on the shield for agent $p$ using the variables $m_p$ and $c_p$. It states for each agent $p \in Agts$ that if a play contains infinitely many occurrences of states in which $m_p = \top$, then it should contain infinitely many occurrences of states in which $c_p = \bot$ and $m_p = \top$.

Thus, $\mathsf{Acc}^f$ is a Streett acceptance condition, and a fair shield can be synthesized by solving a Streett game using well-known methods [2].

## VII. Experimental Evaluation

Now we describe the results of applying our shield synthesis method to several examples. We use the reactive synthesis tool `Slugs` [7] to compute locally-optimal shields under the assumption, that sequences of wrong outputs are bounded by a constant $b$, as discussed in Section VI-C. All experiments were performed on an Intel i5-5300U 2.30 GHz CPU with 8 GB of RAM.

### A. Gridworld

In the first set of experiments, we consider a grid-world with two agents that can move in one of the four cardinal directions at each time step. One grid cell is designated as a charging station. We require global safety property $\varphi = \varphi_{collision} \wedge \varphi_{charge}$, where $\varphi_{collision}$ requires collision avoidance and no simultaneous charging, and $\varphi_{charge}$ describes when and how the agents should use the charging station. The formula $\varphi_{charge}$ is of the form $\varphi_{charge,1} \wedge \varphi_{charge,2}$. These properties are specified in LTL as follows:

$$\begin{aligned} \varphi_{charge,i} &= \quad \square\big(charging_i \rightarrow \neg \bigcirc charging_j\big) \\ \varphi_{collision} &= \quad \square\neg\big(position_i = position_j\big) \end{aligned} \tag{1}$$

for $i, j \in \{1, 2\}$ and $i \neq j$. The formula $\varphi_{charge,i}$ requires that one agent cannot enter the charging area right after the other one has left. The integer variable $position_i$ is the position of agent $i$ in the grid. $\varphi_{collision}$ requires that the agents do not occupy the same position.

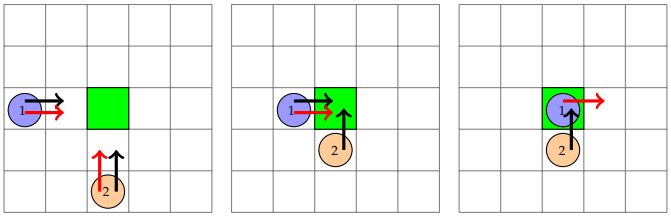
We synthesized locally-optimal shields using an interference cost function $c$ that assigns higher costs for any interferences with the first agent than for interferences with the second one. We consider different values for the bound $b$ on the length of sequences of wrong outputs. A larger bound $b$ results in more robust shields, but also in larger state-spaces of the game, due to the size of the counter that augments the state space.

We report the results in Table I under case (1). The first column gives the bound $b$, the second and the third column state the number of input and output bits of $\varphi$. The fourth column states the total number of variables of the constructed game (including the variables that augment the state space) and the fifth column gives the number of reachable states in the game. In the last two columns, we report the synthesis time (in seconds) to construct the permissive strategy and the locally-optimal strategy.

TABLE I: Results

| Case | $b$ | Inp vars | Out vars | Game vars | Game states | time perm | time cost-opt |
|---|---|---|---|---|---|---|---|
| **(1)** | 2 | 16 | 6 | 28 | 887 | 245 | 67 |
| | 5 | 16 | 6 | 30 | 3456 | 245 | 830 |
| **(2)** | 2 | 24 | 12 | 68 | 41544 | 3545 | 3019 |
| | 5 | 24 | 12 | 70 | $7.5 \times 10^5$ | 3545 | 9822 |

An example for the interference of the shield with the agents during an execution is shown in Figure 2




(a) $t_1$ (b) $t_2$ (c) $t_3$

Fig. 2: Green cell is the charging station. Black arrows correspond to intended actions (outputs from the system) and red arrows correspond to shield outputs. The absence of an arrow indicates that the action chosen was to stay at the same cell. In 2a, no interference is needed. In the next time step in 2b, the shield interferes with agent 2 to prevent collision in the charging area. In 2c, the shield forces agent 1 out of charging area, and agent 2 to wait one time-step as agent 2 is not allowed to enter immediately after agent 1 leaves.

### B. UAV Mission Planning

We synthesized shields for 3 unmanned aerial vehicles (UAVs) simulated using ROS/Gazebo[1] for the case study outlined in Section III. As shown in Figure 1, the environment consists of 2 rows of shelves. We assume there are 12 discrete package pick up points in each row of shelves along with the drop off location and charging station. The input of each UAV controller is its location in $(x, y, z)$. The possible outputs of each UAV consist of 17 trajectories precomputed using a minimum-snap trajectory generator that moves the UAV from one discrete state to another. We consider a safety specification $\varphi_{dist}$ which captures the requirement that the controllers should not choose trajectories that bring them within distance less than a given threshold $r$. The output of each UAV is chosen from the set $T = \{1, 2, \ldots, 17\}$ where each integer corresponds to a particular trajectory choice. Let the function $dist : T \times T \rightarrow \mathbb{N}$ map each pair of trajectories $traj_i, traj_j \in T$ to the closest distance between them. We define the safety specification as

$$\varphi_{dist} = \square \wedge_{i \neq j} \big(dist(traj_i, traj_j) \geq r\big). \tag{2}$$

[1]We thank Jesse Quattrociocchi for his help with the simulations.

Additionally, we specify $\varphi_{shelf}$ which states that no more than one UAV can move into the row of shelves at the same time. Lastly, like VII-A, only one UAV can be at the charging station or drop-off point at any given time and UAVs cannot be allowed to run out of power when not in a charging station.

We assign a higher cost to the interference with the orange UAV compared to the other two. Intuitively, this will force the shield, where possible, to avoid interfering with the orange UAV. We synthesize the shield for two different values of $b$ and the synthesis times are reported in Table I under case (2). A video of the simulation can be seen in `https://bit.ly/2OSRhxJ`.

## VIII. Conclusion

In this paper, we proposed a general approach to the synthesis of shields for multi-agent systems from temporal logic specifications. Our key contribution is the study of quantitative objectives in the shield synthesis setting. Our work is also the first to consider fairness requirements on the shields. We introduced the notion of interference cost, and discussed several costs and synthesis objectives that are of interest when considering multi-agent systems. We demonstrated the applicability of the proposed approach on a range of quantitative interference requirements by synthesizing shields for a multi-UAV system.

A promising avenue for future work is to investigate bounded synthesis [11] with quantitative objectives, in order to synthesize distributed shields, which will enhance the efficiency of shields for distributed systems.

**Acknowledgement:** This work was supported in part by grant DARPA W911NF-16-1-0001, grant ARO W911NF-15-1-0592, and grant NSF 1652113.

## References

[1] Netflix: 5 lessons we have learned using aws (2010).

[2] *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*. IEEE Computer Society, 2006.

[3] Andreas Bauer and Yliès Falcone. Decentralised LTL monitoring. *Formal Methods in System Design*, 48(1-2):46–93, 2016.

[4] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, 2011.

[5] Luca Bertuccelli, Han-Lim Choi, Peter Cho, and Jonathan How. Real-time multi-uav task assignment in dynamic and uncertain environments. In *AIAA guidance, navigation, and control conference*, page 5776, 2009.

[6] Roderick Bloem, Bettina Könighofer, Robert Könighofer, and Chao Wang. Shield synthesis. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 533–548. Springer, 2015.

[7] Swarat Chaudhuri and Azadeh Farzan, editors. *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, volume 9780 of *Lecture Notes in Computer Science*. Springer, 2016.

[8] Yliès Falcone. You should better enforce than verify. In *Runtime Verification - First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings*, pages 89–105, 2010.

[9] Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. What can you verify and enforce at runtime? *STTT*, 14(3):349–382, 2012.

[10] Yliès Falcone, Mohamad Jaber, Thanh-Hung Nguyen, Marius Bozga, and Saddek Bensalem. Runtime verification of component-based systems in the BIP framework with formally-proved sound and complete instrumentation. *Software and System Modeling*, 14(1):173–199, 2015.

[11] Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *STTT*, 15(5-6):519–539, 2013.

[12] Adrian Francalanza, Jorge A. Pérez, and César Sánchez. Runtime verification for decentralised and distributed systems. In *Lectures on Runtime Verification - Introductory and Advanced Topics*, pages 176–210. 2018.

[13] Adrian Francalanza and Aldrin Seychell. Synthesising correct concurrent runtime monitors. *Formal Methods in System Design*, 46(3):226–261, 2015.

[14] Gangyuan Jing, Rüdiger Ehlers, and Hadas Kress-Gazit. Shortcut through an evil door: Optimality of correct-by-construction controllers in adversarial environments. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7, 2013*, pages 4796–4802, 2013.

[15] Atishkumar Kalyan and Steven Gregory Dunn. Automated inventory management system, December 22 2015. US Patent 9,216,857.

[16] Bettina Könighofer, Mohammed Alshiekh, Roderick Bloem, Laura Humphrey, Robert Könighofer, Ufuk Topcu, and Chao Wang. Shield synthesis. *Formal Methods in System Design*, 51(2):332–361, 2017.

[17] Jay Ligatti, Lujo Bauer, and David Walker. Run-time enforcement of nonsafety policies. *ACM Trans. Inf. Syst. Secur.*, 12(3):19:1–19:41, 2009.

[18] Qingzhou Luo and Grigore Rosu. Enforcemop: a runtime property enforcement system for multithreaded programs. In *International Symposium on Software Testing and Analysis, ISSTA '13, Lugano, Switzerland, July 15-20, 2013*, pages 156–166. ACM, 2013.

[19] René Mazala. Infinite games. In Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors, *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*, pages 23–42. Springer, 2001.

[20] Craig Olivo and Michael Buzaki. Method and apparatus for warehouse cycle counting using a drone, August 25 2016. US Patent App. 15/013,029.

[21] Jin Hock Ong, Abel Sanchez, and John Williams. Multi-uav system for inventory automation. In *RFID Eurasia, 2007 1st Annual*, pages 1–6. IEEE, 2007.

[22] Corina S. Pasareanu and Darko Marinov, editors. *International Symposium on Software Testing and Analysis, ISSTA '14, San Jose, CA, USA - July 21 - 26, 2014*. ACM, 2014.

[23] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977.

[24] Steven Rasmussen, Derek Kingston, and Laura Humphrey. A brief introduction to unmanned systems autonomy services (uxas). In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 257–268. IEEE, 2018.

[25] Tariq Samad, John S Bay, and Datta Godbole. Network-centric systems for military operations in urban terrain: The role of uavs. *Proceedings of the IEEE*, 95(1):92–107, 2007.

[26] Sven Schewe. *Synthesis of distributed systems*. PhD thesis, Saarland University, Saarbrücken, Germany, 2008.

[27] Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.

[28] Eduard Semsch, Michal Jakob, Dušan Pavlicek, and Michal Pechoucek. Autonomous uav surveillance in complex urban environments. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 02*, pages 82–85. IEEE Computer Society, 2009.

[29] PB Sujit, A Sinha, and D Ghose. Multi-uav task allocation using team theory. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 1497–1502. IEEE, 2005.